UNIVERSIDADE DE VIGO

Dpto. de Enxeñería Telemática
ETSE de Telecomunicación

TESIS DOCTORAL

TRATAMIENTO FORMAL DE SISTEMAS CON REQUISITOS DE TIEMPO REAL CRÍTICOS

Autora: Ana Fernández Vilas
Ingeniera de Telecomunicación

Director: José J. Pazos Arias
Doctor Ingeniero de Telecomunicación

Enero de 2002

Dpto. de Enxeñería Telemática
ETSE de Telecomunicación
Universidade de Vigo
Campus Universitario s/n
E-36200 Vigo

TESIS DOCTORAL

TRATAMIENTO FORMAL DE SISTEMAS CON REQUISITOS DE TIEMPO REAL CRÍTICOS

Autora: Ana Fernández Vilas
Ingeniera de Telecomunicación

Director: José J. Pazos Arias
Doctor Ingeniero de Telecomunicación

Enero de 2002

TESIS DOCTORAL

TRATAMIENTO FORMAL DE SISTEMAS CON REQUISITOS DE TIEMPO REAL CRÍTICOS

Autora: D. Ana Fernández Vilas

Director: Dr. D. José J. Pazos Arias

TRIBUNAL CALIFICADOR

Presidente: Dr. D.

Vocales:

Dr. D.

Dr. D.

Dr. D.

Secretario: Dr. D.

CALIFICACIÓN:

Vigo, a de de .

A mi madre, siempre estaremos contigo.
A mi padre

Agradecimientos

A Dr. Jose J. Pazos Arias por darme la oportunidad y el apoyo en este trabajo de tesis. También por su humanidad que, muy a menudo, no está presente en los entornos de trabajo.

Quisiera también agradecer a la Xunta de Galicia el apoyo que nos ha prestado. La financiación recibida a través de los proyectos XUGA 32206A97 y PGIDT01PXI32203PR nos ha permitido realizar este trabajo con los medios materiales apropiados.

A Dr. Alberto Gil Solla por haber asistido al inicio y al fin de este trabajo. Y qué mejor que compañeros como vosotros. A Andrés, Ardao, Cándido, Belén, Estela, Jorge, Manuel, Manolo y Raúl, por lo que necesité y, por lo que habiendo podido necesitar, hubiese ocasionado una intervención rápida y, por supuesto, eficiente de cada uno de vosotros.

¿Y Rebeca?. Muy especialmente a Rebeca por compartir conmigo su vida profesional, y su otra vida, si cabe más profesional que la primera. Por mantenerme atualizada en los gustos musicales, y por todos esos momentos con Pedro que nos quedan por pasar. Quiero recordarte estos últimos meses, tu seguridad y confianza han sido importantísimas para llegar hasta aquí. Si la profesión nos separa, siempre quedará esa semana de Febrero en Tenerife.

Aunque afortunadamente desligados de los avatares de esta tesis: A mis hermanos Rosi, Julio, Lucía y Eva, y mis cuñados Jose, Bea y Fonsi, por hacer de la familia ese "especial" grupo de amigos, espero que cada vez más grande. A mis amigos y familia de Cangas, por tener ilusiones y objetivos tan alejados de los míos, aportándome la diversidad que ahora ya me resulta imprescindible. Y aunque exiliada en el Morrazo, no me puedo olvidar de mis amigos de Pontevedra, y de todos aquellos que por placer o por trabajo no están tan cerca de mí como yo quisiera. Sé que todos vais a volver a Galicia. Por último, a Amalia y a toda su aún pequeña familia, espero enumerarte las razones dentro de muhos años.

Resumen

Esta tesis se enmarca dentro de la disciplina del tratamiento formal de los sistemas de tiempo real críticos. Un **sistema de tiempo real** es aquel cuya correcta ejecución depende de interacciones cuidadosas y orquestadas con el tiempo. Para que un sistema de computación de tiempo real sea aceptable no sólo debe ser funcionalmente correcto - producir resultados correctos - sino que también debe ser temporalmente correcto - hacerlo dentro de los intervalos especificados. Adicionalmente, un sistema de tiempo real crítico acarrea el riesgo de pérdidas graves cuando se producen tiempos de respuesta que caen fuera de los límites impuestos en la especificación.

El diseño de sistemas de tiempo real es, de por sí, intrínsecamente complejo. Tal complejidad tiene su origen en características específicas como lo crítico de su ejecución, su inherente concurrencia, la existencia de eventos no predecibles, etc. A pesar de su elevada complejidad, el uso de métodos formales se ha retrasado con respecto a los sistemas sin noción explícita de tiempo. Parte de este retraso se debe a la complejidad derivada de la incorporación del tiempo como requisito funcional del sistema, además el diseño de estos sistemas es dependiente, en gran medida, de características propias de su implementación.

Una vez alcanzada una razonable madurez en los fundamentos teóricos de los métodos formales de tiempo real, la historia de su aplicación ha sido similar a la ya vivida para los métodos formales sin tiempo, es decir, reaparecen los mismos problemas para llegar a una transferencia tecnológica real. El acercamiento a la práctica profesional pasa por integrar los métodos formales en el proceso *software*, proporcionando entornos integrados que incorporen facilidades que van desde el análisis estático basado en verificación formal y teoría de la planificación, al análisis dinámico basado en prueba y monitorización en tiempo de ejecución.

Por lo que respecta a la estructura del proceso *software*, los requisitos para sistemas grandes y complejos son a menudo difíciles de identificar en las primeras fases, ya que evolucionan de forma continuada a lo largo del ciclo de vida *software*. Por lo tanto, los procesos *software* deberían ser lo suficientemente flexibles para acomodarse a los cambios en los requisitos. Aun es más, en las primeras fases, el diseñador no suele tener un conocimiento profundo del sistema y, en cualquier caso, muchas veces la complejidad es excesiva para un diseño de un solo paso. Los ciclos de vida tradicionales en cascada se han revelado, en este aspecto, del todo inadecuados. Los métodos formales deben adaptarse para soportar evolución en el ciclo de vida, saliendo de su papel tradicional de verificar que un modelo cumple un conjunto fijo de requisitos.

Una vez han sido completa o parcialmente resueltos los principales problemas en el ámbito de los métodos formales para sistemas de tiempo real, nuestro objetivo es ligeramente diferente; con el fin de mejorar la calidad del *software* de tiempo real, es nuestra intención conjugar dos soluciones de diferente naturaleza: la **formalización** del proceso *software* (aprovechando las ventajas del uso de FDTs) y un enfoque de proceso *software* **iterativo e incremental** (acomodándose a los cambios en los requisitos a lo largo del ciclo de vida). De esta forma combinamos la corrección de los métodos formales, y lo flexible de un ciclo de vida que no congela los requisitos en fases tempranas y divide la complejidad.

Esta tesis es una extensión, para tiempo real, de los estudios realizados en el

ámbito de los sistemas reactivos sin tiempo por el grupo de Redes e Ingeniería del *Software* del Departamento de Ingeniería Telemática de la Universidad de Vigo. El principal resultado de dichos estudios ha sido el desarrollo de la metodología formal SCTL/MUS. Dicha metodología propone el uso de la lógica multivalor SCTL, y el modelo de estados con subespecificación MUS, como la combinación de métodos formales para la articulación de un modelo de proceso *software* iterativo con prototipado y totalmente formalizado.

La **extensión de la metodología formal SCTL/MUS** ha supuesto la revisión de las necesidades que un sistema de tiempo real impone en el modelo de ciclo de vida. Una de las conclusiones inferidas es la adecuación de un modelo de ciclo de vida incremental e iterativo a este tipo de sistemas, en los que es especialmente importante estudiar alternativas de diseño y prototipos desde las primeras fases del ciclo de vida. Independientemente de la estructura del modelo de ciclo de vida utilizado, existe un amplio consenso en que, especialmente en los sistemas con características de tiempo real, el tiempo debe ser tenido en cuenta desde las primeras fases del ciclo de vida. En base a tal aseveración se hace del todo necesario la adecuación tanto del lenguaje de especificación de requisitos SCTL como de la especificación orientada a modelo MUS.

Con el ánimo de realizar dicha extensión ha sido necesario hacer una revisión del estado del arte en busca de la forma más adecuada de incorporar el dominio temporal. Tal revisión nos permite afirmar que los **modelos densos** son más adecuados para composición y refinamiento, además de no imponer la necesidad de seleccionar un cuanto de tiempo a priori. Si a la flexibilidad se suma el hecho de que los procedimientos de decisión sobre dominios densos son cada vez más eficientes, habremos dibujado las principales justificaciones de dicha elección.

Adicionalmente a la definición de la lógica SCTL-T y el modelo de estados MUS-T, extensiones con tiempo denso de SCTL y MUS, es necesario aportar los **métodos de tratamiento formal** en que se basa el modelo de ciclo de vida propuesto. Consideramos de especial importancia, en este sentido, introducir diferentes tipos de análisis, que van desde la garantía de una corrección mínima centrada en la detección de bloqueos y la validación basada en la ejecución simbólica de la especificación formal, hasta una garantía máxima en base a verificación formal. La verificación formal de requisitos SCTL-T sobre modelos MUS-T, basada en *model checking*, opera sobre una abstracción finita del espacio de estados, infinito y denso, del modelo.

Asimismo, se define un método de síntesis incremental, basado en *model checking*, que actualiza los prototipos MUS-T a partir de nuevos requisitos expresados en SCTL-T. La síntesis incremental, como definitoria de un modelo de proceso iterativo e incremental, respeta tanto el nivel de especificación alcanzado en el prototipo, como el nivel de conocimiento de las propiedades exigidas al prototipo.

Finalmente, si bien el grueso de esta tesis no está centrado en los aspectos de eficiencia relacionados con los métodos de decisión propuestos, se aporta una solución que habilita la aplicación de dichos métodos a casos de estudio de tamaño medio-grande. Para ello se define una bisimulación de abstracción de tiempo fuerte sobre modelos MUS-T y, en base a dicha relación, se propone un algoritmo de minimización que permite aliviar la explosión de tamaño del espacio de estados asociada a los algoritmos de *model checking* y síntesis incremental.

Abstract

This thesis focuses on formal treatment of hard real time systems. A real-time system is one in which the correctness of the system depends not only on the logical results, but also on the time at which results are produced. Additionally, a hard real time system results in serious damages when response times are outside the limits in the specification.

Real time systems design are, per se, intrinsically complex. Such complexity has its origins in the specific characteristics of this kind of systems: criticalness, inherent concurrency, the presence of non predictable events, etc. In spite of such great complexity, there can not be any doubt that real-time formal methods have been delayed with respect to untimed ones, going back few years. This delay is largely due to the complexity derived from integrating time as a functional requirement. Moreover, in this kind of systems, design often depends on implementation details (hardware speed, resource sharing, etc.) that are essential issues when deciding if it is feasible to fulfill timing requirements.

Once maturity in formal basis of timed systems has been reached, these timed theoretical foundations come up against industry reluctance about formal methods adoption in the software process; technology-transfer problems, which had been discovered in conventional untimed systems, appear once again. In order to get closer to the professional practice, formal methods are in urgent need of being incorporated into the software process and being supported by integrated environments which provide from static verification, by means of formal verification or schedulability theory, to dynamic analysis based on test or run time monitoring.

With regard to the structure of the software process, requirements, in large and complex systems, are often difficult to elicit in the earlier phases, since these requirements continuously evolve throughout the life cycle. Therefore, software process models should be flexible enough to follow the inevitable and often continuous stream of changes. What is more, usually, in earlier phases, designer has no a deep knowledge about the system and, in any case, the complexity could be excessive for a one-step design. In this respect, traditional cascade life cycles have showed themselves unlikely suitable. Formal methods should be adapted in order to support evolution in life cycle, getting over their traditional role of verifying that a model meets a set of fixed requirements.

Once major problems in real time formal methods have been completely or partially resolved, our aim is slightly different. In order to improve software quality, our intend is to merge two different nature solutions: the **formalization** of the software process (gaining the advantages of FDTs); and a process approach which is both **iterative and incremental** (fitting in with requirements change) over the whole life cycle. In this way we combine the correctness of formal methods, and the suitability of a life cycle that follows user requirements and splits complexity.

This thesis is a timed extension of previous works, in the field of untimed reactive and distributed systems, tackled by "Redes e Ingeniería del *Software*" group in Department of Telematics Engineering (Vigo University). The main result of these works is the SCTL/MUS formal methodology. This methodology suggests the use of the many-valued temporal logic SCTL, and the model of unspecified states MUS, as a combination of formal methods to articulate a totally formal software process model which is iterative an prototyping-based.

Extending the SCTL/MUS methodology has entailed the inspection of the special characteristics that a real time system forces in a life cycle model. Since, specially in this kind of systems, it is advisable to inspect design alternatives and provide prototypes from the earlier phases, one of the resulting conclusions is that an incremental and iterative life cycle fits appropriately. Apart form the life cycle structure, there is a broad consensus in that time has to be taken into account from the earlier phases in this kind of systems. On the basis of this statement, it is totally needed extending both the requirements language SCTL, and the model-oriented specification MUS-T.

With the aim of achieving these extensions, the state of the art has been revised in order to find the right way of incorporating the time domain. This revision allow us to state that **dense models** are more suitable for composition and refinement, besides, these models do not impose a fixed quantum of time a priori. In addition to flexibility, the decision methods for dense domains are increasingly efficient every day.

Apart from defining the real time logic SCTL-T and the timed model MUS-T, real time extensions of their untimed counterparts, it has been necessary to provide the formal methods on which the proposed life cycle model is based. We consider providing different kinds of analysis methods is particularly important. Analysis covers from minimal correction guarantee, by means of validation and lock detection, to more general guarantee based on formal verification. The formal verification of SCTL-T requirements over MUS-T models, model-checking based, is computed by means of a finite abstraction of the infinite and dense state space of the model.

Also, it is defined a method of incremental synthesis which adapts a prototype MUS-T form a new set of requirements expressed in SCTL-T. The incremental synthesis, as defining an iterative and incremental process model, respects both the specification level reached by the prototype, and the knowledge level of the properties which are demanded in the prototype.

Finally, although the primary aim of this thesis is not focused on efficiency issues related to formal analysis, a solution which makes the methodology amenable to medium-large case studies is provided. For that, it is defined both a strong time abstracting bisimulation for MUS-T, and a minimization algorithm which, on the basis of that relation, reduces the explosion of state space linked to model checking and incremental synthesis algorithms.

Índice

I Introducción **1**

1 Ámbito y objetivos de la tesis **3**

1.1 Sistemas de Tiempo Real (RTS) 3

1.2 Requisitos de tiempo 5

1.3 Tratamiento formal de sistemas de tiempo real 7

1.3.1 Modelado de concurrencia en RTS 9

1.3.2 Modelado de tiempo en RTS 10

1.4 Cronología del tratamiento formal en RTS 12

1.5 Conclusiones y motivación de la tesis 15

1.6 La metodología formal SCTL/MUS 17

1.7 Objetivos 18

II Estado del Arte **21**

2 Especificación formal para RTS **23**

2.1 Introducción 23

2.2 Lógicas 24

2.2.1 Lógicas temporales 24

2.2.2 Expresividad y complejidad 29

2.2.3 Otras lógicas 32

2.3 Álgebras de Procesos 33

2.4 Máquinas de estados 35

2.4.1 Variantes del autómata temporizado 37

3 Análisis y síntesis formal para RTS **39**

3.1 Introducción 39

3.2 Verificación formal 40
3.2.1 *Proof-Theoretic* 41
3.2.2 *Model-Theoretic* 41
3.3 *Model Checking* 42
3.3.1 Mejoras a la verificación basada en regiones 43
3.4 Herramientas de verificación 47
3.4.1 RT-SPIN 48
3.4.2 Timed-COSPAN 48
3.4.3 KRONOS 48
3.4.4 UPPAAL 48
3.5 Aplicación a problemas reales 49
3.6 Otras formas de análisis 49
3.6.1 Simulación simbólica 50
3.6.2 Prueba 51
3.6.3 Análisis de planificabilidad 51
3.7 Síntesis 51

4 Trabajo relacionado 55
4.1 Introducción 55
4.2 Formalismos híbridos 56
4.3 Sistemas de tiempo real *soft* 56
4.4 Lógicas multivaloradas 57

III Metodología formal SCTL/MUS-T 59

5 Proceso *Software* 61
5.1 Introducción 61
5.2 Modelos de ciclo de vida 62
5.3 Los modelos de ciclo de vida y las técnicas formales 63
5.4 Los modelos de ciclo de vida y los sistemas de tiempo real 65
5.5 Revisión del ciclo de vida SCTL/MUS 66
5.6 Modelo de ciclo de vida SCTL/MUS-T 69
5.6.1 *Objetivos iniciales* 69
5.6.2 Características específicas del ámbito de tiempo real 72
5.7 Conclusiones 73

6 MUS-T **75**
6.1 Introducción . . . 75
6.2 De MUS a MUS-T . . . 76
6.2.1 Evolución discreta . . . 80
6.2.2 Evolución temporal . . . 81
6.3 Algo más que el control discreto . . . 81
6.4 Preliminares: relojes, interpretaciones, predicados de tiempo y estados . . 84
6.5 Sintaxis . . . 86
6.5.1 Completitud y no solapamiento de condiciones de especificación . . 87
6.5.2 Localización sumidero subespecificado y no posible . . . 88
6.6 Semántica . . . 89
6.6.1 Estados . . . 89
6.6.2 Transiciones . . . 89
6.6.3 Semántica operacional . . . 90
6.6.4 Nivel de especificación de estados . . . 92
6.6.5 Modelos MUS-T deterministas . . . 93
6.6.6 Modelos MUS-T no deterministas . . . 93
6.7 Requisitos de avance en un sistema temporizado . . . 93
6.8 Composición . . . 94
6.9 Traducción MUS-T a *Timed Automaton* . . . 96
6.10 Conclusiones . . . 98

7 SCTL-T **99**
7.1 Introducción . . . 99
7.2 Valores de verdad en SCTL . . . 100
7.3 Semántica con tiempo de los operadores temporales . . . 101
7.3.1 El operador temporal *a la vez* . . . 102
7.3.2 Los operadores temporales *antes* y *después* temporizados . . . 104
7.3.3 Cuantificación . . . 107
7.3.4 Operadores de pasado . . . 109
7.3.5 En resumen . . . 110
7.4 IPM: Álgebra de **I**ncretidumbre del **P**unto **M**edio . . . 110
7.4.1 Introducción a IPM . . . 110
7.4.2 Definición alternativa en función de un orden parcial . . . 113
7.4.3 Obtención de $\mathcal{L}_6$ a partir de $\mathcal{L}_3$ . . . 115
7.4.4 Orden en nivel de conocimiento . . . 115
7.5 Sintaxis . . . 116
7.6 Semántica . . . 119

7.6.1 Grado de satisfacción 119
7.6.2 Grado de satisfacción de los predicados de tiempo 120
7.6.3 Grado de satisfacción de acciones 121
7.6.4 Grado de satisfacción de constantes proposicionales de estado . . . 121
7.6.5 Grado de satisfacción de fórmulas causales 121
7.7 Forma Normal Positiva 123
7.8 Grado de satisfacción de una fórmula SCTL-T 124
7.9 Recursión e Instanciación en SCTL-T 125
7.9.1 Definiciones preliminares 125
7.9.2 Sintaxis y Semántica de la recursión e instanciación 126
7.9.3 Monotonicidad de transformadores 128
7.9.4 Dualidad de recursiones mínima y máxima 129
7.10 Operadores macro en SCTL-T 130
7.10.1 Operadores macro 131
7.10.2 Expresividad 133
7.11 Conclusiones 133
7.A Almacenamiento de Requisitos 135
7.B Algoritmo de requisitos bien formados y acciones de aplicabilidad 135

IV Mecanismos de análisis y síntesis en el proceso 139

8 *Model Checking* SCTL-T sobre MUS-T 141
8.1 Introducción 141
8.2 Regiones de tiempo (*clock regions*) 143
8.3 Grafo de regiones RMUS-T 146
8.3.1 Estados interpretados del grafo de regiones 146
8.3.2 Transiciones 147
8.3.3 Construcción del grafo de regiones RMUS-T 149
8.4 *Model Checking* SCTL/MUS-T 150
8.4.1 Operador estados de aplicabilidad sobre regiones 151
8.4.2 Relojes de especificación 153
8.4.3 Grado de satisfacción de proposiciones atómicas 155
8.4.4 Cómputo de recursiones mínimas y máximas 156
8.4.5 Grado de satisfacción de una fórmula SCTL-T 157
8.4.6 Algoritmo de verificación SCTL-T sobre RMUS-T 158
8.5 Conclusiones 158
8.A Implementación 161

8.A.1 Representación de regiones 161
8.A.2 Predecesor y sucesor en tiempo 161

9 Síntesis 165

9.1 Introducción 165
9.2 Crecimiento de un modelo MUS-T 167
9.3 Esquema de síntesis incremental en SCTL/MUS-T 167
9.4 Consideraciones previas 169
9.5 Reglas de síntesis 171
9.6 Relojes en las reglas de síntesis 173
9.7 Formas básicas de síntesis en una región 175
9.8 Significado de la parte imperativa 176
9.8.1 Síntesis de proposiciones atómicas 176
9.8.2 Síntesis de fórmulas causales 177
9.8.3 Síntesis de elementos $a\{x\}$ 178
9.9 Ejemplos de síntesis 179
9.9.1 Formas básicas 179
9.9.2 Reglas de síntesis 181
9.10 Selección de un estado de control sucesor discreto 183
9.10.1 Criterios de compatibilidad entre regiones 185
9.11 Recomputaciones necesarias y notificación de conflicto 188
9.11.1 Notificación de conflicto 189
9.12 Algoritmo de Síntesis sobre MUS-T 189
9.12.1 Esbozo del Algoritmo 189
9.12.2 Aplicación de una regla de síntesis en una región RMUS-T 190
9.13 Traducción RMUS-T a MUS-T 190
9.14 Conclusiones 192
9.A Implementación 193
9.A.1 Obtención de los predicados MUS-T 193

10 Minimización y corrección del algoritmo de síntesis 195

10.1 Introducción 195
10.2 Definiciones preliminares 196
10.2.1 Simulaciones y Bisimulaciones 196
10.2.2 Propiedades de las simulaciones y bisimulaciones 197
10.2.3 Particularización para MUS-T 199
10.3 Bisimulación de abstracción de tiempo fuerte ($\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$) 199
10.3.1 Propiedades de STaB 201

10.3.2 Preservación de fórmulas SCTL-T 202
10.3.3 *Model Checking* y síntesis SCTL-T 203
10.4 Simulación de abstracción de tiempo especificada ($\mathrm{ETaS}_{\text{MUS-T}}$) 204
10.4.1 $\mathrm{ETaS}_{\text{MUS-T}}$ es más débil que $\mathrm{STaB}_{\text{MUS-T}}$ 205
10.4.2 Corrección del algoritmo de síntesis 205
10.4.3 Preservación del nivel de conocimiento 205
10.5 Relación de abstracción de tiempo total posible ($\mathrm{TP}_{\text{MUS-T}}$) 207
10.6 Análisis basado en minimización 208
10.6.1 Esbozo de los algoritmos de minimización 209
10.6.2 Particularización para MUS-T 211
10.6.3 Manipulación de los elementos de la partición 211
10.6.4 Algoritmo de minimización 213
10.6.5 Post-estabilización para el cálculo de operadores de pasado 213
10.6.6 Verificación y síntesis basada en minimización para SCTL-T 215
10.7 Conclusiones 215

11 Grafo de simulación: animación y garantía de avance 217
11.1 Introducción 217
11.2 Abstracción en el grafo de simulación 218
11.3 Operadores de simulación 218
11.4 Grafo de simulación posible para MUS-T 220
11.5 Propiedades del grafo de simulación 220
11.5.1 Pre-estabilización del grafo de simulación 221
11.5.2 Corrección mínima del modelo 222
11.5.3 Detección de *deadlocks* 224
11.5.4 Detección de *timelocks* 227
11.6 Conclusiones 231

V Caso de estudio y conclusiones 233

12 Caso de estudio 235
12.1 Introducción y motivación del caso de estudio 235
12.2 Especificación informal 237
12.3 Arquitectura 238
12.3.1 Arquitectura del sistema para modo normal 238
12.3.2 Otros modos de operación 239
12.4 Modelado del entorno: totalmente especificado 240
12.4.1 Modelado del entorno sin fallos 240

12.4.2 Modelado del entorno con fallos 241
12.4.3 Consideraciones sobre el modelo 242
12.4.4 Propiedades del entorno 243
12.5 Propiedades del sistema cerrado 244
12.5.1 Entorno si fallos 244
12.5.2 Entorno con fallos 244
12.6 Especificación inicial del controlador (sin fallos) 245
12.7 Síntesis del controlador I (sin fallos) 248
12.7.1 Corrección mínima del modelo 254
12.7.2 Incorporación de la generación de informes 257
12.7.3 Verificación de propiedades sobre el sistema cerrado (sin fallos) . . 259
12.8 Síntesis del controlador II (con fallos) 260
12.8.1 Especificación informal de los modos de fallo 260
12.8.2 Especificación formal de los modos de fallo 260
12.8.3 Síntesis del controlador II (con fallos) 261
12.8.4 Incorporación de parada de emergencia 263
12.8.5 Verificación de propiedades sobre el sistema cerrado (con fallos) . . 265
12.8.6 Protección frente a errores de bomba no detectados 265
12.9 Utilización del algoritmo de minimización 267
12.9.1 Ejemplo de post-estabilización 270
12.10 Conclusiones 270

13 Conclusiones y líneas de trabajo futuro 275
13.1 Conclusiones 275
13.2 Líneas de trabajo futuro 281
13.2.1 Directas 281
13.2.2 Indirectas 283

VI Apéndices y bibliografía 287

Apéndice A. DBMs 289
A.1 Poliedros 289
A.2 DBM y representación de poliedros convexos 290
A.3 Relojes de especificación 291
A.4 DBM canónico 291
A.4.1 Obtención de la forma canónica 292
A.4.2 Detección de poliedros vacíos 293

A.4.3 Aplicación de un algoritmo *all-pairs shortest path* para obtener la forma canónica . . . 293
A.4.4 DBM universal . . . 293
A.5 Operaciones sobre DBMs . . . 295
A.5.1 Intersección o conjunción . . . 295
A.5.2 Inclusión o implicación . . . 295
A.5.3 Sucesores en tiempo . . . 297
A.5.4 Predecesores en tiempo . . . 297
A.5.5 *until* . . . 298
A.5.6 *since* . . . 298
A.5.7 Reset de un conjunto de relojes . . . 298
A.5.8 PreReset . . . 298
A.5.9 *cierre* . . . 299
A.5.10 Unión o disyunción . . . 299
A.6 Representación de poliedros no convexos: listas de DBMs . . . 300
A.6.1 Complemento . . . 300
A.6.2 Intersección . . . 301
A.6.3 Inclusión . . . 301
A.6.4 Reset, Sucesores, Predecesores, PreReset y cierre . . . 301
A.6.5 *until* y *since* . . . 301
A.6.6 Reducción . . . 302
A.7 Expresión de guardas de tiempo de MUS-T en DBMs . . . 302

Bibliografía **305**

PARTE I

Introducción

CAPÍTULO 1

Ámbito y objetivos de la tesis

Esta tesis se enmarca en el ámbito de la aplicación de técnicas formales para la obtención de sistemas software con características de tiempo de naturaleza crítica. El principal objetivo de esta tesis es la formalización de un modelo de ciclo de vida iterativo e incremental, para lo cual se propone la extensión de la metodología formal SCTL/MUS (García Duque, 2000).

1.1 Sistemas de Tiempo Real (RTS)

La noción de sistema de tiempo real (RTS) tiene interpretaciones muy variadas. En no pocas ocasiones, este término se ha utilizado para referirse a un cómputo muy rápido, a la adquisición inmediata de datos *on line*, y a la computación guiada por plazos. Encontramos en la literatura multitud de definiciones, pero quizás la definición más comúnmente aceptada, por su generalidad y concisión, es la formulada por *Stankovic*:

> (Stankovic, 1988): *"A real-time system is one in which the correctness of the system depends not only on the logical results, but also on the time at which results are produced"*

Un sistema de computación en tiempo real es aquel que debe obtener sus resultados dentro de intervalos de tiempo específicos, su correcta ejecución depende de interacciones cuidadosas y orquestadas con el tiempo, ya sea mediante el cumplimiento de plazos, sincronización con relojes, etc. Para que un sistema de computación de tiempo real sea aceptable no sólo debe ser funcionalmente correcto – producir resultados correctos – sino que también debe ser temporalmente correcto – hacerlo dentro de los intervalos especificados –.

La computación de tiempo real se distingue de otras formas de computación por la incorporación explícita de la dimensión **tiempo**. Ante una petición de un

proceso externo, la adquisición de datos, su evaluación y las respuestas adecuadas deben ser realizadas "a tiempo". Este comportamiento no requiere necesariamente rapidez de procesado, sino más bien la "puntualidad" de las respuestas dentro de límites predefinidos y previsibles.

Mientras que el objetivo de una computación rápida es minimizar el tiempo medio de respuesta de un conjunto de tareas, el objetivo de la computación de tiempo real es hacer frente a los requisitos individuales de duración de cada tarea. Por tanto podemos decir que la restricción más importante que impone un sistema de tiempo real no es su rapidez, sino más bien su previsibilidad (Halang., 1990), es decir, su comportamiento funcional y su "ritmo" debe ser tan determinista como sea necesario para satisfacer las especificaciones del sistema.

Atendiendo a sus requisitos de puntualidad se suelen distinguir dos tipos de sistemas de tiempo real:

- *Hard real time systems* (RTS críticos): En estos sistemas un tiempo de respuesta fuera de los límites impuestos en la especificación supone riesgo de pérdidas graves. La función de coste asociada al incumplimiento del plazo se caracteriza por tener un incremento rápido discontinuo (se trata de un plazo "duro").

- *Soft real time systems* (RTS acríticos): En estos sistemas las respuestas no puntuales no son críticas, comprometiéndose sólo la utilidad y/o precisión de los resultados. En este caso los costes asociados a la terminación de la tarea después del plazo se incrementan gradualmente como una función continua del tiempo (el plazo es "blando").

La computación de tiempo real es un ámbito que sustenta diferentes áreas de aplicación: control de procesos, plantas nucleares, aereoelectrónica, control de tráfico aéreo, telecomunicaciones (autopistas de la información), realidad virtual, aplicaciones médicas (telemedicina y monitorización), aplicaciones de defensa, etc. En concreto, casi todos los sistemas críticos y muchos sistemas empotrados son sistemas de tiempo real.

El diseño de los sistemas de tiempo real es, de por sí, intrínsecamente complejo. Tal complejidad tiene su origen en las características específicas de este tipo de sistemas: su inherente concurrencia, la existencia de eventos no predecibles, las limitaciones de memoria y procesamiento, su comportamiento ante carga variable, etc. Si bien el uso de técnicas formales parece adecuado en el proceso de diseño para este tipo de sistemas de elevada complejidad, su aplicación a los sistemas de tiempo real es relativamente reciente (sección 1.3).

El tratamiento formal de este tipo de sistemas se ha abordado, una vez alcanzada la madurez en la formalización de sistemas sin tiempo, extendiendo una gran cantidad de formalismos recurrentemente investigados en el área de las técnicas formales sin tiempo. Parte de este retraso se debe a la complejidad derivada de la incorporación del tiempo como requisito funcional del sistema. Además el diseño de estos sistemas es dependiente en gran medida de características propias de su implementación, como la velocidad del *hardware* o la política de compartición de recursos, que son esenciales a la hora de determinar cuándo es posible cumplir los requisitos de tiempo estrictos. En el *software* convencional, por cuestiones de

robustez, se ocultan tales dependencias de forma deliberada. La imposición de restricciones de tiempo reintroduce estas dependencias.

A pesar de la amplia variedad de sistemas recogidos bajo el término "sistemas de tiempo real", una característica común se encuentra presente en todos y cada uno de ellos: los requisitos de tiempo cuantitativos (sección 1.2). En general, dentro del ámbito de la ingeniería de requisitos, no existe acuerdo en cómo catalogar los requisitos de tiempo. Por un lado, es coherente pensar que los requisitos de tiempo estrictos que impone un RTS crítico son en sí requisitos funcionales, pues son requisitos que han de cumplirse para que el sistema sea considerado correcto. En otros sistemas, los requisitos de tiempo se incluyen dentro de los requisitos ligados a la implementación del sistema, con lo cual estarán dentro del grupo de requisitos no funcionales (eficiencia). No es nuestro objetivo establecer una taxonomía de requisitos de RTSs, sino que, en cualquier caso, se presenta la necesidad de poder expresar y verificar requisitos de tiempo cuantitativos o de medida de tiempo.

1.2 Requisitos de tiempo

La puntualidad, como característica definitoria de un sistema de tiempo real, puede identificarse como el conjunto de requisitos de tiempo cuantitativos sobre el sistema y/o su entorno. Para aquellos sistemas no específicamente de "tiempo real" estos requisitos son poco estrictos, sin embargo, los requisitos de tiempo para los RTS – inherentemente dinámicos – son mucho más complejos y precisan de especial atención. A diferencia de los sistemas estáticos, en los que todas las entradas de datos se encuentran disponibles antes de comenzar el procesado, en las aplicaciones dinámicas, los datos de entrada continúan llegando al sistema durante el procesado y afectando a su comportamiento.

En 1985 *Dasarathy* (Dasarathy, 1985) propone una clasificación de los requisitos de tiempo básicamente en dos categorías: los requisitos de eficiencia (que hacen referencia al tiempo de respuesta del sistema) y los requisitos de comportamiento (que hacen referencia a los estímulos del entorno). Dentro de estas dos categorías básicas de requisitos de tiempo, incluimos 4 tipos de relaciones entre los eventos del sistema:

- Estímulo-Respuesta: El sistema debe producir una respuesta acorde con la relación de tiempo especificada con respecto a un estímulo (usuario o entorno) anterior.
- Respuesta-Respuesta: Permite especificar la relación temporal que debe existir entre dos respuestas arbitrarias del sistema.
- Estímulo-Estímulo: Permite especificar el comportamiento esperado del entorno o del usuario de un sistema en función de restricciones de tiempo entre dos estímulos.
- Respuesta-Estímulo: Permite especificar la restricción de tiempo que debe existir entre una respuesta del sistema y un posterior estímulo del entorno.

Por estímulo se entiende una acción que proviene del usuario o del entorno del sistema, y por respuesta una acción que es ejecutada por el sistema hacia el usuario

o el entorno. En general, las restricciones serán de mínima cantidad de tiempo, máxima cantidad de tiempo, o ambas. Dado un estímulo de entrada, el sistema debe generar la respuesta correspondiente. En este sentido el comportamiento de un sistema de tiempo real se puede definir listando los estímulos, sus respuestas asociadas y los instantes en que dichas respuestas deben ser llevadas a cabo.

En cuanto a los estímulos, *Sommerville* (Sommerville, 1997) propone la siguiente clasificación:

- Estímulos periódicos: Ocurren a intervalos de tiempo previsibles. Por ejemplo, "el sistema debe examinar el sensor cada x segundos y llevar a cabo una acción de respuesta dependiendo del valor del sensor (estímulo)".
- Estímulos aperiódicos: Son aquellos que ocurren de una forma irregular. Generalmente, se señalizan utilizando mecanismos de interrupción. Por ejemplo, "interrupción indicando que una transferencia I/O se ha completado y los datos están disponibles".

Los estímulos periódicos en un sistema de tiempo real son generalmente producidos por sensores asociados al sistema, que proporcionan información sobre el estado del entorno. Las respuestas a estos estímulos van dirigidas a un conjunto de actuadores que controlan alguna unidad *hardware* que influye en el entorno del sistema. Los estímulos aperiódicos pueden ser generados por actuadores o por sensores; suelen indicar condiciones excepcionales, como fallo *hardware*, que el sistema debería manejar.

Los primeros dos tipos de restricciones de tiempo mencionados – estímulo-respuesta y respuesta-respuesta – definen los requisitos temporales del sistema que estamos especificando, es decir, los diseñadores del sistema pueden emplear cualquier solución arquitectónica o algorítmica, pero la funcionalidad debe ser implementada de tal forma que se ejecute lo suficientemente rápido (despacio) para cumplir dichos requisitos.

Los últimos dos tipos – estímulo-estímulo y respuesta-estímulo – implican que el sistema sea capaz de detectar una violación de las restricciones temporales (ya sea por parte del usuario o por parte del entorno) y en función de ello llevar a cabo una acción alternativa. Es decir, no implican que el sistema deba ser rápido o lento, sino que deben existir elementos adicionales en el sistema para detectar estímulos del usuario en instantes no apropiados y generar respuestas alternativas para el usuario.

Otras aproximaciones consideran el modelar el sistema y el entorno como dos procesos cooperantes, con lo cual se podría reducir la clasificación anterior a sólo dos tipos: estímulo-respuesta, respuesta-respuesta. Es decir, la clasificación realizada por *Dasarathy* pero tomando dos perspectivas diferentes: la del sistema y la del entorno.

Desde una perspectiva de este tipo, en (Koymans, 1990) se identifican dos tipos de relaciones básicas entre ocurrencias de eventos, ya sean del sistema o del entorno: tiempo de respuesta, que relaciona el tiempo de ocurrencia de un evento y la respuesta resultante; y frecuencia, para relacionar ocurrencias del mismo evento.

Ya que los requisitos de tiempo cuantitativos, arriba catalogados, establecen una relación entre eventos en el entorno y en el sistema, o entre eventos en dife-

rentes componentes del sistema, estos requisitos se refieren necesariamente a una noción global de tiempo. Esta noción global de tiempo no se debe identificar con la introducción de un reloj global, ya que un reloj siempre deriva. En otras palabras, el tiempo se puede considerar con un reloj idealizado y perfecto.

Independientemente del tipo de clasificación utilizada, se atisba la necesidad de introducir medidas temporales entre eventos, siendo un evento cualquier "hecho" de relevancia en el sistema. Si se pretende sacar ventaja del tratamiento formal en el diseño de sistemas de tiempo real, se debe comenzar por incorporar ese tipo de requisitos, para luego matizarlos en función del dominio de aplicación. Así, un sistema de tiempo real crítico debe modelarse con determinismo temporal, un sistema empotrado necesita un modelo de paralelismo real y un sistema con limitación de recursos necesita una metodología de multiprogramación con un componente planificador.

1.3 Tratamiento formal de sistemas de tiempo real

Sin duda, el *software* siempre se ha considerado caro y poco fiable, además, demasiado a menudo, el *software* se entrega con retraso. A diferencia de otras ingenierías (ingeniería civil, mecánica, química, etc.) con más tradición y por tanto más asentadas, en las que las herramientas de análisis tienen una fuerte base matemática, la ingeniería del *software* se ha considerado, hasta hace bien poco, más como una actividad artesanal, basada en la prueba y error en vez de en el cálculo y la predicción. Parte de esta diferencia, radica en que el *software* tiene dos características distintivas: la complejidad de comportamiento que el *software* puede alcanzar y su discontinuidad, dado que ligeros cambios en las entradas pueden producir comportamientos muy diferentes (Rushby, 1995). Estas características distintivas hacen que la formalización matemática de la disciplina *software*, métodos formales, sea mucho más difícil que en otras ingenierías.

El término "métodos formales" alude al uso del modelado, cálculo y predicción matemáticos en la especificación, diseño, análisis, construcción y garantía de corrección de sistemas *software* (por supuesto *hw/sw*).

En principio hubo posturas defensoras de que no era necesario el desarrollo de metodologías formales específicas para requisitos de tiempo, sino que los programas debían ser funcionalmente correctos independientemente de la velocidad *hardware* y, por tanto tener en cuenta dichos requisitos supondría una sobreespecificación del sistema. Sin embargo, pronto se resaltó que el tiempo no es necesariamente una característica de la implementación; para ello se expone el hecho de que forzar la velocidad de la computadora no lleva en todos los casos a sistemas de tiempo real funcionalmente correctos (Ostroff, 1992).

Ya desde el principio de los 90 existe un amplio consenso en que la especificación, diseño y análisis de los sistemas de tiempo real resulta difícil fundamentalmente por la inherente concurrencia que caracteriza a dichos sistemas y por la interacción intensiva y sin final del sistema con su entorno, viendo aumentada su complejidad por la presencia del tiempo. Por tanto, a la hora de obtener sistemas de tiempo real fiables y seguros, resultaba urgente el desarrollo de méto-

dos formales que incorporasen la dimensión tiempo, beneficiándose de la ventajas ampliamente aceptadas de tales métodos:

- En el proceso de formalización de los requisitos suelen aparecer ambigüedades, omisiones y contradicciones.
- Se puede verificar la corrección del modelo formal mediante métodos matemáticos.
- Un subsistema formalmente verificado puede ser incorporado en un sistema mayor con gran confianza de que se comporte como se ha especificado.
- Se pueden comparar y evaluar diferentes diseños.
- El método formal puede ser semi-automatizado o automatizado.

El principal reto en la especificación y tratamiento formal de sistemas de tiempo real es el decidir cómo incorporar la medida del tiempo. Se deben incluir los requisitos de tiempo en la especificación y se deben desarrollar métodos para determinar que un sistema satisface dichos requisitos. Necesitamos un análisis cuantitativo (plazos, tasas de repetición), en vez de un análisis cualitativo (satisfacción final) que es lo que manejan los formalismos sin tiempo. Según *Koymans* (Koymans, 1990), las razones que explican el hecho de que los métodos formales para sistemas de tiempo real se hayan retrasado con respecto a los de otras áreas de aplicación se pueden resumir en:

- Por ser los requisitos de tiempo mucho más estrictos para los sistemas de tiempo real que para otros sistemas, las tecnologías de implementación necesarias son mucho más exigentes.
- La complejidad intrínseca de los sistemas de tiempo real hace mucho más difícil el desarrollo de métodos formales adecuados.
- La mayoría de los investigadores en el área de la computación han considerado el tiempo real o bien como un caso especial de sistemas concurrentes, o bien como un elemento cuyo estudio debería ser aplazado hasta que se comprenda mejor la concurrencia básica.

Una vez aceptada la necesidad de aplicar los métodos formales a la especificación, análisis y diseño de los sistemas de tiempo real, se han utilizado diferentes semánticas, técnicas y enfoques para abordar tal objetivo. Existen, en este sentido, algunos debates todavía abiertos, de los que muchos de ellos son herencia de su aplicación a los sistemas concurrentes, por ejemplo, la elección de una semántica de trazas o una semántica de árboles de ejecución; sin embargo hay dos que son específicos de la incorporación del tiempo:

- Generalmente en el tratamiento formal de sistemas concurrentes y distribuidos, se adopta un modelado entrelazado (*interleaving*) de la concurrencia, lo que implica las mínimas suposiciones en la implementación realizada. Sin embargo, cuando se toma el tiempo en consideración, ¿es realista un modelado de este tipo?.

- Una vez aceptada la necesidad de incorporar el tiempo en la descripción del sistema, y dado que, en última instancia el software es, a cierto nivel de abstracción, discreto, ¿es suficiente un modelo de tiempo discreto?.

Abordaremos la discusión de ambos aspectos en las siguientes secciones.

1.3.1 Modelado de concurrencia en RTS

Para poder razonar sobre los sistemas de tiempo real, y en general sobre cualquier sistema concurrente, se debe asumir un modelo de ejecución de los procesos concurrentes. Dicho modelo define el entorno de ejecución del sistema, es decir, el número de procesadores disponibles, sus velocidades, la planificación de los procesos, etc. Aunque existe un amplio consenso en que el diseño formal de sistemas debe hacer las mínimas suposiciones sobre su posible implementación, ya que es poco prudente el ligar la semántica de un lenguaje de especificación a los recursos disponibles en un entorno real (que puede que aún no haya sido decidido en las primeras fases del ciclo de vida), este hecho ha suscitado cierta controversia en el ámbito del tratamiento formal de los sistemas de tiempo real.

En la teoría de la concurrencia clásica (sin tiempo), la ejecución concurrente de un conjunto de procesos se modela mediante el entrelazamiento (*interleaving*) no determinista de las acciones atómicas de cada uno de los procesos. Se trata de un modelo ampliamente aceptado, la concurrencia real puede modelarse mediante entrelazamiento siempre que consideremos un nivel de granularidad suficientemente fino y que incorporemos suposiciones de planificación equitativas en el sistema (*fairness*). Por contra, en el tratamiento formal de sistemas de tiempo real la capacidad de un sistema de cumplir sus requisitos de tiempo cuantitativos depende, en gran medida, de su entorno de ejecución.

El modelo de entrelazamiento es el modelo de ejecución menos restrictivo ya que en este modelo se ignoran las velocidades de los procesadores. El comportamiento del sistema se define como una secuencia de acciones proveniente del entrelazamiento, en cualquiera de las formas posibles y con la única restricción impuesta por el orden relativo de las acciones en cada uno de los procesos individuales. Una de las principales ventajas de este modelo es que impone las mínimas restricciones de implementación. Desde el punto de vista cualitativo, una ejecución legal de un programa en cualquier otro modelo de concurrencia será también legal en el modelo de entrelazamiento; además un programa probado como correcto utilizando entrelazamiento será correcto en cualquier implementación física.

Mientras el entrelazamiento sin tiempo de actividades concurrentes identifica el paralelismo real con no determinismo, cuando se toma el tiempo en consideración, no se puede ignorar la diferencia entre multiprocesado y multiprogramación, ya que elementos como prioridades, interrupciones y planificación de tareas influyen en la capacidad del sistema para cumplir sus requisitos de tiempo. En esta línea, algunas aproximaciones han optado por modelos de concurrencia más restrictivos: máximo paralelismo (*maximal parallelism*) y paralelismo limitado (*limited parallelism*).

En el modelo de máximo paralelismo (Koymans et al., 1988; Mok, 1991) se supone que el número de procesadores es igual al número de procesos. Cada proceso reserva un procesador que utiliza hasta su finalización. Los relojes locales de los

procesadores se sincronizan con un reloj global. Además, se fuerza en el modelo un rendimiento (*throughput*) máximo. Las acciones de un proceso no se retardarán o interrumpirán a no ser que se haya programado de forma explícita. Este modelo es, de entre los modelos de concurrencia, el más fuerte. Desafortunadamente, no es un modelo práctico, ya que en sistemas grandes (cientos o miles de procesos), no resulta posible el tener un procesador por proceso. Además, un sistema *hardware* suele constar de diferentes dispositivos físicos y de computación; es, por tanto, optimista esperar que se ejecutarán a la misma velocidad. Con esta perspectiva, *Mok* (Mok, 1991) propone un modelo computacional para sistemas de tiempo real basado en TTMs (*Timed Transition Models*), donde una ejecución del sistema será una secuencia de conjuntos de eventos, los eventos en un mismo conjunto ocurren "realmente" en paralelo.

Se pueden encontrar otras aproximaciones intermedias entre el modelo de entrelazamiento y el modelo de máximo paralelismo, dependiendo de la cantidad de paralelismo permitido en el modelo de concurrencia. Dichos modelos de paralelismo limitado se propusieron como una solución de compromiso entre el entrelazamiento y el máximo paralelismo, ya que el número de procesadores puede ser menor que el número de procesos.

Sin embargo, actualmente, se considera probado que mediante una incorporación adecuada del tiempo en los modelos de entrelazamiento se puede modelar de forma adecuada la mayoría de los fenómenos cuantitativos propios de la ejecución de sistemas de tiempo real, beneficiándonos, al mismo tiempo, del mínimo conjunto de suposiciones en el entorno de ejecución. En (Henzinger et al., 1991) se justifica la validez del entrelazamiento para el modelado de multiprocesado y multiprogramación, ya que conceptos como prioridades, interrupciones o planificación (fundamentales a la hora de determinar la capacidad de un sistema para cumplir sus requisitos de tiempo) tienen cabida en este modelo de concurrencia.

Los modelos entrelazados representan las suposiciones mínimas necesarias para asegurar una corrección cualitativa en la ejecución de programas concurrentes. De la misma forma que en los sistemas de tiempo es necesario incorporar la noción de equidad (*fairness*) para que el entrelazamiento sea cualitativamente adecuado para representar concurrencia, en el modelado de sistemas de tiempo real es necesario introducir estructuras adicionales, sin duda más fuertes que la noción de equidad, para que el modelado con entrelazamiento sea cuantitativamente adecuado para representar la concurrencia.

1.3.2 Modelado de tiempo en RTS

Ha habido un largo y todavía inestable debate sobre la elección del modelo cuantitativo para el tiempo; dicho debate se centra en decidir si es suficiente un modelo discreto o, por contra, es necesario un modelo denso. En un modelo discreto, los eventos ocurren sólo a intervalos regulares de un reloj ficticio; en un modelo denso, los eventos pueden ocurrir en cualquier instante.[1]

Mientras que las aproximaciones con tiempo discreto se pueden tratar formal-

[1] Dicho debate se ha centrado en la naturaleza densa o discreta del dominio temporal, dado que es la densidad, y no la continuidad, la que aporta el elemento principal: entre dos eventos cualesquiera, siempre puede existir otro evento

mente como sistemas no temporizados con la simple inclusión de una acción para el paso de la unidad de tiempo, en el modelado denso el sello temporal de los eventos es una magnitud en un dominio denso. La transformación de un conjunto de trazas cuyos eventos son marcados con un valor denso en un sistema tradicional (autómata, red de Petri, sistema de transiciones etiquetado) no resulta obvia y es necesaria la incorporación de nuevos modelos, semánticas y razonamientos más complejos para su tratamiento.

Los criterios que han guiado la elección de un modelo u otro son esencialmente dos: por un lado, la **expresividad** del modelo elegido, es decir, lo adecuado del modelo a la hora de describir la realidad; y, en segundo lugar, su **eficiencia**, es decir, cuál es el coste de las técnicas de análisis asociadas al modelo.

El primer estudio de **expresividad** realizado lo encontramos en (Alur, 1991). *Alur* comparó dos modelos discretos: uno en el que los eventos ocurrían con los "*ticks*" del reloj, y otro en el que los eventos pueden ocurrir en cualquier instante, pero la única información cuantitativa registrada es el número de *ticks* entre dos eventos. Tras comparar estos modelos discretos y el modelo denso bajo el paradigma de circuitos asíncronos, se demuestra, si bien informalmente, que el modelo denso es más expresivo. Resultados más formales los encontramos en (Gollu et al., 1994), (Asarin et al., 1998a) y (Henzinger et al., 1992a). Una conclusión general de todos estos resultados es que el tiempo denso es estrictamente más expresivo que el tiempo discreto.

Quizás la característica más sobresaliente del modelo denso es el hecho de que no impone un cuanto (*quantum*) de tiempo específico, por lo que es posible tratar retardos arbitrariamente pequeños. Esta propiedad resulta de fundamental interés en dos casos: en primer lugar, a la hora de implementar un modelo, la corrección de dicha implementación no depende de la velocidad de la máquina; en segundo lugar, la composición de los sistemas es independiente del cuanto de tiempo, sólo es necesario normalizar las constantes que aparecen en las restricciones de los componentes, pero no las granularidades de tiempo como es necesario en el caso discreto.

Una aproximación diferente consiste en utilizar un dominio denso, pero realizar la verificación discreta pasando por la discretización de los comportamientos del sistema. En (Henzinger et al., 1992a), se demuestra que la resolución del problema de verificación sobre un dominio denso se puede reducir a una verificación discreta bajo ciertas condiciones de "digitalización". Dichas condiciones nos aseguran que el problema de verificación es el mismo para todas las propiedades cualitativas y para algunos casos de propiedades cuantitativas (restringiendo el tipo de intervalos que aparecen en la fórmula) en el dominio discretizado y en el dominio denso, pero todavía sería necesaria una metodología que nos permitiese encontrar el cuanto de tiempo necesario para tal discretización. Siguiendo esta línea, (Asarin et al., 1998a) presenta un método de discretización que, en casos especiales como el modelado de circuitos electrónicos acíclicos, preserva el orden de eventos.

En cuanto a la **eficiencia**, muchas veces se argumenta que el modelo discreto es más adecuado, ya que admite técnicas de verificación no temporizadas potentes como las técnicas simbólicas que utilizan BDDs –*Binary Decision Diagrams* (Bryant, 1986)– u órdenes parciales. Aunque esta afirmación es cierta, debe matizarse que:

- La utilización de técnicas discretas no es siempre sin costo, ya que el discretizar un modelo de tiempo denso o realizar el modelado directamente en tiempo discreto puede llevar a una especificación menos compacta. Por ejemplo, los modelos que incorporan constantes de tiempo grandes pueden llevar a una explosión de estados si se tratan con tiempo discreto. En (Bozga et al., 1999) se presenta una comparativa de la eficiencia obtenida en la verificación de STARI (*chip* asíncrono para la prevención de desajustes de reloj en transmisiones síncronas). Aunque para el caso presentado en el artículo los resultados obtenidos son mejores para la verificación discreta (debido a que, en la verificación densa, las variables discretas son enumeradas, mientras que, en el caso discreto, tanto las variables discretas como los relojes se manejan con BDDs), existen otros ejemplos en los que el tamaño de las constantes (o equivalentemente, la necesidad de una menor granularidad en tiempo) hace que la verificación con tiempo discreto sea menos atractiva (Tripakis, 1998; Asarin et al., 1997), ya que su eficiencia depende del número de bits utilizados para codificar los valores de los relojes de forma discreta.

- Algunas veces, la naturaleza síncrona del paso del tiempo afecta a la eficiencia de los BDDs y de los órdenes parciales que trabajaban bien en sistemas donde las acciones son tan independientes como sea posible.

Por tanto, podemos concluir que, en general, la aproximación discreta puede ser adecuada para la descripción del comportamiento de sistemas síncronos y para la resolución de algunos problemas como el análisis de planificabilidad (Alur, 1999). Sin embargo, para el modelado de sistemas asíncronos se hace necesario discretizar el tiempo denso eligiendo un cuanto fijo a priori; esta elección limita la exactitud con que se modela el sistema, ya que esto permite que determinados comportamientos puedan ser pasados por alto. Aunque el cuanto elegido para la discretización sea adecuado, su elección a priori podría presentar problemas para la composición y refinamiento. Además la elección de un cuanto de tiempo lo suficientemente pequeño como para ser "razonablemente" seguro puede hacer crecer el espacio de estados del sistema hasta un punto en el que la verificación del sistema no sea factible.

Dado que, además, las diferencias de eficiencia en la verificación parecen tender a converger, el modelo de tiempo denso resulta, por tanto, más natural y correcto, especialmente porque existen áreas de aplicación, como por ejemplo los sistemas híbridos, en las que el dominio del tiempo no puede ser discretizado de una forma lo suficientemente fina como para conseguir una semántica correcta.

1.4 Cronología del tratamiento formal en RTS

En esta sección presentamos un resumen cronológico de los logros alcanzados en el tratamiento formal de los sistemas de tiempo real (tabla 1.1). Dicho resumen se centra, fundamentalmente, en los aspectos más relacionados con el trabajo realizado en esta tesis.

La historia de los métodos formales para sistemas de tiempo real comienza a finales de los 80, cuando, después de un largo período caracterizado por considerar que los requisitos de tiempo estaban relacionados con la velocidad *hardware*,

- Aproximaciones discretas:
  - 1986: RTL –*Real Time Logic* (Jahanian y Mok, 1986)–.
  - 1989: TTM/RTTL –*Timed Transition Models/Real Time Temporal Logic* (Ostroff, 1989)–.
  - 1991: TPL –*Temporal Process Language* (Henessy y Regan, 1991; Henessy y Regan, 1995)–.
  - 1990: TTS –*Timed Transition Systems* (Henzinger et al., 1990)–.
- Primeras aproximaciones densas (álgebras de procesos):
  - 1988: Timed CSP –*Timed Communicating Sequential Processes* (Reed y Roscoe, 1988)–.
  - 1990: Timed CCS –*Timed Calculus of Communicating Systems* (Yi, 1990)–.
  - 1990: ATP –*Algebra of Timed Processes* (Nicollin et al., 1990)–.
- 1989: DBM –*Difference Bound Matrix* (Dill, 1989)–.
- 1990: Aparición de TA (*Timed Automaton*) y resolución al *model checking* con tiempo denso –grafo de regiones– (Alur et al., 1993a). Versión preliminar en *Proc. 5th LICS, 1990*.
  - 1991: Modelado de sistemas híbridos (Alur et al., 1993b; Maler et al., 1992).
  - 1992: Diferencias de relojes en TA (Nicollin, 1992).
  - 1992: STA (*Safety Timed Automaton*), solución simbólica al *model checking* e introducción de T_μ-*calculus* (Henzinger et al., 1992b).
- Avances en el análisis y síntesis basado en TA:
  - 1991: Primer algoritmo *tableau* para una lógica de tiempo denso (MITL) que no permite puntualidad (Alur et al., 1996). Por primera vez en *Proc. of the Tenth Annual Symposium on Principles of Distributed Computing, 1991*.
  - 1991: Síntesis de controladores (Wong Toi y Hoffmann, 1991).
  - 1992: Minimización (Alur et al., 1992).
  - 1993: Herramienta KRONOS (Yovine, 1993).
  - 1995: Herramienta UPPAAL (Bengtsson et al., 1996).
  - 1995: Caso Real con UPPAAL (Ho y Wong-Toi, 1995).
  - 1995: *Bounded Satisfiability* (Laroussinie et al., 1995)–.
- Nuevas variantes de TA: en busca de mejoras en composición y la búsqueda de una base formal completa para el modelado con tiempo denso.
  - 1994: ECA –*Event-Clock timed Automaton* (Alur et al., 1994)–. Asocia relojes a eventos consiguiendo una variante cerrada bajo complementación.
  - 1996: TAD –*Timed Automaton with Deadlines* (Sifakis y Yovine, 1996)–. Incorporación de plazos en las transiciones como alternativa a las invariantes de tiempo para mejorar la composición.
  - 1997: SCL –*State Clock Logic* (Raskin y Schobbens, 1997)–. Lógica de satisfactibilidad decidible con puntualidad sobre *State Clock Automaton*.
  - 1998: (Henzinger et al., 1998) formaliza la conexión entre las lógicas de tiempo denso sin puntualidad y ECA.
  - 1999: Se proponen CDDs – *Clocked Difference Diagrams* (Larsen et al., 1999)– como el equivalente a los BDD para sistemas con tiempo.
  - 2000: TCTL (*Timed Computation Tree Logic*) se muestra satisfactible sin puntualidad (La Torre y Napoli, 2000), al igual que para el caso lineal MITL.
- Más casos reales 1998- : KRONOS: *Fast Reservation Protocol with Delayed Transmission* (Tripakis y Yovine, 1998). UPPAAL: *Audio Control Protocol* (Bengtsson et al., 1999), *Power Controller* (Havelund et al., 1999), etc. Para más ejemplos [http://www.docs.uu.se/docs/rtmv/uppaal/] y [http://www-verimag.imag.fr//TEMPORISE/kronos/].

Tabla 1.1. Cronología de hechos relevantes en los métodos formales con tiempo

Stankovic (Stankovic, 1988) postula la necesidad de técnicas de especificación y verificación específicas. En sus primeros pasos, los métodos formales para sistemas de tiempo real surgieron bajo una perspectiva de tiempo discreto, que facilitaba la aplicación de las técnicas para sistemas sin tiempo largamente utilizadas.

Los primeros trabajos en tiempo denso se localizan en el ámbito de las álgebras de procesos a finales de los 80. Inmediatamente después, en el año 1990, *Rajeev Alur* propone el autómata temporizado (TA – *Timed Automaton*–), siendo una extensión de un autómata con un número finito de relojes que toman valores en $\Re_+$. Las transiciones del autómata se acompañan de predicados de tiempo, sobre dichos relojes, que limitan el "tiempo" en el que el sistema puede transicionar de un estado a otro. Si bien la propuesta del modelo es importante de por sí, su gran impacto fue debido a que el modelo se acompaña del primer algoritmo de *model checking*, sobre un dominio de tiempo denso, mediante la abstracción al grafo de regiones para la lógica TCTL (*Timed Computation Tree Logic*). En 1992, *Nicollin* (Nicollin, 1992) demuestra que el problema de *model checking* era igualmente resoluble en el caso de ampliar la gramática de los predicados de tiempo, incluyendo las diferencias de valores de relojes. Al mismo tiempo, se introduce en (Henzinger et al., 1992b) la variante STA (*Safety Timed Automaton)* que impone invariantes de tiempo en los estados del autómata, permitiendo modelar urgencia en el autómata temporizado. A la par de esta introducción, dicho trabajo presenta el primer algoritmo de *model checking* simbólico sin construcción explícita del grafo de regiones, mediante la utilización de DBMs (*Difference Bound Matrix*), y la definición de T_μ-*calculus*, para el que el problema de *model checking* se muestra decidible.

Una vez alcanzada una madurez media en los métodos formales de tiempo real, y como resultado de los logros alcanzados, se abre el campo de los métodos formales para sistemas híbridos (Maler et al., 1992), es decir, aquellos que, además del tiempo, permiten el modelado de magnitudes físicas que cambian de forma continua con el tiempo. A partir de este momento, ambos ámbitos de aplicación se han desarrollado de una forma conjunta, aunque siempre con los problemas de complejidad asociados a los modelos híbridos (de los que los modelos con tiempo denso se pueden considerar una particularización).

En años posteriores, salen a la luz distintos resultados relacionados con la eficiencia de la verificación formal basada en *model checking*, en un intento de reducir el problema de la explosión de estados propia del grafo de regiones (resumidos en la sección 3.3.1). Resaltamos, por su relación directa con esta tesis, la extensión de las técnicas de minimización para modelos de estados sin tiempo a su aplicación para el autómata temporizado (Alur et al., 1992). La minimización, como técnica de reducción del espacio de estados, intenta encontrar una abstracción mínima en la que poder resolver, de forma exacta, un problema (análisis o síntesis) concreto.

En esta misma línea de extensión de métodos de análisis ya aplicados en el tratamiento formal sin tiempo, se plantea la resolución al problema de síntesis de controladores para sistemas con tiempo. El problema de síntesis de controladores, como búsqueda de estrategias seguras que garanticen el cumplimiento de la especificación, se resuelve en (Wong Toi y Hoffmann, 1991) mediante la utilización de la técnica de regiones para, posteriormente, encontrar soluciones más eficientes al problema (Asarin et al., 1995; Tripakis y Altisen, 1999).

En torno a 1993 aparecen las primeras herramientas académicas de verificación,

primero KRONOS (Yovine, 1993), seguida de UPPAAL (Bengtsson et al., 1996) y HyTech (Henzinger et al., 1995) (para sistemas híbridos).

Ya a principios de los noventa, y al tiempo que se empezaban a dar los primeros pasos en *model checking* sobre tiempo denso, se atisba que el problema de satisfactibilidad (consistente en la búsqueda de un modelo para una fórmula lógica) y, por tanto, el de síntesis, no iba a ser tan fácilmente trasladable a los sistemas reactivos con tiempo. La definición de lógicas temporales con tiempo denso con satisfactibilidad decidible se consigue sólo a costa de prohibir la puntualidad (precisión infinita) de los requisitos de tiempo (Alur et al., 1996) (por primera vez en 1991).

A partir del año 1994 empiezan a producirse posturas defensoras de que el modelo del autómata temporizado, aunque avalado por la resolución de casos prácticos es "demasiado" expresivo y como consecuencia, en base a dicho modelo, no se encuentra solución a todos los problemas (verificación homogénea basada en autómata), dado que se trata de un modelo no cerrado bajo complementación. Como alternativa se propone ECA –*Event-Clock timed Automaton* (Alur et al., 1994)– que, asociando los relojes a los eventos en el modelo, supone una variante cerrada bajo todas las operaciones booleanas. La conexión entre la prohibición de puntualidad y la restricción de los relojes a relojes de eventos se formaliza en (Henzinger et al., 1998).

A pesar de no haber alcanzado la madurez de las técnicas formales para los sistemas reactivos sin tiempo, los casos de estudio reales (como los abordados en UPPAAL y KRONOS) han corroborado que los modelos y métodos temporizados se encuentran cerca de estar preparados para su uso en la industria del software.

1.5 Conclusiones y motivación de la tesis

Como conclusión central de los resultados expuestos, podemos afirmar que el área del tratamiento formal de los sistemas de tiempo real, además de constituir una disciplina con entidad propia, ha alcanzado una madurez teórica y práctica razonable. A la par de los estudios teóricos al respecto, se han ido desarrollando herramientas de especificación y análisis, principalmente en el ámbito académico (KRONOS (Yovine, 1993) seguida de UPPAAL (Bengtsson et al., 1996), RT-SPIN (Tripakis y Courcoubetis, 1996) y Timed-COSPAN (Alur y Kurshan, 1996)). La mayoría de dichas herramientas se basan, en mayor o menor medida, en el modelo de autómata temporizado –*Timed Automaton* (Alur et al., 1993a)–, aunque difieren en los lenguajes de especificación de requisitos utilizados. Los casos de estudio reales (como los abordados en UPPAAL y KRONOS) han recalcado que los modelos y métodos temporizados están preparados para su uso profesional.

Sin embargo, dichos logros tropiezan con el rechazo de la industria *software* a la adopción de los métodos formales; los problemas de transferencia tecnológica, ya descubiertos en sistemas sin tiempo convencionales, aparecen una vez más. El reto de alcanzar la práctica profesional pasa por la incorporación de los métodos formales en el proceso *software*. Para ello, es necesario que sean fáciles de usar y de entender, resultando esencial la disposición de entornos integrados que incorporen facilidades como el análisis estático, basado en verificación formal y teoría de la

planificación, o el análisis dinámico, basado en prueba y monitorización en tiempo de ejecución (*run time monitoring*). Stankovic (Stankovic, 1996) y Heitmeyer (Heitmeyer, 1998) abogan por la necesidad de un marco común que permita la aplicación de todas éstas técnicas a sistemas de gran escala. En dicho marco se utilizaría un único modelo del sistema para realizar el análisis estático y dinámico, lo que evitaría la especificación de modelos propios para cada una de las técnicas y la necesidad de la comprobación de consistencia entre estos modelos.

Por lo que respecta a la estructura del proceso *software*, los requisitos para sistemas grandes y complejos son a menudo difíciles de identificar en las primeras fases, ya que evolucionan de forma continuada a lo largo del ciclo de vida *software*. Por lo tanto, los procesos *software* deberían ser lo suficientemente flexibles para acomodarse a los cambios en los requisitos. Aun es más, en las primeras fases, el diseñador no suele tener un conocimiento profundo del sistema y, en cualquier caso, muchas veces la complejidad es excesiva para un diseño de un solo paso. Los ciclos de vida tradicionales en cascada se han revelado, en este aspecto, del todo inadecuados. Los métodos formales deben adaptarse para soportar evolución en el ciclo de vida, saliendo de su papel tradicional de verificar que un modelo cumple un conjunto fijo de requisitos. Referimos a (Stankovic, 1996) para escenarios de tiempo real donde resulta imprescindible soportar la evolución de requisitos en el ciclo de vida.

En tal contexto se sitúa la motivación de esta tesis. Una vez han sido completa o parcialmente resueltos los principales problemas en el ámbito de los métodos formales para sistemas de tiempo real, nuestro objetivo es ligeramente diferente; con el fin de mejorar la calidad del *software* de tiempo real crítico, es nuestra intención conjugar dos soluciones de diferente naturaleza: la **formalización** del proceso *software* (aprovechando las ventajas del uso de métodos formales) y un enfoque de proceso *software* **iterativo e incremental** (acomodándose a los cambios en los requisitos a lo largo del ciclo de vida). De esta forma combinamos la corrección de los métodos formales, y lo flexible de un ciclo de vida que no congela los requisitos en fases tempranas y divide la complejidad.

Para tal objetivo, los resultados teóricos relacionados con el diseño y análisis formal de sistemas de tiempo real (recogidos en el estado del arte, parte II) son los elementos constructivos para la definición de un modelo formal de proceso *software*. Sin embargo, dado que el ciclo de vida propuesto es iterativo e incremental, para alcanzar un diseño sólido es necesario relacionar fases entre sí y registrar la información de diseño obtenida hasta la fase actual. En cada ciclo, la capacidad del sistema de satisfacer los requisitos de usuario depende de los requisitos ya alcanzados en el sistema. Una vez obtenido un modelo de diseño, es aconsejable utilizar dicho modelo para diferentes tipos de análisis (formal e incluso informal).

Bajo este punto de partida, se propone la extensión, para sistemas de tiempo real críticos, de los estudios realizados en este ámbito por el grupo de Redes e Ingeniería del *Software* (Pazos Arias, 1995; Gil Solla, 1999; García Duque, 2000) del Departamento de Ingeniería Telemática de la Universidad de Vigo. El principal resultado de dichos estudios, cuyo objetivo era el uso de los métodos formales sobre modelos de ciclo de vida iterativos e incrementales, ha sido el desarrollo de la metodología formal SCTL/MUS (Pazos Arias y García Duque, 2001), que resumimos brevemente en la siguiente sección.

1.6 La metodología formal SCTL/MUS

Dadas las ventajas ampliamente aceptadas en la ingeniería del *software* de los modelos de proceso *software* iterativos e incrementales, la metodología SCTL/MUS (Pazos Arias y García Duque, 2001) persigue, utilizando las bases formales SCTL y MUS, la definición de un marco formal para el diseño y desarrollo de sistemas *software*, en especial para sistemas distribuidos, que habilite procesos *software* con dichas características (iterativos e incrementales).

La metodología SCTL/MUS afronta el problema de diseño *software* mediante una aproximación dual a la especificación (orientada a propiedades y orientada a modelo). La especificación dual resulta útil debido a que aúna las ventajas de dos estilos de especificación ampliamente utilizados: el estilo operacional y el estilo orientado a propiedad. Mientras que en una especificación operacional es menos probable omitir un comportamiento requerido y pueden incluso ser estilos de especificación ejecutables, en una especificación orientada a propiedades se permite una especificación concisa, abstracta y no influenciada por factores de implementación. Una aproximación dual, además de integrar todas las ventajas anteriores, permite detectar inconsistencias entre dos especificaciones diferentes del mismo comportamiento, lo que nos permite depurar tanto la declaración de las propiedades deseadas como la especificación operacional.

En la metodología formal SCTL/MUS, y bajo un enfoque de este tipo, se propone:

- El uso de la lógica temporal **SCTL** (*Simple Causal Temporal Logic*), como un estilo de especificación orientado a propiedades adecuado en las primeras fases del ciclo de vida. El objetivo de SCTL es permitir el tratamiento de información incompleta e inconsistente, características inherentes a estas primeras fases en las que aún no hemos alcanzado un conocimiento profundo del sistema. Por ello, se ha optado por imprimir a SCTL-T una naturaleza multivalorada como un mecanismo atractivo para el tratamiento de información tanto incompleta como inconsistente.

- El modelo de estados **MUS** (*Model of Unspecified States*) como especificación orientada a modelo, dado que este estilo operacional resulta mucho más adecuado que las álgebras de procesos para las fases tempranas del ciclo de vida, en las que la noción de estructura es mínima o no existe. En base a la estructura incremental del modelo de ciclo de vida, es necesario permitir la definición de modelos incompletos, siendo este hecho el origen de MUS.

- Dada la mejor adecuación de las álgebras de procesos a la hora de manejar refinamientos, la metodología propone el uso del álgebra de procesos LOTOS (Bolognesi y Brinksma, 1989) en fases no centradas en la captura de requisitos.

El sistema, posiblemente incompleto, se modela en cualquier fase de su diseño mediante un grafo MUS que guarda la información conocida hasta el momento sobre el comportamiento funcional del sistema. La información no "conocida" se cataloga como "subespecificada". MUS aporta, en este sentido, un modelo de estados con **subespecificación** en el dominio de las acciones que componen los

eventos identificados en el sistema. Dicho modelo, además de caracterizar las acciones como de ocurrencia posible, no posible o subespecificada; almacena las relaciones temporales cualitativas entre dichas acciones.

Los requisitos se expresan en la lógica temporal SCTL que, por tener una naturaleza multivalor, permite distinguir las distintas condiciones de especificación (posible, no posible, subespecificada) definidas en un modelo MUS. La verificación formal de un requisito SCTL en un modelo MUS permite conocer el **grado de satisfacción** con que el sistema soporta el requisito, expresando cualitativamente la capacidad del sistema para cumplir o incumplir el requisito, y dependiendo, dicha capacidad, del nivel de subespecificación que exista tanto en el sistema como en el requisito.

La metodología SCTL/MUS se define sobre un modelo de ciclo de vida iterativo e incremental con prototipado. Las distintas iteraciones en el proceso *software* conllevan, en mayor o menor medida, todas las tareas formales (especificación, síntesis, análisis) definidas en el modelo de ciclo de vida propuesto. Los prototipos, identificados como modelos MUS, reflejan un proceso de diseño incremental en el que, fase a fase, requisitos SCTL adicionales son incorporados al prototipo actual mediante una síntesis automática. En el capítulo 5 se presenta una exposición más detallada del modelo de ciclo de vida propuesto como base para la metodología SCTL/MUS en (García Duque, 2000).

1.7 Objetivos

Es nuestro objetivo (Fernández Vilas et al., 1999; Fernández Vilas et al., 2000; Fernández Vilas y Pazos Arias, 2000), dado el creciente uso de los sistemas de tiempo real en innumerables ámbitos, el dotar a la metodología formal SCTL/MUS de las bases teóricas, así como del soporte práctico necesarios, para la incorporación de requisitos de tiempo críticos. Si bien en el tratamiento formal de los sistemas con tiempo ha habido un largo debate sobre la naturaleza discreta o densa del dominio temporal, se considera ampliamente aceptada la mayor expresividad de los modelos densos. Aunque existen escenarios concretos (Bozga et al., 1997) en los que el modelado discreto se ha aplicado con éxito, su utilización arrastra problemas en composición y refinamiento. Dado que además, la mayor adecuación del modelado denso ha ido acompañada de una gran madurez en la búsqueda de técnicas que permitan una aplicación eficiente de los métodos de decisión, hemos optado por un modelado denso del tiempo (ver sección 1.3.2).

Partiendo de las ventajas, ya postuladas por la metodología SCTL/MUS, asociadas al uso una especificación dual, y a la flexibilidad aportada por un modelo de ciclo de vida iterativo e incremental, se establecen los siguientes objetivos:

- **Adaptación del modelo de ciclo de vida** para su aplicación al ámbito de los sistemas de tiempo real críticos. En este sentido, se deben estudiar las peculiaridades que un sistema de este tipo impone en el modelo de ciclo de vida propuesto en la metodología SCTL/MUS, y lo adecuado de su estructura iterativa e incremental. Más concretamente, y dado que los requisitos de tiempo son el origen del "tiempo real", será necesario decidir cuándo incor-

porar estos requisitos en el modelo de ciclo de vida SCTL/MUS. (capítulo 5)

- **Extensión de los formalismos MUS y SCTL** para la incorporación del tiempo como una característica que determina la corrección del sistema:
  - Definición del modelo de estados **MUS-T** (*Timed Model of Unspecified States*) (capítulo 6), su sintaxis y su semántica, en base al paradigma de autómata temporizado. Un modelo MUS-T debe almacenar toda la información disponible, en la fase de diseño actual, sobre los ordenamientos relativos de las acciones (al igual que el modelo MUS) y, adicionalmente, las distancias temporales relativas entre ellas. La semántica debe asociar con cada modelo MUS-T un grafo denso, conforme al dominio de tiempo seleccionado, recogiendo el conjunto de comportamientos temporizados especificados, o subespecificados, en el prototipo actual.
  - Definición de la lógica temporal causal multivalor **SCTL-T** (*Timed Simple Causal Temporal Logic*) (capítulo 7), su sintaxis y su semántica, como una extensión conservativa y temporizada de SCTL. Dicha extensión debe respetar los principios básicos de SCTL: causalidad, simplicidad y naturaleza multivalorada. La semántica SCTL-T, asociando un grado de satisfacción a un fórmula SCTL-T, se debe interpretar, al igual que en su correspondiente sin tiempo SCTL, sobre el modelo semántico definido para MUS-T.

- Proporcionar los **métodos de tratamiento formal**, tanto de análisis como de síntesis, en que se basa el modelo de ciclo de vida propuesto. Como se ha argumentado en la presentación de la motivación de la tesis (sección 1.5), consideramos de especial importancia introducir diferentes tipos de análisis:
  - **Corrección mínima del modelo**: La metodología propuesta deberá proporcionar herramientas de análisis que garanticen una corrección mínima del modelo MUS-T del sistema en desarrollo. Como forma sistemática de aportar una garantía de corrección mínima es necesario asegurar que el modelo se encuentra libre de bloqueos en control, suponemos que el sistema real nunca se para, y en tiempo, suponemos que el tiempo nunca se para (capítulo 11).
  - **Verificación estática**: Se propone un método de verificación automático en la línea de los algoritmos de *model checking* introducidos por *Clarke*, utilizando la técnica de regiones de *Alur* (Alur et al., 1993a). La verificación estática basada en *model checking* permite conocer el grado de satisfacción de un requisito SCTL-T en un modelo MUS-T (capítulo 8).
  - **Simulación simbólica**: La simulación simbólica del sistema a partir de la especificación formal, además de aumentar la comprensión del problema, es un método adecuado para la validación del sistema. Asimismo, la simulación proporciona el soporte para la evaluación de contraejemplos y la evaluación de alternativas de diseño.
  - **Síntesis**: Se propone un método automático de actualización de modelos MUS-T mediante la incorporación de nuevos requisitos, como la

correspondencia, con tiempo, de la síntesis propuesta en la metodología SCTL/MUS. El conjunto de requisitos se expresa en una versión reducida de la lógica SCTL-T, lo suficientemente expresiva como para realizar una especificación basada en escenarios.

- **Garantizar la viabilidad de la metodología para su aplicación a casos reales**: Los métodos de decisión basados en la técnica de regiones sufren una gran explosión del espacio de estados, siendo poco eficientes en la práctica. Para posibilitar la aplicación de la metodología propuesta a casos de estudio de tamaño medio-grande, es necesario estudiar abstracciones más gruesas que permitan una aplicación eficiente de los métodos de decisión inmersos en el proceso *software* (capítulo 10).

Además de los capítulos citados, en los que se recogen las principales contribuciones de esta tesis, la memoria contiene una parte dedicada a la revisión del actual estado del arte en el tratamiento formal de los sistemas de tiempo real (parte II), estructurada como sigue:

- Capítulo 2: Contiene una revisión de los lenguajes de especificación para sistemas de tiempo real, haciendo especial hincapié en la presentación de las lógicas temporales y las máquinas de estados más relevantes. Adicionalmente, aunque más alejado de los objetivos de ésta tesis, se presenta una breve introducción a los lenguajes de especificación basados en álgebras de procesos.

- Capítulo 3: Contiene una revisión de los métodos de decisión utilizados en el tratamiento formal de sistemas de tiempo real. En este capítulo se hace una revisión de los logros obtenidos en el *model checking* para modelos de tiempo denso. Adicionalmente, se presenta una visión general del problema de síntesis de máquinas de estados a partir de propiedades expresadas en lógica. La resolución al problema específico de síntesis (capítulo 9) en la metodología propuesta (SCTL/MUS-T) proporciona, además, un breve revisión de los aspectos más relacionados con la solución adoptada.

- Capítulo 4: Se introduce de forma breve tres conceptos relacionados en menor medida con los objetivos de la tesis. En primer lugar, se hace una breve introducción al tratamiento formal de los sistemas de tiempo real *soft* y a los sistemas híbridos. Al final del capítulo, se hace una breve revisión de las lógicas multivalor.

Por último, en el capítulo 13, tras enumerar las principales contribuciones de esta tesis, se analizan los resultados obtenidos, se exponen las conclusiones, y se perfilan posibles líneas de trabajo futuro. La memoria contiene, además, un apéndice dedicado a la representación del dominio de tiempo mediante la utilización de DBMs (apéndice A), y la bibliografía utilizada.

PARTE II

Estado del Arte

CAPÍTULO 2

Especificación formal para RTS

En este capítulo se introduce la revisión realizada en el estado del arte en el ámbito de la especificación formal de sistemas de tiempo real. La presentación se realiza atendiendo a tres categorías básicas: lógicas; álgebras de procesos; y máquinas de estados, las mismas que han sido utilizadas en el tratamiento formal de sistemas sin tiempo.

2.1 Introducción

Si bien los métodos formales han llegado a un elevado grado de madurez en nuestros días, su aplicación a los sistemas de tiempo real es relativamente reciente. La aplicación de métodos formales a los sistemas de tiempo real se hace especialmente compleja ya no debido a lo crítico de estos sistemas, pues esta característica la podemos encontrar en otros sistemas que no incorporan el tiempo como un requisito funcional sino como un requisito de eficiencia, sino más bien a la incorporación del tiempo como requisito funcional del sistema. El tratamiento formal de este tipo de sistemas se ha abordado bajo las mismas categorías de formalismos utilizadas con éxito en los sistemas sin tiempo.

Dichos formalismos abarcan una gran variedad de aproximaciones que difieren considerablemente en sus técnicas, objetivos, pretensiones y filosofía. Muchas de estas diferencias radican en el nivel de formalidad utilizado, la fase del ciclo de vida en la que se aplican y los propósitos analíticos o descriptivos de dicha aplicación. En general, las distintas aproximaciones al modelado formal se asocian a diferentes clases de lenguajes de especificación, que se clasifican en orientados a modelo y orientados a propiedad.

Las notaciones formales orientadas a modelo especifican los comportamientos deseados o exhibidos mediante un modelo matemático. Para las estructuras de datos, estos modelos utilizan nociones de teoría de conjuntos (conjuntos, funciones, relaciones), mientras que para la descripción del control se utilizan conceptos de matemática discreta, como grafos, órdenes parciales y máquinas de estados.

En una especificación orientada a modelo es menos probable omitir un comportamiento requerido y proporcionan especificaciones ejecutables.

En contraste con el estilo orientado a modelo, a menudo preferido para descripciones más cercanas a la implementación, las notaciones orientadas a propiedad son más adecuadas a las primeras fases del ciclo de vida *software*, dado que permiten una especificación concisa, abstracta y no influenciada por factores de implementación. Las notaciones orientadas a propiedad, usualmente basadas en lógica, utilizan un estilo axiomático para establecer las propiedades que se requieren en el sistema o en sus componentes, sin entrar a sugerir como alcanzar dicho comportamiento.

Dentro de las notaciones formales orientadas a propiedad, las más extendidas son, sin duda, las basadas en lógica, ya sea temporal o modal. En cuanto a las notaciones orientadas a modelo, encontramos categorías de especificación diferentes dependiendo del nivel de abstracción utilizado: notaciones muy próximas a la implementación como los lenguajes de especificación síncronos – ESTEREL (Boussinot y de Simone, 1991), LUSTRE (Halbwachs et al., 1991)) y asíncronos (ASTRAL (Coen Porisini et al., 1997) –; una aproximación que podemos considerar intermedia en lo que serían las álgebras de procesos; y la aproximación más abstracta, en la que enmarcamos todos aquellos formalismos basados en máquinas de estados.

En la mayoría de los casos, los formalismos basados en máquinas de estados utilizan notaciones gráficas, resultando estas especificaciones más legibles e intuitivas que las producidas por notaciones textuales, comúnmente basadas en lógica o en álgebra de procesos.

En las siguientes secciones presentamos las diferentes notaciones formales que han sido propuestas para la especificación de sistemas de tiempo real atendiendo a estas categorías básicas, las mismas que han sido tradicionalmente utilizadas para la clasificación de los métodos formales sin tiempo.

2.2 Lógicas

2.2.1 Lógicas temporales

Las lógicas temporales, obtenidas de interpretar la lógica modal bajo modos de verdad con semántica temporal, son lógicas ampliamente utilizadas en el área de la computación. Tienen la capacidad de representar de forma cualitativa un conocimiento temporal y de diferenciar el pasado y el futuro, aunque no permiten razonar sobre el tiempo en un sentido cuantitativo.

Dada la ventaja de especificar y verificar propiedades de un sistema mediante lógica temporal, y trasladándonos al ámbito de los sistemas de tiempo real, en estos últimos años se han desarrollado diferentes extensiones para la incorporación de medida de tiempo. Estas lógicas, comúnmente denominadas **lógicas temporales de tiempo real**, permiten, en mayor o menor medida, la especificación de propiedades típicas de sistemas RTS como periodicidad, retardos o plazos. Entre las lógicas propuestas encontramos versiones proposicionales y de primer orden, así

como de semántica puntual y de intervalo, de tiempo lineal y de tiempo ramificado, bajo las mismas categorías utilizadas en las lógicas temporales sin tiempo.

Con una semántica puntual las fórmulas se interpretan sobre un punto en el dominio temporal. Por contra, las lógicas temporales de intervalo basan su semántica en intervalos de tiempo que representan trozos finitos del comportamiento del sistema. En la sección 2.2.3 presentamos una breve reseña sobre este tipo de lógicas temporales, pero, en cualquier caso, creemos que la actividad en este campo no es comparable con la obtenida en el caso de lógicas puntuales.

En lo referente a la estructura temporal, el debate sobre la ventajas e inconvenientes de las semánticas lineal y ramificada ha sido profundo; sin embargo, parece existir un acuerdo en que se trata de semánticas incomparables. Así, por ejemplo, las lógicas temporales de tiempo ramificado no pueden expresar restricciones de equidad (*fairness*) en la sintaxis del lenguaje, aunque se pueden trasladar ciertas condiciones de equidad, por ejemplo, mediante restricciones internas en el grafo de alcanzabilidad al utilizar *model checking*; las de tiempo lineal pueden especificar equidad de forma directa. Por otro lado, las lógicas de tiempo ramificado pueden expresar condiciones existenciales sobre evoluciones temporales futuras (pasadas), tales restricciones en los caminos no se pueden expresar en general con una lógica de tiempo lineal.

Independientemente del tipo de estructura, las lógicas temporales de semántica puntual se han extendido para la incorporación de la medida de tiempo utilizando dos enfoques principales (Alur y Henzinger, 1992b): ***hidden clock*** y ***explicit clock***.

Las lógicas ***hidden clock*** utilizan operadores temporales acotados (***bounded operator***). Estas lógicas introducen para cada operador de la lógica temporal una o más versiones acotadas en tiempo. Se trata, pues, de introducir el tiempo en la sintaxis de la lógica reemplazando los operadores temporales no acotados por versiones acotadas en tiempo. Por ejemplo, el operador acotado $\Diamond_{[2,4]}$ se interpreta como "finalmente entre 2 y 4 unidades de tiempo". Así, para una versión acotada en tiempo de la lógica PTL (*Propositional Temporal Logic*), la propiedad de respuesta acotada en tiempo se formularía:

$$\Box(p \rightarrow \Diamond_{[0,3]} q)$$

Como de operadores temporales acotados podemos clasificar la lógicas de tiempo lineal MTL –*Metric Temporal Logic* (Koymans, 1990)– , MITL –*Metric Interval Temporal Logic* (Alur et al., 1996)– , y rEventClockTL – *Event Clock Logic* (Raskin y Schobbens, 1999)– ; así como la de tiempo ramificado RTCTL –*Real Time Computation Tree Logic* (Emerson et al., 1990)–.

Las extensiones con operadores acotados sólo pueden relacionar contextos temporales adyacentes. Es decir, si consideramos la propiedad "todo estímulo p es seguido por una respuesta q y, ésta seguida por una respuesta r en 5 unidades de tiempo desde el estímulo p", no existe una forma directa de expresar este requisito cuantitativo "no local" utilizando versiones acotadas de operadores temporales. Se puede salvar esta deficiencia si extendemos la lógica temporal con referencias explícitas (***explicit clock***) a los tiempos de los contextos temporales. Se han utilizado para ello dos métodos: la cuantificación congelada (***freeze quantification***) y la

utilización de una variable reloj explícita (***explicit clock variable***). A diferencia de las lógicas que utilizan operadores acotados (incluidas en las *hidden clock*), las lógicas extendidas con estos dos métodos no se abstraen totalmente del tiempo, apareciendo referencias explícitas en las fórmulas.

En la cuantificación congelada, se accede al tiempo de un estado por medio de un operador que liga una variable a dicho instante. Por ejemplo, para una lógica temporal lineal, el operador *freeze* "$x.$" liga la variable x al tiempo del contexto temporal actual; la fórmula $x.\phi(x)$ se cumple para el tiempo t si se cumple $\phi(t)$. La fórmula para la propiedad no local sería:

$$\Box x.(p \rightarrow \Diamond(q \wedge \Diamond z.(r \wedge z \leq x + 5)))$$

Como se sigue del ejemplo anterior, la cuantificación congelada sólo permite referencias a tiempos asociados a estados, por lo que se comporta de forma diferente a los cuantificadores estándar de primer orden. Por ser la potencia expresiva de una lógica temporal con cuantificación congelada intermedia entre la potencia expresiva de las versiones proposicionales y las versiones de primer orden, suele referirse a este tipo de lógicas como lógicas temporales de medio orden.

Aunque con origen en la idea de *freeze quantification* introducida en (Alur y Henzinger., 1989), las lógicas catalogadas bajo este tipo de cuantificación utilizan más comúnmente su variante *reset quantification*, como una forma de ligar y reiniciar una variable reloj mediante el operador $x.$.

Como de cuantificación congelada podemos clasificar la lógica de tiempo lineal TPTL –*Timed Propositional Temporal Logic* (Alur y Henzinger., 1989)– y las de tiempo ramificado TCTL –*Timed Computation Tree Logic* (Alur et al., 1993a)– y T_μ –*Timed μ-calculus* (Henzinger et al., 1992b)– , de la que TCTL se puede considerar un fragmento.

Las lógicas que utilizan una variable reloj explícita (*explicit clock variable*) no introducen nuevos operadores temporales, sino que interpretan una variable de estado no rígida, como el tiempo actual en cada estado. El único elemento nuevo en este tipo de lógicas es la capacidad de hacer referencia explícita al reloj. En este caso, la extensión a tiempo real se basa en utilizar una lógica temporal de primer orden. La sintaxis de la lógica utiliza una variable de estado dinámica - T, la variable reloj – y cuantificación de primer orden para variables globales en el dominio temporal. La variable reloj T toma, en cada estado, el valor de tiempo actual. Por ejemplo, con una extensión de este tipo, la propiedad de respuesta acotada en tiempo se formula:

$$\forall x.\Box((p \wedge (T = x)) \rightarrow \Diamond(q \wedge (T \leq x + 3)))$$

donde la variable global x está limitada al tiempo de los estados en que se satisface p.

Como de reloj explícito podemos clasificar las lógicas de tiempo lineal RTTL –*Real Time Temporal Logic* (Ostroff, 1990a)– y XCTL –*eXplicit Clock Temporal Logic* (Harel et al., 1990)–.

Sin embargo, como veremos en la sección 2.2.2, las lógicas *explicit clock*, aunque más expresivas que las *hidden clock*, presentan problemas en decidibilidad para dominio denso si no se limita el tipo de expresiones que involucran a los relojes.

Aunque no mencionado en esta introducción, otra de las características destacables en la definición de lógicas temporales de tiempo real es el dominio de tiempo sobre el que se interpretan. A este respecto nos remitimos a la exposición realizada en la sección 1.3.2. En las siguientes secciones presentaremos brevemente las características de las lógicas temporales de tiempo real más relevantes, tanto lineales como ramificadas, y en la sección 2.2.2 los resultados obtenidos sobre el estudio de su expresividad y complejidad para *model checking* y satisfactibilidad.

Lógicas de tiempo lineal

Las enumeradas a continuación, son lógicas temporales de tiempo lineal que se interpretan sobre trazas de ejecución en un dominio de tiempo discreto:

- MTL –*Metric Temporal Logic* (Koymans, 1990)– es una lógica temporal proposicional con operadores acotados. Sus operadores temporales incluyen versiones acotadas en tiempo de los operadores *until*, *next*, *since* y *previous*.
- TPTL –*Timed Propositional Temporal Logic* (Alur y Henzinger., 1989)– es una lógica temporal de medio orden que utiliza *freeze quantification*. Las expresiones sobre los relojes utilizados en la cuantificación incluyen las primitivas de comparación ($\leq$), congruencia módulo una constante ($\equiv_c$), y la adición de una constante entera ($+c$).
- RTTL –*Real Time Temporal Logic* (Ostroff, 1990a)– es una lógica temporal de primer orden de tipo *explicit clock*. Una fórmula RTTL puede utilizar explícitamente la variable reloj T con cualquier operador aritmético ($+,-,=,\leq,>$), y utilizar cuantificación de primer orden sobre variables de tiempo rígidas. Por tanto, RTTL es muy expresiva pero, como veremos, no decidible en satisfactibilidad y *model checking*.
- XCTL –*eXplicit Clock Temporal Logic* (Harel et al., 1990)– es una lógica proposicional de tipo *explicit clock*, cuyo lenguaje para restricciones de tiempo atómicas permite primitivas de comparación y adición. Por tanto, las restricciones de tiempo de XCTL son más ricas que las de MTL y TPTL, las cuales prohibían la suma de variables temporales. XCTL, sin embargo, prohíbe la cuantificación explícita sobre las variables de tiempo, es decir, se asume que todas las variables globales de tiempo se cuantifican universalmente de forma implícita fuera de cualquier fórmula dada.

Para el caso de dominio denso destacamos:

- MITL –*Metric Interval Temporal Logic* (Alur et al., 1996)– es una lógica temporal de operadores acotados que sólo permite la medida de tiempo con una precisión finita. Como veremos en la siguiente sección, tal restricción hace decidible el problema de satisfactibilidad para dicha lógica, dado que, es posible dividir el dominio temporal en subintervalos salvando así su densidad.
- rEventClockTL –*Event Clock Temporal Logic* (Raskin y Schobbens, 1999) – presentada previamente en (Raskin y Schobbens, 1997) como SCL (*State Clock Logic*), es una lógica temporal de operadores acotados que si bien permite la precisión infinita en sus fórmulas, introduce una medida de tiempo asociada a eventos. Informalmente, dado que los sistemas son de variabilidad finita, se introduce otra forma de salvar la densidad del dominio temporal.

Además, en la literatura se pueden encontrar una gran cantidad de lógicas de aplicación más específica, entre las que destacamos ATPL y TRIO:

ATPL –*Asynchronous Propositional Temporal Logic* (Wang et al., 1993)– es una lógica temporal de tiempo lineal que permite la utilización de relojes de granularidades distintas, proporcionando un marco formal para el razonamiento sobre *clock jitter* (incertidumbre en la medida del tiempo). Al introducir varios relojes, y aunque utiliza tiempo discreto, evita el problema de composición de componentes distribuidos con relojes locales.

TRIO –(Felder y Morzenti, 1994; Ghezzi et al., 1990) – es una lógica de primer orden, aumentada con operadores temporales de tiempo lineal, cuya semántica puede interpretarse sobre dominios temporales que incluyen los enteros, los racionales y los reales. Los dos operadores temporales básicos de TRIO $Futr(A, t)$ y $Past(A, t)$ expresan, intuitivamente, que A se cumple en el instante t en el futuro (resp. en el pasado) con respecto al instante actual. Se trata, pues, de una lógica de tipo *hidden clock*.

Aunque TRIO tiene capacidades expresivas similares a otras lógicas de tiempo real – MTL utiliza los mismos operadores básicos – se diferencia de éstas en el uso de la especificación. Ya que el principal objetivo de TRIO es la ejecución de las especificaciones, no utiliza una metodología de verificación formal. TRIO, más que una verificación, realiza una validación basada en una estructura temporal.

Por tanto, el objetivo de TRIO es proporcionar un lenguaje que de soporte a las especificaciones de sistemas de tiempo real, en particular un entorno que permita la ejecución formal de las especificaciones. Ejecutando las especificaciones formales y observando el comportamiento del sistema especificado, se puede juzgar si las especificaciones capturan los requisitos deseados. La ejecución formal de la especificación TRIO sólo va a ser posible en dominios finitos, a partir de los cuales el usuario debe extrapolar a dominios infinitos. Este método conlleva el riesgo de que el usuario crea que una fórmula se satisface (mediante una comprobación en un dominio finito), cuando realmente la propiedad no se satisface. Aunque el análisis basado en dominios finitos aumenta el grado de confianza en la corrección de la especificación, la inferencia del comportamiento del sistema a dominios infinitos, a partir del análisis realizado en dominios finitos, no puede ser considerada como formalmente probada, siendo, por tanto, responsabilidad del usuario de la herramienta.

Lógicas de tiempo ramificado

Las siguientes son lógicas temporales de tiempo real con semántica ramificada:

- RTCTL –*Real Time Computation Tree Logic* (Emerson et al., 1990)–: Lógica proposicional de tiempo ramificado para sistemas síncronos. Se trata de una extensión del tipo *bounded operator* con una semántica de tiempo discreto puntual. Hace la suposición simplificadora de modelar sistemas de tiempo real síncronos, es decir, considera sistemas en los que todos los eventos ocurren al compás de los *ticks* de un reloj global.
- TCTL –*Timed Computation Tree Logic* (Alur et al., 1993a) –: Lógica proposicional de tiempo ramificado con una semántica menos restrictiva que

RTCTL. Se define sobre una semántica de tiempo denso puntual, en la que la medida de tiempo se introduce mediante *freeze quantification*, permitiendo primitivas de comparación y adición de constante. Con la introducción de TCTL en (Alur et al., 1993a) se aporta el primer algoritmo de *model checking* para tiempo denso.

- T_μ –*Timed μ-calculus* (Henzinger et al., 1992b)–: T_μ es la extensión con tiempo denso de *μ-calculus* mediante la introducción de *freeze quantification*. Las fórmulas en T_μ se construyen utilizando predicados de estado, un operador temporal *next*, el cuantificador de reinicio de variables relojes (*freeze*) y un cuantificador de punto fijo mínimo. Dicha lógica está avalada por el éxito de conseguir el primer algoritmo de *model checking* simbólico para tiempo denso.

2.2.2 Expresividad y complejidad

De los resultados obtenidos en (Alur y Henzinger, 1992b; Ostroff, 1992; Alur y Henzinger, 1993), además de otras lógicas investigadas, se puede realizar una comparativa de la expresividad y complejidad (para satisfactibilidad y *model checking*) de lógicas temporales de tiempo real para dominios de tiempo densos y discretos

En general, aunque las extensiones que utilizan *explicit clock* y *freeze quantification* llevan a lógicas de tiempo real más expresivas, no se conocen resultados positivos en cuanto a decidibilidad (Ostroff, 1992) si no se restringe el tipo de expresiones que involucran en el primer caso a las variables globales y en el segundo caso a las variables utilizadas en la cuantificación *freeze*. Por otro lado, aunque estas extensiones de la lógica temporal explicitan el tiempo, alejándose de la filosofía original de las lógicas temporales sin tiempo (Koymans, 1990), permiten expresar requisitos de tiempo sobre contextos no adyacentes.

En cuanto a la complejidad, desde el punto de vista del análisis formal, son importantes los problemas de satisfactibilidad y *model checking*.

El problema de **satisfactibilidad** afecta a la verificación formal homogénea. Dadas una implementación I y una especificación S (que la implementación debe satisfacer) expresadas como fórmulas Φ_I y Φ_S de la lógica temporal $\mathcal{L}$, entonces, la implementación I cumple la especificación S si la implicación $\Phi_I \rightarrow \Phi_S$ es válida, o de forma equivalente, si $\Phi_I \wedge \neg\Phi_S$ es no satisfactible.

El problema de ***model checking*** es importante en el caso de verificación heterogénea basada en modelo. La implementación I viene dada por un modelo finito basado en una máquina de estados con tiempo (sección 2.4) $\mathcal{M}_I$. La especificación S viene dada por una fórmula lógica Φ_S. Si bien en esta sección se estudia el *model checking* desde el punto de vista de la complejidad asociada a distintas lógicas temporales, en la sección 3.2.2 se detalla el estado del arte en lo referente a las técnicas de *model checking* para sistemas con características de tiempo real.

En cuanto a complejidad, veamos primero el caso de las lógicas sin extensión de tiempo real. En estas lógicas, el problema de satisfactibilidad es, para semántica ramificada, de complejidad exponencial, mientras que el *model checking* es de complejidad lineal. El *model checking* se lleva a cabo mediante un algoritmo específico para las lógicas ramificadas que es lineal en el tamaño del grafo de alcanzabilidad, y lineal en el tamaño de la fórmula que se está verificando. Este

algoritmo no es aplicable a las lógicas lineales, por lo que el *model checking* para estas últimas es exponencial en el tamaño de la fórmula. Por tanto, la aplicación del *model checking* es más eficiente en el caso de lógicas ramificadas. Es por esto que, en la práctica, se utilizan lógicas ramificadas para llevar a cabo la verificación sobre sistemas de estados finitos – mediante *model checking* –, dejándose las lógicas lineales para la verificación de sistemas de estados infinitos mediante un sistema de prueba deductivo.

En los siguientes apartados recapitulamos la complejidad de los problemas de satisfactibilidad y *model checking* para las lógicas de tiempo real más relevantes.

Satisfactibilidad

En cuanto a las lógicas temporales de tiempo real, debemos decir que, como era de esperar, y al igual que ocurre en las lógicas sin tiempo, sólo las lógicas proposicionales son decidibles. Para analizar la complejidad de las lógicas proposicionales se deben tener en cuenta dos parámetros: el dominio del tiempo (Dom) y las operaciones permitidas sobre dicho dominio (Ops).

El conjunto de operaciones permitidas Ops es un parámetro sintáctico que define las fórmulas permitidas en una lógica $\mathcal{L}_{Dom,Ops}$. Como mínimo, el predicado de orden $\leq$ y la adición de una constante entera $+c$ son necesarios para especificar límites inferiores y superiores constantes en la distancia temporal entre eventos – $Ops = Suc$ –. Las lógicas de tipo *bounded operator* se pueden enmarcar dentro de la clase $\mathcal{L}_{Dom,Suc}$, ya que cada fórmula de estas lógicas se puede reescribir como fórmulas *explicit clock* que utilizan predicados de tiempo con las operaciones definidas en Suc.

Existe una caracterización intrínseca a las lógicas de tiempo real con satisfactibilidad decidible. Dicha caracterización es independiente de los detalles de la lógica de tiempo real específica si la lógica es suficientemente expresiva para definir propiedades de puntualidad de la forma:

$$\Box(p \rightarrow \Diamond_{=n} q)$$

En (Alur y Henzinger, 1992b) se demuestra que si $\mathcal{L}$ es un lenguaje que (1) es cerrado bajo todas las operaciones booleanas y (2) puede expresar puntualidad, entonces el problema de satisfactibilidad es no decidible para $\mathcal{L}_{Denso,Suc}$ y $\mathcal{L}_{Discreto,Mas}$, donde el conjunto de operaciones Mas incluye la adición de variables en el dominico temporal.

Dentro del conjunto de lógicas $\mathcal{L}_{Denso,Suc}$, MITL trata de aportar decidibilidad en un modelo de tiempo denso a costa de relajar la noción de puntualidad. Bajo una estructura lineal y una sintaxis de operadores acotados, impone la restricción adicional de que los operadores temporales no pueden estar acotados por intervalos singulares (por ejemplo, $[3, 3]$). Mientras otras lógicas con el propósito de conseguir decidibilidad realizan una abstracción semántica consistente en que en cada cambio de estado sólo se registra una aproximación discreta a tiempo real, MITL hace una concesión sintáctica. En vez de discretizar el significado de una sentencia, se prohíben las restricciones temporales entre estados con precisión infinita:

$$\Box(p \rightarrow \Diamond_{=5} q) \text{ como } \Box(p \rightarrow \Diamond_{(4.9,5.1)} q)$$

Se define por tanto una sintaxis que registra la diferencia de tiempos entre eventos sólo con una precisión finita arbitraria. MITL no puede expresar propiedades de puntualidad, y es por tanto menos expresiva que otras lógicas, pero es decidible para satisfactibilidad incluso en el caso de dominio denso de tiempo. La complejidad de MITL es exponencial en el tamaño de la fórmula, tanto para satisfactibilidad como para *model checking*.

Igualmente bajo un dominio denso, rEventClockTL es de satisfactibilidad decidible. Sin embargo, en este último caso se opta por permitir la puntualidad en las fórmulas y limitar la medida de tiempo a distancias temporales entre eventos.

En cuanto a las versiones discretas, como ya hemos mencionado, el problema de satisfactibilidad es decidible para todas las lógicas en la categoría $\mathcal{L}_{Discrete,Suc}$ (TPTL, MTL), y no decidible para $\mathcal{L}_{Discrete,Mas}$ (RTTL).

El problema de satisfactibilidad para la lógica XCTL es decidible a pesar de admitir la adición en tiempo. XCTL, es igualmente decidible para *model checking*, sin embargo, es un lenguaje no cerrado bajo negación y, por lo tanto, no se puede utilizar para resolver el problema de verificación homogénea.

Model Checking

En el caso de lógicas lineales, el problema del *model checking* es de igual dificultad al de satisfactibilidad, es decir, es no decidible para lógicas de tiempo denso capaces de expresar puntualidad y de complejidad exponencial para las lógicas de tiempo discreto TPTL y MTL. Para XCTL, un algoritmo diferente permite el *model checking* de propiedades de tiempo, con dominio discreto y adición en tiempo.

En (Alur et al., 1996) se presenta un algoritmo de *model checking* para MITL, el primer algoritmo de *model checking* decidible para dominio denso con semántica de tiempo lineal. La complejidad del algoritmo tiene una dependencia exponencial con el tamaño de la especificación y doblemente exponencial con el tamaño de la fórmula a verificar. Por otro lado, limitando la ligadura de los relojes a la ocurrencia de eventos, la lógica rEventClockTL aporta una complejidad exponencial en la fórmula y la especificación, incluso para fórmulas con puntualidad.

En el caso de las lógicas ramificadas, es posible el *model checking* de propiedades (incluyendo puntualidad) incluso sobre un dominio denso. El problema es decidible para TCTL – $\mathcal{L}_{Denso,Suc}$ – tanto para su extensión *bounded operator* como su sintaxis de medio orden (Alur et al., 1993a; Alur, 1991). La complejidad del *model checking* es lineal en los tamaños de la fórmula y de la especificación, pero exponencial en el número de relojes en el modelo y en el tamaño de las constantes de tiempo utilizadas tanto en la especificación como en la fórmula a verificar. La complejidad del *model checking* para las lógicas ramificadas es la misma bajo una interpretación discreta.

2.2.3 Otras lógicas

Además de las lógicas temporales, las lógicas clásicas también se han utilizado para el razonamiento sobre el tiempo. Uno de los primeros intentos fue el realizado por *Jahanian* y *Mok* sobre RTL –*Real Time Logic* (Jahanian y Mok, 1986; Mok, 1991)–.

RTL es una lógica de primer orden para el razonamiento sobre eventos y sus instantes de ocurrencia. Un evento instantáneo se denota como ΩBUT, mientras que las acciones no instantáneas se denotan mediante los eventos de inicio y fin de la acción ($\downarrow ACT$ y $\uparrow ACT$). El tiempo, sobre un dominio discreto, se introduce en el lenguaje mediante la función de ocurrencia @, que asigna valores de tiempo a las ocurrencias de los eventos. Una especificación en RTL podría ser:

$$\forall x[@(\Omega BUT, x) \leq @(\uparrow ACT, x) \wedge @(\downarrow ACT, x) \leq @(\Omega BUT, x) + 30]$$

afirmando que, una vez presionado el botón (evento ΩBUT), la acción se ejecutará en 30 segundos.

La función de ocurrencia de RTL permite una gran expresividad de propiedades de tiempo real periódicas y no periódicas; sin embargo el problema de satisfactibilidad es no decidible. Para llegar a un problema decidible se limitan las fórmulas al conjunto de propiedades de seguridad y se estructuran, incorporando el conocimiento de las restricciones de tiempo en la especificación, en grafos de computación (abstracción finita del modelo). En cualquier caso, su aplicación no es fácil para sistemas grandes.

Por último, y como ya introdujimos en la sección 2.2.1, existen algunas aproximaciones sobre lógicas temporales con semántica de intervalo, basándose la mayoría de ellas en la lógica DC –*Duration Calculus* (Chaochen et al., 1991)–. DC, y en general las lógicas con semántica de intervalo, son muy adecuadas para la especificación de la cantidad de tiempo acumulada durante la que se cumple un predicado de estado (*duration*).

Una fórmula DC, $\int P = 5$, es cierta para un intervalo I cuando la integral en el intervalo satisface $\int_I P = 5$, mientras que una fórmula del tipo $\int P \leq 20 \int Q$ intuitivamente significa que "Q se cumple durante un intervalo por lo menos $\frac{1}{20}$ de tiempo del que se cumple P". Además de la operaciones booleanas incorpora un operador *chop* $(P; Q)$ que se satisface en un intervalo si dicho intervalo puede dividirse en dos intervalos adyacentes, el primero satisfaciendo P y el segundo Q.

Las fórmulas expresadas en DC se pueden interpretar sobre dominios de tiempo denso o discreto, pero en general son no decidibles; sin embargo, también se han identificado fragmentos decidibles de la misma (Chaochen et al., 1993). Aunque alejado del marco de esta tesis, la utilización de DC ha acaparado una gran interés en el campo de los sistemas de tiempo real, prueba de ello es la gran cantidad de resultados encontrados que incluyen el problema de especificación (Veloudis y Nissanke, 1998), verificación mediante *model checking* (Bouajjani et al., 1993) e incluso síntesis (Dierks et al., 1998; Franzle, 1996).

2.3 Álgebras de Procesos

Un álgebra de procesos es un cálculo con operadores para la construcción de procesos complejos a partir de procesos simples. En general, los procesos más simples serán eventos individuales y las operaciones suelen incluir la composición secuencial, composición paralelo, ocultación de eventos, sincronización y elección no determinista.

Con un pequeño conjunto de construcciones, las álgebras de procesos históricas como CCS –*Calculus of Communicating Systems* (Milner, 1980)– y CSP – *Communicating Sequential Processes* (Hoare, 1985)– proporcionan un lenguaje capaz de capturar toda la complejidad de los sistemas paralelos y distribuidos, basándose en las premisas de que las nociones fundamentales para el razonamiento sobre dichos sistemas dinámicos y complejos son la concurrencia y la comunicación. Las aproximaciones basadas en álgebras de procesos vienen a introducir la noción de estructura dentro del conjunto de los métodos formales, pudiendo ser aplicadas para composición de elementos individuales modelados con otros formalismos sin noción de estructura (lógicas, autómatas, sistemas de transiciones, etc). Sin duda, su aportación principal es la especificación y verificación modular.

Para extender la utilidad de las álgebras de procesos a los sistemas de tiempo real se han desarrollados diferentes formalismos que introducen la noción de tiempo mediante la definición de operadores con tiempo sobre procesos. Dichos operadores permiten expresar tres capacidades básicas: retardo de una ejecución t unidades de tiempo, temporización a la espera de la ocurrencia de acciones y la acotación del tiempo asociado a la ejecución de una secuencia de acciones. Así, por ejemplo, la extensión con tiempo de CSP – *Timed* CSP (Reed et al., 1991; Davies y Schneider, 1995)– propone tres operadores con tiempo básicos:

- Operador *timeout*: $P\rho^t Q$ transfiere el control de P a Q si no ocurre ninguna comunicación antes del tiempo t.
- Operador prefijo con tiempo: $a \rightarrow^t P$, que se comporta como P, t unidades de tiempo después de que se haya producido una sincronización en a.
- Terminación con tiempo $WAIT\ t$ representa un proceso que no hace nada salvo terminar con éxito después de un tiempo t.

Aunque, sin duda, *Timed* CSP es una de las álgebras más maduras y desarrolladas de entre las álgebras de procesos con tiempo, encontramos en la literatura gran cantidad de propuestas, entre las que cabe citar TeCCS –*Temporal* CCS (Moller y Tofts, 1990)–, TiCCS –*Timed* CCS (Yi, 1990)–, TPL –*Temporal Process Language* (Henessy y Regan, 1995)–, E-LOTOS – *Enhancements to Language of Temporal Ordering Specifications* (ISO/IECJTC1/SC21/WG7, 1998)– y ATP –*Algebra of Timed Processes* (Nicollin y Sifakis, 1994)–.

En (Nicollin y Sifakis, 1991) se desarrolla una comparativa de las álgebras de procesos con tiempo más importantes. Con tal objetivo, se analizan varias características que se consideran de mayor interés en un álgebra de procesos con tiempo, naturaleza del dominio temporal, determinismo temporal, urgencia, variabilidad finita y acotada, etc. En base a tal estudio y como ya hemos comentado *Timed*

CSP es, de entre las anteriores, el álgebra de procesos con características más adecuadas para la especificación de sistemas distribuidos de tiempo real.

De relación directa con el paradigma de autómata temporizado, en (Yi et al., 1994) se presenta un álgebra de procesos con tiempo denso que proporciona operadores algebraicos para la construcción de autómatas complejos en función de autómatas simples. Se trata de un álgebra de *Timed Automaton*.

Sin embargo, el comportamiento en tiempo de los sistemas de tiempo real depende no sólo de los retardos debidos a la sincronización entre procesos, sino también de la disponibilidad de los recursos compartidos. Las álgebras que acabamos de presentar permiten expresar dichos retardos, pero se abstraen de los detalles específicos de los recursos, suponiendo entornos de operación ideales. Por otro lado, los algoritmos de planificación y de reserva de recursos ignoran los efectos de la sincronización entre procesos.

Aunque el primer reto en la extensión de las álgebras de procesos para su aplicación a RTS fue la introducción de tiempo cuantitativo, sin duda, la compartición de recursos y, por tanto, la planificación es otra característica que, si bien está presente en la mayoría de los sistemas reales, resulta primordial en estos sistemas donde la mayor o menor disponibilidad de los recursos compartidos no se traduce en un problema de eficiencia, sino en la base fundamental de su corrección. Con este objetivo, se han incorporado los conceptos de prioridades y recursos en las álgebras temporizadas, resultando formalismos a caballo entre la especificación y la implementación que no hacen la suposición de *unbounded parallelism*. Así, ACSR –*Algebra of Communicating Shared Resources* (Brémond Grégoire y Lee, 1997)– es un álgebra de procesos basada en el modelo de sincronización de CCS, que incorpora explícitamente recursos y prioridades. En esta misma línea se define TAM –*Temporal Agent Model* (Lowe, 1995)–.

En general, las especificaciones con álgebras de procesos son mucho más concretas y cercanas a la implementación que las realizadas en lógica temporal, siendo estas últimas mucho más apropiadas para las primeras fases del diseño de un sistema, puesto que permiten manejar especificaciones parciales e "imprecisas". Dada una especificación en álgebra de procesos, el conjunto de implementaciones derivadas está limitado a una sola clase de equivalencia de procesos (por ejemplo, los procesos equivalentes bajo bisimulación) y, además, dado que la noción de corrección es definida a partir de dichas equivalencias, el conjunto de implementaciones posibles permanece constante durante los pasos de refinamiento. En (Cerans et al., 1993) se propone TMS –*Timed Modal Specifications*– un álgebra de procesos extensión de TiCCS, que permite el tratamiento de especificaciones más vagas que las de las álgebras de procesos habituales mediante la incorporación de dos modalidades en el prefijo de acción: *may* y *must*. Dichas modalidades determinan qué transiciones son "necesarias" y qué transiciones son "admisibles". La definición de dos tipos de modalidades en las transiciones (*must* y *may*) permite definir una noción de refinamiento más relajada, intuitivamente, cuantas más transiciones *must* y menos transiciones *may* haya en una especificación, mucho más fina será ésta. TMS propone un formalismo que se sitúa entre las álgebras de procesos y las lógicas, haciendo uso de las capacidades composicionales de las primeras y de las especificaciones no influenciadas por factores de implementación de las últimas. Como veremos, este concepto está ligeramente ligado al concepto de subespecificación manejado en esta tesis, siendo por tanto un álgebra de procesos interesante como

punto de partida para encarar la parte constructiva de la metodología propuesta.

Por último, citar que, como veremos en 4.3, recientemente también se han estudiado diferentes extensiones de estas álgebras para el tratamiento de sistemas con requisitos de tiempo *soft*.

2.4 Máquinas de estados

Conviene resaltar que si bien las notaciones formales basadas en máquinas de estados pueden utilizarse para la especificación a bajo nivel de sistemas de tiempo real, en su mayoría son la base para la semántica operacional de notaciones formales basadas en álgebra de procesos o en lógicas. En consecuencia, al mismo tiempo que las lógicas y álgebras se extendían para poder describir propiedades cuantitativas de tiempo o de duración, los modelos de máquina de estados clásicos también fueron adaptados de diferentes formas para el razonamiento sobre el tiempo. En general, la definición de formalismos para especificación de RTS basados en máquinas de estados generalizan sus correspondientes sin tiempo, pudiéndose clasificar en extensiones con tiempo de sistemas de transiciones, autómatas y redes de Petri. En general, los modelos basados en máquinas de estados incorporan el tiempo suponiendo que las transiciones del sistema ocurren de forma instantánea, limitándose a imponer restricciones en el momento en que dichas transiciones están habilitadas.

Aunque, como ya hemos mencionado, en un primer momento existieron posturas favorables a que el tiempo es una característica propia de la implementación del sistema y, por tanto, no debe quedar reflejado en la especificación del mismo, pronto se aceptó la necesidad de incorporación de requisitos de tiempo cuantitativos en el modelo del sistema.

En las primeras aproximaciones al diseño de sistemas de tiempo real se adoptó una hipótesis **síncrona**, en la que se asumía que los eventos externos al sistema estaban ordenados en el tiempo y el programa respondía instantáneamente a dichos eventos – *Statecharts* (Harel, 1987) –. Tratar la respuesta del sistema como instantánea es una idealización que podría aplicarse si siempre el tiempo de respuesta es menor que el tiempo mínimo entre dos eventos externos y, por tanto, la incorporación del tiempo en el modelo del sistema se hacía necesaria para un modelado más realista y de mayor aplicación (Joseph, 1996).

Las primeras aproximaciones con tiempo, basadas en máquinas de estados, asumen un dominio **discreto** utilizando una aproximación de **reloj global ficticio**. Se introduce un transición especial *tick* en el modelado del sistema; de esta forma, el tiempo puede verse como una variable de estado global cuyos valores abarcan el dominio de los números naturales y que se incrementa con cada transición *tick*. El retardo de tiempo entre dos eventos se mide contando el número de transiciones *ticks* entre ellos. La incorporación del evento *tick* permite utilizar las metodologías de razonamiento sobre sistemas sin tiempo; pero, cuando existan k *ticks* entre dos eventos, sólo se podrá inferir que el retardo entre ellos es, por lo menos, $(k-1)$ unidades de tiempo y, como mucho, $(k+1)$ unidades de tiempo. En consecuencia, es imposible establecer de forma precisa requisitos en los retardos tales como "el retardo entre dos transiciones es igual a 2 segundos".

Bajo esta aproximación de reloj global ficticio se desarrollaron modelos como TTM –*Timed Transition Model* (Ostroff, 1990b)–, *Modechart* –(Jahanian y Mok, 1994)– y la primera versión de TTS – *Timed Transition System* (Henzinger et al., 1991)–.

Aproximaciones posteriores al modelado de sistemas de tiempo real incorporan el tiempo de forma más realista, como una magnitud **densa**. Con cada transición del sistema modelado asociamos un valor de tiempo del conjunto de reales no negativos ($\in \Re_+$). Vemos las computaciones del sistema como árboles densos. En un árbol denso, los posibles caminos del árbol son correspondencias del dominio de tiempo $\Re_+$ a los estados del sistema. Este modelado denso difiere del modelado discreto, ya que permite expresar límites exactos en las distancias entre eventos.

Bajo esta aproximación de tiempo denso, se encuentra la segunda versión de TTS y su evolución hacia CTS –*Clocked Transition Systems* (Manna y Pnueli, 1995)–, y la mayoría de los modelos basados en autómata como *Timed I/O Automaton*, en sus dos versiones, la de *Meritt-Modugno-Tuttle* (Meritt et al., 1991) y la de *Lynch-Vaandrager* (Lynch y Vaandrager, 1992), TA –*Timed Automaton* (Alur y Dill, 1990a)– , así como sus distintas variantes (ver sección 2.4.1).

Independientemente del dominio de tiempo utilizado, la incorporación de restricciones de tiempo en el modelado del sistema se ha realizado mediante dos mecanismo básicos. En la extensión de los sistemas de transiciones y de las redes de Petri se introducen límites inferiores (I) y superiores (S) en las transiciones. La transición estará habilitada cuando hayan transcurrido I unidades de tiempo y dejará de estarlo cuando hayan transcurrido S.

Bajo este tipo de introducción de restricciones de tiempo, los TTSs (*Timed Transition Systems*) son la extensión de los sistemas de transiciones para la incorporación de tiempo realizada por *Henzinger*, *Manna* y *Pnueli* en (Henzinger et al., 1991). Los sistemas de transiciones temporizados generalizan el modelo computacional propuesto por *Pnueli* y *Keller* (FTS –*Fair Transition Systems*–). Similar extensión se ha realizado en (Meritt et al., 1991; Lynch y Vaandrager, 1992) para *I/O Automaton* y por *Ostroff* para los TTMs.

Por otro lado, en la extensión del autómata finito con tiempo TA – *Timed Automaton*–, se permite representar explícitamente restricciones del tiempo transcurrido al atravesar un camino cualquiera en el autómata, no sólo el empleado en una transición. El TA fue propuesto como un modelo abstracto para RTS en (Dill, 1989) y (Alur y Dill, 1990a).

Un autómata temporizado TA opera con un control finito, un conjunto finito de localizaciones de control y un conjunto finito de relojes con valores reales. No se trata de relojes locales del sistema, sino que todos evolucionan a un mismo ritmo (según un reloj global común) registrando el tiempo transcurrido desde que han sido iniciados o reiniciados. Para incorporar las restricciones de tiempo del sistema, las transiciones se etiquetan con expresiones sobre los valores de dichos relojes. Este formalismo generaliza, para la inclusión de tiempo, el autómata sobre palabras o grafos de longitud infinita (w-autómata). Aunque la aplicación de análisis formal a una especificación de este tipo (como veremos en el capítulo 3) limita el conjunto de expresiones a combinaciones de límites inferiores o superiores sobre la diferencia entre relojes, este modelo se ha mostrado suficientemente expresivo para el modelado de una gran cantidad de sistemas de tiempo real.

Con posterioridad a la aparición del TA, *Manna* y *Pnueli* (Manna y Pnueli, 1995) adaptan el modelo de *Timed Transition Systems*, introduciendo cambios basados en TA, proponiendo CTS (*Clocked Transition System*) como modelo computacional para los sistemas de tiempo real. Este nuevo modelo sobre dominio denso, al igual que TA, representa el tiempo mediante un conjunto de relojes que se incrementan uniformemente a medida que el tiempo progresa, y que se pueden reasignar en las transiciones del sistema. Al igual que los TTS, los CTS se basan en los FTS e intentan reducir el problema de verificación basado en prueba deductiva a la verificación sobre FTS. En consecuencia, y a diferencia del TA, se propone un modelo no limitado a sistemas de estados finitos, que utiliza las metodologías ya desarrolladas para los FTS (Bjøner et al., 1996) para la verificación de algunas propiedades de seguridad.

Por último, y aunque los modelos más evolucionados son las extensiones de los sistemas de transiciones y los basados en la teoría de autómatas, también se han investigado la posibilidad de utilizar formalismos basados en redes de Petri, originados en los trabajos (Ramchandani, 1976; Merlin, 1976). En (Bruno et al., 1993; Berthomieu y Diaz, 1991; Leveson y Stolzy, 1987) encontramos propuestas de análisis de sistemas de tiempo real basadas en este tipo de extensiones.

En su mayoría, los formalismos con semántica de tiempo denso orientados a modelo (redes de Petri, álgebras de procesos, autómatas, etc) se basan en la suposición de ortogonalidad entre los cambios continuos y discretos, lo que simplifica de forma drástica la semántica subyacente: un ejecución de un sistema temporizado es un secuencia de pasos que alterna avances de tiempo densos y transiciones discretas. Un punto delicado en la definición de la semántica es la forma en que dichos cambios interaccionan, introduciéndose para ello o bien invariantes de tiempo o plazos en las transiciones (ahondaremos en este hecho en el capítulo 6).

2.4.1 Variantes del autómata temporizado

A pesar de que el paradigma de *Timed Automaton* es el modelo con mayor éxito en el campo del tratamiento formal denso de los sistemas de tiempo real, son muchas las variaciones de dicho paradigma que se pueden encontrar en la literatura. A lo largo de esta tesis haremos referencia a algunas de estas variantes, haciendo hincapié en las ventajas que aportan.

Especial atención han merecido las variantes cerradas bajo la operación de complementación. Así como el problema de satisfactibilidad para lógicas de tiempo denso es no decidible si se permite puntualidad en los predicados de tiempo, el modelo TA original con relojes en un dominio denso no es cerrado bajo complementación. Así, de la misma forma que las lógicas de tiempo real se limitan no permitiendo precisión infinita, los relojes de un autómata temporizado se limitan a relojes asociados a eventos, resultando en la variante ECA –*Event-Clock timed Automaton* (Alur et al., 1999)– que es cerrada bajo todas las operaciones booleanas. En (Henzinger et al., 1998) se presenta la relación entre las clases de lenguajes resultantes de la aplicación de ambas restricciones. Retomaremos esta cuestión en el capítulo 9 dedicado a la síntesis en la metodología propuesta.

CAPÍTULO 3

Análisis y síntesis formal para RTS

Si bien las notaciones formales son importantes, por permitir construir una especificación no ambigua del sistema o de sus propiedades, el éxito del tratamiento formal de los sistemas software se basa principalmente en la obtención de procedimientos de decisión para la verificación y para la síntesis. En este capítulo se resume el estado del arte a este respecto, centrando mayoritariamente la exposición en los métodos de decisión asociados al paradigma del autómata temporizado

3.1 Introducción

En el concepto más puro de análisis, la ingeniería del software debe incorporar, como otras disciplinas, métodos formales cuyo propósito sea la predicción del comportamiento y las propiedades del software, basándose en cálculos realizados sobre un modelo matemático, es decir, los métodos formales ligados a la ingeniería del software no se pueden quedar en un uso puramente descriptivo. Idealmente, las notaciones formales deberían permitir una metodología de deducción eficiente, y ser lo suficientemente legibles como para alcanzar sus objetivos descriptivos. Desgraciadamente, en el estado del arte actual, las notaciones que son ampliamente aceptadas como legibles son mucho menos tratables para el razonamiento automático y viceversa.

Entre el uso puramente descriptivo de las especificaciones formales, y su utilización para hacer predicciones generales sobre su comportamiento, existen diferentes niveles de análisis. El nivel más bajo de análisis no intenta deducir propiedades del sistema descrito mediante una especificación formal, sino probar que la especificación está lo suficientemente bien construida como para ser la especificación de algo. Este tipo de análisis, análisis de consistencia, está presente en todos los formalismos del estado del arte actual.

Aunque el objetivo final, el que incorpora un mayor grado de formalidad, del uso de métodos formales es la verificación formal, dicha tarea es fuertemente dependiente de la validación. Dado que las predicciones que nos permiten realizar

los métodos formales se basan en un modelo matemático del sistema, si el modelo es inadecuado las predicciones no serán ciertas en el sistema real. Es por ello que los métodos de verificación deben de ir acompañados de diferentes técnicas de validación; mientras que el análisis de consistencia nos garantiza que la especificación significa algo, la validación nos permite establecer si lo que significa es lo que se estaba buscando. Como de validación clasificaremos algunas técnicas de análisis como son la simulación simbólica y el prototipado (sección 3.6). Una vez llegados a este punto podremos realizar la verificación del sistema.

En cuanto a la verificación del sistema especificado, debemos tener en cuenta que, debido a su dominio de aplicación, una gran cantidad de los sistemas de tiempo real son inherentemente críticos. Una metodología de análisis dinámico basada en prueba no puede proporcionar ningún resultado de corrección absoluto. Además. el conjunto de comportamientos posibles en un RTS sobrepasa el conjunto de comportamientos que puede ser alcanzado incluso con billones de pruebas. Entre el conjunto de comportamientos no examinados pueden existir combinaciones críticas no probadas, o lo que es peor, modos de fallo del sistema desconocidos.

Por tanto, la forma de análisis que se considera más conveniente es la verificación formal, que intenta demostrar que el sistema satisface ciertas propiedades dadas. Una verificación formal permite encontrar errores no obvios en los diseños. En la verificación formal se consideran TODAS las posibles interacciones de los procesos concurrentes en sistemas de tiempo real. Con frecuencia, las diferentes interacciones entre procesos llevan a errores sutiles, difíciles de descubrir utilizando técnicas de simulación y prueba.

De todas formas, debemos notar que la verificación formal del sistema no prueba necesariamente la ausencia de errores en el sistema final si en el diseño se han abstraído detalles importantes; y que, aunque la prueba no puede proporcionar una garantía de corrección absoluta, se considera que es una técnica complementaria a la verificación formal debido a los problemas de escalado de esta última. (sección 3.6.2)

En este capítulo presentaremos los distintos enfoques de verificación formal, haciendo especial hincapié en la verificación formal basada en *model checking*. También haremos una breve introducción a las técnicas de validación y prueba (sección 3.6). Por último, en la sección 3.7, nos movemos del análisis al uso más ambicioso de las técnicas formales para la síntesis *software*. Dado que este tesis se enmarca en las primeras fases del ciclo de vida, nos centraremos en la síntesis a partir de propiedades expresadas en lógica.

3.2 Verificación formal

En el ámbito de las técnicas de descripción formal encontramos fundamentalmente dos grandes enfoques para la verificación formal: **demostradores de teoremas** (*theorem proving* o *proof-theoretic*) y ***model checking*** (o *model-theoretic*). Ambos van un paso más allá de la especificación, al tratar de utilizar las técnicas formales para analizar el sistema en busca de las propiedades deseadas.

En las técnicas de verificación con demostradores de teoremas se utilizan axiomas y reglas de inferencia para probar formalmente que una especificación satisface

una propiedad de interés, o que una especificación de un sistema es equivalente o está incluida en otra. Las técnicas de verificación *model-theoretic* realizan la verificación mediante la exploración exhaustiva de una abstracción finita del sistema (modelo).

3.2.1 *Proof-Theoretic*

Las técnicas de reglas deductivas resultan difíciles de automatizar y conllevan, generalmente, una gran cantidad de interacción con el diseñador, al que se le presupone un profundo conocimiento matemático. Las herramientas desarrolladas en base a este enfoque no son, por tanto, totalmente automáticas. El proceso es lento y a menudo propenso a error, aunque en dicho proceso el usuario adquiere un conocimiento mayor tanto del sistema como de la propiedad. En contrapartida, las técnicas *proof-theoretic* permiten manejar, de forma directa, espacios de estados infinitos, mediante la utilización de técnicas de inducción.

Para el caso de los sistemas de tiempo real, TAME –*Timed Automaton Modeling Environment* (Archer y Heitmeyer, 1996)– propone un método formal basado en *Lynch-Vaandrager I/0 Automaton* para la especificación y análisis de sistemas de tiempo real, que utiliza el *theorem prover* PVS (*Prototype Verification System*). Si bien se han incorporado una gran cantidad de estrategias y heurísticos, y aunque algunos pasos en la prueba se han automatizado, sigue siendo bastante interactivo.

Aunque los demostradores de teoremas han sido utilizados en investigación para la verificación de diferentes algoritmos y protocolos, su utilización en la industria del software ha sido muy reducida. Según (Heitmeyer, 1998), para que la utilización de los demostradores de teoremas sea amplia, los lenguajes de especificación deberían ser más naturales, y los pasos de razonamiento que realizan deberían estar más próximos a los pasos en la prueba realizada "a mano", ya que actualmente los pasos son demasiado detallados.

3.2.2 *Model-Theoretic*

En la verificación basada en modelo, el usuario utiliza una herramienta *software* que examina un modelo finito de máquina de estados del sistema para la verificación de una propiedad específica. La principal ventaja de este enfoque es que es automático y puede ser utilizado por usuarios con una mínima formación matemática. Además, se puede utilizar para la prueba de especificaciones parciales por lo que puede proporcionar información sobre la corrección del sistema incluso si éste no está completamente especificado. Por último, y no por ello menos importante, permite la obtención de contraejemplos que revelan errores sutiles en el sistema.

En líneas generales, la prueba se realiza mediante una búsqueda exhaustiva del espacio de estados del modelo. Dicha búsqueda siempre finaliza dado que el modelo es finito. Desgraciadamente, éstas técnicas, que exploran el total del espacio de estados, se encuentran limitadas por la explosión en el tamaño del espacio de estados, que es exponencial en el tamaño de las descripciones en la composición de especificaciones. El reto tecnológico de este tipo de técnicas es la mejora de eficiencia de los algoritmos y de las estructuras de datos con el fin de permitir el

manejo de espacios de búsqueda de gran tamaño. En este sentido, se han utilizado diversas aproximaciones para atacar el problema que se basan en eliminar los estados no alcanzables y los estados imposibles; describir simbólicamente conjuntos grandes de estados; y utilizar procedimientos de decisión especiales, como los BDDs (Bryant, 1986), para la evaluación eficiente de fórmulas booleanas.

Más concretamente, hoy en día, se utilizan bajo la denominación común de *model checking* básicamente dos tipos de enfoques: ***temporal model checking*** y ***behaviour conformance checking***.

En el enfoque de *model checking* temporal, la especificación se expresa en lógica temporal y el sistema se modela mediante una máquina de estados. Se utiliza un procedimiento de búsqueda para comprobar si el sistema es un modelo para la especificación.

En el segundo enfoque, la especificación es una máquina de estados; entonces, se compara el sistema (también modelado como una máquina de estados) para determinar si su comportamiento se ajusta o no la especificación. Se pueden utilizar diferentes nociones de cumplimiento de la especificación: inclusión del lenguaje generado por la especificación y el modelo; refinamiento, equivalencia observacional, etc.

Ambos enfoques *model-theoretic* no son totalmente independientes entre sí, siendo este último el más genérico, dado que el problema de *model checking* temporal se puede reescribir en términos de máquinas de estados.

En el caso que nos ocupa, las aproximaciones al problema del *model checking* en sistemas con un espacio de estados infinito ha sido la clave para la extensión de la verificación automática mediante *model checking* a tiempo real, además de para el estudio de sistemas parametrizados y protocolos de comunicación.

Sin duda, dentro del estudio de las técnicas de verificación formal, una de las áreas de investigación más activas en los últimos años ha sido el estudio de algoritmos de *model checking* para sistemas de tiempo real. Dichos estudios han resultado en diferentes algoritmos de *model checking* para especificaciones en lógicas de tiempo real (sección 3.3).

En lo que sigue nos centraremos en la presentación de los resultados obtenidos en cuanto a *model checking* temporal con lógicas temporales ramificadas de tiempo denso, dado que éstas son las únicas decidibles sin relajar la noción de puntualidad (sección 2.2.2). Este problema ha sido abordado recurrentemente estos últimos años, dando como resultado la aparición de distintas herramientas (ver sección 3.4).

3.3 *Model Checking*

La solución al problema del *model checking* con semántica de tiempo denso tiene su origen en (Alur et al., 1993a) (por primera vez en 1990). En este trabajo, *Alur et al* llegan a la verificación de un autómata extendido con relojes de valores reales para la lógica de tiempo real TCTL. El avance crucial que lleva a la resolución de la verificación sobre un dominio de tiempo denso es que el espacio de estados infinito resultante de la asignación de valores a los relojes del sistema se puede particionar

en un espacio cociente finito. De esta forma, el dominio infinito de interpretaciones de los relojes se abstrae a un conjunto finito de clases de equivalencia (con un número de elementos infinito), a los que se denomina **regiones**, tales que las interpretaciones en la misma región inducen estados que satisfacen las mismas propiedades lógicas en TCTL. Posteriormente, dicho resultado fue extendido a la lógica T_μ (Henzinger et al., 1992b), de la cual TCTL se puede considerar un fragmento. Además del *model checking* sobre TCTL y T_μ, la construcción de dicho espacio cociente finito se ha aplicado a la resolución de problemas como vacuidad de un autómata, alcanzabilidad, relaciones de orden entre autómatas (simulaciones y bisimulaciones) y síntesis de controladores.

Desafortunadamente, el grafo de regiones (resultante de la partición en regiones) crece exponencialmente no sólo con el número de elementos que intervienen en la composición paralelo, sino también con el número de relojes y la magnitud de la constante más grande que interviene en el modelo y en la fórmula a verificar. Este hecho hace que el principal inconveniente que se achaca al *model checking* sin tiempo se vea aún más agudizado. Si bien incorporando las regiones hacemos decidible un problema que hasta el momento no lo era, el algoritmo sigue presentando problemas de explosión del espacio de estados a explorar. Dicha explosión se traduce ahora en dos explosiones potenciales: explosión en el espacio de regiones y explosión en el espacio de estados de control. Los algoritmos basados en la construcción explícita de la partición en regiones serían muy poco eficientes en la práctica, de hecho, en la revisión del estado del arte no hemos encontrado ninguna herramienta de verificación formal basada estrictamente en el grafo de regiones.

Ante el problema localizado de la explosión de tamaño, y con el fin de que los resultados obtenidos de forma teórica lleven a herramientas que puedan ser aplicadas a problemas reales, resulta imprescindible aplicar algún tipo de técnica de reducción. Las técnicas de reducción utilizadas son, en general, de naturaleza bastante diversa, pero se pueden clasificar bajo dos grandes enfoques: se trata de, o bien reducir la complejidad técnica mediante la representación eficiente de la información de control (usualmente mediante BDDs – *Binary Decision Diagrams* (Bryant, 1986) –) y de la información de tiempo (usualmente mediante DBMs – *Difference Bound Matrix* (Dill, 1989)), o bien simplificar la complejidad teórica mediante la aplicación de abstracciones del espacios de estados (Heitmeyer et al., 1998; Nitsche, 1994), detección de simetrías (Emerson y Trefler, 1998; Ajami et al., 1998), etc. En la siguiente sección veremos las técnicas más relevantes que han sido utilizadas en la práctica.

3.3.1 Mejoras a la verificación basada en regiones

Para evitar el problema de explosión en el número de estados del grafo de regiones se han utilizado técnicas de diferente naturaleza, que resumimos a continuación. Existen tres técnicas que, por su entidad, constituyen un cambio conceptual básico: la utilización de representantes discretos de las regiones, el cómputo simbólico de la verificación, y la utilización de particiones menos finas que la utilizada en el grafo de regiones. Además, existen un sinfín de técnicas subsidiarias que, en su mayoría, tratan de aplicar conceptos ya utilizados con éxito en el tratamiento de sistemas sin tiempo y que suponen variaciones de menor entidad que las anteriores.

Análisis mediante representantes

Esta aproximación consiste en discretizar el espacio de las interpretaciones de los relojes y el progreso del tiempo de forma que, por lo menos, un representante de cada una de la regiones pertenezca al espacio discreto construido. Aunque esta aproximación puede sufrir de la misma explosión en el tamaño de estados que la técnica basada en regiones (ya que contiene por lo menos un representante por región), su principal ventaja es la posibilidad de utilizar algoritmos y estructuras de datos ya aplicados con éxito en el análisis de sistemas discretos.

El análisis utilizando representantes se ha mostrado eficiente para redes booleanas asíncronas y circuitos MOS (Bozga et al., 1997). La eficiencia obtenida reside en la utilización de NDDs (*Numeric Decision Diagrams*) que no son más que BDDs aplicados a la codificación binaria de los relojes una vez discretizados.

La idea que subyace bajo la utilización de NDDs es sencilla. Si los relojes pueden tomar valores en un rango $[0, k)$ y consideramos un dominio discreto del tiempo $\{0, \cdots, k-1\}$, cada reloj puede ser codificado en binario utilizando $\log k$ bits. En consecuencia, cualquier subconjunto de K^n puede ser considerado como un subconjunto de $\{0,1\}^{n \log k}$ y representado por un BDD sobre $n \log k$ variables booleanas.

Análisis simbólico

La principal limitación del *model checking*, que aparece también en el caso de modelos sin tiempo, radica en el tamaño del espacio de estados que crece exponencialmente con el número de componentes. Un método ampliamente utilizado, a la hora de atajar el problema de explosión de estados, es la representación simbólica (en lugar de enumerativa) de conjuntos de estados, que permite calcular el conjunto de estados que satisface una fórmula como un punto fijo sobre una función de predicados lógicos sobre los estados. Aunque dicha posibilidad ya se apunta como teóricamente posible en (Emerson y Clarke, 1981), la eficiencia práctica del método sólo se alcanzó con la incorporación de estructuras de datos eficientes para el cálculo (BDDs).

La historia se repite en el caso de *model checking* para el análisis de sistemas con tiempo, donde, además, como ya hemos mencionado, la explosión del tamaño del espacio de estados se ve agudizada por la incorporación del tiempo.

El grafo de regiones es el cociente obtenido del modelo operacional del sistema con respecto a la equivalencia de regiones. Aunque un nodo del grafo de regiones es un conjunto de estados "equivalentes", es decir, o todos los estados de una región verifican una propiedad o bien ninguno la verifica, desde el punto de vista de los algoritmos de verificación basados en regiones, una región puede considerarse como un estado y el grafo de regiones como el espacio de estados de entrada al algoritmo de verificación. Desde este punto de vista, podemos clasificar a los algoritmos basados en el grafo de regiones como algoritmos enumerativos.

La solución simbólica al problema de la explosión de tamaño del grafo de regiones manipula conjuntos de regiones en vez de regiones individuales. De los resultados obtenidos en (Alur et al., 1993a) se deduce que un conjunto de regiones se puede caracterizar mediante la conjunción y disyunción de relaciones lineales

sobre los relojes del sistema. Este hecho ha llevado de forma natural a realizar la verificación sin necesidad de construir de forma explícita el grafo de regiones, sino realizando el análisis mediante la manipulación simbólica de dichas relaciones lineales entre los relojes.

El *model checking* simbólico para sistemas con tiempo fue postulado por primera vez en (Henzinger et al., 1992b) para el caso de verificación de propiedades expresadas en Tμ. Los resultados empíricos muestran que la complejidad debida a la magnitud de las constantes de tiempo se elimina en la práctica mediante un algoritmo de verificación simbólico. Sin embargo, el número de relojes sigue siendo un obstáculo importante a la hora de desarrollar algoritmos de verificación eficientes.

Al igual que en el caso de la solución simbólica para el *model checking* de sistemas sin tiempo, la aplicación práctica de la técnica pasa por encontrar estructuras de datos eficientes a la hora de manipular los conjuntos de regiones. La elección de dicha estructura de datos en la gran mayoría de los algoritmos simbólicos se ha decantado a favor de los DBMs (Dill, 1989).

Análisis mediante particiones menos finas (abstracciones)

La última aproximación consiste en encontrar una partición del espacio de estados con dos propiedades básicas. En primer lugar, que la partición sea menos fina y, por tanto, tenga menor número de clases que la partición en regiones; y, en segundo lugar, que la resolución del problema de verificación para el que se construye la nueva partición tenga la misma respuesta para todos los estados de la misma clase.

Por tanto, se trataría de encontrar una relación de equivalencia que proporcionase un grafo cociente más pequeño que el grafo de regiones. Para la obtención de dichas relaciones se utilizan bisimulaciones de abstracción de tiempo, donde el aspecto cuantitativo del tiempo se oculta en mayor o menor medida: sabemos que el tiempo pasa, pero no sabemos cuánto. Evidentemente, la relación de equivalencia que define las regiones en (Alur et al., 1993a) es una bisimulación de abstracción de tiempo, aunque no es necesariamente la mayor para la resolución de un problema concreto.

Debemos tener presente que, además de reducir el tamaño del espacio de estados cociente, una abstracción también implica la pérdida de parte de la información que contenía el grafo original. Cuanto más fuerte sea la abstracción, menor será la pérdida de dicha información, pero también menor la reducción alcanzada en el tamaño del espacio de estados.

En (Tripakis, 1998) se definen tres tipos de bisimulaciones de abstracción de tiempo. Estas bisimulaciones han sido incorporadas en la herramienta KRONOS, y se demuestra que, aunque sólo la más fuerte preserva propiedades de tiempo ramificado, todas preservan propiedades de tiempo lineal.

En realidad, el concepto de abstracción se encuentra en la obtención del grafo de regiones, ya que el espacio de regiones no es más que una abstracción exacta para propiedades expresadas en TCTL. Por tanto, cuando hablamos de abstracciones como un medio para aliviar el problema de la explosión de estados, nos

referimos a abstracciones más gruesas que la de regiones, e incluso de abstracciones con pérdidas de información, aunque seguras para la resolución de un problema concreto.

Otras aproximaciones

En la literatura encontramos, además, gran cantidad de aproximaciones de menor entidad, entre las que resaltamos las siguientes:

- La naturaleza independiente de la propiedad a verificar del espacio de regiones lleva a un particionado del conjunto de interpretaciones muy fino (y, por tanto, con un número de regiones muy grande). En (Larsen et al., 1995a) se propone una técnica simbólica de manipulación de las restricciones sobre los relojes en la que se tiene en cuenta la propiedad a verificar a la hora de realizar la partición del espacio de estados. Adicionalmente, se propone una estrategia composicional que va eliminando cada uno de los componentes en la especificación al mismo tiempo que modifica la propiedad a verificar de acuerdo al componente eliminado.

- En base al momento en el que se genera el espacio de estados a explorar, distinguimos los algoritmos **al vuelo** y **a priori**. En los algoritmos al vuelo, los estados se computan y almacenan conforme se van necesitando, mientras que en los algoritmos a priori se genera el espacio de estados total antes de empezar la exploración. El algoritmo de *model checking* para TCTL propuesto en (Alur et al., 1993a), además de enumerativo, es un algoritmo a priori. En (Henzinger et al., 1996) se presenta un método basado en la construcción del grafo de regiones al vuelo, pero, al no tratarse de una técnica simbólica conlleva, en el peor caso, la construcción de todo el grafo de regiones.

- Aunque la solución simbólica para TCTL (Henzinger et al., 1992b) opera sobre conjuntos de estados, presenta la desventaja de que el algoritmo propuesto no se trata de un algoritmo al vuelo. La integración de técnicas simbólicas y al vuelo se ha aplicado a la resolución del *model checking* (Bouajjani et al., 1997) para una lógica basada en autómata $\text{TECTL}_\exists$ (Bouajjani et al., 1996).

- Aunque las técnicas presentadas y las estructuras de datos que utilizan se han diseñado para intentar aliviar los problemas de explosión de tamaño, en la práctica, el espacio de estados que debe ser explorado podría aun así exceder la memoria disponible. Para salvar este problema, muchos de los algoritmos se han modificado para realizar el análisis no sobre el espacio de estados exacto, sino sobre una sobre-aproximación o sub-aproximación de dicho espacio. En (Wong Toi, 1995) se realizan tanto sobre como sub-aproximaciones para el cálculo del conjunto de estados alcanzables de un autómata temporizado. El conjunto de estados alcanzables se aproxima por arriba y por abajo. Si la sobre-aproximación no contiene estados que violan la especificación el sistema, ya ha sido verificada la corrección; además si la sub-aproximación contiene estados que violan la especificación, se ha probado que el sistema no es correcto.

- En (Daws y Yovine, 1996) se realiza una aproximación basada en la experiencia para reducir el número de relojes. Se observa que el número de relojes en

la especificación de un sistema crece principalmente por dos motivos. Por un lado, las especificaciones se suelen escribir en lenguajes de alto nivel que, al traducirse a un formalismo basado en máquina de estados, involucran un número de relojes proporcional al número de *time outs* en la especificación. Sin embargo, estos *time outs* rara vez están activos al mismo tiempo. Por otro lado, es habitual modelar los sistemas complejos mediante la composición de componentes más simples (con un número pequeño de relojes); debido a la sincronización en las transiciones, muchos relojes son reinicializados a la vez y, por tanto, serán iguales por algún tiempo (ya que se incrementan al mismo ritmo). Se proponen, en consecuencia, dos algoritmos: uno para detectar relojes inactivos y otro para detectar relojes iguales.

- Existen diversas aproximaciones que intentan extender las técnicas de órdenes parciales a modelos de estados con tiempo. Los órdenes parciales se han utilizado con éxito para reducir la explosión del tamaño del espacio de estados resultante del modelado *interleaving* de la concurrencia. La técnica se basa en construir un grafo para verificación en el que, para cada estado, sólo se incluyen una transición por cada conjunto de transiciones equivalentes. Informalmente, dos transiciones que parten del mismo estado son equivalentes si la ejecución de una de ellas no elimina la posibilidad de tomar la otra y la ejecución de ambas es conmutativa. En (Pagani, 1996) se aplica este tipo de técnica para la reducción del número de estados del grafo de regiones, y más recientemente, el trabajo presentado en (Bengtsson et al., 1998).

- En (Emerson y Trefler, 1998) se propone una reducción basada en la detección de simetrías en el modelo. La detección de simetrías, ya aplicada en los sistemas sin tiempo, es una abstracción que aporta una importante reducción del espacio de estados a explorar cuando el sistema presenta un elevado nivel de simetría, es decir, el modelo a explorar se corresponde con la composición de un conjunto de componentes idénticos (protocolos, exclusión mutua, etc.).

3.4 Herramientas de verificación

En paralelo con los resultados teóricos, se han desarrollado una serie de herramientas de verificación con tiempo denso. KRONOS ha sido la primera, seguida de Uppaal, RT-SPIN y Timed-COSPAN. Todas estas herramientas se basan, en mayor o menor medida, en el modelo de autómata temporizado, aunque difieren en los lenguajes de especificación de propiedades que utilizan.

La existencia de herramientas ha sido de primordial importancia a la hora de introducir la aproximación de tiempo denso, no sólo dentro del ámbito de la investigación, sino también como una solución factible en problemas reales. El tratamiento de casos de estudio reales, como los abordados con Uppaal y KRONOS, ha puesto de manifiesto que el modelo, las técnicas y, en menor medida, las herramientas han alcanzado el grado de madurez suficiente para ser utilizadas en la práctica.

3.4.1 RT-SPIN

En (Tripakis y Courcoubetis, 1996) se extiende el lenguaje de especificación PROMELA a RT-PROMELA, un lenguaje de especificación para sistemas de tiempo real asíncronos. Asimismo, se adecúa el módulo de verificación SPIN (Bell Labs) a RT-SPIN, realizando verificación homogénea con tiempo mediante la utilización de un autómata generado a partir de la especificación RT-PROMELA. Los algoritmos utilizan uniones convexas de regiones, y DBMs para el almacenamiento de dichas uniones convexas.

3.4.2 Timed-COSPAN

La herramienta COSPAN es una herramienta de modelado y verificación homogénea basada en autómata y desarrollada en Bell Labs. Su extensión para tiempo real (Alur y Kurshan, 1996), basada en *Timed Automaton*, realiza el análisis utilizando un cociente basado en regiones, o más exactamente zonas (uniones convexas de regiones). Permite la utilización tanto de un algoritmo enumerativo y al vuelo como de un algoritmo simbólico basado en BDDs. Además, introduce dos métodos aproximados. Por un lado, en vez de analizar todas las restricciones de tiempo a la vez, las va incorporando a medida que son necesarias de una forma automática guiada por los resultados obtenidos en las iteraciones anteriores (Alur et al., 1995). Por otro lado, se aproxima conservativamente la semántica densa por una semántica de enteros y la semántica original sólo será utilizada cuando sea necesario.

3.4.3 KRONOS

La herramienta KRONOS es un *model checker* para *Timed Automaton* desarrollado en VERIMAG [1] (Yovine, 1997). El análisis puede llevarse a cabo mediante la exploración del autómata basado en zonas, ya sea mediante un algoritmo enumerativo o simbólico. La herramienta soporta *model checking* de propiedades expresadas en TCTL, e interfaces para una gran variedad de notaciones basadas en álgebras de procesos. Si la propiedad TCTL no se satisface en el *Timed Automaton* se genera una traza de error. Además, se han incorporado distintos algoritmos de minimización para alcanzar mayor reducción en el espacio de estados a explorar.

3.4.4 Uppaal

La herramienta Uppaal se ha desarrollado conjuntamente en las universidades de Aalborg y Uppsala [2] (Larsen et al., 1997b). Es, como el anterior, un *model checker* para *Timed Automaton*, sin embargo, la lógica utilizada en la formulación de propiedades es más simple que TCTL, habilitando algoritmos más eficientes. Los sistemas se modelan como autómatas temporizados con tipos de datos simples, que se comunican mediante el uso de canales o de variables compartidas. Permite

[1] http://www-verimag.imag.fr//TEMPORISE/kronos/
[2] http://www.docs.uu.se/docs/rtmv/uppaal/

la verificación de propiedades de seguridad y de viveza acotada, mediante un algoritmo de alcanzabilidad al vuelo que opera sobre estados simbólicos representados como DBMS. Incorpora, adicionalmente, un animador gráfico que puede interactuar con trazas de diagnosis generadas por el *model checker*. Además, se utilizan técnicas composicionales para reducir el espacio de estados.

3.5 Aplicación a problemas reales

En el estudio de los distintos formalismos de especificación y verificación de sistemas de tiempo real se han utilizado diferentes ejemplos de tamaño reducido que han ayudado al avance teórico de dichos formalismos. Entre ellos podemos encontrar el problema de exclusión mutua de *Fischer*, el controlador de un paso a nivel, etc. Pero, sin duda, la aplicación práctica de estos formalismos supone la resolución de problemas de tamaño medio-grande más cercanos a los problemas reales.

- UPPAAL ha sido utilizada para la verificación de un protocolo de audio-control con manejo de colisión propiedad de Philips en (Larsen et al., 1995b). En (Havelund et al., 1997) se estudia otro protocolo de audio/video control propiedad de Bang & Olufsen. Aunque se conocía que el protocolo perdía ocasionalmente mensajes, el error no pudo ser detectado mediante prueba convencional. Con la utilización de UPPAAL se llega a la generación de una traza de error que revela el error en el protocolo. Podemos encontrar más ejemplos en [http://www.docs.uu.se/docs/rtmv/uppaal/].
- KRONOS ha sido utilizado para la verificación del protocolo FRP-DT (*Fast Reservation Protocol with Delayed Transmission*) (Tripakis y Yovine, 1998). Dicho protocolo se ha propuesto con el objetivo de alcanzar una gestión más eficiente del ancho de banda en una conexión ATM. La especificación se ha realizado utilizando *Timed Automaton* a partir de una especificación anterior en SDL. La verificación del protocolo ha revelado que uno de los *timeouts* inicializado para controlar la pérdida de mensajes nunca expira, y algunas de las transiciones nunca llegan a ocurrir. No se trata, pues, de errores serios del protocolo; sin embargo, su detección ha permitido la eliminación de temporizadores y transiciones innecesarias. Sin embargo, también se ha detectado una inconsistencia potencial en el ancho de banda reservado a lo largo de una conexión (*conection path*).

3.6 Otras formas de análisis

En términos de corrección, un sistema es correcto si hace lo que se espera y requiere que haga. Los métodos formales no pueden demostrar la corrección de un sistema en este sentido general, dado que, como hemos visto, se basan en la utilización de modelos formales que pueden ser inadecuados e incompletos. La diferencia entre el problema real y el problema modelado es una fuente de error potencial que debe ser tenida en cuenta, debiendo controlarse mediante la validación de los

modelos utilizados para el problema en concreto. Igualmente, encontramos fuentes de error en la diferencia entre lo que se espera y los requisitos que se documentan. Aunque los métodos formales permiten la construcción de especificaciones precisas, tanto del sistema objetivo como de sus requisitos, se debe utilizar la validación formal para garantizar que las especificaciones son correctas y completas. En este sentido, no debe desdeñarse el uso de técnicas paralelas de validación como el prototipado y la simulación simbólica. Adicionalmente, a medida que se avanza en el ciclo de vida, el problema se hace, a menudo, inmanejable para la verificación estática. Es en este punto donde la verificación dinámica basada en prueba, o la monitorización en tiempo de ejecución (*run time monitoting*), introducen métodos de análisis complementarios que, sin embargo, no deberían perder de vista la especificación formal del sistema y de sus requisitos.

El prototipado rápido de sistemas permite a los diseñadores mostrar el producto al usuario durante la fase de diseño. El usuario puede determinar si el prototipo refleja o no el producto deseado. En caso negativo, se realizarán las aclaraciones oportunas, evitando desperdiciar esfuerzo en la implementación de requisitos no correctos. El prototipado ha sido utilizado ampliamente en diferentes tipos de sistemas, aunque no en el dominio de los sistemas de tiempo real. Esto es debido fundamentalmente a que los prototipos de RTS deben reflejar no sólo que el comportamiento funcional es el correcto, sino que no se violan las restricciones de tiempo. Cuando se construyen los prototipos, a menudo no se consideran ni la velocidad ni la eficiencia, para reducir el tiempo de desarrollo del prototipo, sin embargo, el comportamiento en tiempo se ve influenciado por dichos factores. El prototipado de sistemas de tiempo real basado en técnicas formales es una disciplina con escasa actividad en nuestros días, entre los resultados encontrados en la literatura cabe resaltar (Krupnova et al., 1998; Doerfel et al., 1999).

3.6.1 Simulación simbólica

La simulación simbólica se puede considerar como una forma de prueba aplicada a una especificación formal. Para iniciar la simulación, el usuario proporciona una secuencia de eventos de entrada, generados ya sea manualmente o automáticamente, al simulador. El simulador ejecuta el sistema simbólicamente, utilizando la especificación del sistema, para determinar la respuesta de éste a cada entrada.

Mediante la ejecución simbólica del sistema basada en la especificación formal, el usuario puede asegurarse de que la especificación captura el comportamiento deseado en el sistema. Es decir, a diferencia de las técnicas de verificación formal, que garantizan la corrección de una especificación con respecto a propiedades de interés, la simulación proporciona un medio de validar la especificación. Ejecutando escenarios, el usuario puede, además, utilizar el simulador para una mayor comprensión de las razones del incumplimiento de propiedades específicas (utilizando los contraejemplos obtenidos mediante *model checking*)

Por tanto, la simulación de especificaciones formales puede considerarse una técnica de validación de sistemas; ejecutando la especificación formal y observando el comportamiento del sistema especificado, se puede comprobar si las especificaciones capturan el comportamiento que se le pide al sistema. Este tipo de validación ha sido incorporada en la mayoría de herramientas de apoyo a la especificación y verificación con métodos formales.

3.6.2 Prueba

Aunque la técnica de análisis más atractiva es la verificación formal, cuyo objetivo es probar formalmente que el sistema satisface ciertas propiedades, esta se encuentra limitada por problemas teóricos de decidibilidad y problemas técnicos de coste (espacio y tiempo). La prueba es una técnica de verificación y validación ampliamente extendida, que puede entenderse como complementaria a la verificación formal. El objetivo de la prueba es bastante diferente al de la verificación, ya que se centra en encontrar errores, pero en general no puede proporcionar un resultado de corrección absoluto.

Existen varios trabajos sobre la prueba de propiedades de tiempo real basado en especificación formal, donde conjuntos de *tests* se generan automáticamente a partir de las especificaciones. En (Clarke y Lee, 1995) se propone un método de generación de *tests* a partir de una especificación en álgebra de procesos ACSR. La definición de un *test* no será más que colecciones de entradas, salidas y retardos expresados desde el punto de vista de un agente externo que interactúa con el sistema bajo *test*. En esta misma línea, encontramos trabajos de prueba sobre modelos basados en *Timed I/O Automaton* (Springintveld et al., 1997) y especificaciones en LUSTRE (Raymond et al., 1998).

Si bien la prueba trata de cubrir el paso de diseño a implementación, no permite aportar garantías de la corrección de la implementación sobre todas las posibles ejecuciones. Intentando acercarse a la garantías de la verificación formal, se propone la monitorización en tiempo de ejecución basada en la especificación formal del sistema. A diferencia de la prueba y de la verificación formal, la monitorización no intenta garantizar que todas las ejecuciones (probadas) son correctas, sino sólo que la ejecución actual del sistema es correcta. La monitorización basada en la especificación formal ha recabado un significativo interés en la comunidad investigadora, notables ejemplos los podemos encontrar en (Lee et al., 1998; Mok y Liu, 1997).

3.6.3 Análisis de planificabilidad

Si bien la verificación formal cubre el aspecto de análisis estático, no nos podemos olvidar del análisis de planificabilidad, fundamental en los sistemas de tiempo real. Es común el ver los sistemas de tiempo real como una colección de procesos concurrentes que se comunican; pero dicha concurrencia no es sin costo, debe existir un sistema que gestione la ejecución de los procesos. En la mayoría de los formalismos se considera que la planificación es una disciplina y la verificación otra diferente, siendo pocos los ejemplos de integración propuestos en la literatura (Mok et al., 1996; Choi et al., 1995; Fredette y Cleaveland, 1993).

3.7 Síntesis

Podemos decir que el tratamiento automatizado de los sistemas reactivos se sustenta, principalmente, en la resolución algorítmica de dos problemas fundamentales:

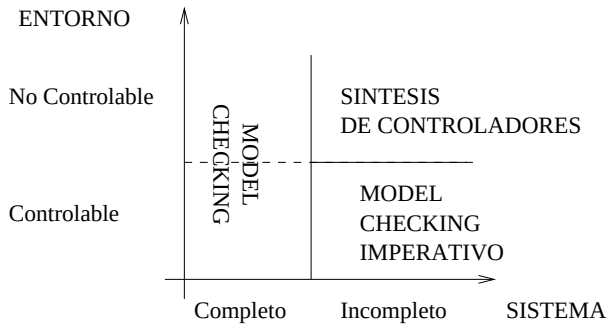


Figura 3.1. Estudio de sistemas reactivos

- El problema de *model checking* (Verificación): Dado un sistema de transiciones finito $\mathcal{M}$ y una especificación en lógica temporal Φ_S, ¿es $\mathcal{M}$ un modelo de Φ_S?.

- El problema de satisfactibilidad (Síntesis): Aunque el problema de satisfactibilidad es fundamental a la hora de llevar a cabo la verificación homogénea basada en lógica (sección 2.2.2), también es importante a la hora de abordar el problema de síntesis, dado que sólo serán sintetizables aquellas fórmulas que son satisfactibles. Una fórmula es satisfactible bajo un semántica dada, si existe un modelo semántico en el que la formula se satisface. Es decir, dada una especificación en lógica temporal Φ_S, ¿existe un modelo $\mathcal{M}$ de Φ_S?.

Si bien el análisis formal para sistemas de tiempo real ha arrojado una inmensa cantidad de resultados en los últimos años, el área de la síntesis, todavía en un estado de menor madurez formal, parece de radical importancia sobre todo en aquellos sistemas embebidos donde el diseño de un controlador es la base de su correcto funcionamiento.

Considerando los sistemas reactivos como aquellas entidades con una continua interacción con el entorno, podemos clasificar su estudio en las categorías representadas en la figura 3.1.

- Si consideramos un sistema completo (en especificación), independientemente de la existencia de un entorno controlable o no controlable, el estudio del sistema reactivo se centra en las tareas de verificación de la especificación en lógica temporal.

- Si, por contra, la especificación del sistema es incompleta y el entorno es controlable (o no se considera en el caso de síntesis de un componente aislado de un sistema reactivo), completar el modelo según una especificación en lógica temporal se reduce a la aplicación de métodos *tableau*, o técnicas menos genéricas basadas en una forma "imperativa" de *model checking*. En dichas formas "imperativas" de *model checking*, dado un modelo incompleto, se utiliza un algoritmo de *model checking* "aumentado", que fuerza la satisfacción de una nueva especificación. En esta categoría se incluye también la síntesis de planificadores, para el caso de tareas periódicas con límite duro, dado que

mediante *model checking* es posible buscar un escenario de ejecución en el que todas las tareas sean servidas.

- Si, por contra, el entorno es no controlable y el sistema está incompleto, nos movemos al concepto más puro de síntesis que, bajo la denominación genérica de síntesis de controladores, enmarca:
  - Síntesis de controladores (supervisores): Dado un modelo del entorno (planta) $\mathcal{M}$ y una especificación en lógica temporal Φ_S de los comportamientos aceptables, se sintetiza un controlador que supervisa la planta, garantizando que se cumple la especificación Φ_S independientemente del comportamiento del entorno.
  - Síntesis de planificadores a medida: Bajo esta denominación tiene cabida la síntesis de planificadores (sin política de planificación a priori) de tareas no necesariamente periódicas que adaptan sus decisiones en función del comportamiento del entorno (recursos). En este caso, el modelo $\mathcal{M}$ representa el conjunto de tareas a planificar y la especificación Φ_S establece que todas las tareas a planificar han de cumplir sus plazos.
  - Síntesis de procesos sincronizadores: Dados dos procesos componentes $\mathcal{P}_1$ y $\mathcal{P}_2$ que representan el entorno $\mathcal{M}$ y una especificación en lógica temporal Φ_S de su adecuada sincronización, se busca la síntesis de un proceso sincronizador que asegure que el sistema compuesto respete la especificación.

  Como se sigue del enunciado de los tres escenarios, su solución se integra en un único ámbito en el marco más general de la teoría de juegos. En concreto, todos ellos se basan en el cómputo de una estrategia que mantenga al sistema dentro de la especificación, distinguiendo para ello entre el conjunto de acciones controlables y no controlables. El proceso sintetizado debe hacer una elección adecuada de entre el conjunto de acciones controlables sin despreciar ninguna de las acciones no controlables en el sistema. En el caso de la síntesis de sincronizadores, se asume que todas las acciones son controlables.

La **síntesis** de máquinas de estados a partir de especificaciones en lógica temporal para sistemas reactivos sin tiempo se sustenta principalmente en la utilización de métodos de *tableau*. Para el caso de los sistemas de tiempo real, dichos métodos sólo serán aplicables para aquellas lógicas de tiempo real con satisfactibilidad decidible. El primer algoritmo *tableau* para una lógica de tiempo denso está en el haber de los trabajos con MITL en (Alur et al., 1996). Sólo muy recientemente se han obtenido resultados similares para otro tipo de lógicas, ya sea relajando la noción de puntualidad, o variando el modelo sobre el que se interpreta la fórmula a distintas variantes del autómata temporizado (Raskin y Schobbens, 1997; La Torre y Napoli, 1997; Dickhofer y Wilke, 1999; La Torre y Napoli, 2000). En el capítulo 9 profundizaremos en este problema como introducción a la solución adoptada para la síntesis automática de modelos MUS-T.

A diferencia de la síntesis de modelos a partir de propiedades expresadas en lógica temporal, en la **síntesis de controladores** no se añade funcionalidad adicional al sistema diseñado y, por tanto, su resolución es menos genérica. En este

ámbito se ha alcanzado un éxito comparable al obtenido en los sistemas reactivos sin tiempo.

En lo que se refiere al paradigma del autómata temporizado, citamos como especialmente relevante el trabajo (Wong Toi y Hoffmann, 1991) que opera sobre el grafo de regiones, y soluciones que evitan dicha construcción (Asarin et al., 1995; Maler et al., 1995; Asarin et al., 1998b; Tripakis y Altisen, 1999), así como soluciones específicas para la síntesis de planificadores (Altisen et al., 1999; Altisen et al., 2000). Aunque no abordado en el curso de esta tesis, consideramos este tipo de síntesis de especial interés para el ámbito en que se enmarcan, los sistemas de tiempo real.

CAPÍTULO 4

Trabajo relacionado

En este capítulo introducimos brevemente dos ámbitos de investigación relacionados con el tratamiento formal de los sistemas de tiempo real: el tratamiento formal de sistemas de tiempo real soft, y la extensión natural del tratamiento formal de sistemas de tiempo real a sistemas híbridos (con magnitudes continuas y discretas). En último lugar, haremos una breve reseña del origen y aplicación de las lógicas multivalor.

4.1 Introducción

Aunque la incorporación de tiempo, en especial el tiempo denso, ha venido a llenar el vacío existente en el tratamiento formal de requisitos de tiempo cuantitativos en sistemas de tiempo real críticos, existen dos ámbitos directamente relacionados con el que nos ocupa. Por un lado, aunque los requisitos de tiempo críticos son especialmente importantes dadas las catastróficas consecuencias de su incumplimiento, existen numerosos sistemas de tiempo real con requisitos de tiempo *soft*. Se trata de sistemas donde es importante un límite en el tiempo de respuesta, pero el fallo del cumplimiento de dicho límite no lleva a un desastre, pérdida de vidas, etc. Ejemplos de este tipo de sistemas son conmutadores telefónicos y las redes de ordenadores.

Por otro lado, y dado que la mayoría de los sistemas de tiempo real son sistemas embebidos, es necesario el desarrollo de formalismos en los que, además del tiempo, se permita el tratamiento de magnitudes, en general magnitudes físicas, que cambian de forma continua con el tiempo. El estudio de estos sistemas híbridos ha sido abordado estos últimos años como extensión de los formalismos de tiempo denso.

Por último, en este capítulo se introduce una breve reseña referente al uso de las lógicas multivaloradas en su aplicación a la ingeniería del software. Desgraciadamente, aunque las bases teóricas que sustentan dichas lógicas son tan antiguas como las utilizadas en la lógica temporal, su aplicación en este ámbito es más bien reducida.

4.2 Formalismos híbridos

Los sistemas híbridos son sistemas de tiempo real que permiten cambios de estado continuos sobre períodos de tiempo de duración positiva, así como cambios de estado discretos instantáneos.

La utilización de sistemas híbridos es muy habitual en la práctica en todos aquellos entornos en los que magnitudes físicas, de naturaleza analógica y generalmente modeladas mediante ecuaciones diferenciales, interaccionan con dispositivos digitales modelados como máquinas de estados. Se trata de procesos reactivos embebidos en un entorno continuo que deben reaccionar a los cambios de ese entorno en tiempo real. Es, sin duda, esta interacción de cambios continuos y discretos lo que hace a los sistemas híbridos especialmente atractivos a la aplicación de análisis formal.

Aunque los métodos matemáticos sobre ecuaciones continuas y sobre transiciones discretas se han estudiado de forma independiente, el desarrollo de métodos de razonamiento formal sobre los sistemas híbridos es relativamente reciente. Aunque nacidos al amparo del estudio de los formalismos con tiempo, hoy constituyen una disciplina con entidad propia. Encontramos, en este ámbito, extensiones de los modelos con tiempo más prometedores como de *Timed Automaton* a *Hybrid Automaton* (Alur et al., 1993b); de *Clocked Transition Sytem* a *Phase Transtion System* (Kapur et al., 1994); y de *Timed I/O Automaton* a *Hybrid I/O Automaton* (Lynch et al., 1996). La extensión de estos modelos ha ido acompañada de la adecuación de distintas lógicas temporales para el tratamiento de magnitudes de cambio continuo.

Por último, el desarrollo de herramientas, aunque en su infancia debido a la mayor complejidad de los problemas de análisis asociados con los sistemas híbridos, ha dado lugar a la implementación de semi-algoritmos y métodos de análisis aproximados para modelos menos generales (Alur et al., 1993c; Henzinger y Ho, 1994), como los incluidos en HyTech . HyTech (Henzinger et al., 1997) permite el análisis formal, basado en *model checking*, de *Hybrid Automaton* con leyes de evolución especificadas mediante ecuaciones diferenciales lineales. Los métodos de decisión utilizan mecanismos muy próximos a los utilizados en el *model checking* sobre tiempo denso.

4.3 Sistemas de tiempo real *soft*

A diferencia de los sistemas de tiempo real *hard*, en los sistemas de tiempo real no críticos o *soft real time systems*, interesa razonar sobre el tiempo de respuesta medio del sistema o la probabilidad de cumplimiento de un plazo en el sistema. Mientras que el razonamiento formal sobre sistemas de tiempo real críticos utiliza el dominio temporal, el análisis de requisitos *soft* requiere la introducción de información relativa a la distribución estocástica de la ocurrencia de los eventos en el modelo. La mayoría de los formalismos existentes modelan este tipo de sistemas como procesos de *Markov*, incluyendo en el modelo las distribuciones de los retardos entre acciones y las distribuciones de las elecciones de las distintas transiciones que parten de un estado del sistema.

En la literatura encontramos diversas extensiones probabilísticas de máquinas de estados, lógicas y álgebras de procesos, siendo uno de los estudios más destacables, en la integración de tiempo y probabilidades, el desarrollado en torno a PCTL y TPCCS, extensiones con tiempo y probabilidades de CTL y CCS (Hansson y Jonsson, 1994; Yi, 1994; Jonsson et al., 1999).

PCTL – *Probabilistic real time Computation Tree Logic* – es una lógica temporal de tiempo ramificado que permite la especificación de restricciones de tiempo y de fiabilidad y, por tanto, adecuada para la especificación y verificación de *soft deadlines*. Por ejemplo, se podría especificar una propiedad como: "después de una petición de servicio, existe por lo menos un 90% de probabilidad de que el servicio sea llevado a cabo en 5 segundos". PCTL es una de las pocas lógicas que puede expresar plazos *soft* y *hard*, lo que hace que sea especialmente adecuada en la verificación de protocolos de comunicación en medios con ruido. Con ella se pueden sustituir suposiciones fuertes sobre el comportamiento del medio ("el medio nunca pierde más de tres mensajes consecutivos"), por suposiciones más débiles ("la transmisión tiene éxito con una probabilidad x").

PCTL extiende la lógica CTL con tiempo y probabilidades, obteniendo fórmulas que se interpretan sobre cadenas de *Markov* de tiempo discreto. Un ejemplo de una fórmula PCTL con el operador *until* sería:

$$p\ U^{\leq 2}_{\geq 0.98}\ q$$

que, intuitivamente, afirma que existe una probabilidad no menor de 0.98 de que "q se cumpla en no más de 2 unidades de tiempo y p se cumpla desde el instante actual hasta que q sea cierto".

Para la especificación del sistema se utiliza TPCCS, extensión de CCS con dos tipos de estados: estados reactivos y estados probabilísticos. Las transiciones que parten de estados reactivos se etiquetan sólo con eventos, mientras que las transiciones que parten de los estados probabilísticos se etiquetan sólo con probabilidades. Paralelamente a los estudios teóricos se ha desarrollado la herramienta de análisis TPWB [1], desarrollada en *Swedish Institute of Computer Science*.

4.4 Lógicas multivaloradas

Las lógicas multivaloradas arrancan del problema de aceptación de la ley del tercero excluido. Dichas lógicas son similares a la lógica clásica ya que el valor de verdad de una sentencia compuesta se determina por el valor de verdad de sus sentencias componentes. Sin embargo, difieren de las lógicas clásicas en su definición del valor de verdad, al no limitar el número de valores de verdad a 2.

Las lógicas multivaloradas, como una rama específica de la lógica, se remontan a los trabajos del matemático polaco *Lukasiewicz* (1920). La idea de *Lukasiewicz* se basaba la utilización de un tercer valor de verdad ("posible"), recogiendo, de esta forma, las modalidades "es necesario que" y "es posible que". Paralelamente a esta aplicación de la lógica modal (no finalmente materializada), *Post* (1921) introdujo valores de verdad adicionales para su aplicación al campo de la representación de funciones.

[1] http://www.sics.se/~fred/tpwb.html

En general, los valores de verdad adicionales pueden ser de motivación variada, dependiendo de su ámbito de aplicación. Así, por ejemplo, *Belnap* (Belnap, 1977) introduce un conjunto de verdad de cardinalidad 4 para la actualización de información en una base de datos. En su larga historia, el ámbito de aplicación de las lógicas multivaloradas ha sido muy variado: sistemas expertos, códigos de detección de errores, razonamiento inconsistente, sistemas multi-agente, AI (Inteligencia Artificial), verificación de circuitos digitales, etc (recopilados en (Bolc y Borowik, 2000; Hähnle y Escalada-Imaz, 1997)).

En lo que se refiere a su uso en la ingeniería *software*, la motivación del uso de lógicas multivaloradas arranca de tres hechos principales (Chechik et al., 2001):

- La necesidad de contemplar la incertidumbre propia de las primeras fases del ciclo de vida.
- La necesidad de contemplar desacuerdos entre los agentes involucrados (*stakeholders*).
- La necesidad de representar importancia relativa entre comportamientos a implementar.

siendo el primero el directamente relacionado con los objetivos de esta tesis. A pesar de tal justificación, pocos son los ejemplos encontrados en la literatura:

- En (Fitting, 1992) *Melvin Fitting* sugiere, en vez de utilizar lógica clásica para razonar sobre varios expertos, utilizar lógicas multivaloradas cuya cardinalidad dependerá del número de expertos, tales valores de verdad son una medida del acuerdo entre expertos sobre la especificación del sistema. En esta misma línea, (Messing, 1997) sugiere un entorno para el procesamiento y combinado de conocimiento distribuido.

- En (Hunter y Nuseibeh, 1998), *Hunter* y *Nuseibeh* proponen el uso de lo que se denomina lógica quasi-clásica para la representación de especificaciones parciales y la identificación de inconsistencias entre ellas. En este trabajo se sugiere el uso de este tipo de lógicas para el tratamiento formal de puntos de vista en la especificación. Siguiendo esta línea de trabajo se encuentra (Easterbrook y Chechik, 2001).

En lo que se refiere a métodos automatizados de análisis, existe una gran madurez en el campo de los demostradores de teoremas, pudiendo citar los distintos trabajos de *Hähnle* (Beckert et al., 1996). Sin embargo, la verificación heterogénea basada en *model checking* ha recibido una atención más bien reducida, pudiendo únicamente citar los trabajos en el Departamento de Ciencias de la Computación de la Universidad de Toronto [2]. $\mathcal{X}$check es un *model checker* parametrizable por distintas lógicas definidas a partir de retículos. Aunque dicha implementación no considera explícitamente el tiempo, la utilización de cualquier técnica de abstracción que lleve el conjunto de computaciones de un modelo con tiempo a un conjunto finito, podría utilizar tal *model checker*. El *model checker* desarrollado en esta tesis es un *model checker* de propósito específico para SCTL/MUS-T, que, a diferencia de $\mathcal{X}$Check, sigue una estrategia local.

[2] [http://www.cs.toronto.edu/].

PARTE III

Metodología formal SCTL/MUS-T

CAPÍTULO 5

Proceso *Software*

En este capítulo se presenta la metodología SCTL-MUS cuya principal característica es que propone un modelo de ciclo de vida iterativo e incremental con prototipado. Por considerar este modelo de ciclo de vida igualmente adecuado para el ámbito de sistemas de tiempo real, en este capítulo se propone una extensión de los fundamentos teóricos de esta metodología, así como sus procedimientos de decisión para su aplicación al diseño y desarrollo de sistemas de tiempo real críticos.

5.1 Introducción

Un proceso *software* define un conjunto estructurado de actividades necesarias para desarrollar un sistemas *software*, las relaciones entre ellas y los productos (código, documentación, etc.) que se obtienen de cada una de estas actividades.

Los principales objetivos de los procesos de ingeniería *software* son facilitar la comunicación y la comprensión humana del problema y permitir una gestión adecuada.

Los procesos *software* se pueden definir con diferentes niveles de abstracción. A este respecto, se suele utilizar el término modelo de ciclo de vida para aquellas definiciones de alto nivel de las actividades que tienen lugar durante la producción *software*; utilizándose el término modelo de proceso de ciclo de vida *software* para definiciones más detalladas. Como modelos de ciclo de vida clásicos podemos citar: cascada, prototipado desechable, prototipado evolutivo, desarrollo iterativo e incremental, modelo en espiral, modelo *software* reutilizable, etc.

En (Sommerville, 1997), *Sommerville* propone un conjunto de criterios generales y no excluyentes como base para la clasificación de los modelos de ciclos de vida (informal, gestionado, metódico, con mejora del proceso). Una de las principales razones que justifica la poca penetración de los métodos formales en la industria del *software* es su deficiente integración en el proceso de desarrollo *software*. Según estos criterios de clasificación, el uso de métodos formales en la ingeniería *software*

persigue, como fin último, la definición de modelos de ciclo de vida formales y metódicos. No obstante, la situación actual es que la incorporación de los métodos formales en un modelo de proceso *software* se puede considerar excepcional.

En este capítulo resaltaremos las ventajas un ciclo de vida iterativo, paradigma adoptado en la metodología SCTL/MUS, así como las características específicas que un sistema de tiempo real impone en dicho modelo.

5.2 Modelos de ciclo de vida

Podemos clasificar los modelos de ciclo de vida más usuales (Sommerville, 1997; Pfleeger, 1998; Jalote, 1997), desde el punto de vista del ordenamiento de las tareas necesarias en la producción *software*, en secuenciales e iterativos. En los modelos secuenciales, cada una de las tareas definidas se realiza "de una vez"; por contra, en los modelos iterativos, estas tareas se repiten en distintas iteraciones hasta alcanzar el producto *software* final. Es decir, en los modelos secuenciales se realiza toda la captura de requisitos, toda la especificación, todo el diseño y toda la implementación en una secuencia; en los modelos iterativos, cada iteración incluye todas las tareas, si bien, dependiendo de la fase actual, se pone mayor atención en unas tareas u otras. Por supuesto, no todos los modelos de ciclo de vida se pueden clasificar como puramente secuenciales o puramente iterativos, existiendo modelos de ciclo de vida que sólo adoptan un enfoque iterativo en alguna de sus fases.

El modelo de ciclo de vida en cascada es el ejemplo más conocido de modelo secuencial puro. Si bien son muchas las deficiencias achacadas a este modelo, no debemos olvidar que es el más utilizado, fundamentalmente por permitir una gestión más fácil que los modelos de ciclo de vida iterativos.

El principal inconveniente del modelo en cascada es la imposibilidad de probar la corrección del diseño hasta el final del ciclo de vida y, en consecuencia, tampoco es posible explorar y validar distintas alternativas de diseño. Esta situación redunda en una pérdida de calidad del sistema final, entendida ésta como el grado de satisfacción de los clientes o usuarios del sistema. La realización de cambios de diseño en la fase de implementación, o incluso una vez el producto haya sido entregado, introduce desajuste entre el diseño y el sistema final. Además, tal práctica es excesivamente cara y, a menudo, hace imposible el cumplimiento de los plazos de entrega, por la aparición de labores no planeadas. Además, en un modelo secuencial un fallo puede suponer, en el peor de los casos, la vuelta a las fases iniciales y, en consecuencia, una pérdida elevada de trabajo. Otros ejemplos de modelos secuenciales son el modelo V y el modelo de especificación operacional.

Podemos afirmar que los tiempos en que se discutía sobre la mayor o menor adecuación de una u otra estructura en el modelo de ciclo de vida han quedado atrás. Hoy son ampliamente reconocidas las ventajas de los modelos basados en un paradigma iterativo, en especial, cuando se aplica a la producción de sistemas complejos o sistemas en los que la experiencia del equipo de desarrollo es limitada. Dichos modelos no congelan la especificación de requisitos en fases tempranas del proceso *software*, permitiendo una realimentación de clientes al equipo de desarrollo que se ha mostrado imprescindible para la obtención de *software* de calidad. Además, los enfoques iterativos permiten la exploración de alternativas, pueden

asumir las tareas de mantenimiento como una iteración más en el proceso, y, en presencia de errores, sólo producen la pérdida de trabajo de una iteración. Como modelos iterativos clásicos podemos catalogar, por ejemplo, el modelo en espiral e *iterative enhancement*.

Por último, notar que hemos referido como modelos iterativos a aquellos que llevan a cabo un conjunto de tareas de forma iterativa. En el ámbito de los modelos de ciclo de vida se suele hablar de modelo iterativo para referirse al ciclo de vida en el que la salida de cada una de las iteraciones es el sistema *software* completo (a distinto nivel de calidad), utilizándose el término modelo incremental para modelos de ciclo de vida que producen el sistema *software* en incrementos de funcionalidad. Sin embargo, y en la práctica, suele utilizarse una combinación de ambos modelos en lo que se denomina modelo iterativo e incremental.

5.3 Los modelos de ciclo de vida y las técnicas formales

Los métodos formales han probado su utilidad a distintos niveles de abstracción, que van desde la captura de requisitos, la especificación, el diseño, la codificación, e incluso el hardware subyacente. Sin embargo, la mayor parte de investigación se ha centrado en la resolución de problemas concretos y pocos son los ejemplos de modelos de ciclo de vida totalmente formalizados.

El modelo de ciclo de vida comúnmente asociado a las técnicas formales es el modelo transformacional. No se trata de un modelo iterativo puro, dado que reduce las iteraciones a fases de transformación de la especificación (figura 5.1). Si las transformaciones aplicadas son correctas, es posible asegurar que el sistema construido satisface la especificación.

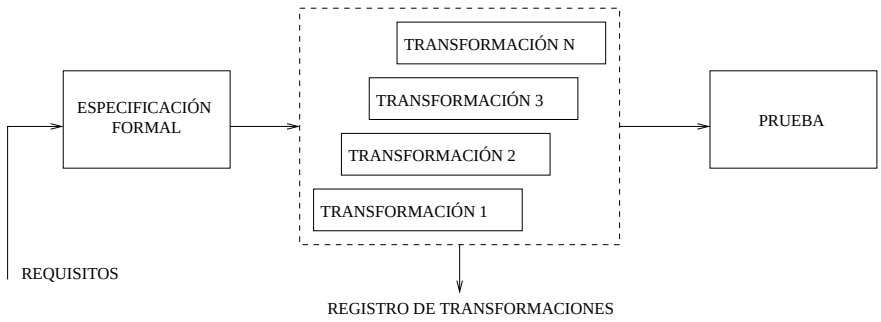


Figura 5.1. Modelo de ciclo de vida transformacional

Como cita de mérito, cabe resaltar la visión totalmente formalizada en FOCUS (Broy et al., 1992). Bajo un modelo transformacional de refinamientos sucesivos, FOCUS propone una metodología y las herramientas de apoyo para un modelo de proceso *software* formal desde la especificación a la implementación. Si bien no es una metodología específica de sistemas de tiempo real, incorpora el tiempo con un enfoque de reloj global ficticio.

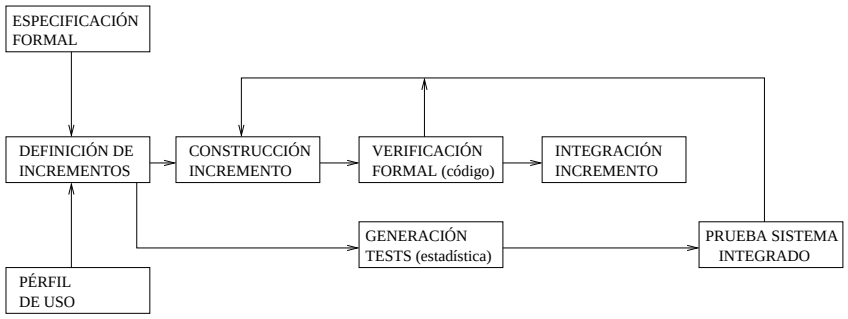


Figura 5.2. Modelo de ciclo de vida *Cleanroom*

Como de carácter iterativo e incremental destacamos la filosofía *Cleanroom* (Linger, 1994) (figura 5.2), cuyo objetivo es reemplazar la prueba de unidad por la certificación del *software* respecto a la especificación utilizando verificación estática. Las ideas básicas de la metodología son:

- Desarrollo incremental: El *software* se produce mediante incrementos sucesivos, cada uno de los cuales se añade a la funcionalidad de la versión anterior.

- Especificación y diseño formal: Utiliza una estructura jerárquica en cajas en las que el comportamiento de los objetos del sistema se define formalmente. En este marco, se utilizan métodos formales para demostrar la corrección del diseño.

- Separación de implementación y prueba: Los desarrolladores no prueban su código, trasladándose dicha tarea a un equipo de revisores.

- Prueba estadística: Basado en el perfil de funcionamiento definido para el sistema, se realiza prueba estadística basada en la frecuencia de uso con el objetivo de maximizar el tiempo medio entre fallos.

Aunque distintas evaluaciones empíricas de la metodología sugieren que *Cleanroom* mejora la calidad del *software*, existen posturas más reticentes apuntando tres objeciones básicas (Pfleeger, 1998): la falta de prueba unitaria (*unit testing*) es una práctica peligrosa que contradice las bases de la teoría de pruebas; las evaluaciones empíricas realizadas no son del todo completas; y la no existencia de garantías sobre la corrección del perfil de funcionamiento.

Por último, destacar que la gran barrera que dificulta el uso extendido de ambos modelos, tanto de FOCUS como de *Cleanroom*, radica, precisamente, en su motivación: la formalidad. Tanto uno como otro sólo han sido aplicados en ámbitos científicos, y en el caso de *Cleanroom*, sólo en prácticas *software* realizadas por el mismo equipo de desarrollo de *Cleanroom*.

5.4 Los modelos de ciclo de vida y los sistemas de tiempo real

En esta sección abordaremos las peculiaridades que el sistema de tiempo real introduce o reintroduce en los modelos de ciclo de vida. Tales peculiaridades complican, en gran medida, las labores de producción de este tipo de sistemas, centrándose en: su inherente concurrencia, la existencia de eventos no predecibles, su carga variable y las limitaciones de memoria y procesamiento en la plataforma elegida para el producto *software*. A la hora de definir una metodología de aplicación a sistemas de tiempo real se debe tener en cuenta:

Consideración de los requisitos de tiempo durante todo el ciclo de vida

O lo que es lo mismo, considerar los requisitos de tiempo al mismo nivel que los requisitos funcionales. Aunque tal práctica debería ser consecuencia de la definición de corrección de un sistema de tiempo real (los resultados se deben producir en los tiempos especificados), era a menudo una práctica habitual en la ingeniería *software* dedicar la máxima atención a los requisitos funcionales y retrasar la consideración de los requisitos no funcionales sólo al final del ciclo de vida. Para la producción de sistemas de tiempo real no es suficiente ni correcto la utilización de técnicas de ingeniería *software* tradicionales en las que se retrasa la consideración del tiempo a las fases de implementación y prueba. Tal práctica asume que un sistema de tiempo real es un sistema *software* con un tiempo de ejecución "rápido", no siendo la rapidez la característica definitoria del *software* de tiempo real. Considerado el tiempo como una característica del desempeño final del sistema, si el desempeño requerido no era alcanzado, la solución pasaba por la realización de cambios *ad hoc* en el sistema final. Además, sin tomar en consideración los requisitos de tiempo, no existe una visión clara de si el *hardware* subyacente es adecuado para el producto *software*. En consecuencia, el tiempo debe ser considerado como una parte integral del proceso *software*, y todas las tareas habituales en el proceso *software* deben ser adaptadas para la producción de *software* de tiempo real

Modelo de ciclo de vida

En lo que se refiere al modelo de ciclo de vida, nos situamos en el lado de metodologías de tiempo real ampliamente utilizadas como HRT-HOOD (Burns y Wellings, 1994) o ROOM (Selic et al., 1994), y distintos trabajos en el ámbito académico, como (Angilotti et al., 1998; Puchol y Mok, 1998). El modelo de ciclo de vida ha de ser un modelo iterativo de prototipado rápido (automático) que permita una pronta realimentación, dado que especialmente en el *software* de tiempo real es importante disponer tan pronto como sea posible de una versión ejecutable que avale lo factible de las decisiones de diseño.

Además, *Stankovic*, en (Stankovic, 1996), plantea la necesidad de infraestructuras que den un soporte eficiente al mantenimiento del *software* de tiempo real, permitiendo cambios con el menor tiempo de entrega posible. Como ya hemos

mencionado, un modelo iterativo se adapta a cambios asimilándolos como una iteración más en el proceso *software*.

Con el menor nivel de formalidad comúnmente encontrado en la industria *software*, podemos encontrar notables ejemplos de modelos de ciclo de vida y herramientas integradas específicos para sistemas de tiempo real: ObjectGeode [1] , Objectime [2] y RT Rational Rose [3]. Todas ellas utilizan un modelo de ciclo de vida iterativo e incremental, que permite una rápida realimentación para cotejar alternativas de diseño. Todas estas herramientas se basan en la verificación dinámica de los requisitos de tiempo mediante la generación semi-automática de un prototipo final; la verificación formal se reduce a la búsqueda de bloqueos y la detección de estados no alcanzables. En general, cabe destacar el paradigma OO utilizado en tales metodologías que, hasta hace muy poco, se consideraba incompatible con tiempo real.

Análisis específico para tiempo real

Existen varias técnicas que han sido desarrolladas y utilizadas en la práctica para aportar garantías de corrección en los sistemas de tiempo real, que van desde el análisis estático basado en métodos formales y teoría de la planificación al análisis dinámico basado en prueba y monitorización en tiempo de ejecución. Sin embargo, tales técnicas sólo han sido aplicadas con éxito a sistemas de pequeña escala de forma independiente.

Stankovic (Stankovic, 1996) y *Heitmeyer* (Heitmeyer, 1998) abogan por la necesidad de un marco común que permita la aplicación de todas las técnicas a sistemas de gran escala. En dicho marco se utilizaría un único modelo del sistema para realizar el análisis estático y dinámico, lo que evitaría la especificación de modelos propios para cada una de las técnicas y la necesidad de la comprobación de consistencia en estos modelos. Además, los métodos deberían permitir el análisis de planificabilidad tan pronto como sea posible en el proceso *software*, la generación automática de casos de prueba a partir de la especificación formal, e incluso la monitorización *run time* basada en tal especificación.

5.5 Revisión del ciclo de vida SCTL/MUS

En (García Duque, 2000) se propone un modelo de ciclo de vida formal basado en la aplicación combinada de varios tipos de FDTs (*Formal Description Techniques*) – MUS, SCTL, LOTOS –. Se trata de un modelo de ciclo de vida iterativo e incremental con prototipado, en el que se pueden diferenciar, a alto nivel, tres fases de proceso (figura 5.3):

- En una primera fase de *objetivos iniciales*, de naturaleza iterativa, los requisitos se especifican incrementalmente utilizando la lógica temporal SCTL. Tales requisitos son traducidos de forma automática a modelos de estados

[1] [http://www.telelogic.com]
[2] [http://www.objectime.com]
[3] [http://www.rational.com]

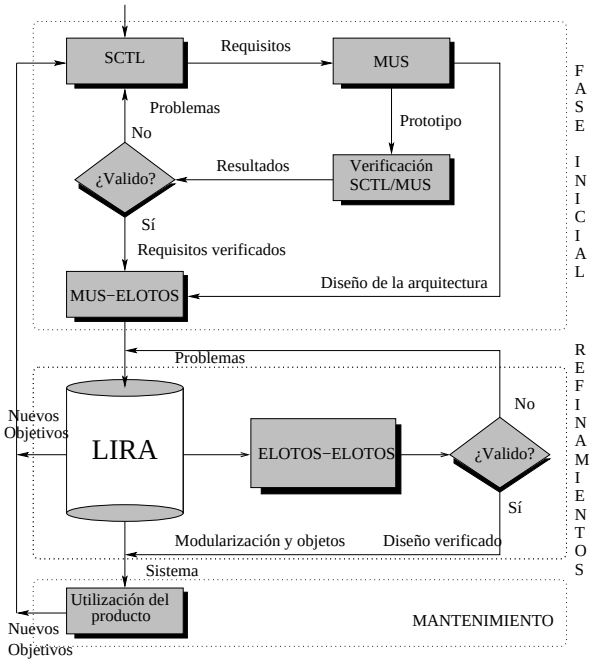


Figura 5.3. Modelo de ciclo de vida SCTL/MUS

MUS, que registran los comportamientos especificados hasta el momento, incluyendo el concepto de subespecificación para aquellos comportamientos no especificados. Esta fase se sustenta principalmente en procedimientos de verificación de requisitos SCTL sobre modelos MUS, y la materialización de nuevos requisitos SCTL sobre modelos MUS resultantes de ciclos anteriores.

- Partiendo de la especificación resultante de la fase anterior, se articula una fase de *refinamientos*. Dada la mejor adecuación de las álgebras de procesos a la hora de manejar refinamientos y estructura, se propone la utilización de una FDT basada en álgebra de procesos (LOTOS) (Bolognesi y Brinksma, 1989).

- Por último, la fase de *mantenimiento*, que comienza cuando el sistema se entrega al cliente o usuario y se pone en funcionamiento, se asume como una iteración más en el proceso *software*.

Objetivos iniciales

En esta primera fase se articula una estructura de proceso iterativa e incremental. Para dotar de formalidad al proceso, se atiende a las exigencias que una estructura de este tipo fuerza en la metodología definida:

- Dada la naturaleza iterativa del proceso, es necesaria la sincronización formal de los productos de salida en el proceso *software* (modelos y requisitos)

entre ciclos consecutivos. Dicha sincronización formal debe ir destinada a la detección de inconsistencias entre requisitos y modelos.

- Dada la naturaleza incremental del proceso, es necesario dar soporte a modelos incompletos como base para prototipos intermedios no totalmente especificados.

En la fase de *objetivos iniciales* se propone una aproximación dual al sistema *software* mediante la utilización de un formalismo orientado a propiedades (SCTL) y un formalismo orientado a modelo (MUS). Como ya hemos mencionado en la introducción a SCTL/MUS (sección 1.6), la especificación combinada resulta útil debido a que aúna las ventajas de la especificación orientada a modelo y la especificación orientada a propiedades; además, el uso de dos estilos de especificación diferentes permite la detección de inconsistencias entre ambos como una forma de depurar la especificación.

Como FDT orientada a propiedades se propone el uso de la lógica **SCTL**. SCTL es una lógica temporal causal, de naturaleza multivalor. Un modelo de ciclo de vida iterativo e incremental conlleva una especificación de requisitos que puede ser tanto incompleta como inconsistente, dado que el usuario va adquiriendo conocimiento del sistema a lo largo del proceso *software*. Para capturar un conocimiento de este tipo, las lógicas multivalor suponen una solución atractiva.

Como FDT orientada a modelo se utiliza el modelo de estados **MUS**, dado que en las primeras fases un modelo de este tipo es suficiente para la mínima noción de estructura en la especificación. El modelo MUS extiende los formalismos basados en máquinas de estados clásicas con el concepto de subespecificación. Bajo el término subespecificación consideramos todos aquellos elementos en el modelo cuyo comportamiento no ha sido especificado. Por tanto, el modelo permite la especificación incompleta del sistema, característica típicamente asociada a técnicas de especificación orientadas a propiedades. Adicionalmente, y como resultado de no asumir los comportamientos no especificados, como comportamientos no posibles, permite un chequeo de consistencia implícito al modelo.

En base a la lógica SCTL y al modelo MUS, se articula un proceso iterativo sustentado, principalmente, por métodos de decisión orientados a verificación y a síntesis:

- Se proporciona un algoritmo de *model checking* de una fórmula SCTL sobre un modelo MUS (sobre el que se interpreta la fórmula), que decide el grado de satisfacción con que el modelo "realiza" el requisito.

- Se proporciona un algoritmo de síntesis incremental que, a partir de un modelo MUS y de un nuevo requisito expresado en SCTL, sintetiza (si es posible) un modelo más evolucionado que "realiza" dicho requisito.

Adicionalmente y como un punto intermedio a la fase de *refinamientos*, se define un método para la síntesis de procesos sincronizadores, a partir de una especificación global del comportamiento deseado.

5.6 Modelo de ciclo de vida SCTL/MUS-T

Dada la adecuación de un ciclo de vida iterativo con prototipado a las tendencias actuales en el ámbito de los sistemas de tiempo real, es nuestro objetivo la adaptación de la metodología desarrollada a dicho ámbito. Para tal fin, y dado que los requisitos de tiempo deben ser considerados desde las primeras fases del ciclo de vida, es necesario la extensión de los formalismos SCTL y MUS para la incorporación de requisitos de tiempo. Asimismo, y siguiendo el proceso iterativo en la fase de *objetivos iniciales*, es necesario adecuar los procedimientos de decisión (verificación y síntesis incremental) a las extensiones realizadas.

En la siguiente sección se detalla la estructura propuesta para la fase de objetivos iniciales. Sin embargo, y si bien el paradigma iterativo es totalmente acomodable al ámbito de tiempo real, existen puntos importantes a tener en cuenta para la adecuación del modelo de ciclo de vida completo que se detallan en la sección 5.6.2.

5.6.1 *Objetivos iniciales*

El objetivo prioritario de la presente tesis, y dada la necesidad de considerar el tiempo como una característica básica para la corrección del sistema, es la adecuación de los formalismos MUS y SCTL para la incorporación de requisitos de tiempo. Tal extensión se ha realizado conservando el espíritu de los modelos subyacentes sin tiempo: dar soporte a un modelo de ciclo de vida iterativo e incremental. Para ello, se define el modelo MUS-T y la lógica SCTL-T que, como habíamos adelantado en el capítulo 1, asumen un modelado continuo del tiempo y un modelo de concurrencia entrelazado.

- Para la especificación de requisitos, se define la lógica SCTL-T, extensión temporizada de la lógica SCTL sobre un dominio temporal denso.
- El modelo de estados MUS-T se define para poder realizar prototipos y para servir de base para la interpretación de fórmulas SCTL-T. MUS-T es un modelo temporizado que extiende el modelo MUS siguiendo el paradigma de autómata temporizado. A diferencia de un autómata temporizado, el modelo MUS-T registra la información generada durante fases previas, permitiendo, al igual que el modelo MUS, una incorporación implícita de la consistencia entre ciclos. MUS-T propone un modelado del sistema dirigido por eventos (los dirigidos por tiempo sólo pueden modelar sistemas síncronos) donde, por evento entendemos cualquier acción llevada a cabo por el sistema o cualquier estímulo del entorno, y donde consideramos los eventos instantáneos. De esta forma, el modelado de una acción con duración no despreciable en el sistema se modelará mediante dos eventos instantáneos de inicio de la acción y final de la acción.

En el ámbito específico de tiempo real, a las ventajas asociadas a la especificación combinada se suma la facilidad de particionado que ello supone. En su mayoría, los sistemas de tiempo real se pueden catalogar como sistemas empotrados en los que existe una alta integración entre hardware y *software*. Por tanto,

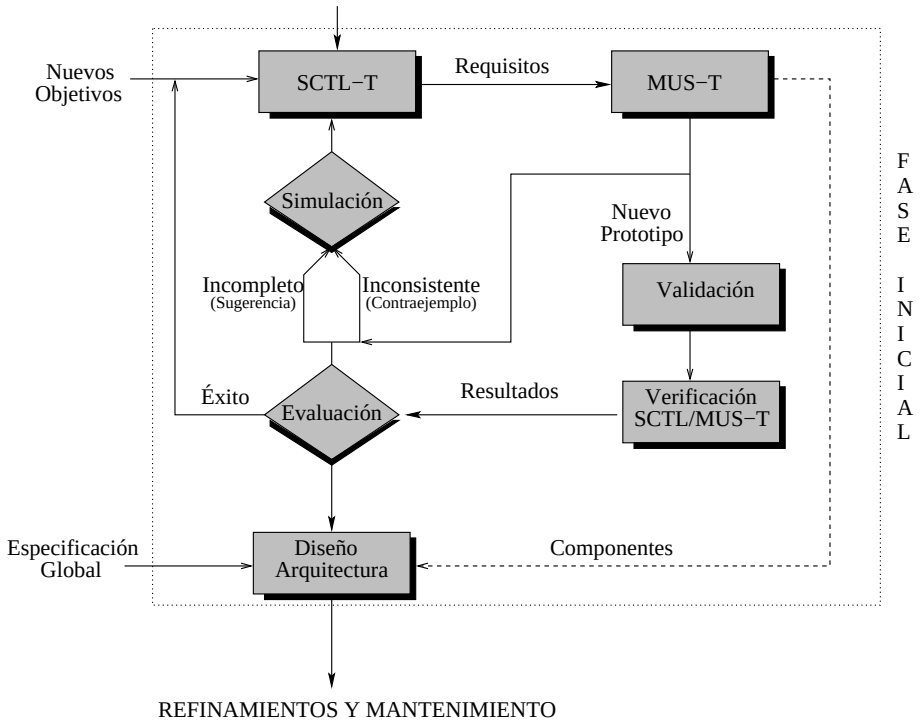


Figura 5.4. Modelo de ciclo de vida SCTL/MUS-T para la fase de *objetivos iniciales*

parte del proceso formal supondrá decidir qué funcionalidades del sistema son implementadas en *software* y cuáles en *hardware*. A nivel de coste, retrasar las decisiones sobre particionado se ha mostrado más efectivo: la utilización de un modelo operacional permite a la vez una fácil traducción a *hardware* y, pasando por la utilización de un álgebra de procesos, a *software*.

El secuenciamiento de tareas en la fase de *objetivos iniciales* se presenta en la figura 5.4. En cada iteración, el usuario identifica y especifica un conjunto de nuevos requisitos. Estos requisitos se verifican en los modelos de diseño actual del sistema (componentes MUS-T). El resultado de la verificación permitirá conocer si el sistema ya satisface los requisitos; si el sistema no será capaz de satisfacer dichos requisitos en un ciclo futuro, a partir del modelo de diseño actual (inconsistencia); o, si el sistema no satisface los requisitos pero puede llegar a satisfacerlos en ciclos posteriores (incompleto). La estructura de la fase de objetivos iniciales se articula, por tanto, como un ciclo con las siguientes tareas:

[1] **División en componentes**: La mínima noción de estructura se materializa bajo la delimitación de los distintos componentes en la especificación del sistema, guiada por el conocimiento preliminar de la arquitectura lógica y física del sistema final.

[2] **Ciclo de Componente**:

[2.1] **Especificación de requisitos (SCTL-T)**: Usando una versión restringida para síntesis de SCTL-T, se especifican los nuevos requisitos a

incorporar en el componente.

[2.2] **Síntesis incremental del prototipo (MUS-T)**: Con los nuevos requisitos especificados se sintetiza (si es posible), de forma incremental, el nuevo prototipo MUS-T del componente. La salida de esta tarea puede ser o bien un nuevo prototipo para el componente, o bien un error de inconsistencia. Si el nuevo prototipo es sintetizado, se pasa a la tarea de validación ([2.3]); si se produce un error de inconsistencia, se pasa a la tarea de evaluación de resultados ([2.5]).

[2.3] **Validación**: Desafortunadamente, la formalidad introducida no es suficiente. Un sistema es correcto cuando funciona como el usuario espera. Pero los métodos formales no pueden demostrar corrección en este sentido absoluto, ya que se basan en modelos formales que pueden ser inadecuados o incompletos; y, además, pueden aparecer errores originados en la diferencia entre las expectativas de los usuarios y los requisitos formales identificados. Para la validación del nuevo prototipo se debe, por un lado, garantizar una corrección mínima y, por otro lado, corroborar que el modelo responde al comportamiento esperado mediante la simulación basada en la especificación formal. Como de mínimamente correctos caracterizamos a aquellos prototipos que no exhiben bloqueos (de control o de tiempo).

[2.4] **Verificación SCTL/MUS-T**: Se verifican requisitos SCTL-T genéricos sobre los componentes MUS-T. La verificación formal se lleva a cabo mediante *model checking* sobre una abstracción del componente MUS-T. El resultado de la tarea de verificación es el nivel de satisfacción de un requisito SCTL-T sobre un modelo MUS-T. Dicho nivel refleja, de forma cualitativa, la capacidad del sistema de proporcionar el requisito en el ciclo actual, o en ciclos posteriores del proceso *software*. Dicha capacidad dependerá de la cantidad de subespecificación en el sistema. Si el requisito ya se satisface en el modelo actual, empezamos un nuevo ciclo; si el requisito es inconsistente se genera un contraejemplo; si el requisito es incompleto, se generan sugerencias de modificación del modelo actual. En caso de que el resultado de la tarea de verificación no tenga éxito, se pasa a la fase de evaluación de resultados ([2.5]).

[2.5] **Evaluación de resultados**: Esta tarea se centra en la simulación simbólica del sistema mediante el prototipo MUS-T:

- Error de **consistencia**: El requisito no se puede alcanzar con el modelo actual, ni en un modelo obtenido por evolución del actual en ciclos posteriores. Es necesario proporcionar el conjunto de requisitos que entran en conflicto y un contraejemplo. El usuario simulará el contraejemplo, para poder decidir cuál de los requisitos en conflicto es el origen del error.
- Error de **incompletitud**: El requisito puede ser alcanzado en un ciclo posterior. Se deben especificar un conjunto de sugerencias de modificación sobre el modelo actual. Informalmente, dichas sugerencias tratan de cambiar la condición de especificación de los elementos subespecificados, para poder satisfacer el requisito. Este mecanismo puede ser entendido como una forma de síntesis incremental. El usuario debe poder simular el modelo actual con las

sugerencias aportadas por la metodología con el objetivo de decidir qué sugerencias concuerdan con sus expectativas.

[3] **Diseño de la arquitectura**: Considerando un sistema concurrente como la composición de una parte funcional y una parte de sincronización, el diseño de la arquitectura lógica consiste en la síntesis del proceso sincronizador a partir de los componentes funcionales y la especificación del sistema global. Aunque no abordado en el grueso de esta tesis, se propone el uso de estrategias para la búsqueda de procesos sincronizadores o controladores para sistemas empotrados.

5.6.2 Características específicas del ámbito de tiempo real

La fase de *refinamientos* en la metodología SCTL/MUS propone la traducción MUS a LOTOS con la incorporación de decisiones arquitectónicas y la construcción de prototipos cada vez con más nivel de detalle, hasta una correspondencia semiautomática con la implementación del sistema final.

Dado que, en base a la definición de un sistema de tiempo real, su correcta ejecución depende de la satisfacción de los requisitos de tiempo, se trata de sistemas que reintroducen en la especificación detalles de implementación que comúnmente se abstraen en el diseño de los sistemas convencionales. Tal dependencia de la implementación hace que la disponibilidad de los recursos y la política de planificación sean básicas a la hora de garantizar la correcta ejecución del sistema final.

Por estar centrado el trabajo de esta tesis en la fase de *objetivos iniciales*, se postergan tales consideraciones a fases de diseño posteriores, centradas en la fase de *refinamientos*. En esta fase, se deben incorporar elementos de modelado más cercanos a la implementación, que permitan la especificación de recursos y prioridades. Sin una noción explícita de recurso, la especificación de los sistemas de tiempo real (sistemas fuertemente dependientes de la arquitectura física) debe modelar el uso de los recursos como un proceso que se ejecuta concurrentemente con las tareas del sistema; y, análogamente, la política de prioridades. Por tanto, la no disponibilidad de tales herramientas de diseño, nos llevaría a especificaciones que se alejan de la arquitectura lógica del sistema (integrando detalles dependientes de la arquitectura física) y, consecuentemente, a especificaciones difíciles de mantener ante el cambio de la ocupación de recursos o la política de planificación. Es por ello que se propone o bien la extensión de E-LOTOS (ISO/IECJTC1/SC21/WG7, 1998) (extensión con tiempo de LOTOS) en este sentido, o la utilización de otra álgebra de procesos, apuntando ACSR o TAM (ver sección 2.3) como la solución más apropiada, dada su incorporación explícita de recursos y prioridades.

Dado que se trata de una metodología de aplicación al ámbito de los sistemas de tiempo real críticos, es primordial el proporcionar un análisis de planificabilidad integrado. Desde el punto de vista del tratamiento formal, el análisis de planificabilidad con la elección a priori de una política de planificación, puede ser realizado utilizando *model checking*. Una vez modelado el planificador como un componente MUS-T de ejecución concurrente con los componentes MUS-T a planificar, se trataría de asegurar que todas las tareas cumplen sus plazos. Igualmente en el campo del tratamiento formal, y haciendo uso de técnicas aplicadas a la síntesis

de controladores, sería posible a partir de los componentes MUS-T realizar la síntesis de un planificador a medida (si ello es posible) que garantice que todas las tareas son servidas. Sin embargo, y en base a las consideraciones antes expuestas, consideramos más oportuno retrasar tal análisis a la fase de *refinamientos*, en la que se introducen las características físicas relevantes para el diseño del sistema.

Como ya habíamos adelantado, el dominio de aplicación crítico impone la verificación formal como prueba completa de la corrección del sistema. Sin embargo, si bien la verificación formal tiene el ambicioso objetivo de dar una garantía de corrección absoluta sobre el sistema desarrollado, tal objetivo es difícilmente alcanzable cuando se trata de un sistema de tamaño grande, o en las últimas fases, en las que el proceso *software* se torna menos automático y formal. Técnicas como la prueba o la monitorización en tiempo de ejecución no pueden ser descartadas. Sin embargo, y aún en este campo, nos podemos beneficiar de la formalidad introducida en el proceso *software*. De esta forma, se propone la generación de *tests* basada en la especificación formal obtenida en la fase de *refinamientos*, y la construcción de un monitor que permita detectar las diferencias del sistema final con dicha especificación (sección 3.6.2).

Como resultado, se hace del todo necesario adecuar la fase de refinamientos específicamente para tiempo real, atendiendo a todas las consideraciones arriba expuestas.

5.7 Conclusiones

Como conclusión, nos remitimos a las pautas dadas por *Heitmeyer* (Heitmeyer, 1998) para producir métodos formales prácticos. Si bien los objetivos de esta tesis no los cubren en su totalidad, intentaremos exponer la adecuación, a este respecto, del modelo de ciclo de vida propuesto:

- **Integrar el método en un proceso *software***: Bajo este objetivo se encuentra la motivación de la metodología formal SCTL/MUS; la definición de unos formalismos y unos procedimientos de decisión que se integren en un proceso *software*.

- **Minimizar el esfuerzo y la experiencia necesaria para la aplicación del método**: La sola inclusión de una lógica temporal para la captura de requisitos semeja alejarnos de esta pauta. Aunque no tratado en esta tesis, se sugieren dos líneas básicas. Por un lado, la definición de patrones de diseño, con una presentación en lenguaje natural del problema, y la solución en el formalismo, en este caso SCTL/MUS-T, facilita en gran medida la experiencia necesaria para la aplicación de una metodología formal (Dwyer et al., 1999). Por otro lado, y de relación directa con los requisitos de tiempo, consideramos adecuada la especificación gráfica de las distancias relativas de los requisitos de tiempo, que permita una traducción automática a fórmulas SCTL-T.

- **Hacer el análisis formal tan automático como sea posible**. En este punto nos referimos a técnicas *pushbutton*. En el uso de las técnicas formales podemos catalogar de *pushbutton* el chequeo de consistencia y la utilización

de *model checking*. En cuanto al chequeo de consistencia, especialmente importante en modelos de ciclo de vida iterativos, cabe resaltar que, en SCTL/MUS-T, tal chequeo es implícito a la metodología desarrollada, no necesitando procedimientos de decisión externos. Por otro lado, si bien la principal ventaja del uso de *model checkers* como garantes de corrección es precisamente su total automatización, tal ventaja tropieza con la explosión del tamaño del espacio de estados. Tal explosión, principalmente originada en la composición de sistemas, se agudiza, en nuestro caso, por la utilización de un dominio de tiempo denso. En este sentido, se propone, mediante minimización, la búsqueda de abstracciones lo más gruesas posibles para el problema concreto (capítulo 10).

- **Proporcionar realimentación:** En caso de error, el usuario debe disponer de una realimentación, útil y fácil de entender, destinada a corregir dicho error. La no posibilidad de cumplimiento (error de consistencia) de un requisito SCTL debe, en consecuencia, ir acompañada de un contraejemplo localizando el origen del error. Igualmente, la no satisfacción originada en la incompletitud del modelo, debe ir acompañada de una sugerencia localizando la forma en que la satisfacción del requisito puede ser alcanzada. Con esta perspectiva, se ha optado por el diseño de algoritmos de verificación (capítulo 8) y síntesis (capítulo 9) que operan de formal local sobre una abstracción del modelo MUS-T bajo estudio. Dicha decisión de diseño posibilita, de forma directa, la construcción de contraejemplos y sugerencias durante el cómputo del grado de satisfacción de un requisito SCTL-T.

- **Proporcionar un conjunto de herramientas de análisis**: Esta pauta de diseño, especialmente apuntada para el caso de sistemas de tiempo real por *Stankovic* (Stankovic, 1996), será alcanzable mediante las propuestas realizadas para la fase de *refinamientos*.

- **Proporcionar capacidades de simulación potentes y adaptables**: Se propone el uso de un simulador sobre MUS-T que permita la validación de la especificación. Tal simulador será la entrada de los contraejemplos y sugerencias generados por el *model checker*.

Adicionalmente, *Heitmeyer* apunta al desarrollo de **demostradores de teoremas fáciles de usar**. En este sentido, el ámbito del tratamiento formal de los sistemas de tiempo real se encuentra en fases muy tempranas para una satisfacción razonable de tal objetivo.

CAPÍTULO 6

MUS-T

En este capítulo se presenta el modelo **MUS-T** *(**T**imed **M**odel of **U**nspecified **S**tates), extensión temporizada del modelo MUS. Tal extensión permite la formalización de un modelo de proceso software de naturaleza iterativa e incremental, acorde con las tendencias actuales de la ingeniería del software, específico para su aplicación en el ámbito de los sistemas de tiempo real críticos. Para ello, se utilizará un esquema de relojes con valores en $\Re_+$ semejante al utilizado en los modelos de estados basados en el paradigma del autómata temporizado.*

6.1 Introducción

El origen de las máquinas de estados con tiempo denso se remonta a principios de la década de los noventa, cuando, después de numerosas aproximaciones con tiempo discreto, surgen las primeros resultados positivos con modelos densos. Si bien su historia es corta, la investigación ha sido prolija. De entre todas las máquinas de estados que podemos encontrar en la literatura (ver sección 2.4), el paradigma del autómata temporizado es el que ha tenido mayor éxito éxito en lo que se refiere a resultados teóricos y aplicaciones a casos reales.

No obstante, la principal limitación del autómata temporizado (TA) radica en que su versión no determinista (NDTA) no es cerrada bajo complementación, lo que resulta del hecho de que, a diferencia de los modelos sin tiempo, en los modelos con tiempo denso el autómata no determinista es más expresivo que el determinista (DTA). Este hecho imposibilita la verificación basada en autómata de propiedades (sección 3.2.2), dado que dicha verificación conlleva la complementación del autómata de especificación de la propiedad.

Se ha argumentado acerca de la naturaleza demasiado restrictiva de DTA, y la naturaleza demasiado permisiva de NDTA, llevando, en el primer caso, a ser incapaz de expresar determinadas propiedades, y, en el segundo caso, a un problema de inclusión de lenguajes no decidible. Intentando resolver las carencias del autómata temporizado, se han definido un serie de modelos alternativos. De

especial mención resultan los modelos que evitan el reinicio explícito de los relojes, utilizando relojes en el modelo, que o bien están asociados a estados del modelo (Raskin y Schobbens, 1997) (se reinician con los estados), o bien están asociados a eventos en el modelo (Alur et al., 1999) (se reinician con los eventos). Dichos modelos, aunque menos expresivos que TA, son cerrados bajo todas la operaciones booleanas. Lo que resta es su uso para la resolución de problemas reales que garanticen que la pérdida de expresividad es irrelevante para el ámbito de aplicación. Por último citar (Alur y Henzinger, 1992a), donde el problema se resuelve sólo en parte, es decir, aunque la versión no determinista sigue siendo no cerrada para complementación, se proporcionan versiones deterministas más expresivas, dotando al autómata de la posibilidad de moverse hacia delante y hacia atrás en el tiempo.

Dada su mayor expresividad, avalada, a su vez, por su aplicación a casos reales (sección 3.5), se ha optado por una extensión del modelo MUS con tiempo basada en *Timed Automaton*. Además, los métodos de decisión utilizados en la metodología SCTL/MUS son métodos basados en verificación heterogénea; tanto el algoritmo de *model checking* como el algoritmo de síntesis incremental operan sobre el modelo de estados MUS en base a la estructura de la fórmula a verificar o sintetizar.

Introducidas las razones que nos han decantado por el paradigma del autómata temporizado, el objetivo de este capítulo es presentar la extensión con tiempo MUS-T (*Timed Model of Unspecified States*) del modelo de estados con subespecificación MUS (*Model of Unspecified States*).

A continuación introduciremos, de manera informal, las diferencias fundamentales entre el modelo MUS y el modelo MUS-T. Para establecer la definición formal, en la sección 6.4 introducimos un conjunto de definiciones preliminares comúnmente utilizadas en las máquinas de estados con tiempo denso. A continuación, definiremos formalmente la sintaxis de los modelos MUS-T (sección 6.5) en forma de sistemas de transiciones etiquetados en un alfabeto de acciones y un conjunto de valores de verdad; y, después, su semántica asociada en forma de grafos densos multivalorados (sección 6.6) que se adecúe a la naturaleza ramificada de SCTL-T. Dado el ámbito de aplicación de los sistemas de tiempo real críticos, consideramos la reactividad una característica básica en estos sistemas, que debemos apoyar ofreciendo garantías de avance en el sistema (sección 6.7). Además, en general, el modelado de un sistema de tiempo real crítico conllevará el modelado de m procesos cooperantes y comunicantes con un objetivo común que impone requisitos funcionales y de tiempo sobre el sistema total. Con tal propósito, en la sección 6.8 se define la composición de modelos MUS-T que se sincronizan en un conjunto de acciones, interpretadas como canales de sincronización.

6.2 De MUS a MUS-T

El modelo MUS refleja la estructura discreta de un sistema reactivo, es decir, un conjunto de estados discretos y un conjunto de transiciones. Con el objetivo de permitir el tratamiento de sistemas incompletos en las primeras fases de diseño y el chequeo de consistencia en un ciclo de vida iterativo e incremental, el modelo MUS permite definir grafos que almacenan los comportamientos identi-

ficados en un sistema, de forma que comportamientos fuera de los identificados serán comportamientos subespecificados. Un grafo MUS caracteriza las acciones y, consecuentemente, las transiciones en el modelo, como posibles, no posibles o subespecificadas (condición de especificación). La condición de especificación permite registrar la información conocida hasta el momento de los comportamientos exhibidos por el sistema. Un nuevo ciclo en el proceso, cuando supone un incremento en la funcionalidad, se traduce en un cambio en la condición de especificación de los elementos subespecificados del modelo a elementos posibles o no posibles.

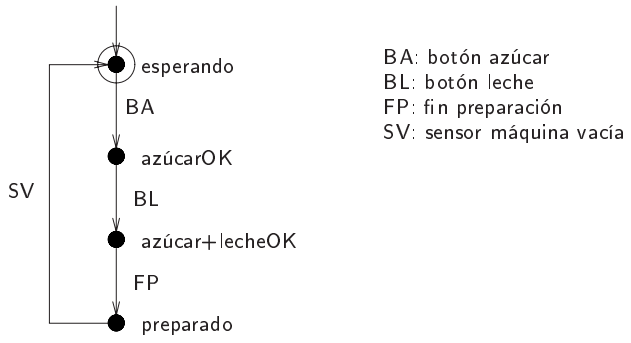


Figura 6.1. Modelo MUS de una funcionalidad inicial de la máquina de café.

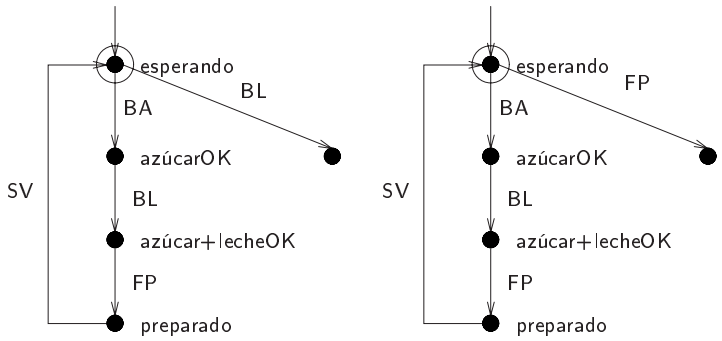


Figura 6.2. Distintas funcionalidades que un formalismo sin control de consistencia permitiría añadir al modelo inicial de la figura 6.1.

En el modelo MUS, los estados representan el modo de operación en que se encuentra el sistema, mientras las transiciones describen las formas en que el control cambia de un modo a otro. En la figura 6.1 se muestra el modelo MUS con los comportamientos posibles de una máquina de café. Sin duda, ésta sería la perspectiva de una máquina de estados típica. Como se muestra en la figura 6.2, sería factible incrementar su funcionalidad, por ejemplo, permitiendo que se seleccione primero la leche y después el azúcar. De la misma forma, el diseñador podría optar por añadir en el estado esperando una nueva transición para el sensor FP (fin preparación). Pero, seguramente, en la idea inicial del diseñador no estaba la segunda opción y sí la primera. Con este sencillo ejemplo, vemos como la falta de especificación de los elementos de comportamiento en un modelo puede obedecer a dos causas: o bien los elementos de comportamiento no son posibles, o bien los

elementos de comportamiento no han sido especificados. En este punto es donde la idea de condición de especificación, como caracterizadora de los elementos en un modelo de estados, imprime una característica distintiva al modelo MUS.

Dado que en los sistemas de tiempo real críticos la puntualidad de los eventos caracteriza la corrección del sistema, la extensión del modelo MUS ha de permitir el modelado de comportamientos con evolución temporal cuantitativa y no sólo cualitativa. Los estados y transiciones del modelo MUS capturan toda la información discreta de la especificación actual del sistema, pero no la información de tiempo. Es decir, la noción de tiempo en el modelo MUS es puramente cualitativa: existe un orden temporal, pero no una métrica temporal. Como ya hemos mencionado, a la hora de introducir dicha información cuantitativa, de la que depende la corrección de los sistemas de tiempo real, nos basamos en el paradigma del autómata temporizado. Un autómata temporizado representa el tiempo mediante un conjunto de relojes que evolucionan al mismo ritmo. Los relojes pueden ser eventualmente reiniciados en una transición del autómata y, además, las transiciones se asocian a guardas de tiempo que habilitan el cambio de estado. Sin embargo, mientras las proposiciones atómicas en un estado de control del autómata temporizado son proposiciones ciertas o falsas, en un enfoque iterativo e incremental, tales proposiciones serán ciertas o falsas en el sistema final, pero en un ciclo de proceso intermedio cualquier proposición puede ser cierta, falsa o no especificada (subespecificada).

En consecuencia, MUS-T extiende el modelo MUS con un conjunto de relojes y predicados de tiempo sobre dichos relojes. Los predicados de tiempo, sobre los relojes incorporados, se asocian a las acciones definidas en el formalismo MUS. Dichos predicados o guardas de tiempo permiten fijar la condición de especificación de las acciones (posible, no posible o subespecificada) en función del contexto temporal actual en la evolución del sistema. Dicho contexto, como veremos, está marcado por los valores de relojes (interpretaciones) incorporados en el modelo para registrar distancias temporales.

Pasamos, pues, de un modelo MUS en el que la condición de especificación de un evento o acción (de entre los identificados en el sistema) depende únicamente del estado en el que reside el sistema, a un modelo MUS-T en el que dicha condición depende del "estado" y, adicionalmente, del "tiempo" en el que reside el sistema.

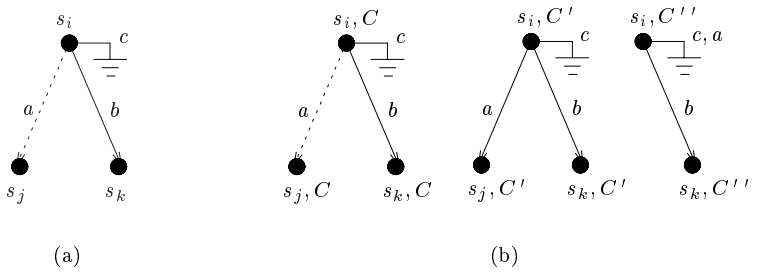


Figura 6.3. Condición de especificación cualitativa (MUS) 6.3(a) frente a condición de especificación cuantitativa (MUS-T) 6.3(b)

En la figura 6.3, podemos ver dicha diferencia. En el modelo MUS-T, la condi-

ción de especificación es cuantitativa, es decir, para una localización en el modelo, la condición de especificación de los eventos varía en función del contexto temporal en el sistema. Así, en el estado s_i (figura 6.3(b)), para el contexto temporal C la acción a está subespecificada, mientras que para el contexto temporal C'' dicha acción está prohibida.

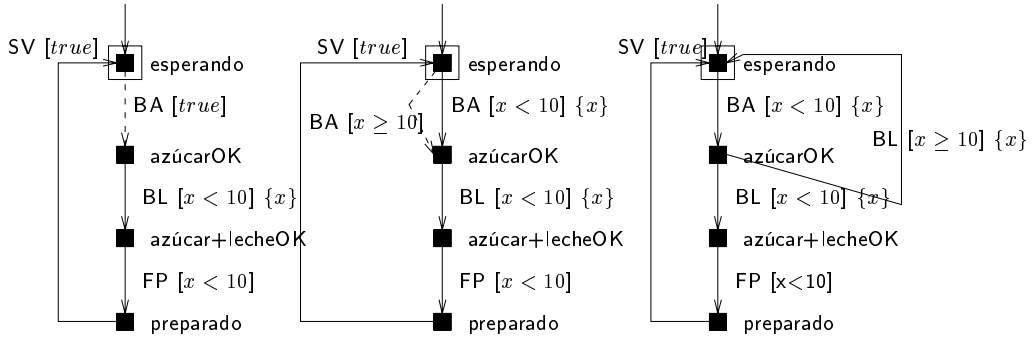


Figura 6.4. Modelo MUS-T: estructura discreta + información de tiempo

En el modelo MUS-T de la máquina de café se especifican los distintos estados de control del sistema y, adicionalmente, la información de tiempo que marca las transiciones de un estado de control a otro. Así, en el primer modelo de la figura 6.4, vemos que una vez seleccionados el azúcar y la leche (poniendo a cero el reloj x, denotado por $\{x\}$), el café se prepara en menos de 10 segundos (FP $[x < 10]$). La capacidad de evolución en funcionalidad en el modelo MUS se conserva en MUS-T, pero, además, se multiplica. Si el control del sistema reside en la localización `azúcar+lecheOK`:

- ¿Cómo se comporta el sistema si llega una señal del sensor `SV`?
- ¿Cómo se comporta el sistema si le llega la señal del sensor `FP` en un tiempo mayor de 10 segundos?

Intentando responder a estas preguntas, podemos ver como la capacidad de evolución del sistema a partir de la localización azúcar+lecheOK, que en el modelo MUS se limita a la especificación de los comportamientos cuando llegan las señales de los sensores BL, BA y SV, se multiplica de forma infinita en el modelo MUS-T. Así, en el caso límite, podríamos introducir en esta localización infinitos comportamientos adicionales, en función de cuál sea la señal y del instante temporal en que dicha señal es recibida.

En la figura 6.4 se pueden observar dos modificaciones que el modelo original permite realizar en la especificación. Así en la especificación central, se ha decidido especificar la acción BA como posible para aquellos instantes temporales que cumplan $x < 10$, quedando subespecificada en otros casos. Además, en la especificación derecha se ha decidido que si el usuario no presiona el botón BL en 10 unidades de tiempo desde la selección del azúcar, dicha opción quedará deshabilitada, volviendo al estado esperando.

Bajo este sencillo ejemplo subyace la idea principal en el modelado de sistemas de tiempo real con máquinas de estados: hay una evolución discreta (de un modo a otro), pero también hay una evolución continua en el tiempo.

6.2.1 Evolución discreta

Para dar soporte al modelado de información incompleta y a la comprobación de consistencia, los eventos temporizados (eventos con información de tiempo de ocurrencia) en un estado del modelo MUS-T se pueden caracterizar como posibles (línea continua), prohibidos o no posibles (sin transición) o subespecificados (línea discontinua o ausencia de transición).

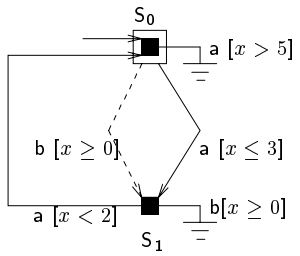


Figura 6.5. Los eventos temporizados en MUS-T

Así, en la figura 6.5:

- El evento temporizado a$[x \leq 3]$ es un evento posible que lleva al estado S_1, y el evento temporizado a$[x > 5]$ está prohibido en el estado S_0.
- El evento temporizado b$[x \geq 0]$ está parcialmente subespecificado en el estado S_0; es decir, si dicho evento se caracterizase como posible, su estado sucesor sería S_1.
- El evento a en el estado S_1 está totalmente subespecificado para instantes de ocurrencia que no coincidan con su especificación posible $[x < 2]$.

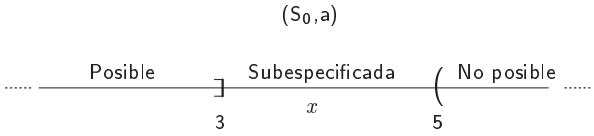


Figura 6.6. Zonas de especificación para S_0 y a

Por tanto, para cada localización y para cada evento identificado en el sistema, el dominio temporal se puede dividir en tres zonas con distinta naturaleza de especificación (figura 6.6 para S_0 y a): la zona posible, la zona no posible y la zona subespecificada. Para cualquier localización del modelo, y para cualquier evento, las ejecuciones del sistema seguirán las siguientes reglas:

- Si el contexto temporal pertenece a la zona no posible, el sistema no puede evolucionar (de forma discreta) en dicho evento.
- Si el contexto temporal pertenece a la zona posible, el sistema puede evolucionar (de forma discreta) en dicho evento, pasando el control a la localización sucesora para dicho evento.
- Si el contexto temporal pertenece a la zona subespecificada, el evento temporizado podrá ser caracterizado, en una evolución futura del modelo, como posible o no posible para dicho contexto temporal.

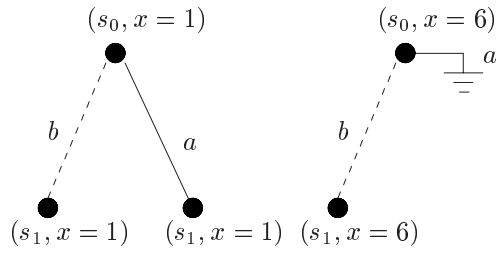


Figura 6.7. Comportamiento discreto en MUS-T

En la figura 6.7 se muestran dos ejemplos del modelo de ejecución discreto para el modelo MUS-T de la figura 6.5: en una transición discreta, el sistema pasa de un estado de control a otro en el modelo, conservando el contexto temporal; es decir, las transiciones discretas son instantáneas. Por tanto, la evolución discreta en el modelo de ejecución MUS-T será idéntica a la definida en el modelo MUS. Sin embargo, dado que, bajo distintos contextos temporales, un mismo estado de control puede tener configuraciones de especificación diferentes, el modelo de ejecución se torna en un máquina de estados con un espacio de estados infinito.

6.2.2 Evolución temporal

En la figura 6.8 podemos ver una representación de la evolución continua en un modelo MUS-T. Por evolución continua entendemos las transiciones temporales en el modelo de ejecución. Mientras en la figura 6.7 veíamos las transiciones discretas que hacen que el modelo de ejecución pase de un estado de control a otro, mediante una transición continua o temporal el sistema permanece en el mismo modo, sólo produciéndose un incremento temporal τ. Si la evolución discreta era finita, aunque sobre un número infinito de estados, las transiciones temporales imprimen en el modelo MUS-T una naturaleza infinitamente ramificada.

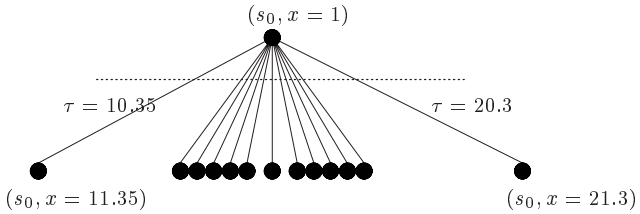


Figura 6.8. Comportamiento continuo en MUS-T

6.3 Algo más que el control discreto

Una simple comparación entre las figuras 6.7 y 6.8 nos descubre la asimetría entre las transiciones temporales y discretas presentadas para el modelo MUS-T: el conjunto de transiciones temporales continuas en una localización del modelo es independiente del contexto temporal en que se encuentre el sistema, es decir, las

restricciones de tiempo acompañan a los eventos formando eventos temporizados y controlan el instante de ocurrencia de las transiciones discretas, pero nada controla las transiciones temporales, siempre se pueden producir.

Si bien en el modelo de *Timed Automaton* (TA) original (Alur y Dill, 1990b), las restricciones de tiempo sólo determinan los momentos de ocurrencia de las transiciones discretas, la variante introducida por (Henzinger et al., 1992b) (*Safety Timed Automaton* - STA) generaliza la noción de control temporizado, de forma que es posible especificar restricciones de tiempo sobre la evolución temporal y no sólo sobre la evolución discreta del sistema. Para ello, STA define invariantes en los estados de control del modelo. Una invariante es una restricción de tiempo del mismo tipo que las utilizadas en las guardas de TA, pero, mientras una guarda se asocia a una transición discreta y controla el momento en que el sistema puede evolucionar en dicha transición, las invariantes se asocian a los estados de control del TA y permiten controlar cuánto tiempo el sistema puede residir en dicho estado de control. Dicha variante se ha revelado como más realista para el modelado de sistemas de tiempo real, además de permitir el tratamiento de urgencia en dichos sistemas.

En general, las restricciones de avance temporal se han incorporado en los modelos de tiempo real mediante la utilización o bien de invariantes o bien de plazos. Las invariantes caracterizan los estados de una localización en los que el tiempo puede avanzar, mientras que los plazos caracterizan los estados de una localización en los que el avance del tiempo se detiene. En (Bornot y Sifakis, 1997) podemos encontrar un estudio formal sobre el uso de invariantes o de plazos para la caracterización de las condiciones de avance temporal más habituales en la práctica. El expresar el control sobre el avance temporal de una forma u otra depende de si deseamos una formulación positiva o negativa en la especificación, pudiéndose considerar la utilización de invariantes como una especificación positiva (afirmativa) y la utilización de plazos como una especificación negativa.

Dado que en nuestro proceso software incremental se incluyen tanto especificaciones afirmativas como especificaciones negativas, dejando un punto intermedio para las especificaciones inciertas (subespecificadas), decidimos incorporar en el modelo los requisitos de avance mediante los dos mecanismos. El objetivo será alcanzar un sistema final en el que las invariantes y los plazos se complementen, de forma que no exista ningún avance temporal que se encuentre subespecificado.

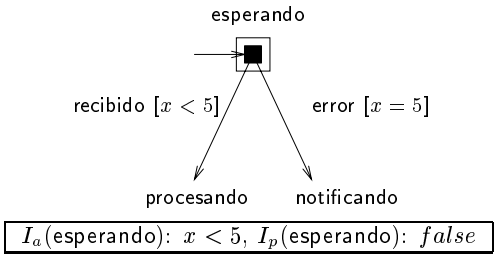


Figura 6.9. Introducción de control sobre el avance temporal

Siguiendo la filosofía del concepto de invariantes para el control de la evolución temporal del modelo, se introducen las invariantes en el modelo MUS-T de una

forma simétrica a la utilizada en el tratamiento de las acciones temporizadas. Así, para cada localización del modelo, definimos [1]:

- Invariante de avance (I_a): determina los contextos temporales alcanzables mediante una transición temporal.
- Invariante de parada (I_p): determina los contextos temporales no alcanzables mediante una transición temporal.
- Los contextos temporales no incluidos en ninguno de los anteriores serán contextos temporales en los que el acceso por paso temporal está subespecificado.

Para el ejemplo del modelo MUS-T con invariantes de la figura 6.9, podemos observar como las transiciones temporales posibles y subespecificadas (el ejemplo no define invariante de parada) dependen del contexto temporal en el que se encuentre el sistema (figura 6.10).

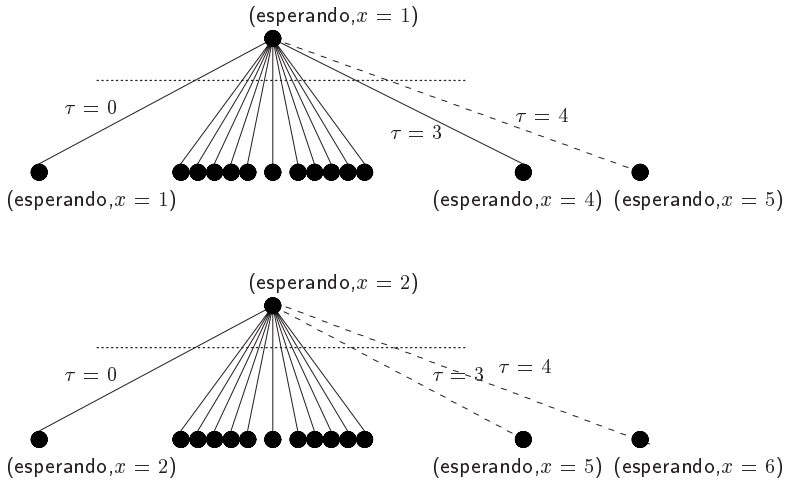


Figura 6.10. Control del progreso del tiempo: dependencia del contexto temporal

Como habíamos anticipado, la introducción de invariantes permite el modelado de urgencia. Así, si en el ejemplo de la figura 6.9 modificamos la invariante de parada a $I_p(\text{esperando}) : x \geq 5$, habremos especificado el evento error como urgente. En la figura 6.11 se ilustra el uso de las invariantes para el modelado de urgencia. Una acción se puede diferir (diferible) si, una vez habilitada su ocurrencia, puede ser relegada por una transición temporal. Una acción no se puede diferir (inmediata) si no puede ser pospuesta por una transición temporal. La introducción de tres invariantes hace que la urgencia de una acción en un estado pueda estar subespecificada.

Hasta aquí hemos conformado el boceto de un modelo MUS-T: los eventos con su condición de especificación temporizada (posible, no posible o subespecificada), y las invariantes con su condición de especificación también temporizada (de parada y de avance). La dualidad de comportamiento (discreto y continuo)

[1] La definición de invariantes es ligeramente diferente a la comúnmente utilizada, que especifica los contextos temporales en los que puede residir un estado de control.

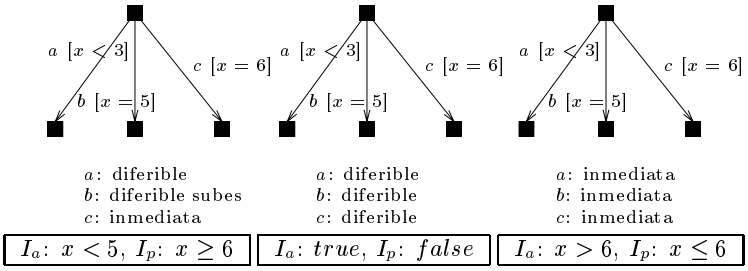


Figura 6.11. Invariantes en el modelado de urgencia

en los sistemas de tiempo real lleva a un tratamiento dual del diseño incremental basado en MUS-T, definiendo similares condiciones de especificación para ambos comportamientos.

En las siguientes secciones de este capítulo introducimos la definición formal del modelo MUS-T (su sintaxis y su semántica), recogiendo los conceptos informalmente presentados hasta el momento.

Por último, notar que las invariantes introducidas en STA, como un mecanismo para el modelado de los requisitos de avance temporal, presentan desventajas a la hora de realizar la composición de modelos, pudiendo llevar a situaciones de *deadlock* difícilmente identificables por el diseñador. Para salvar dichos problemas en composición, se ha introducido una nueva variante del autómata temporizado TAD –*Timed Automaton with Deadlines* (Bornot et al., 1997)–; dicha variante traslada la información de urgencia de las localizaciones a las transiciones, revelándose como más composicional.

6.4 Preliminares: relojes, interpretaciones, predicados de tiempo y estados

Como ya hemos mencionado, el tiempo se modela mediante la incorporación de un conjunto finito de relojes $\mathcal{C}$ que pueden ser reiniciados en cualquier transición discreta del modelo de estados. Todos los relojes progresan a la misma velocidad (la de un reloj global ficticio) y registran la cantidad de tiempo transcurrido desde el momento en que son reiniciados. Acorde a nuestro modelado denso del tiempo, optamos por relojes que toman valores en un dominio $\Re_+$.

Definición 6.1. **Interpretación y vector de interpretaciones** Dado un reloj $x \in \mathcal{C}$, definimos la interpretación de x, representada por $\gamma(x)$, como el valor actual de dicho reloj en el sistema:

$$\gamma : x \in \mathcal{C} \rightarrow \Re_+$$

y el vector de interpretaciones, $\gamma(\mathcal{C}) \in \Re_+^{\#\mathcal{C}}$:

$$\gamma(\mathcal{C}) = \{\gamma(x),\ \forall x \in \mathcal{C}\}$$

como el vector de valores actuales de todos los relojes en el sistema. En adelante,

utilizaremos γ para referirnos al vector de interpretaciones de los relojes $\mathcal{C}$ en el modelo y $\gamma(x)$ para referirnos a la componente x de dicho vector.

DEFINICIÓN 6.2. **Incremento de una interpretación.** Dado γ un vector de interpretaciones en $\Re_+^{\#\mathcal{C}}$, y $\tau \in \Re_+$ un avance temporal, definimos $\gamma + \tau$ como el vector de interpretaciones tal que $(\gamma + \tau)(x) = \gamma(x) + \tau,\ \forall x \in \mathcal{C}$.

DEFINICIÓN 6.3. **Reinicio de relojes en una interpretación.** Dado γ un vector de interpretaciones en $\Re_+^{\#\mathcal{C}}$, y $\lambda \subseteq \mathcal{C}$ un conjunto de relojes, definimos γ_λ como el vector de interpretaciones tal que $\gamma_\lambda(x) = 0$ si $x \in \lambda$ y $\gamma_\lambda(x) = \gamma(x)$ en otro caso.

Las características de tiempo, tales como retardos de propagación, tiempos de ejecución y tiempos de respuesta, se expresan como predicados sobre los valores de los relojes. Dichos **predicados de tiempo** no serán más que guardas e invariantes de tiempo asociadas a los elementos en un modelo MUS-T (eventos y estados de control). Una acción posible lo será bajo una determinada guarda de tiempo; análogamente, una acción que el diseñador ha caracterizado como no posible, lo será bajo una guarda de tiempo. Fuera del alcance de dichas guardas, el momento en que la transición puede ocurrir se encontrará subespecificado.

Para poder definir el modelo MUS-T de un sistema, debemos formalizar los predicados de tiempo (condiciones sobre los valores de los relojes) que se contemplan en el modelo. Un predicado de tiempo será cualquier combinación booleana de fórmulas atómicas que comparan interpretaciones o diferencias de interpretaciones de los relojes con constantes enteras. En (Alur, 1991) se demuestra que la incorporación de condiciones de tiempo más expresivas ($x > y + z$) no sería realizable en un autómata con memoria finita, incluso si consideramos un modelo de tiempo discreto.

DEFINICIÓN 6.4. **Predicados de tiempo.** Para un conjunto de relojes finito $\mathcal{C}$, el conjunto $\Psi(\mathcal{C})$ de predicados de tiempo ψ se define:

$$\Psi(\mathcal{C}) ::= \{x \prec c \mid x - x' \prec c \mid \neg\ \psi \mid \psi\ \wedge\ \psi' \mid \psi\ \vee\ \psi'\}$$

donde $x, x' \in \mathcal{C}$, $\prec \in \{<, \leq, >, \geq, =\}$, y $c \in \aleph$, siendo $\aleph$ el conjunto de los números naturales. Por simplicidad, se definen en $\Psi(\mathcal{C})$ las constantes:

- $true \equiv \bigwedge_{\forall x \in \mathcal{C}} x \geq 0$
- $false \equiv \bigwedge_{\forall x \in \mathcal{C}} x < 0.$

DEFINICIÓN 6.5. **Satisfacción booleana de un predicado de tiempo.** Bajo la satisfacción booleana habitual, diremos que una interpretación γ satisface un predicado de tiempo $\psi \in \Psi(\mathcal{C})$ sobre un conjunto de relojes $\mathcal{C}$, si la instanciación $x \to \gamma(x)\ \forall\ x \in \mathcal{C}$ cumple la relación aritmética ψ (denotado $\psi(\gamma) = 1$). En otro caso, el predicado de tiempo no se satisface (denotado $\psi(\gamma) = 0$).

El **estado** de un sistema es la información sobre el pasado del sistema que es necesaria para determinar su futuro. De esta forma, en los modelos de estados

sin tiempo, como autómatas, sistemas de transiciones etiquetados o máquinas de estados, se definen los estados. Al incorporar el tiempo a un modelo de estados, el estado de control o localización actual del sistema no es suficiente para determinar su "estado", siendo necesario, además, "alguna medida" del tiempo en el sistema. Es por ello que en los modelos de estados con tiempo se suele hablar de modos o localizaciones para referirnos al estado de control en que reside la máquina de estados y de estado como la información concerniente a la parte de control del sistema y a dicha medida de tiempo.

Definición 6.6. **Estado de un modelo MUS-T.** Sea $\mathcal{M}$ un modelo MUS-T con un conjunto de localizaciones S. Un estado de $\mathcal{M}$ será un par (s, γ), donde $s \in S$ es la localización en la que se encuentra el control y γ la interpretación actual de todos los relojes $\in \mathcal{C}$.

Por tanto, el modelo operacional que le da semántica será, o bien un grafo denso, o bien un conjunto de trazas densas para semántica lineal, con estados (s, γ) como vértices, donde s marca el estado de control en que reside el modelo, y γ la interpretación actual de los relojes (definición 6.1).

6.5 Sintaxis

Formalmente, un grafo MUS-T es un grafo (temporizado) multivalorado de transiciones etiquetadas, definido por la 6-tupla $\langle S_0, S, T, I, A, \mathcal{C}\rangle$ sobre el conjunto de verdad $\mathcal{L}_3 = \{0, \frac{1}{2}, 1\}$, donde identificamos:

- El conjunto de verdad, $\mathcal{L}_3 = \{0, \frac{1}{2}, 1\}$, o de condición de especificación, que determina la condición de especificación de las transiciones en el modelo.
- Un conjunto finito de acciones o eventos A.
- Un conjunto finito de localizaciones discretas S, que incluye las localizaciones ficticias sumidero subespecificado s_s y sumidero no posible s_{np} (sección 6.5.2).
- Un conjunto $S_0 \subseteq S$ de localizaciones iniciales.
- Un conjunto finito de relojes $\mathcal{C}$.
- Un conjunto de transiciones T:

  $$T \subseteq S \ \times \ A \ \times \ \Psi(\mathcal{C}) \ \times \ 2^{\mathcal{C}} \ \times \ S$$

  y una función $CE : T \rightarrow \mathcal{L}_3$ asignando un valor de verdad (condición de especificación) a las transiciones.

  Cada elemento $t \in T$ con $\mathbf{CE(t) = 1}$ define:

  $$(s, < a, g_p(s, a), \lambda >, s\,')$$

  una transición posible con localización origen s, localización destino $s\,'$, y una acción guardada posible $< a, g_p(s, a), \lambda >$ especificando una acción $a \in A$,

una guarda $g_p(s,a) \in \Psi(\mathcal{C})$ y un conjunto $\lambda \subseteq \mathcal{C}$ de relojes que serán reiniciados en la transición. La localización destino s' es un a-sucesor posible de la localización origen s.

De forma análoga, cada elemento $t \in T$ con $\mathbf{CE(t)} = \frac{1}{2}$ define:

$$(s, < a, g_s^P(s,a), \lambda >, s')$$

una transición parcialmente subespecificada con localización origen s, localización destino s' y una acción guardada subespecificada $< a, g_s^P(s,a), \lambda >$. La localización destino s' es un a-sucesor subespecificado de la localización origen s.

Las transiciones $t \in T$ con $\mathbf{CE(t)} = \mathbf{0}$ definen no transiciones:

$$(s, < a, g_{np}(s,a), \{\} >, s_{np})$$

siendo la localización sumidero no posible s_{np} su a-sucesor no posible.

- Dos funciones etiqueta I_a e I_p: $S \rightarrow \Psi(\mathcal{C})$, que asignan a cada localización $s \in S$:
  - Una invariante de avance $I_a(s)$ que determina las interpretaciones de los relojes en s alcanzables mediante una transición temporal, de forma que sólo son alcanzables temporalmente aquellas interpretaciones satisfaciendo la invariante de avance.
  - Una invariante de parada $I_p(s)$ que determina las interpretaciones de relojes que fuerzan al sistema a realizar una transición discreta o, si no, bloquearse en tiempo. No son alcanzables temporalmente aquellas interpretaciones satisfaciendo la invariante de parada.

6.5.1 Completitud y no solapamiento de condiciones de especificación

Como habíamos adelantado informalmente en la sección 6.2, las condiciones de especificación (para transiciones discretas y para transiciones temporales) en MUS-T particionan el espacio de interpretaciones en tres subconjuntos (posible, no posible y subespecificado). Dicha partición es completa y disjunta. Se imponen, por tanto, las siguientes restricciones de completitud y no solapamiento:

- Completitud:
  - Para todo $s \in S$, $a \in A$, a partir de los conjuntos $\{g_{p_i}\}$, $\{g_{s_j}^P\}$ y la guarda no posible g_{np}, se construye la función etiqueta de interpretaciones totalmente subespecificadas:

    $$g_s^T(s,a) = \neg(\bigvee_i g_{p_i}(s,a)) \wedge \neg(\bigvee_j g_{s_j}^P(s,a)) \wedge \neg(g_{np}(s,a))$$

    Por tanto, el conjunto de interpretaciones definidas por g_p, g_{np}, g_s^P, y g_s^T cubren $\Re_+^{\#\mathcal{C}}$, siendo $\#\mathcal{C}$ el número de relojes en $\mathcal{C}$.

La función etiqueta $g_s^T(s,a)$ define, para cada localización s y cada acción a, el conjunto de interpretaciones de relojes con un a-sucesor discreto totalmente subespecificado, definiendo implícitamente la transición:

$$(s, < a, g_s^T(s,a), \{\} >, s_s)$$

En adelante, denotaremos como $g_s(s,a)$ al predicado de tiempo que caracteriza las a-transiciones discretas subespecificadas en la localización s, es decir, $g_s(s,a) = g_s^P(s,a) \vee g_s^T(s,a)$.

– A partir de I_a e I_p se construye la función etiqueta I_s (invariante subespecificada): $I_s(s) = \neg I_p(s) \wedge \neg I_a(s)$. Por tanto, el conjunto de interpretaciones definidas por I_a, I_p e I_s cubren $\Re_+^{\#\mathcal{C}}$.

- No solapamiento de guardas e invariantes explícitas en modelo:

  – Guardas: $\nexists\ \gamma \in \Re_+^{\#\mathcal{C}}$ tal que se cumple alguna de las siguientes condiciones:

$$\bigvee_i g_{p_i}(s,a)(\gamma) \ \wedge\ g_{np}(s,a)(\gamma) = 1$$
$$\bigvee_i g_{p_i}(s,a)(\gamma) \ \wedge\ \bigvee_j g_{s_j}^P(s,a)(\gamma) = 1$$
$$g_{np}(s,a)(\gamma) \ \wedge\ \bigvee_j g_{s_j}^P(s,a)(\gamma) = 1$$

  – Invariantes: $\nexists\ \gamma \in \Re_+^{\#\mathcal{C}}$ tal que se cumple $I_a(s)(\gamma) \ \wedge\ I_p(s)(\gamma) = 1$

6.5.2 Localización sumidero subespecificado y no posible

Las localizaciones sumidero son el destino ficticio de las "no transiciones" en el modelo, es decir, las transiciones no posibles y las transiciones totalmente subespecificadas. Sin embargo, su naturaleza es radicalmente distinta en uno u otro caso: mientras la localización sumidero no posible es de crecimiento cero (no es origen de ninguna evolución discreta o temporal), la localización sumidero subespecificado es de crecimiento máximo (puede ser origen de cualquier evolución discreta o temporal).

Conforme a la anterior introducción informal, definimos la localización s_s con las siguientes características:

$$\forall\ a \in A,\ g_s^T(s_s,a) = true$$
$$I_s(s_s) = true$$

y la localización s_{np}:

$$\forall\ a \in A,\ g_{np}(s_{np},a) = true$$
$$I_p(s_{np}) = true$$

6.6 Semántica

A pesar de que el modelo MUS-T basa su definición en el paradigma de *Timed Automaton*, es más correcto decir que es una versión de su variante ramificada (*Timed Graph*). El grafo temporizado (Alur et al., 1993a) define su semántica sobre un árbol de estados denso, mientras el autómata temporizado se interpreta sobre secuencias de estados. Dada la mayor adecuación de la semántica ramificada a la resolución del problema de verificación mediante *model checking*, definimos una semántica ramificada para MUS-T que ya había sido utilizada en el modelo MUS. Sin embargo, mientras los grafos temporizados son dirigidos por tiempo, MUS-T define modelos dirigidos por eventos. Sin duda, en el diseño de sistemas reactivos, caracterizados por su intensiva comunicación con el entorno, un modelo dirigido por eventos parece más adecuado.

La semántica operacional asocia a un modelo MUS-T un grafo denso multivalorado, cuyos vértices son los estados del modelo (información de control más información de tiempo) y cuyos arcos (multivalorados) son, o bien transiciones discretas, o bien transiciones temporales.

6.6.1 Estados

El **estado** de un modelo MUS-T está completamente determinado por la localización $s \in S$ en la que se encuentra el control y la interpretación actual de todos los relojes γ (definición 6.6).

6.6.2 Transiciones

En su mayoría, los formalismos con semántica de tiempo denso orientados a modelo se basan en la suposición de ortogonalidad entre los cambios continuos y discretos, lo que simplifica de forma drástica la semántica subyacente: una ejecución de un sistema temporizado es una secuencia de pasos que alternan transiciones de tiempo continuas y transiciones de estado de control discretas:

- Transiciones de tiempo: No producen un cambio de estado de control en el modelo, sólo varían los valores de los relojes del sistema una cantidad de tiempo $\tau \in \Re_+$: $(s, \gamma) \xrightarrow{\tau} (s, \gamma + \tau)$.
- Transiciones discretas: Producen un cambio de estado de control en el modelo mediante la aceptación de un evento $a \in A$. Se trata de una transición instantánea que no incrementa los valores de los relojes, sólo eventualmente reiniciando un subconjunto λ de ellos: $(s, \gamma) \xrightarrow{a} (s', \gamma_\lambda)$.

Aunque conservando la misma naturaleza ortogonal (entre transiciones discretas y temporales) y, como consecuencia de la naturaleza multivalorada de un modelo MUS-T, las transiciones en el grafo semántico serán transiciones discretas o temporales con una condición de especificación: o bien posible (denotadas por $\rightarrow$), o bien subespecificada (denotadas por $\rightsquigarrow$), o bien no posible (denotadas por $\nrightarrow$). El conjunto de transiciones, $\mathcal{T} \subseteq (S \times \Re_+^{\#\mathcal{C}}) \times (A \cup \Re_+) \times \mathcal{L}_3 \times (S \times \Re_+^{\#\mathcal{C}})$,

multivaloradas con una condición de especificación $\mathcal{CE}(t) \in \mathcal{L}_3 = \{0, \frac{1}{2}, 1\}$, serán las transiciones discretas (etiquetadas en A) definidas explícitamente por T y CE, y las transiciones temporales (etiquetadas en $\Re_+^{\#\mathcal{C}}$) definidas implícitamente por I_a e I_p.

Dado $s \in S$ y $\gamma \in \Re_+^{\#\mathcal{C}}$:

- Transiciones discretas $t \in \mathcal{T}$, con etiqueta $a \in A$:
  - Posible (con $\mathcal{CE}(t) = 1$): Para $(s, < a, g_p(s,a), \lambda >, s\,') \in T$, existe una transición discreta posible $t = (s, \gamma) \xrightarrow{a} (s\,', \gamma_\lambda)$, sii $g_p(s,a)(\gamma) = 1$.
  - Subespecificada ($\mathcal{CE}(t) = \frac{1}{2}$): Para $(s, < a, g_s^P(s,a), \lambda >, s\,') \in T$, existe una transición discreta subespecificada $t = (s, \gamma) \overset{a}{\rightsquigarrow} (s\,', \gamma_\lambda)$, sii $g_s^P(s,a)(\gamma) = 1$. Alternativamente, habrá una transición discreta totalmente subespecificada $t = (s, \gamma) \overset{a}{\rightsquigarrow} (s_s, \gamma)$ sii $g_s^T(s,a)(\gamma) = 1$.
  - No transición ($\mathcal{CE}(t) = 0$): Para $(s, < a, g_{np}(s,a), \{\} >, s_{np}) \in T$, no existe una a-transición discreta $(s, \gamma) \overset{a}{\nrightarrow}$ (denotando $t = (s, \gamma) \overset{a}{\nrightarrow} (s_{np}, \gamma)$) sii $g_{np}(s,a)(\gamma) = 1$.
- Transiciones temporales $t \in \mathcal{T}$, con etiqueta $\tau \in \Re_+^{\#\mathcal{C}}$:
  - Posible (con $\mathcal{CE}(t) = 1$): Existe una transición temporal posible $t = (s, \gamma) \xrightarrow{\tau} (s, \gamma + \tau)$ sii:
    - $\forall\ 0 \leq \tau\,' \leq \tau, I_a(s)(\gamma + \tau\,') = 1$.
  - Subespecificada (con $\mathcal{CE}(t) = \frac{1}{2}$): Existe una transición temporal subespecificada $t = (s, \gamma) \overset{\tau}{\rightsquigarrow} (s, \gamma + \tau)$ sii:
    - $\forall\ 0 \leq \tau\,' \leq \tau,\ I_p(s)(\gamma + \tau\,') = 0$.
    - $\exists\ 0 \leq \tau\,' \leq \tau,\ I_s(s)(\gamma + \tau\,') = 1$.
  - No transición (con $\mathcal{CE}(t) = 0$): Existe una transición temporal no posible $(s, \gamma) \overset{\tau}{\nrightarrow}$ (por $t = (s, \gamma) \xrightarrow{\tau} (s_{np}, \gamma + \tau)$) sii:
    - $\exists\ 0 \leq \tau\,' \leq \tau,\ I_p(s)(\gamma + \tau\,') = 1$.

6.6.3 Semántica operacional

Al igual que en el modelo no temporizado MUS, la operación de una especificación MUS-T se modelará mediante un grafo de estados que representa su evolución temporal cualitativa (con las relaciones de orden impuestas en los estados del sistema) y, adicionalmente, su evolución temporal cuantitativa (con los predicados de tiempo asociados a las versiones posible, no posible y subespecificadas de las acciones en los estados de control).

Al incorporar la medida cuantitativa del tiempo en el modelo, ampliamos la especificación y subespecificación del dominio de las acciones (espacial) al dominio de acciones temporizadas (temporal). Si en el caso no temporizado teníamos una subespecificación en el dominio del universo de acciones, ahora además tenemos una subespecificación en el dominio temporal. Una acción podrá ser posible, no posible, o subespecificada en un estado dependiendo del instante temporal del que se trate, determinado mediante las interpretaciones de los relojes.

La semántica ramificada se define asociando a cada modelo MUS-T $\mathcal{M}$ un grafo etiquetado denso, $\mathcal{S}_{\mathcal{M}} = \langle\{(s_0, \gamma^0)\}, \mathcal{ST} = \{S \times \Re_+^{\#C}\}, \mathcal{T}, A, \mathcal{C}\rangle$, sobre el conjunto de verdad $\mathcal{L}_3$, siendo:

- $\{(s_0, \gamma^0)\}$: El conjunto de estados iniciales; cada estado inicial (s_0, γ^0) se compone de una localización inicial $s_0 \in S_0$ y la interpretación inicial γ^0 tal que $\gamma^0(x) = 0 \ \forall \ x \ \in \mathcal{C}$.
- $(S \times \Re_+^{\#\mathcal{C}})$: El conjunto de estados (s, γ) definidos en la sección 6.6.1.
- $\mathcal{T}$: Conjunto de transiciones, $(S \times \Re_+^{\#\mathcal{C}}) \times (A \cup \Re_+) \times \mathcal{L}_3 \times (S \times \Re_+^{\#\mathcal{C}})$, definidas en la sección 6.6.2, que representan la evolución del sistema mediante pasos discretos (A) que llevan de un estado de control a otro, y pasos temporales $(\Re_+)$ que incrementan el tiempo de forma síncrona en todos los relojes definidos en el modelo.

El grafo semántico definido es el usual en los modelos con tiempo, simplemente añadiendo un valor de verdad a las transiciones temporales y de control.

Es posible definir la semántica lineal de un modelo MUS-T a partir del grafo semántico, de la misma forma que en (Tripakis, 1998). La semántica lineal asocia a un modelo MUS-T un conjunto de ejecuciones, que no son más que secuencias finitas o infinitas:

$$(s_0, \gamma^0) \overset{\tau_0}{\multimap} (s_0, \gamma^0 + \tau_0) \overset{a_0}{\multimap} (s_1, \gamma_1 = (\gamma^0 + \tau_0)_{\lambda_0}) \overset{\tau_1}{\multimap} (s_1, \gamma_1 + \tau_1) \overset{a_1}{\multimap} \cdots$$

Es decir, una ejecución será un camino en el grafo semántico del modelo donde se alternan transiciones discretas y transiciones temporales con distintas condiciones de especificación ($\multimap \in \{\rightarrow, \rightsquigarrow, \nrightarrow\}$), de forma que:

- Transiciones temporales consecutivas se concatenan con absorción por $\nrightarrow$ y absorción de posibles por $\rightsquigarrow$, es decir, la concatenación de transiciones es equivalente al producto de sus condiciones de especificación $(\wedge)$. Es decir, para toda subtraza:

  $$(s, \gamma) \overset{(\tau_1, \mathcal{CE}_1)}{\multimap} (s, \gamma + \tau_1) \overset{(\tau_2, \mathcal{CE}_2)}{\multimap} (s, \gamma_1 + \tau_2) \cdots \overset{(\tau_n, \mathcal{CE}_n)}{\multimap} (s, \gamma_{n-1} + \tau_n)$$

  con transiciones sólo temporales, el cierre de dicha subtraza será:

  $$(s, \gamma) \overset{(\tau, \mathcal{CE})}{\multimap} (s, \gamma + \tau) \text{ con } \tau = \sum_{i=1 \cdots n} \tau_i \text{ y } \mathcal{CE} = \bigwedge_{i=1 \cdots n} \mathcal{CE}_i$$

- Transiciones discretas consecutivas se separan mediante un paso temporal $\tau = 0$ con condición de especificación 1. Es decir, para toda subtraza:

  $$(s, \gamma) \overset{(a_1, \mathcal{CE}_1)}{\multimap} (s', \gamma' = \gamma_{\lambda_1}) \overset{(a_2, \mathcal{CE}_2)}{\multimap} (s'', \gamma'_{\lambda_2})$$

  con transiciones sólo discretas, el cierre de dicha subtraza será:

  $$(s, \gamma) \overset{(a_1, \mathcal{CE}_1)}{\multimap} (s', \gamma' = \gamma_{\lambda_1}) \overset{\tau=0, \mathcal{CE}=1}{\multimap} (s', \gamma') \overset{(a_2, \mathcal{CE}_2)}{\multimap} (s'', \gamma'_{\lambda_2})$$

Al igual que en el diseño de sistemas distribuidos con MUS se pasa de un gran estado subespecificado a etapas de modelado con menor subespecificación, en el modelo con tiempo MUS-T se partirá, para cada acción, de un conjunto de interpretaciones totalmente subespecificado para, a posteriori, en las sucesivas fases de diseño, ir especificando las interpretaciones posibles y no posibles, reduciendo así el conjunto de interpretaciones subespecificadas.

El modelo MUS-T es una extensión conservativa del modelo MUS, ya que las guardas temporales impuestas sobre las acciones de las localizaciones del modelo temporizado pueden verse como filtros que prohíben ciertos comportamientos del modelo no temporizado subyacente. Cualquier propiedad no temporizada satisfecha por el diagrama de estados temporizado es satisfecha por el diagrama de estados sin tiempo subyacente. Lo inverso no es siempre cierto.

Como ejemplo citar el caso de una acción con guarda temporal referente a un reloj del sistema x tal que $[0 \leq x < \infty]$. Mientras que un límite inferior 0 no filtra ninguno de los posibles comportamientos del caso no temporizado, el límite superior ∞, añade una condición de justicia – *justice* o *weak fairness* –, ya que la acción no puede permanecer continuamente habilitada sin ser tomada su transición.

6.6.4 Nivel de especificación de estados

Definición 6.7. **Estado Especificado.** Un estado (s, γ) es un estado especificado sii $\forall\ a \in A$ la configuración de tiempo γ se encuentra bajo la guardas especificadas (posibles o no posibles) y bajo las invariantes especificadas (de avance y de parada), es decir, $g_p(s,a)(\gamma) \vee g_{np}(s,a)(\gamma) = 1$ (especificado en control), e $I_a(s)(\gamma) \vee I_p(s)(\gamma) = 1$ (especificado en tiempo).

Definición 6.8. **Estado No Especificado.** Un estado (s, γ) es un estado no especificado sii se encuentra bajo la invariante subespecificada $I_s(s)(\gamma) = 1$ y, $\forall\ a \in A$, la configuración de tiempo γ se encuentra bajo la guarda de tiempo subespecificada $g_s(s,a)(\gamma) = 1$.

Definición 6.9. **Estado Parcialmente Especificado o Parcialmente No Especificado.** Un estado (s, γ) es un estado parcialmente especificado o parcialmente no especificado sii:

- $\exists\ a \in A$, tal que la interpretación de tiempo γ se encuentra bajo la guarda de tiempo subespecificada, es decir, $g_s(s,a)(\gamma) = 1$ (control parcialmente especificado),
- o, la interpretación de tiempo γ se encuentra bajo la invariante subespecificada, es decir, $I_s(s)(\gamma) = 1$ (tiempo parcialmente especificado).

Por extensión, podemos decir que un estado de control s es especificado (no especificado), si todas sus versiones temporales (s, γ) son estados especificados (no especificados). En caso contrario, el estado de control s será parcialmente especificado o parcialmente subespecificado.

6.6.5 Modelos MUS-T deterministas

Introducimos en esta sección los modelos MUS-T deterministas que imponen restricciones adicionales a un modelo MUS-T genérico.

DEFINICIÓN 6.10. **Modelo MUS-T determinista.** En general, un modelo determinista es aquél que posee un único estado inicial y, además, para cada estado del sistema, si determinamos el próximo evento, el estado sucesor debe estar determinado de forma única.

Aplicamos este mismo criterio determinista al modelo MUS-T. Dada una localización del modelo y las interpretaciones actuales de los relojes, si determinamos el próximo evento, así como su instante de ocurrencia, el estado siguiente debe estar determinado de forma única. Es decir, un modelo MUS-T $\mathcal{M}$ es determinista sii:

[1] Existe un único estado de control inicial ($\#S_0 = 1$).

[2] $\forall\ s \in S$, y $\forall\ a \in A$, y $G(s,a) = \{g_{p_i}(s,a), g^P_{s_j}(s,a)\}$ el conjunto de guardas, para cualquier $\gamma \in \Re_+^{\#\mathcal{C}}$, y para todo $g_i \in G(s,a)$ tal que $g_i(\gamma) = 1$, el estado de control sucesor s' es único.

DEFINICIÓN 6.11. **Modelo MUS-T de control determinista.** Un modelo MUS-T es de control determinista sii para todo estado de control s en el modelo y para toda acción a, el estado de control s' del a-sucesor de s, de existir, es idéntico para todas las posibles configuraciones (s,γ) del estado de control s.

Los modelos MUS-T de control determinista contienen un único a-sucesor por cada acción a, es decir, si bien el tiempo puede habilitar o prohibir la ocurrencia de un evento a, cuando el evento es posible el estado de control sucesor es único.

6.6.6 Modelos MUS-T no deterministas

Por contraposición a la definición 6.10, un modelo MUS-T será no determinista si $\forall\ (s,\gamma) \in \mathcal{ST}$ y $\forall\ a \in A$, pueden existir $(s',\gamma') \neq (s'',\gamma'')$, tal que $(s,\gamma) \overset{(a,\mathcal{CE})}{\multimap} (s',\gamma')$ y $(s,\gamma) \overset{(a,\mathcal{CE})}{\multimap} (s'',\gamma'')$. En base a tal definición un modelo no determinista, con alguna transición t tal que $\mathcal{CE}(t) \neq 0$, es implícitamente un modelo incompleto. Es decir $\forall\ (s,\gamma) \in \mathcal{ST}$ y $\forall\ a \in A$ tal que $(s,\gamma) \overset{(a,\mathcal{CE})}{\multimap} (s',\gamma')$, para $\mathcal{CE} \in \{1, \frac{1}{2}\}$, existe, por definición, $(s,\gamma) \overset{(a,\mathcal{CE})}{\multimap} (s_s,\gamma)$

6.7 Requisitos de avance en un sistema temporizado

En general, los sistemas reactivos se ejecutan de forma continuada, sin una terminación previsible; denominamos a esta característica de un sistema reactivo requisito de avance (*progress*).

Las reglas de evolución del grafo denso $\mathcal{S}_{\mathcal{M}}$ (sección 6.6.2) definen los posibles comportamientos temporizados legales en el modelo $\mathcal{M}$, pero no tienen en cuenta los requisitos de avance en el sistema, es decir, no hemos garantizado la progresión, ya sea discreta o temporal, del sistema.

Mientras en el caso de sistemas no temporizados, o en los que se ha abstraído el tiempo, la garantía de avance coincide con la ausencia de *deadlocks*, en los sistemas con tiempo se caracterizan dos clases de requisitos de avance, resultantes de imponer restricciones sobre el conjunto de transiciones discretas y temporales (Tripakis, 1998):

- **Avance discreto**: El número de transiciones discretas en una ejecución es no finito.
- **Avance temporal**: Análogamente, el número de pasos temporales en una ejecución es no finito, pero, además, carece de cota superior. Vemos que el requisito de avance temporal es más fuerte que su correspondiente discreto, ya que no sólo impone la existencia (infinita) de pasos temporales, sino también la divergencia de dichos pasos.

Dado que los requisitos de avance en los sistemas garantizan una mínima corrección, en el capítulo 11 se tratan dichos requisitos para el formalismo MUS-T.

6.8 Composición

En general, un sistema se modelará como un conjunto de componentes, representados por modelos MUS-T, que se ejecutan de forma concurrente y sincronizada. Utilizamos el modelo de concurrencia usual en los sistemas de transiciones con tiempo: las acciones se entrelazan mientras que el tiempo en ambos sistemas se solapa (el tiempo transcurre de forma síncrona en todos los componentes del sistema). La comunicación se modela mediante la sincronización de acciones en el sistema.

Sean dos modelos MUS-T tales que $\mathcal{M}_i = \langle S_0^i, S^i, T^i, I^i, A^i, \mathcal{C}^i \rangle$ para $i = 1, 2$ con relojes $\mathcal{C}^1$ y $\mathcal{C}^2$ disjuntos, donde las acciones de sincronización son $A_S = A^1 \cap A^2$ y las acciones independientes son $A_I = \{A^1 \cup A^2\} - A_S$. La composición paralela de $\mathcal{M}_1$ y $\mathcal{M}_2$, $\mathcal{M}_1|[A_S]|\mathcal{M}_2$, se define como el modelo MUS-T $\mathcal{M}_{1x2} = \langle S_0, S, T, I, A, \mathcal{C} \rangle$, donde:

- $A = A^1 \cup A^2 = A_I \cup A_S$.
- $\mathcal{C} = \mathcal{C}^1 \cup \mathcal{C}^2$.
- Un conjunto de estados de control $S = \{s^{1\times 2}\}$, con $s^{1\times 2} = (s^1, s^2) \in S^1 \times S^2$.
- Un conjunto de estados de control iniciales

$$S_0 = \{s^{1\times 2} \in S \mid s^1 \in S_0^1, \text{ y } s^2 \in S_0^2\}.$$

- Dos funciones etiqueta de invariantes I_a, I_p, tales que $\forall s^{1\times 2} \in S$:

- $I_a(s^{1\times 2}) = I_a^1(s^1) \wedge I_a^2(s^2)$
- $I_p(s^{1\times 2}) = I_p^1(s^1) \vee I_p^2(s^2)$

- Un conjunto de transiciones $T \subseteq S \times A \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times S$, y una función $CE : T \to \mathcal{L}_3$, tal que:
  - Si $a \in A_I$ y $a \in A^1$, para todo $t^1 \in T^1$ etiquetada con a:
    - con $CE(t^1) = 1$, y $t^1 = (s^1, < a, g_p(s^1, a), \lambda^1 >, s^{1\,\prime})$, la transición $t = ((s^1, s^2), < a, g_p(s^1, a), \lambda^1 >, (s^{1\,\prime}, s^2))$ pertenece a T, con $CE(t) = 1$.
    - con $CE(t^1) = \frac{1}{2}$, y $t^1 = (s^1, < a, g_s^P(s^1, a), \lambda^1 >, s^{1\,\prime})$, la transición $t = ((s^1, s^2), < a, g_s^P(s^1, a), \lambda^1 >, (s^{1\,\prime}, s^2))$ pertenece a T, con $CE(t) = \frac{1}{2}$.
    - con $CE(t^1) = 0$, y $t^1 = (s^1, < a, g_{np}(s^1, a), \{\} >, s_{np}^1)$, la transición $t = ((s^1, s^2), < a, g_{np}(s^1, a), \{\} >, s_{np})$ pertenece a T, con $CE(t) = 0$.
  - Análogo para $a \in A_I$ y $a \in A^2$.
  - Si $a \in A_s$, para todo $t^1 \in T^1$, $t^2 \in T^2$ etiquetadas con a:
    - Si $CE(t^1) = 1$, y $CE(t^2) = 1$, con

      $$t^1 = (s^1, < a, g_p(s^1, a), \lambda^1 >, s^{1\,\prime})$$
      $$t^2 = (s^2, < a, g_p(s^2, a), \lambda^2 >, s^{2\,\prime})$$

      la transición:

      $$t = ((s^1, s^2), < a, g_p(s^1, a) \wedge g_p(s^2, a), \lambda^1 \cup \lambda^2 >, (s^{1\,\prime}, s^{2\,\prime}))$$

      pertenece a T, con $CE(t) = 1$.
    - Si $CE(t^1) = 0$, o $CE(t^2) = 0$, con

      $$t^1 = (s^1, < a, g_{np}(s^1, a), \{\} >, s_{np}^1)$$
      $$t^2 =< s^2, < a, g_{np}(s^2, a), \{\} >, s_{np}^2)$$

      la transición

      $$t = ((s^1, s^2), < a, g_{np}(s^1, a) \vee g_{np}(s^2, a), \{\} >, s_{np})$$

      pertenece a T, con $CE(t) = 0$.
    - En cualquier otro caso

      $$\begin{cases} CE(t^1) = \frac{1}{2}, \text{ y } CE(t^2) = \frac{1}{2} \\ CE(t^1) = 1, \text{ y } CE(t^2) = \frac{1}{2} \\ CE(t^1) = \frac{1}{2}, \text{ y } CE(t^2) = 1 \end{cases}$$

      con $g_x, g_y \in \{g_p, g_s^P\}$ y

      $$t^1 = (s^1 < a, g_x(s^1, a), \lambda^1 >, s^{1\,\prime})$$
      $$t^2 = (s^2, < a, g_y(s^2, a), \lambda^2 >, s^{2\,\prime})$$

      la transición

      $$t = ((s^1, s^2), < a, g_x(s^1, a) \wedge g_y(s^2, a), \lambda^1 \cup \lambda^2 >, (s^{1\,\prime}, s^{2\,\prime}))$$

      pertenece a T, con $CE(t) = 1/2$.

Aunque la anterior es la definición de composición más cercana a la usual, en (Sifakis y Yovine, 1996) se argumenta sobre la necesidad de la definición de diferentes tipos de composición entre modelos temporizados, reflejando el distinto acoplamiento entre componentes en un modelo real: fuertemente o débilmente acoplados. Aplicado a nuestro modelo, la composición "flexible" de menor acoplamiento seguiría las siguientes reglas:

- $I_a(s^{1\times 2}) = I_a^1(s^1) \vee I_a^2(s^2)$
- $I_p(s^{1\times 2}) = I_p^1(s^1) \wedge I_p^2(s^2)$

Es decir, el sistema global puede avanzar en tiempo si alguno de los dos componentes puede avanzar o lo que es lo mismo, un componente puede esperar al otro aun incumpliendo su invariante de avance. De la misma forma, un componente sólo detiene su avance temporal si el otro componente también lo hace, o lo que es lo mismo, un componente puede "esperar" aun incumpliendo su invariante de parada, siempre que el otro componente pueda avanzar.

Igualmente se puede flexibilizar la parte discreta de la composición:

- Para $CE(t^1) = 0$ y $CE(t^2) = 0$, relajando la guarda de la transición compuesta a $g_{np} = g_{np}^1 \wedge g_{np}^2$. Una acción es no posible cuando lo es en los dos componentes.
- Para $CE(t^1) = \frac{1}{2}$ y $CE(t^2) = 1$, o $CE(t^1) = 1$, y $CE(t^2) = \frac{1}{2}$, definiendo la condición de especificación de la transición compuesta como posible $CE(t) = 1$, en vez de subespecificada.

6.9 Traducción MUS-T a *Timed Automaton*

Con el objetivo de comparar la complejidad de los métodos de decisión sobre modelos MUS-T se detalla, en esta sección, la traducción de MUS-T a *Timed Automaton*. Sea una modelo MUS-T $\mathcal{M} = \langle S_0, S, T, I, A, \mathcal{C}\rangle$, la traducción a un *Timed Automaton* pasa por encontrar una estructura, sin comportamientos no posibles ni comportamientos subespecificados y que, sin embargo, represente toda la información en un modelo MUS-T. Definimos:

- Para todo $a \in A$, las acciones a^P y a^S, como sus versiones posible y subespecificada.
- Para todo $s \in S$, los estados de control s^A y s^S, las versiones cuya invariantes son $I_a(s)$ y $I_s(s)$ respectivamente.
- Para todo par (s^A, s^S), definimos dos transiciones discretas ficticias T^A y T^S, tales que $(s^A, < T^S, I_s(s), \{\} >, s^S)$, $(s^S, < T^A, I_a(s), \{\} >, s^A)$ y, por tanto, permiten modelar el cambio de la condición de especificación bajo la que el tiempo progresa.

En base a las definiciones anteriores, el *Timed Automaton* correspondiente al modelo $\mathcal{M}$, $TA_{\mathcal{M}}$, será una estructura:

$$\langle S_0^{TA}, S^{TA}, T^{TA}, I^{TA}, A^{TA}, \mathcal{C}\rangle$$

donde $T^{TA} \subseteq S^{TA} \times A^{TA} \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times S^{TA}$, e I^{TA} es la función de etiqueta invariante de los estados de control S^{TA}, tal que:

- Para todo $s_0 \in S^0$, $s_0^A \in S_0^{TA}$ y $s_0^S \in S_0^{TA}$
- Para todo $s \in S$, s^A y $s^S \in S^{TA}$
- Para todo $a \in A$, $a^P \in A^{TA}$ y $a^S \in A^{TA}$
- $s_s \in S^{TA}$ con una invariante $I^{TA}(s_s) = true$.
- Para todo $(s, < a, g_p(s,a), \lambda >, s\,') \in T$, sea el conjunto:

$$\begin{aligned} T(s, < a, g_p(s,a), \lambda >, s\,') = \{&(s^A, < a^P, g_p(s,a), \lambda >, s\,'^A),\\ &(s^A, < a^P, g_p(s,a), \lambda >, s\,'^S),\\ &(s^S, < a^P, g_p(s,a), \lambda >, s\,'^A),\\ &(s^S, < a^P, g_p(s,a), \lambda >, s\,'^S)\} \end{aligned}$$

  entonces $T(s, < a, g_p(s,a), \lambda >, s\,') \in T^{TA}$.

- Para todo $(s, < a, g_s^P(s,a), \lambda >, s\,') \in T$, sea el conjunto:

$$\begin{aligned} T(s, < a, g_s^P(s,a), \lambda >, s\,') = \{&(s^A, < a^S, g_s^P(s,a), \lambda >, s\,'^A),\\ &(s^A, < a^S, g_s^P(s,a), \lambda >, s\,'^S),\\ &(s^S, < a^S, g_s^P(s,a), \lambda >, s\,'^A),\\ &(s^S, < a^S, g_s^P(s,a), \lambda >, s\,'^S)\} \end{aligned}$$

  entonces $T(s, < a, g_s^P(s,a), \lambda >, s\,') \in T^{TA}$.

- Para todo $s \in S$ y $a \in A$, sea el conjunto:

$$\begin{aligned} T(s, < a, g_s^T(s,a), \lambda >, s_s) = \{&(s^A, < a^S, g_s^T(s,a), \lambda >, s_s),\\ &(s^S, < a^S, g_s^T(s,a), \lambda >, s_s)\} \end{aligned}$$

  entonces $T(s, < a, g_s^T(s,a), \lambda >, s_s) \in T^{TA}$.

- Para todo par (s^A, s^S):

$$\{(s^A, < T^S, I_s(s), \{\} >, s^S), (s^S, < T^A, I_a(s), \{\} >, s^A)\} \in T^{TA}$$

En base a la traducción anterior, podemos adelantar, que los métodos de decisión desarrollados, aunque no más complejos que sus correspondientes sobre *Timed Automaton*, operan sobre un espacio de estados de un tamaño mayor, resultando, por tanto, en algoritmos más costosos.

6.10 Conclusiones

Con la definición en este capítulo del modelo MUS-T, proporcionamos un formalismo de especificación orientado a modelo para sistemas de tiempo real incompletos. Tal definición habilita un modelo de proceso *software* formal, que es iterativo en diseño e incremental en funcionalidad. Si bien no recogido en los objetivos de esta tesis, el modelo proporciona la base para un diseño cooperativo con varios agentes especificadores, sólo restando la definición de las reglas que permitan construir el modelo cooperativo a partir de las vistas que cada uno de los expertos proporciona del sistema.

Retomamos, como conclusión, la discusión sobre el paradigma seleccionado para la incorporación de tiempo denso. Si bien hemos seleccionado el *Timed Automaton* como base para la extensión MUS-T, creemos del todo inmediato la extensión del modelo MUS utilizando un paradigma que utilice relojes asociados a eventos (*Event-Clock Timed Automaton*), dado que, comúnmente, las propiedades deseables en un sistema especificado como un modelo MUS relacionan contextos temporales asociados a eventos. Sin embargo, como ya hemos mencionado, dicha extensión sería menos expresiva que la proporcionada en este capítulo.

CAPÍTULO 7

SCTL-T

*En este capítulo se presentan las bases formales de **SCTL-T** (**T**imed **S**imple **C**ausal **T**emporal **L**ogic), extensión con tiempo denso de la lógica atemporizada SCTL. La extensión se ha realizado conservando las características principales de la primera: causalidad, simplicidad y naturaleza ramificada y multivalorada, e introduciendo la referencia al tiempo mediante relojes de especificación con cuantificación freeze.*

7.1 Introducción

La extensión SCTL-T se ha realizado conservando el espíritu de la lógica SCTL: causalidad, simplicidad y naturaleza ramificada y multivalorada.

La principal motivación que se encuentra detrás de la elección de una lógica **causal** en la especificación de requisitos es salvar la distancia entre el lenguaje natural, en el que se expresan los usuarios, y la especificación formal de los requisitos. La causalidad respeta tres principios fundamentales de la ingeniería de requisitos: la especificación es clara para los usuarios, clara para los desarrolladores y, por medio de causalidad formal, puede ser analizada formalmente (Moffett et al., 1996). Las propiedades en SCTL siguen el siguiente patrón causal:

$$\text{Premisa } Æ \otimes \text{ Consecuencia}$$

Una propiedad genérica como la anterior establece una condición causante (premisa), un operador temporal que determina el ámbito de aplicación de la causa ($\otimes$), un cuantificador de camino ($Æ \in \{\forall, \exists\}$), y una condición que es el efecto originado por dicha causa (consecuencia).

Además de causal, SCTL es una lógica **multivalorada**. El punto de partida de las lógicas multivaloradas es que no "todo es cierto o falso" (principio de bivalencia). En general, las lógicas multivaloradas resultan adecuadas para el tratamiento de la información incompleta e inconsistente obtenida en la captura de requisitos.

Desafortunadamente, la investigación en este ámbito es bastante deficiente en resultados teóricos y, especialmente, en herramientas prácticas. En concreto, los valores de verdad en SCTL se refieren a la característica incompleta en la captura de requisitos. Los valores lógicos fuera del principio de bivalencia provienen de la interpretación de la lógica SCTL sobre modelos MUS (con subespecificación), y de la diferencia entre la implicación y la causalidad. En esta línea, la relación de satisfacción de una fórmula SCTL sobre un estado de un modelo MUS se define sobre un conjunto finito de valores de verdad de cardinalidad 6.

Con el objetivo de **simplicidad**, en la definición de SCTL se han utilizado únicamente tres operadores temporales que marcan el ordenamiento temporal de los estados sobre los que se interpreta la fórmula: *antes*, *después* y *a la vez*. Así, el operador *antes* (*después*) permite expresar propiedades que relacionan proposiciones sobre un estado del sistema y sus estados predecesores (sucesores), y el operador *a la vez* permite expresar propiedades que relacionan proposiciones sobre un mismo estado.

SCTL-T (*Timed Simple Causal Temporal Logic*) es una lógica de tiempo real densa, con semántica ramificada, y que puede catalogarse como de tipo *explicit clock*. Para expresar restricciones de tiempo, incorpora relojes de especificación fijados por cuantificación *freeze*, y predicados de tiempo sobre dichos relojes (ver sección 2.2.1). Como ya hemos mencionado, las lógicas *explicit clock* permiten expresar propiedades cuantitativas entre contextos no necesariamente adyacentes.

Con la extensión propuesta, las propiedades cuantitativas en SCTL-T presentarán un patrón causal del tipo:

$$x.\ \text{Premisa Æ} \otimes \text{Consecuencia } (f(x))$$

donde, además de la premisa, la consecuencia y el operador temporal, proporcionamos una medida de la distancia temporal entre la premisa y la consecuencia. La consecuencia ha de satisfacerse en un estado cuya distancia temporal, con respecto al estado en el que se aplica la premisa, cumpla la restricción de tiempo $f(x)$.

Al igual que SCTL, SCTL-T es una lógica temporal causal multivalorada de tiempo ramificado, conservando los mismos grados de satisfacción definidos en SCTL.

7.2 Valores de verdad en SCTL

En los sistemas clásicos, que utilizan lógicas binarias, el grado de satisfacción de una propiedad o requisito se reduce a un valor booleano — *verdadero* o *falso*. La lógica SCTL, al ser una lógica multivalorada donde se ha introducido el concepto de subespecificación, permite un mayor refinamiento en estos resultados, ampliando el abanico de posibilidades a seis, donde cuatro de estos valores son resultados intermedios entre la plena satisfacción y la no satisfacción de la propiedad. De esta forma se amplía la expresividad de la lógica y se permite la detección de errores en la versión actual del sistema e, incluso, errores potenciales, es decir, errores que se producirían con una determinada evolución en su diseño.

Desde el momento que una propiedad no ha sido especificada en la versión actual del sistema, su grado de satisfacción no debería ser *falso*, como ocurre en

las lógicas binarias, sino que deberá ser *subespecificado* ya que, dependiendo de cómo evolucione el diseño del sistema, podrá converger en un futuro hacia un grado de satisfacción *verdadero* o *falso*.

Dentro de estas propiedades subespecificadas, es posible diferenciar, a su vez, varios grados de subespecificación, ya que puede que la propiedad se encuentre parcialmente especificada, es decir, algunos elementos que la componen sí están plenamente especificados y otros no. El englobar toda esta casuística bajo un mismo criterio implicaría una reducción de la expresividad de la lógica y, por esta razón, se ha definido un conjunto de valores de verdad $\mathcal{L}_6 = \{0, \frac{1}{4}, \widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$. $\mathcal{L}_6$, además de los valores *verdadero* (1) y *falso* (0), incluye 4 valores intermedios para reflejar las siguientes situaciones:

[$\frac{1}{4}$] Requisitos potencialmente *falsos*, o requisitos que no pueden llegar a ser *verdaderos*. Su grado actual de satisfacción es subespecificado, pero si se perdiese dicha subespecificación, evolucionaría hacia un grado de satisfacción *falso*.

[$\frac{3}{4}$] Requisitos potencialmente *verdaderos*, o requisitos que no pueden llegar a ser *falsos*. Similares a los anteriores, salvo que en este caso al perder subespecificación se evolucionaría hacia un grado de satisfacción *verdadero*.

[$\widehat{\frac{1}{2}}$] Requisitos que no pueden llegar a ser *falsos* ni *verdaderos*. Estos requisitos no pueden ser estudiados por no satisfacerse su premisa o condición de aplicabilidad. Este grado de satisfacción recibe el nombre de *no aplicable*.

[$\frac{1}{2}$] Requisitos que podrían llegar a ser *falsos* o *verdaderos* ya que están totalmente subespecificados y no se tiene ninguna información a propósito de su posterior evolución.

7.3 Semántica con tiempo de los operadores temporales

Los operadores temporales en SCTL definen el patrón de ordenamiento temporal entre pares de estados en los que se aplica la premisa y la consecuencia. Dichos operadores se basan en una semántica orientada a estado o, más bien, a estados de control, marcando:

- Operador *a la vez* ($\Rightarrow$): Las proposiciones premisa y consecuencia se aplican a un mismo estado, aquél en que se formula la propiedad SCTL.
- Operador *después* ($\Rightarrow \bigcirc$): La proposición premisa se refiere al estado en el que se aplica la fórmula SCTL, y la proposición consecuencia a sus estados sucesores.
- Operador *antes* ($\Rightarrow \odot$): La proposición premisa se refiere al estado en el que se aplica la fórmula SCTL, y la proposición consecuencia a sus estados predecesores.

Se trata, pues, de operadores temporales que imponen un patrón en el orden de los estados del sistema. En la definición de la extensión temporizada de SCTL se

intentó mantener la misma semántica, entendiendo ahora por estados los distintos estados de control definidos en el modelo MUS-T (ver sección 6.5).

En consecuencia, redefinimos los operadores temporales SCTL para que impongan un patrón de ordenamiento en los estados de control o localizaciones del modelo MUS-T. Recordamos que, si bien en un modelo MUS los estados de control se instancian de forma única en cada ejecución, en la extensión con tiempo MUS-T, y como se sigue de la semántica definida en la sección 6.6, cada estado de control tendrá un número infinito de representantes en función del instante temporal (interpretaciones actuales de los relojes). Los operadores temporales en la versión temporizada relacionarán los contextos temporales siguientes:

- Operador *después*: El operador *después* relaciona el estado del sistema donde se aplica con estados accesibles mediante una transición discreta.
- Operador *antes*: El operador *antes* relaciona el estado del sistema donde se aplica con estados que pueden acceder a dicho estado mediante una transición discreta.
- Operador *a la vez*: El operador *a la vez* relaciona el estado del sistema donde se aplica con otras instancias de la misma localización.

Cada estado de control se corresponderá con un conjunto denso de estados que comparten dicho estado de control. Desde la perspectiva del control, vemos el sistema de tiempo real, representado por su modelo MUS-T, como operando en "modos" y dirigido por eventos. El sistema cambia de un modo a otro cuando algún evento habilitado dispara su transición. Si bien el sistema reside en el mismo modo cuando el tiempo progresa (aunque en un estado diferente), es el progreso del tiempo el responsable de la habilitación de los eventos que suponen el cambio de modo. Dicho evento externo puede ser una señal, el vencimiento de una temporización o el cambio de alguna hipotética variable de estado.

7.3.1 El operador temporal *a la vez*

En SCTL, el operador temporal *a la vez* ($\Rightarrow$) definido por:

$$\phi_{SCTL} := P(\text{remisa}) \ \Rightarrow C(\text{onsecuencia})$$

permite expresar propiedades causales en las que los estados de aplicación de la premisa P y de la consecuencia C son el mismo, aquél en el que se aplica la fórmula ϕ_{SCTL}. La verificación de una propiedad de este tipo en un estado s de un modelo MUS conlleva la verificación de P y C en dicho estado.

A la hora de realizar la extensión temporizada del operador *a la vez*, es necesario preguntarse si dicho operador debe suponer la verificación de la premisa y de la consecuencia en el mismo estado o, por contra, en el mismo estado de control. Aunque sin duda las dos interpretaciones son válidas, hemos considerado que la segunda es más expresiva y coherente con la interpretación que daremos a los operadores *antes* y *después*.

Renombramos, por coherencia, el operador *a la vez* a *actual*, siendo éste un operador que permite relacionar contextos temporales diferentes en una misma

localización del modelo, aportando las versiones futura ($\Rightarrow_+$), pasada ($\Rightarrow_-$) y presente ($\Rightarrow$):

$$\begin{aligned} \Rightarrow_+ &: < s, \gamma > \xrightarrow{\tau} < s, \gamma' > \\ \Rightarrow_- &: < s, \gamma' > \xrightarrow{\tau} < s, \gamma > \\ \Rightarrow &: < s, \gamma > \end{aligned}$$

relacionando el contexto premisa $< s, \gamma >$ con el contexto consecuencia $< s, \gamma' >$ (donde $\gamma' = \gamma$ si $\tau = 0$), o contextos temporales premisa y consecuencia idénticos para el caso del operador $\Rightarrow$.

Las versiones pasada y futura admiten cuantificación universal y existencial; por contra, la versión presente es insensible al tipo de cuantificación, siendo ésta redundante.

En las figuras 7.1 y 7.2 se representa la noción del operador *a la vez* temporizado en sus dos versiones sobre un grafo semántico MUS-T. Se representan como ●, los estados que marca el operador temporal; y como ❍, los que quedan fuera del alcance del operador temporal. Se trata de una interpretación simétrica con respecto a pasado-futuro que también seguiremos para la definición de los operadores *antes* y *después* temporizados.

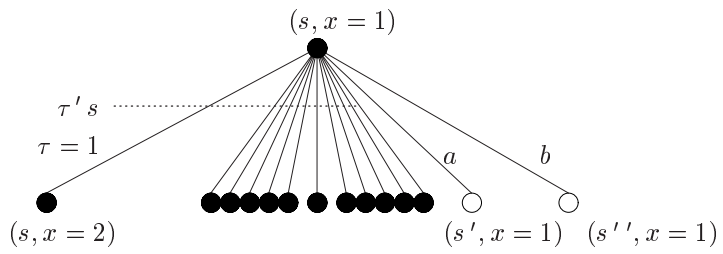


Figura 7.1. Operador *a la vez* sobre el estado $(s, x = 1)$: versión futuro

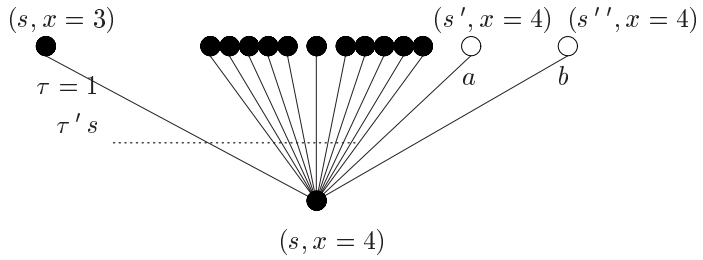


Figura 7.2. Operador *a la vez* sobre el estado $(s, x = 4)$: versión pasado

Ejemplo 7.1. Con esta semántica es posible expresar:

- Simultaneidad:

$$a \Rightarrow b$$

permite especificar que dos acciones a y b se encuentran habilitadas en el mismo instante temporal.

- Temporizador:

$$y.a \quad \forall\Rightarrow_{+} \quad ((y > 3) \Rightarrow b)$$

permite especificar un temporizador: 3 unidades de tiempo después de que una acción a esté habilitada, estará habilitada, en el mismo estado de control, la acción b. No obstante, la definición completa del temporizador b conllevaría la imposición de requisitos de urgencia en el estado.

- Urgencia:

$$\emptyset \quad \forall\Rightarrow_{+} \quad ((x < 3) \Rightarrow true) \wedge ((x \geq 3) \Rightarrow false)$$

donde $\emptyset$ denota la ausencia de proposición y x es un reloj definido en un modelo MUS-T sobre el que se interpreta la fórmula. La fórmula permite especificar invariantes de avance y de parada en un estado.

□

7.3.2 Los operadores temporales *antes* y *después* temporizados

El operador temporal *después* ($\bigcirc$) de la lógica SCTL es un operador temporal común a muchas lógicas temporales lineales y ramificadas sin noción de tiempo (CTL, PTL). En cuanto a su extensión para tiempo real, encontramos dos interpretaciones en las lógicas temporales cuantitativas de tiempo denso:

- Dichas lógicas se interpretan sobre dominios temporales densos y, por tanto, el número de estados no es sólo infinito sino que, además, a partir de un estado cualquiera no se puede decir que exista un único estado sucesor. En consecuencia, algunas lógicas (TCTL, RTCTL, XCTL) optan por no utilizar el operador *después* definiendo su sintaxis temporal a partir del operador *until* ($\mathcal{U}$).

- Una segunda línea de interpretación, como la utilizada en T_μ, considera el operador *después* como un operador que incluye una única transición discreta. De esta forma se redefine la semántica del operador $\bigcirc$ atemporizado, que especifica una única transición en el sistema, a una semántica en dos pasos o transiciones: un paso temporal (considerado ausente si es del tipo $\tau = 0$) y una transición discreta de una localización a otra del modelo. El nuevo operador $\bigcirc$ puede verse, desde el punto de vista de las lógicas que utilizan el operador $\mathcal{U}$, como un operador *until* de un solo paso. Por otro lado, L_ν (*Timed Modal Logic* (Laroussinie et al., 1995)) distingue explícitamente las transiciones discretas y temporales, incluyendo operadores de evolución temporal ($\exists\phi$, $\forall\phi$), y operadores de evolución discreta: $< a > \phi$ con cuantificación existencial, y $[a]\phi$ con cuantificación universal sobre transiciones a. Esta segunda interpretación, aunque más fina, respeta las diferentes modalidades de comportamiento en el autómata temporizado, lo que permite la construcción de fórmulas características para diferentes tipos de equivalencias y preórdenes.

Para la extensión SCTL-T optamos por la interpretación más fina que se corresponde con la encontrada en L_ν, dado que otras interpretaciones pueden ser alcanzadas con la definición de macros de operadores y la aplicación de reglas de reescritura.

- El operador $\Rightarrow \bigcirc$, aplicado sobre un estado (s, γ), nos permite establecer una relación causal entre estados (s, γ) y (s', γ'), que respondan a un comportamiento operacional MUS-T con el siguiente patrón:

$$(s, \gamma) \xrightarrow{a} (s', \gamma') \text{ con } \gamma' = \gamma_\lambda$$

- Análogamente, el operador $\Rightarrow \bigodot$ aplicado sobre un estado (s, γ):

$$(s', \gamma') \xrightarrow{a} (s, \gamma) \text{ con } \gamma = \gamma'_\lambda$$

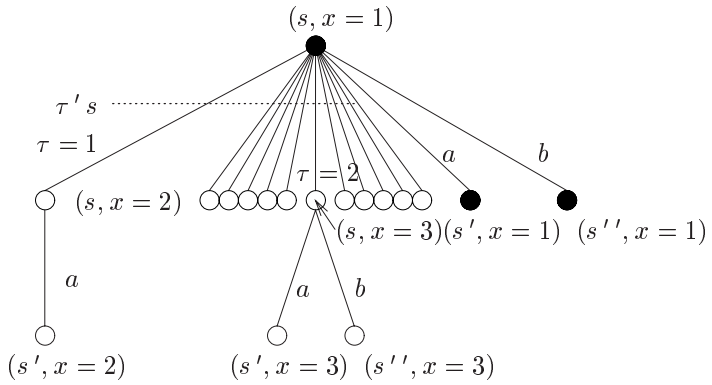


Figura 7.3. Operador *después* temporizado aplicado al estado $(s, x = 1)$

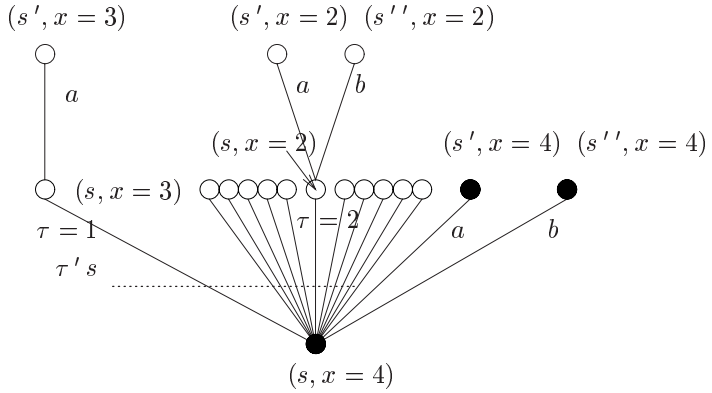


Figura 7.4. Operador *antes* temporizado aplicado al estado $(s, x = 4)$

En las figuras 7.3 y 7.4 se representa la interpretación de los operadores temporales *antes* y *después* temporizados sobre un grafo semántico MUS-T. Igualmente, los estados que marca el operador temporal se representan como ●, y los que quedan fuera del alcance del operador temporal como ❍. Como se sigue de dichas

figuras, se trata de una interpretación del pasado totalmente simétrica a la interpretación de futuro, es decir: el pasado es tan ramificado e infinito [1] como el futuro (ver sección 7.3.4).

Operadores diferidos

Dado que una de las principales ventajas de una lógica temporal, además de su expresividad, es su calidad de concisión, es del todo conveniente la definición de macro-operadores a partir de los operadores básicos. Como mostraremos en la sección 7.10, una amplia gama de operadores usuales en las lógicas ramificadas se pueden expresar en SCTL-T. Sin embargo, de entre todos los posibles macro-operadores, resultan de especial interés aquellas combinaciones de operadores SCTL-T que permiten relacionar dos estados con estados de control consecutivos, por ser esta la semántica de la lógica atemporizada SCTL.

Con el objetivo de proporcionar una intuición orientada a estados de control en el modelo, sugerimos como fundamentales los operadores básicos diferidos, que relacionan un estado con estados diferidos con respecto a la transición discreta que produce el cambio de estado de control, es decir, serán la concatenación de un operador temporal de paso discreto más un operador temporal de paso continuo:

$$Æ \Rightarrow \bigcirc \circ Æ \Rightarrow_{+}$$
$$Æ \Rightarrow \odot \circ Æ \Rightarrow_{-}$$

De la misma forma, se podrían definir los macro-operadores obtenidos de las siguientes combinaciones de paso en el modelo semántico: *temporal-discreto-temporal* y *temporal-discreto*. Sin embargo, no debemos olvidar que las constantes proposicionales en SCTL-T son acciones que etiquetan las transiciones discretas del modelo MUS-T sobre el que se interpreta la fórmula. Por tanto, la aparición de una acción a en la premisa de una fórmula SCTL-T sugiere un paso discreto: el paso discreto con origen en el estado en el que se evalúa la fórmula y etiquetado con la acción a. Los operadores diferidos, tal como se definen a continuación, respetan, por tanto, la intuición de las proposiciones en la premisa de la fórmula.

Definición 7.1. **Operadores diferidos.** Los operadores básicos diferidos se obtienen concatenando un patrón de evolución discreta ($\bigcirc$, $\odot$) y un patrón de evolución temporal ($Æ \Rightarrow_{+|-}$), obteniéndose los operadores recogidos en la tabla 7.1.

	$\forall\Rightarrow\bigcirc$	$\exists\Rightarrow\bigcirc$
$\forall\Rightarrow_{+}$	$\forall\Rightarrow\overset{\forall\rightarrow}{\bigcirc}$	$\exists\Rightarrow\overset{\forall\rightarrow}{\bigcirc}$
$\exists\Rightarrow_{+}$	$\forall\Rightarrow\overset{\exists\rightarrow}{\bigcirc}$	$\exists\Rightarrow\overset{\exists\rightarrow}{\bigcirc}$

	$\forall\Rightarrow\odot$	$\exists\Rightarrow\odot$
$\forall\Rightarrow_{-}$	$\forall\Rightarrow\overset{\forall\rightarrow}{\odot}$	$\exists\Rightarrow\overset{\forall\rightarrow}{\odot}$
$\exists\Rightarrow_{-}$	$\forall\Rightarrow\overset{\exists\rightarrow}{\odot}$	$\exists\Rightarrow\overset{\exists\rightarrow}{\odot}$

Tabla 7.1. Operadores diferidos

Como veremos en el capítulo 9, los operadores diferidos proporcionan una forma más concisa de expresar las reglas de síntesis en la metodología propuesta.

[1] introduciendo infinitos pasos temporales $\tau = 0$

7.3.3 Cuantificación

Si bien los operadores temporales marcan el patrón de ordenamiento en los estados premisa y consecuencia, la cuantificación delimita el "nivel" de satisfacción exigido sobre los estados consecuencia. Incluimos la cuantificación en SCTL-T como la cuantificación de camino usual en las lógicas ramificadas:

- Cuantificación $\exists$: Responde en la lógica ramificada tradicional (con bivalencia) a la existencia de un camino que satisfaga la fórmula cuantificada: propiedad potencial. Aunque la intuición de la cuantificación sigue siendo la misma en una lógica multivalorada, la cuantificación $\exists$ se identifica con el máximo grado de satisfacción de la fórmula en el dominio de los caminos.
- Cuantificación $\forall$: Responde en la lógica ramificada tradicional (con bivalencia) a la satisfacción de la fórmula cuantificada en todos los caminos: propiedad inevitable. En una lógica multivalorada se identifica con el mínimo grado de satisfacción de la fórmula en el dominio de los caminos.

El dominio de la cuantificación es el conjunto de estados consecuencia (de aplicabilidad) seleccionados por el operador temporal. En lo referente a este punto, recordamos que, para el caso del operador *después*, se realiza una filtración causal previa a la cuantificación de camino. La filtración causal para el operador *después*, introducida en SCTL, restringe los estados sucesores sobre los que se aplica la consecuencia a aquellos alcanzados mediante una transición discreta etiquetada con una acción de aplicabilidad. Se definen las acciones de aplicabilidad de una fórmula causal, con operador *después*, como el conjunto de acciones no negadas incluidas en la parte de la premisa que refiere al estado en que se aplica la fórmula.

Ejemplo 7.2. En la figura 7.5 se representa una filtración causal ejemplo en la que el operador *después* incluye a en su premisa:

$$(a \ \exists\Rightarrow_{_} \ b) \ \forall\Rightarrow\overset{\forall\rightarrow}{\bigcirc} \ a \wedge c$$

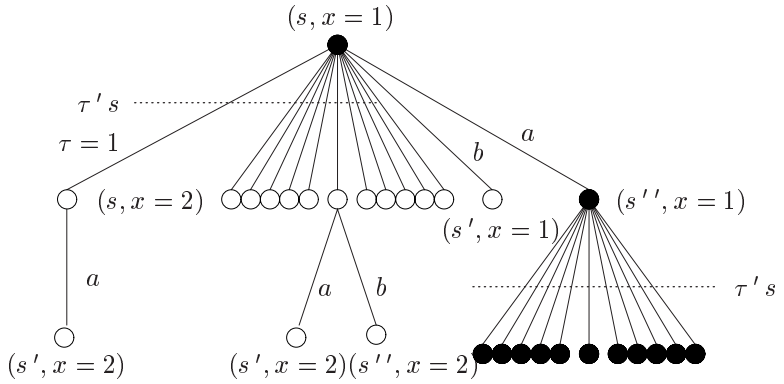


Figura 7.5. Operador *después* temporizado con premisa a aplicado al estado $(s, x = 1)$

□

Los cuantificadores de camino Æ operan sobre los operadores temporales básicos en la lógica. Para un operador temporal $\otimes$ aplicado a un estado s, y $\{s_a\}$ el conjunto de estados de aplicabilidad de dicho operador, se definen las versiones universal (existencial) de $\otimes$ como la disyunción (conjunción) del valor de la consecuencia en cada uno de los estados de aplicabilidad s_a modificada por la condición de accesibilidad (c_e) del estado s al estado s_a. Dicha condición de accesibilidad no es más que la condición de especificación asociada a la transición.

Así, para el caso del cuantificador $\exists$:

$$\bigvee_{\{(s_a,c_e)\}} c_e \wedge \vdash (\phi, s_a)$$

Por tanto, una fórmula del tipo $\exists \otimes \phi$ se satisface (1) en un estado s, sii existe una transición con condición de especificación $c_e = 1$ a un estado de aplicabilidad s_a, en el cual la fórmula ϕ se satisface. Inversamente, la fórmula no se satisface (0), si para todos los estados de aplicabilidad, ϕ no se satisface. Si la fórmula no se satisface, no se satisfará aunque se modifiquen las condiciones de especificación de las transiciones a los estados de aplicabilidad.

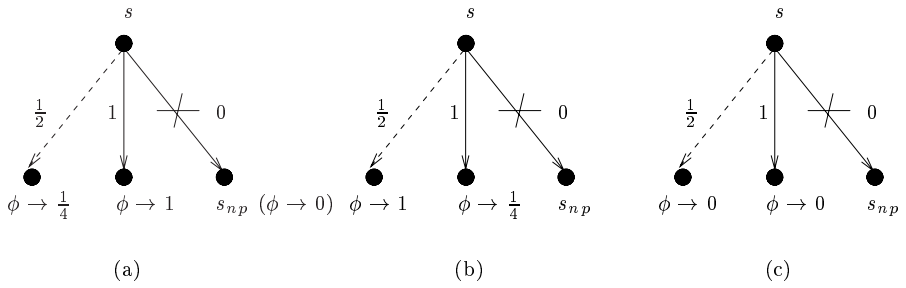


Figura 7.6. Ejemplos de un operador temporal con cuantificación existencial

Algunos ejemplos se muestran en la figura 7.6. El valor de verdad de la cuantificación existencial será 1 en el caso 7.6(a), cualquier evolución posterior del sistema satisfará la fórmula. En el caso 7.6(b), el valor de verdad será $\frac{1}{2}$, en una evolución posterior la fórmula podrá satisfacerse o no satisfacerse. Por último, en el caso 7.6(c), la fórmula no se satisface, ni se satisfará en ninguna evolución posterior.

Para la cuantificación universal $\forall$:

$$\bigwedge_{\{(s_a,c_e)\}} (\neg c_e) \vee \vdash (\phi, s_a)$$

Por tanto, una fórmula del tipo $\forall \otimes \phi$ se satisface (1) en un estado s, sii se satisface en todos los estados de aplicabilidad con $c_e \in \{\frac{1}{2}, 1\}$, indicando que la fórmula se satisfará en cualquier evolución futura del modelo. La fórmula no se satisface (0) si el valor de verdad de ϕ es 0 en algún estado de aplicabilidad con $c_e = 1$, indicando que toda evolución posterior del modelo producirá este mismo resultado. A diferencia de la cuantificación universal, si no existen estados de

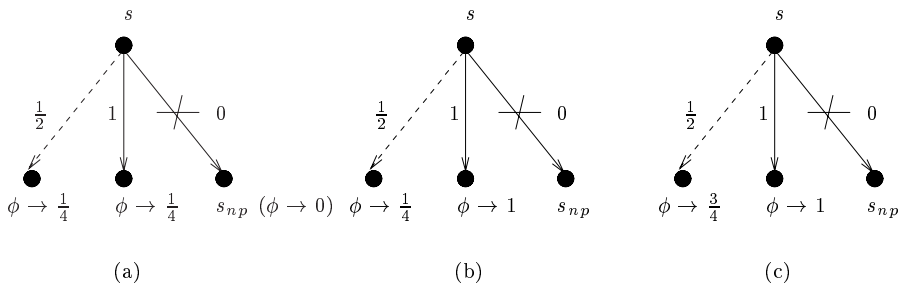


Figura 7.7. Ejemplos de un operador temporal con cuantificación universal

aplicabilidad, o lo que es lo mismo, todos los estados de aplicabilidad son no posibles, la fórmula se satisface.

Algunos ejemplos se muestran en la figura 7.7. El valor de verdad de la cuantificación universal será $\frac{1}{4}$ en el caso 7.7(a), cualquier evolución posterior del sistema no podrá satisfacer la fórmula. En el caso 7.7(b), el valor de verdad será $\frac{1}{2}$, en una evolución posterior la fórmula podrá satisfacerse o no satisfacerse. La fórmula se satisfará, por ejemplo, si la transición subespecificada pasa a ser una transición no posible; por contra, la fórmula no se satisfará si pasa a ser una transición posible. Por último, en el caso 7.7(c), el valor de verdad es $\frac{3}{4}$, la fórmula no podrá no satisfacerse en una evolución posterior. La fórmula se satisfará si la transición con condición de especificación subespecificada pasa a no posible, o si pasa a posible y su estado de aplicabilidad evoluciona a un valor de verdad 1. Asimismo, la fórmula podrá pasar a no ser aplicable, si el estado de aplicabilidad evoluciona a un valor de verdad $\widehat{\frac{1}{2}}$.

7.3.4 Operadores de pasado

Desde los trabajos en lógicas temporales sin tiempo (Lichtenstein et al., 1985; Kupferman y Pnueli, 1995), se considera aceptado que los operadores temporales de pasado permiten especificaciones más sencillas y naturales de ciertas propiedades; por ejemplo, "si se produce una salida abrupta del sistema, entonces, antes se ha actualizado el registro del sistema". Si bien la expresividad de las lógicas de tiempo lineal (con operadores de pasado) es idéntica a las lógicas de sólo futuro, en el caso de las lógicas ramificadas la expresividad es mayor para aquellas lógicas que consideran operadores de pasado ramificados, conservando la complejidad de las lógicas ramificadas de sólo futuro (Kupferman y Pnueli, 1995). Similares resultados en cuanto a expresividad y complejidad nos encontramos en las lógicas cuantitativas de tiempo discreto. Sin embargo, cuando nos movemos al campo de las lógicas de tiempo denso la situación varía ligeramente. En dichas lógicas, la incorporación de operadores de pasado aumenta la expresividad pudiendo definir un conjunto más amplio de lenguajes (Alur et al., 1999; Alur y Henzinger, 1992a).

La naturalidad de los operadores temporales de pasado fue motivo suficiente para su inclusión en SCTL; además, si a la naturalidad sumamos la expresividad adicional obtenida en tiempo denso, hemos dibujado las principales ventajas de tal

decisión. De todas formas, debemos resaltar dos puntos claves en los operadores temporales básicos en SCTL:

- Ambos operadores son operadores ramificados, es decir, para cualquier punto de una computación tenemos un conjunto de posibles sucesores y un conjunto de posibles predecesores.
- La simetría introducida en el punto anterior se rompe debido a la causalidad: sólo se realiza cuantificación dependiente de la premisa en el caso del operador temporal $\Rightarrow \bigcirc$.

7.3.5 En resumen

En las secciones anteriores hemos ilustrado informalmente las ideas que han marcado la extensión de SCTL-T, básicamente en lo que se refiere a la extensión de los operadores temporales con tiempo. Sin embargo, hemos dejado al margen puntos importantes:

- ¿Cómo influye la subespecificación a nivel de acción en el grado de satisfacción de una propiedad SCTL-T?
- ¿Cómo influye la subespecificación temporal, es decir las invariantes, en el grado de satisfacción de una propiedad SCTL-T?
- ¿Cómo se miden las propiedades cuantitativas?

Informalmente, la distintas condiciones de especificación sobre las que opera la lógica definida se traducen en un grafo denso multivalorado, modelo de la fórmula. La verificación de una fórmula SCTL-T sobre un modelo MUS-T se corresponde con la verificación sobre el grafo denso multivalorado que proporciona la semántica operacional a MUS-T. Por otro lado, la medida de distancias temporales se realiza incorporando en el modelo MUS-T y, consecuentemente, en su grafo denso multivalorado, un conjunto de relojes testigo que miden las distancias temporales a verificar.

En las siguientes secciones responderemos a estas preguntas presentando la sintaxis y semántica completa para SCTL-T. Para tal fin, se presenta a continuación el Álgebra de Incertidumbre del Punto Medio (IPM) sobre la que se define SCTL.

7.4 IPM: Álgebra de Incretidumbre del Punto Medio

7.4.1 Introducción a IPM

La fórmulas SCTL-T se interpretan sobre grafos densos multivalorados en el conjunto $\mathcal{L}_3 = \{0, \frac{1}{2}, 1\}$. El cálculo del grado de satisfacción de una fórmula SCTL-T utiliza, al igual que la lógica base SCTL, el álgebra IPM (Incertidumbre del Punto Medio) (García Duque, 2000).

$\rightarrow$		cons					
		0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
prem	0	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$
	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$
	1	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1

$\vee$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
0	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	1
1	1	1	1	1	1	1

$\wedge$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
0	0	0	0	0	0	0
$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$\widehat{\frac{1}{2}}$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
$\frac{1}{2}$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$\frac{3}{4}$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$
1	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1

$\neg$	
0	1
$\frac{1}{4}$	$\frac{3}{4}$
$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
$\frac{1}{2}$	$\frac{1}{2}$
$\frac{3}{4}$	$\frac{1}{4}$
1	0

Tabla 7.2. Operadores en IPM

El álgebra IPM se define sobre un orden total ($\{0, \frac{1}{4}, \widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$), y un conjunto de operadores lógicos ($\{\wedge, \vee, \neg\}$), formando un álgebra de De Morgan con las siguientes propiedades (ver tabla 7.2):

- Idempotencia: $a \wedge a = a$, $a \vee a = a$
- Conmutativo: $a \wedge b = b \wedge a$, $a \vee b = b \vee a$.
- Asociativo: $a \wedge (b \wedge c) = (a \wedge b) \wedge c$, $a \vee (b \vee c) = (a \vee b) \vee c$
- Absorción: $a \vee (a \wedge b) = a$, $a \wedge (a \vee b) = a$
- Distributivo:

    - $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
    - $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

- Existe un único máximo (1) y un único mínimo (0), respectivamente único elemento neutro para $\wedge$ y único elemento neutro para $\vee$.

Complemento

El complemento $\neg$ es único, con propiedades:

- Involución: $\neg\neg\ a = a$
- De Morgan: $\neg(a \wedge b) = \neg\ a \vee \neg\ b$, y $\neg(a \vee b) = \neg\ a \wedge \neg\ b$
- Antimonotónico: $a \leq b$ sii $\neg\ a \geq \neg\ b$

no satisfaciendo las propiedades booleanas:

- Contradicción: $\exists\ a \mid a \wedge \neg\ a \neq 0$
- Totalidad: $\exists\ a \mid a \vee \neg\ a \neq 1$

Es un retículo quasi-booleano, o retículo De Morgan, ya que es un retículo distributivo con un operador unario $\neg$ con propiedades De Morgan, involución, antimonotónico. Es decir, $\neg a$ es un quasi-complemento de a.

Operación causal

Se define la operación interna causal $\rightarrow$ (tabla 7.2), con las siguientes propiedades:

- $a \rightarrow (b \wedge c) = (a \rightarrow b) \wedge (a \rightarrow c)$
- $a \rightarrow (b \vee c) = (a \rightarrow b) \vee (a \rightarrow c)$
- $\neg(a \rightarrow b) =_{\mathcal{B}} (a \rightarrow \neg b)$, sólo si $a \rightarrow b \in \{0, 1\}$.

Monotonicidad de $\wedge$, $\vee$, y causal ($\rightarrow$)

Se comprueba de la definición de los operadores $\wedge$, $\vee$ y causal ($\rightarrow$) sobre los valores de verdad de la lógica que:

- $\wedge$ es monótono:

$$a \leq a' \text{ y } b \leq b', \text{ entonces } a \wedge b \leq a' \wedge b'$$

- $\vee$ es monótono:

$$a \leq a' \text{ y } b \leq b', \text{ entonces } a \vee b \leq a' \vee b'$$

- Causal $\rightarrow$ es monótono:

$$a \leq a' \text{ y } b \leq b', \text{ entonces } a \rightarrow b \leq a' \rightarrow b'$$

La lógica SCTL, como ya hemos mencionado, es una lógica con cuantificación universal implícita, es por esto que no se impone la necesidad de establecer la dualidad entre las cuantificaciones universal y existencial. Con el objetivo de aumentar la expresividad, tanto de SCTL como de SCTL-T, se introduce en esta tesis la cuantificación existencial en SCTL. La dualidad clásica entre la cuantificación universal y existencial se consigue redefiniendo el orden total en SCTL a un orden parcial.

7.4.2 Definición alternativa en función de un orden parcial

Consideremos la definición de la lógica como una relación de orden parcial en el conjunto de verdad $\{0, \frac{1}{4}, \widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$, en el que $\frac{1}{2}$ y $\widehat{\frac{1}{2}}$ no se encuentran ordenados, y siendo la operación $\vee$ el menor límite superior (supremo), y $\wedge$ el mayor límite inferior (ínfimo) de los dos elementos en la relación.

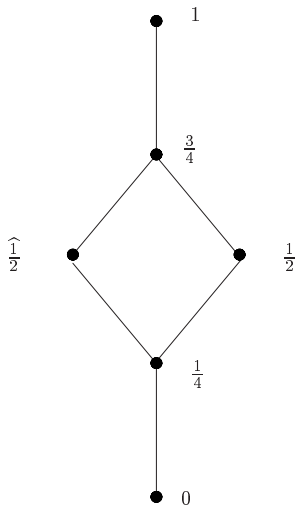


Figura 7.8. Orden Parcial

Con esta definición, es necesario introducir los siguientes cambios, que se siguen del orden parcial en la figura 7.8 y de la definición de los operadores $\vee$ y $\wedge$:

- $\neg \frac{1}{2} = \frac{1}{2}$, y $\neg \widehat{\frac{1}{2}} = \widehat{\frac{1}{2}}$
- $\frac{1}{2} \wedge \widehat{\frac{1}{2}} = \frac{1}{4}$
- $\frac{1}{2} \vee \widehat{\frac{1}{2}} = \frac{3}{4}$

Los cambios respetan la intuición de la lógica. Así, por ejemplo, para $\frac{1}{2} \wedge \widehat{\frac{1}{2}} = \frac{1}{4}$, de la interpretación $\frac{1}{4}$, podemos decir que el requisito nunca puede llegar a satisfacerse; sin embargo, mientras en la definición original su única evolución era a 0, con el orden parcial definido existen dos posibles evoluciones: no se satisface (0), o no es aplicable ($\widehat{\frac{1}{2}}$); es decir, el requisito no puede llegar a satisfacerse.

La ventaja de la definición a partir del orden parcial es que, además de todas la propiedades anteriores (con lo que sigue siendo un álgebra De Morgan), se cumple la dualidad:

$$\neg(a \rightarrow b) = a \rightarrow \neg\, b \tag{7.1}$$

Con la definición en función de un orden total (sección 7.4.1), la equivalencia 7.1 sólo se cumple cuando el valor de a no es $a < \frac{1}{2}$, en cuyo caso, el valor de verdad del requisito es siempre "no aplicable". Esta propiedad es imprescindible para la definición de la forma normal positiva en SCTL-T.

$\rightarrow$		cons					
		0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
prem	0	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$
	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$
	1	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1

$\vee$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
0	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$	$\frac{3}{4}$	$\frac{3}{4}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1
$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	1
1	1	1	1	1	1	1

$\wedge$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1
0	0	0	0	0	0	0
$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$\widehat{\frac{1}{2}}$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
$\frac{1}{2}$	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$\frac{3}{4}$	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{4}$
1	0	$\frac{1}{4}$	$\widehat{\frac{1}{2}}$	$\frac{1}{2}$	$\frac{3}{4}$	1

$\neg$	
0	1
$\frac{1}{4}$	$\frac{3}{4}$
$\widehat{\frac{1}{2}}$	$\widehat{\frac{1}{2}}$
$\frac{1}{2}$	$\frac{1}{2}$
$\frac{3}{4}$	$\frac{1}{4}$
1	0

Tabla 7.3. Operadores en IPM: orden parcial

La definición en función del orden parcial de los operadores IPM se muestra en las tabla 7.3. Donde con el orden parcial definido, la operación interna causal $\rightarrow$, es tal que:

$$\rightarrow (a,b) = (a \vee \frac{\widehat{1}}{2}) \wedge \left(\left((\neg a \wedge \frac{1}{4}) \vee (b \wedge \frac{3}{4}) \right) \vee \left((\neg a \wedge \frac{\widehat{1}}{2}) \vee (b) \right) \right)$$

que podemos simplificar a:

$$\rightarrow (a,b) = (a \vee \frac{\widehat{1}}{2}) \wedge \left(\left(\neg a \wedge \frac{\widehat{1}}{2} \right) \vee b \right)$$

y, por tanto, interpretable como la implicación cuando el requisito es aplicable.

7.4.3 Obtención de $\mathcal{L}_6$ a partir de $\mathcal{L}_3$

Tanto el orden total $\mathcal{L}_3$ como el orden parcial $\mathcal{L}_6$ son cerrados bajo las operaciones conjunción ($\wedge$), disyunción ($\vee$) y complementación ($\neg$). La obtención de $\mathcal{L}_6$ a partir de $\mathcal{L}_3$ se basa en la interpretación causal de los requisitos bajo la operación $\rightarrow$. En la figura 7.9 se puede observar la obtención de $\mathcal{L}_6$ a partir de $\mathcal{L}_3$, aplicando el operador causal a un par ordenado (i, j), donde $i, j \in \mathcal{L}_3$

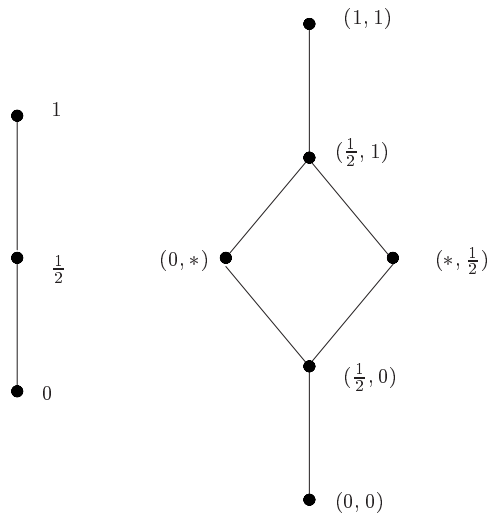


Figura 7.9. Obtención de $\mathcal{L}_6$ a partir de $\mathcal{L}_3$

7.4.4 Orden en nivel de conocimiento

El orden parcial que define la lógica es un orden en el nivel de verdad. Podemos definir un orden alternativo en nivel de conocimiento $\leq_c$, obteniendo tres niveles de conocimiento de una fórmula SCTL-T (figura 7.10):

- $\{0, \widehat{\frac{1}{2}}, 1\}$: Conocemos el grado de satisfacción de la fórmula, sea cual sea la evolución posterior del modelo.
- $\{\frac{1}{4}, \frac{3}{4}\}$: Nivel medio del conocimiento del grado de satisfacción de la fórmula en el modelo. En un modelo obtenido por evolución del sistema actual, el valor de verdad l de la fórmula será $\frac{1}{4} \leq_c l$ (resp. $\frac{3}{4} \leq_c l$) para el valor de verdad $\frac{1}{4}$ (resp. $\frac{3}{4}$).
- $\{\frac{1}{2}\}$: Nivel mínimo de conocimiento del grado de satisfacción de la fórmula; no podemos decir nada sobre su grado de satisfacción sobre el sistema final sin subespecificación.

Como se sigue de la figura, el orden $\leq_c$ no es un retículo, sino un semi-retículo completo, dado que para cada par de elementos existe un ínfimo y, además, existe un mínimo ($\frac{1}{2}$). Sin embargo, para cada par de elementos, no siempre existe un

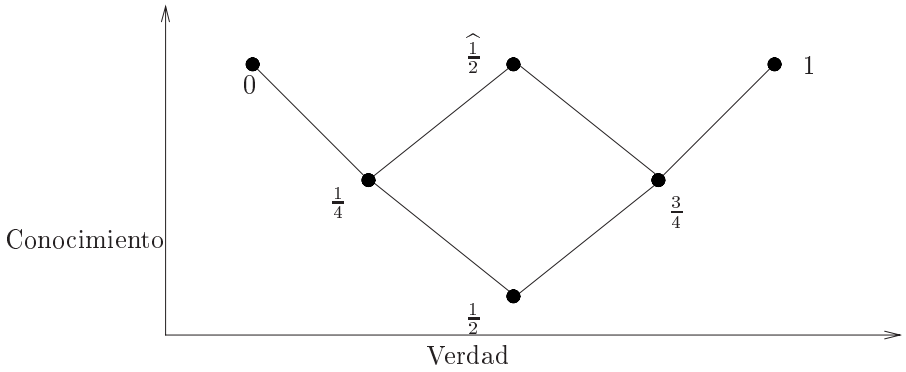


Figura 7.10. Representación del orden de verdad y orden de conocimiento

supremo y, además, no existe un máximo. Como veremos en el capítulo 10, el algoritmo de síntesis definido en la metodología respeta el nivel de conocimiento de cualquier fórmula SCTL-T.

Para el orden $\leq_c$ definido, se cumple que los operadores $\vee$, $\wedge$, $\rightarrow$ y $\neg$ son monótonos:

- $\wedge$ es monótono:

$$a \leq_c a' \text{ y } b \leq_c b', \text{ entonces } a \wedge b \leq_c a' \wedge b'$$

- $\vee$ es monótono:

$$a \leq_c a' \text{ y } b \leq_c b', \text{ entonces } a \vee b \leq_c a' \vee b'$$

- Causal $\rightarrow$ es monótono:

$$a \leq_c a' \text{ y } b \leq_c b', \text{ entonces } a \rightarrow b \leq_c a' \rightarrow b'$$

- $\neg$ es monótono:

$$a \leq_c b \text{ entonces } \neg a \leq_c \neg b$$

7.5 Sintaxis

La lógica SCTL-T es una lógica de tiempo real de tipo *explicit clock* que extiende la lógica SCTL sobre un dominio de tiempo denso ($\Re_+$). La definición de un requisito SCTL-T establece un Identificador de Requisito $I_R \in \mathcal{I}_R$ (conjunto de variables proposicionales de requisito) y su fórmula SCTL-T asociada:

$$I_R := \phi, \text{ con } \phi \in \Phi_{SCTL-T}$$

Siendo A el conjunto de constantes proposicionales de acción, Θ el conjunto de constantes proposicionales de estado, $\mathcal{C}$ el conjunto de constantes proposicionales de relojes en un modelo MUS-T, $\mathcal{E}$ el conjunto de variables proposicionales relojes (relojes de especificación) e $\mathcal{I}_R$ el conjunto de variables proposicionales de requisito, Φ_{SCTL-T} se define por la gramática:

$$
\begin{array}{lll}
\langle \phi_{SCTL-T} \rangle & ::= & \phi_C \forall \otimes \phi_C \\
| & & \phi_C \exists \otimes \phi_C
\end{array}
$$

$$
\begin{array}{lll}
\langle \phi_C \rangle & ::= & \psi \in \Psi \\
| & & a \in A \\
| & & \theta \in \Theta \\
| & & y.\phi_C \\
| & & \phi_C \wedge \phi_C \\
| & & \phi_C \vee \phi_C \\
| & & \neg \phi_C \\
| & & \phi_{SCTL-T}
\end{array}
$$

$$
\begin{array}{lll}
\langle \otimes \rangle & ::= & \Rightarrow \\
| & & \Rightarrow_+ \\
| & & \Rightarrow_- \\
| & & \Rightarrow \bigcirc \\
| & & \Rightarrow \odot
\end{array}
$$

donde:

- $\{\forall, \exists\}$ son los cuantificadores de camino propios de la lógica ramificada.
- $\otimes ::= \{\Rightarrow, \Rightarrow_+, \Rightarrow_-, \Rightarrow \odot, \Rightarrow \bigcirc\}$ es el conjunto de los operadores temporales lineales *actual*, *antes* y *después*.
- $\theta \in \Theta ::= \{true \mid false \mid \emptyset\}$, constantes proposicionales de estado, donde $\emptyset$ denota la ausencia de proposición, y $\{true, false\}$ la condición de accesibilidad posible y no posible respectivamente.
- $a \in A$, el conjunto de acciones en el alfabeto de un modelo MUS-T.
- $y.$ es el cuantificador *freeze* inspirado en el del mismo nombre propuesto por (Alur y Henzinger., 1989) que liga un reloj y, al que denominamos reloj de especificación (para diferenciarlo de los relojes del sistema), al contexto temporal local, reiniciándolo. Un reloj de especificación $y \in \mathcal{E}$ no controla el comportamiento del sistema (como los relojes en el modelo), sino que marca una restricción temporal entre dos contextos temporales del mismo.
- $\psi \in \Psi$, el conjunto de predicados sobre los relojes *freeze* de la fórmula ($\mathcal{E}$) o los relojes definidos en un modelo MUS-T ($\mathcal{C}$). Dichas restricciones se definen según la gramática $\Psi(\mathcal{C}')$ (definición 6.4), donde $\mathcal{C}' = \mathcal{C} \cup \mathcal{E}$.

Las fórmulas SCTL-T son de subespecificación implícita; los comportamientos temporizados no explicitados en las fórmulas se consideran subespecificados en el requisito y no son considerados para el cálculo del grado de satisfacción en un modelo MUS-T.

Definición 7.2. **Fórmulas cerradas.** Diremos que una fórmula SCTL-T ϕ es cerrada si:

[1] Toda variable reloj de especificación $y \in \mathcal{E}$ incluida en un predicado de tiempo ψ_y en ϕ ha sido ligada mediante una operación $y.$ incluida en la premisa de un operador temporal que incluye ψ_y en su consecuencia.

Definición 7.3. **Fórmulas bien formadas.** Una fórmula $\phi \in \Phi_{SCTL-T}$ se dice bien formada si cumple:

[1] ϕ es una fórmula SCTL-T cerrada.

[2] ϕ no tiene relojes alternantes en tiempo (monodireccional): todos los predicados de tiempo que incluyen un reloj de especificación $y \in \mathcal{E}$ forman parte de consecuencias en fórmulas que, o bien sólo incluyen operadores de futuro ($\Rightarrow \bigcirc$, $\Rightarrow_+$), o bien sólo operadores de pasado ($\Rightarrow \odot$, $\Rightarrow_-$). Esta regla lleva a una partición de los relojes de especificación en relojes de pasado $\mathcal{E}_p$ y relojes de futuro $\mathcal{E}_f$.

De ámbito específico para la lógica SCTL-T consideramos el punto [2]. Las fórmulas con relojes de especificación no alternantes en tiempo garantizan una visión del tiempo siempre positiva (bien hacia delante o bien hacia atrás), lo que resulta más conveniente en especificación, dado que la dirección de ejecución viene determinada por el operador temporal. Desde nuestro punto de vista, la no alternancia en tiempo no excluye ninguna propiedad útil en un sistema y, lejos de aportar flexibilidad, podría ser una importante fuente de error en las especificaciones.

Se puede ver el tratamiento de los relojes de especificación en SCTL-T como una versión de *Event-Clock timed Automaton* (ECA) (Alur et al., 1999), pero únicamente aplicada a los relojes de especificación. Los relojes en el conjunto $\mathcal{E}_p$ se identifican con los relojes denominados *event recording clocks* en ECA, y $\mathcal{E}_f$ con *event predicting clocks*.

Por último, citar que la regla [2] no impide combinar operadores de pasado y futuro en una misma fórmula, sólo restringiendo el uso de los relojes de especificación como de sólo pasado o de sólo futuro.

Las fórmulas SCTL-T se pueden clasificar atendiendo al conjunto de operadores temporales utilizados:

Definición 7.4. **Fórmulas de futuro.** Una fórmula SCTL-T ϕ es de futuro estricto si sólo incluye los operadores temporales $\Rightarrow \bigcirc$ y $\Rightarrow_+$. ϕ será una fórmula de futuro no estricto si, además, contiene el operador $\Rightarrow$.

Definición 7.5. **Fórmulas de pasado.** Una fórmula SCTL-T ϕ es de pasado estricto si sólo incluye los operadores temporales $\Rightarrow \odot$ y $\Rightarrow_-$. ϕ será una fórmula de pasado no estricto si, además, contiene el operador $\Rightarrow$.

7.6 Semántica

SCTL se puede catalogar como una lógica ramificada multivalorada y puntual, y, como tal, se interpreta sobre un grafo de estados. Su extensión temporizada SCTL-T, de las mismas características que la anterior, incorpora requisitos de tiempo cuantitativos mediante un mecanismo de relojes de especificación que operan sobre un dominio denso. Con dicha ampliación, SCTL-T se interpreta sobre un grafo denso multivalorado de interpretaciones extendidas con el entorno de la fórmula, que no es más que una extensión del definido en la sección 6.6 con los relojes de especificación.

Definición 7.6. **Estado extendido**. Un estado extendido será una 3-tupla (s, γ, ζ), donde $(s, \gamma) \in \mathcal{ST}$ es un estado de un grafo semántico MUS-T y ζ su entorno, tal que:

- s, es un estado de control $\in S$.
- $\gamma \in [\mathcal{C} \to \Re_+]$ es la interpretación actual de los relojes del sistema.
- $\zeta \in [\mathcal{E} \to \Re_+]$ es el entorno, o interpretación de los relojes de especificación $(\mathcal{E})$, tal que:
  - Evoluciona según:

$$\begin{cases} (s, \gamma, \zeta) \overset{\tau}{\multimap} (s, \gamma + \tau, \zeta + \tau) \\ (s, \gamma, \zeta) \overset{a}{\multimap} (s\,', \gamma_\lambda, \zeta) \end{cases}$$

  - Se reinicia al aplicar el operador $y.$ en un estado extendido:

$$y.(s, \gamma, \zeta) = (s, \gamma, \zeta[y/0])$$

Adicionalmente, y como se ha introducido en la sección 7.3.3, el grado de satisfacción de una fórmula ϕ en un estado (s, γ, ζ) dependerá de la condición de especificación de la transición $t \in \mathcal{T}$ bajo la que hemos alcanzado dicho estado (condición de accesibilidad). Añadimos, por simplicidad en la presentación, un elemento $c_e \in \mathcal{L}_3$ representando dicha condición de especificación, definiendo, por tanto, la 4-tupla (s, γ, ζ, c_e).

7.6.1 Grado de satisfacción

Definición 7.7. **Grado de Satisfacción.** Definimos el grado de satisfacción $\vdash$:

$$\vdash: \left(\Phi_{SCTL-T}, (S \times \Re_+^{\#\mathcal{C}} \times \Re_+^{\#\mathcal{E}} \times \mathcal{L}_3)\right) \to \mathcal{L}_6 = \{0, \frac{1}{4}, \frac{\widehat{1}}{2}, \frac{1}{2}, \frac{3}{4}, 1\}$$

donde $\vdash (\phi, (s, \gamma, \zeta, c_e))$ asigna un valor de verdad $\in \mathcal{L}_6$ a una fórmula ϕ en un estado (s, γ), posiblemente extendido con un entorno ζ, y con condición de especificación, por acceso, c_e.

En la verificación formal de una fórmula SCTL-T en un estado de control del modelo MUS-T se deberá especificar una cuantificación sobre el conjunto de estados que se corresponden con dicho estado de control.

Definimos la relación de satisfacción a nivel de modelo y de estados de control de la siguiente forma:

- La relación de satisfacción de una propiedad especificada en SCTL-T en un modelo MUS-T ($\models_{\mathcal{M}}$) se define de la forma usual como:

$$\models_{\mathcal{M},s_0} (\phi, \mathcal{M}) \equiv \vdash (\phi, (s_0, \gamma^0, \zeta^0, 1))$$

- La relación de satisfacción de una propiedad especificada en SCTL-T en un estado de control de un modelo MUS-T, se define mediante dos reglas de satisfacción, $\models_{\forall}$ y $\models_{\exists}$, que, intuitivamente, reflejan el deseo de que la propiedad se satisfaga por lo menos en un instante, o en todos los instantes en que el estado de control es alcanzado:

$$\begin{cases} \models_{\forall} (\phi, s) \equiv \bigwedge \vdash (\phi, (s, \gamma, \zeta^0, 1)) \\ \models_{\exists} (\phi, s) \equiv \bigvee \vdash (\phi, (s, \gamma, \zeta^0, 1)) \end{cases}$$

La obtención del grado de satisfacción ($\vdash$) de un requisito SCTL-T sobre un estado extendido de un grafo semántico se construye por inducción en la estructura de la fórmula a partir de:

- Grado de satisfacción de los predicados de tiempo (sección 7.6.2).
- Grado de satisfacción de una acción del alfabeto A del modelo (sección 7.6.3).
- Grado de satisfacción de las constantes proposicionales de estado (sección 7.6.4).
- Operador estados de aplicabilidad con tiempo, y grado de satisfacción de fórmulas causales (sección 7.6.5).

Dado que el grado de satisfacción de una acción del alfabeto en un estado del modelo MUS-T depende del contexto temporal de dicho estado, definimos una nueva relación de satisfacción sobre acciones con los mismos valores que en el caso no temporizado $\{0, \frac{1}{2}, 1\}$ (sección 7.6.3). En el caso de requisitos genéricos, las reglas de la relación de satisfacción son las mismas que para el caso no temporizado, modificando el operador estados de aplicabilidad (sección 7.6.5) y añadiendo la regla del operador *freeze*.

7.6.2 Grado de satisfacción de los predicados de tiempo

El grado de satisfacción de un predicado de tiempo $\psi \in \Psi$ asigna a cada estado (s, γ, ζ, c_e) un valor de verdad $\in \{0, 1\}$. Dicho grado de satisfacción $\vdash$ será la satisfacción booleana del predicado de tiempo ψ (definición 6.5):

$$\vdash (\psi, (s, \gamma, \zeta, c_e)) = \psi(\gamma, \zeta)$$

7.6.3 Grado de satisfacción de acciones

El grado de satisfacción de una acción a del alfabeto redefine el grado de satisfacción de las acciones en SCTL, teniendo en cuenta el valor actual de los relojes del sistema.

DEFINICIÓN 7.8. **Condición de especificación de una acción.** La condición de especificación c_A de $a \in A$ en un estado (s, γ):

$$c_A(a, (s, \gamma)) : A \times (S \times \Re_+^{\#\mathcal{C}}) \rightarrow \{0, \frac{1}{2}, 1\}$$

se define como:

$$c_A(a, (s, \gamma)) = \begin{cases} 0 & \text{sii } g_{np}(a, s)(\gamma) = 1 \\ \frac{1}{2} & \text{sii } g_s(a, s)(\gamma) = 1 \\ 1 & \text{sii } g_p(a, s)(\gamma) = 1 \end{cases}$$

Los tres grados de satisfacción $\{0, \frac{1}{2}, 1\}$ reflejan, respectivamente, las tres posibles condiciones de especificación para la acción a en dicho estado: la acción es posible, la acción no es posible, o la acción está subespecificada. Consecuentemente, el grado se satisfacción de a en un estado (s, γ, c_e):

$$\vdash (a, (s, \gamma, \zeta, c_e)) = c_A(a, (s, \gamma))$$

7.6.4 Grado de satisfacción de constantes proposicionales de estado

Dadas las constantes proposicionales de estado $\Theta = \{true, false, \emptyset\}$, donde:

- $\emptyset$: Denota la ausencia de proposición. Su grado de satisfacción es siempre 1.
- $true$: Denota la proposición de accesibilidad posible. Su grado de satisfacción es 1 en un estado si se accede mediante una transición posible ($c_e = 1$).
- $false$: Denota la proposición de accesibilidad no posible. Su grado de satisfacción es 1 en un estado si se accede mediante una transición no posible ($c_e = 0$).

definimos el grado de satisfacción de $\theta \in \Theta$ en un estado (s, γ, ζ, c_e), $\vdash (\theta, (s, \gamma, \zeta, c_e))$, mediante la tabla 7.4.

7.6.5 Grado de satisfacción de fórmulas causales

Redefinición de los estados de aplicabilidad

El grado de satisfacción de un requisito SCTL-T que incluya un operador temporal $\otimes$ conlleva el cálculo de los estados de aplicabilidad de dicho operador. El nuevo operador estados de aplicabilidad ($\perp$) se interpretará sobre los estados

	c_e		
	1	$\frac{1}{2}$	0
$true$	1	$\frac{1}{2}$	0
$false$	0	$\frac{1}{2}$	1
$\emptyset$	1	1	1

Tabla 7.4. Grado de satisfacción de las constantes proposicionales de estado

(extendidos), y su resultado será un conjunto (casi siempre infinito) de estados (extendidos) en los que realizar la verificación. Como se muestra a continuación, los estados de aplicabilidad de un estado (s, γ, ζ, c_e) no dependen de la condición de especificación c_e.

$$\begin{aligned}\perp((\bigcirc, P), (s, \gamma, \zeta)) = \{&(s', \gamma', \zeta', 1) \mid (s, \gamma, \zeta, c_e) \xrightarrow{a} (s', \gamma', \zeta'),\\ &(s', \gamma', \zeta', \frac{1}{2}) \mid (s, \gamma, \zeta) \overset{a}{\rightsquigarrow} (s', \gamma', \zeta')\}\end{aligned}$$

siendo a cualquier acción que en su forma positiva se incluya en la parte de la premisa P aplicable al estado (s, γ, ζ, c_e).

Análogamente, para el operador $\bigodot$:

$$\begin{aligned}\perp((\bigodot, P), (s, \gamma, \zeta)) = \{&(s', \gamma', \zeta', 1) \mid (s', \gamma', \zeta') \xrightarrow{a} (s, \gamma, \zeta),\\ &(s', \gamma', \zeta', \frac{1}{2}) \mid (s', \gamma', \zeta') \overset{a}{\rightsquigarrow} (s, \gamma, \zeta)\}\end{aligned}$$

donde el cálculo de los estados de aplicabilidad no incluye filtración causal.

Por último, definimos el operador estados de aplicabilidad para requisitos incluyendo los operadores temporales actual: $\Rightarrow_+$, $\Rightarrow_-$ y $\Rightarrow$:

$$\begin{aligned}\perp((\Rightarrow_+, P), (s, \gamma, \zeta)) &= \{(s, \gamma', \zeta', 1) \mid (s, \gamma, \zeta) \xrightarrow{\tau} (s, \gamma', \zeta'),\\ &\quad (s, \gamma', \zeta', \frac{1}{2}) \mid (s, \gamma, \zeta) \overset{\tau}{\rightsquigarrow} (s, \gamma', \zeta')\}\\ \perp((\Rightarrow_-, P), (s, \gamma, \zeta)) &= \{(s, \gamma', \zeta', 1) \mid (s, \gamma', \zeta') \xrightarrow{\tau} (s, \gamma, \zeta),\\ &\quad (s, \gamma', \zeta', \frac{1}{2}) \mid (s, \gamma', \zeta') \overset{\tau}{\rightsquigarrow} (s, \gamma, \zeta)\}\\ \perp((\Rightarrow, P), (s, \gamma, \zeta)) &= \{(s, \gamma, \zeta, 1)\}\end{aligned}$$

Grado de satisfacción de fórmulas causales

Como habíamos adelantado, el grado de satisfacción de una fórmula causal ϕ se calcula a partir del grado de satisfacción de la premisa en el estado en que se

formula ϕ, y el grado de satisfacción de la consecuencia en cada uno de los estados de aplicabilidad modificado por la condición de accesibilidad. Así, en base a la cuantificación en la fórmula causal:

- Cuantificación universal: $\vdash (\phi_1 \forall \otimes \phi_2, (s, \gamma, \zeta, c_e)) \equiv$

$$\rightarrow \left(\vdash (\phi_1, (s, \gamma, \zeta, c_e)), \bigwedge_i (\neg c_{e_i}{}') \vee \vdash (\phi_2, (s_i{}', \gamma_i{}', \zeta_i{}', c_{e_i}{}')) \right)$$

- Cuantificación existencial: $\vdash (\phi_1 \exists \otimes \phi_2, (s, \gamma, \zeta, c_e)) \equiv$

$$\rightarrow \left(\vdash (\phi_1, (s, \gamma, \zeta, c_e)), \bigvee_i c_{e_i}{}' \wedge \vdash (\phi_2, (s_i{}', \gamma_i{}', \zeta_i{}', c_{e_i}{}')) \right)$$

siendo $(s_i{}', \gamma_i{}', \zeta_i{}', c_{e_i}{}') \in \perp (\otimes, \phi_1), (s, \gamma, \zeta))$[2]

7.7 Forma Normal Positiva

Diremos que un requisito está en *forma normal positiva* si todos los operadores de negación están aplicados directamente sobre las acciones. Dado que en IPM se cumple la dualidad:

$$\neg(a \rightarrow b) \equiv a \rightarrow \neg b$$

podemos afirmar:

$$\neg(\phi_1 \forall \otimes \phi_2) = \phi_1 \exists \otimes \neg \phi_2 \tag{7.2}$$

$$\neg(\phi_1 \exists \otimes \phi_2) = \phi_1 \forall \otimes \neg \phi_2 \tag{7.3}$$

Demostración. **Expresión 7.2** $\neg(\phi_1 \forall \otimes \phi_2) = \phi_1 \exists \otimes \neg \phi_2$. Para s_t denotando un estado temporizado genérico (s, γ, ζ, c_e), y $ce(s_t)$ la condición de especificación c_e de dicho estado:

$$\begin{aligned}
\vdash (\neg(\phi_1 \forall \otimes \phi_2), s_t) &= \neg(\rightarrow (\vdash (\phi_1, s_t), \bigwedge_{s_t{}' \in \perp(s_t)} \neg ce(s_t{}') \vee \vdash (\phi_2, s_t{}'))) \\
&= \rightarrow (\vdash (\phi_1, s_t), \neg(\bigwedge_{s_t{}' \in \perp(s_t)} \neg ce(s_t{}') \vee \vdash (\phi_2, s_t{}'))) \\
&= \rightarrow \left(\vdash (\phi_1, s_t), \bigvee_{s_t{}' \in \perp(s_t)} ce(s_t{}') \wedge \neg \vdash (\phi_2, s_t{}') \right) \\
&= \vdash (\phi_1 \exists \otimes \neg \phi_2, s_t)
\end{aligned}$$

□

[2] $s_i{}' = s$ para $\otimes \in \{\Rightarrow_+, \Rightarrow_-, \Rightarrow\}$, y $\gamma_i{}' = \gamma$ para $\Rightarrow$

DEMOSTRACIÓN. **Expresión 7.3** $\neg(\phi_1 \exists \otimes \phi_2) = \phi_1 \forall \otimes \neg\phi_2$. Análoga □

La traducción de un requisito a su forma normal positiva es directa y se describe en la tabla 7.5.

Φ_{SCTL-T}	$\mathbf{FNP}(\Phi_{SCTL-T})$
$\neg(\phi \wedge \phi')$	$\neg\phi \vee \neg\phi'$
$\neg(\phi \vee \phi')$	$\neg\phi \wedge \neg\phi'$
$\neg(\phi\forall \otimes \phi')$	$\phi\exists \otimes \neg\phi'$
$\neg(\phi\exists \otimes \phi')$	$\phi\forall \otimes \neg\phi'$
$\neg(x.\phi)$	$x.(\neg\phi)$
$\neg(x > c)$	$x \leq c$
$\neg(x \geq c)$	$x < c$
$\neg(x < c)$	$x \geq c$
$\neg(x \leq c)$	$x > c$
$\neg(x = c)$	$(x > c) \wedge (x < c)$
$\neg true$	$false$
$\neg false$	$true$
$\neg\emptyset$	$\emptyset$

Tabla 7.5. Forma Normal Positiva

7.8 Grado de satisfacción de una fórmula SCTL-T

Siendo ϕ, ϕ_1, ϕ_2 fórmulas SCTL-T genéricas, definimos la relación de satisfacción:

$$\vdash: \left(\Phi_{SCTL-T} \times (S \times \Re_+^{\#\mathcal{C}} \times \Re_+^{\#\mathcal{E}}, \{0, \frac{1}{2}, 1\})\right) \rightarrow \{0, \frac{1}{4}, \widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$$

como sigue:

i) $\vdash (\psi, (s, \gamma, \zeta, c_e))$ con $\psi \in \Psi$ (sección 7.6.2).

ii) $\vdash (a, (s, \gamma, \zeta, c_e))$ con $a \in A$ (sección 7.6.3).

iii) $\vdash (\theta, (s, \gamma, \zeta, c_e))$ con $\theta \in \Theta$ (sección 7.6.4).

iv) $\vdash (y.\phi, (s, \gamma, \zeta, c_e)) \equiv \vdash (\phi, (s, \gamma, \zeta[y/0], c_e))$

v) $\vdash (\neg\phi, (s, \gamma, \zeta, c_e)) \equiv \neg(\vdash (\phi, (s, \gamma, \zeta, c_e)))$

vi) $\vdash (\phi_1 \vee \phi_2, (s, \gamma, \zeta, c_e)) \equiv \vdash (\phi_1, (s, \gamma, \zeta, c_e)) \vee \vdash (\phi_2, (s, \gamma, \zeta, c_e))$

vii) $\vdash (\phi_1 \wedge \phi_2, (s, \gamma, \zeta, c_e)) \equiv \vdash (\phi_1, (s, \gamma, \zeta, c_e)) \wedge \vdash (\phi_2, (s, \gamma, \zeta, c_e))$

viii) $\vdash (\phi_1 \forall \otimes \phi_2, (s, \gamma, \zeta, c_e)) \equiv$

$$\rightarrow \left(\vdash (\phi_1, (s, \gamma, \zeta, c_e)), \bigwedge_i ((\neg c_{e_i}{}') \vee \vdash (\phi_2, (s_i{}', \gamma_i{}', \zeta_i{}', c_{e_i}{}')))\right)$$

con $(s_i{}', \gamma_i{}', \zeta_i{}', c_{e_i}{}') \in \perp((\otimes, \phi_1), (s, \gamma, \zeta))$[3]

ix) $\vdash (\phi_1 \exists \otimes \phi_2, (s, \gamma, \zeta, c_e)) \equiv$

$$\rightarrow \left(\vdash (\phi_1, (s, \gamma, \zeta, c_e)), \bigvee_i c_{e_i}{}' \wedge \vdash (\phi_2, (s_i{}', \gamma_i{}', \zeta_i{}', c_{e_i}{}')) \right)$$

con $(s_i{}', \gamma_i{}', \zeta_i{}', c_{e_i}{}') \in \perp((\otimes, \phi_1), (s, \gamma, \zeta))$ [4]

Definición 7.9. **Satisfacción.** Una fórmula SCTL-T ϕ se satisface en un estado (s, γ, ζ, c_e) si su grado de satisfacción es 1:

$$\vdash (\phi, (s, \gamma, \zeta, c_e)) = 1$$

Definición 7.10. **Satisfactibilidad finita.** Una fórmula SCTL-T ϕ se dice finitamente satisfactible si existe un modelo MUS-T $\mathcal{M}$ y un estado (s, γ) de su grafo semántico asociado $\mathcal{S}_{\mathcal{M}}$, tal que ϕ se satisface en $(s, \gamma, \zeta = 0)$.

Mientras la mayoría de las técnicas formales para requisitos de tiempo real consideran la verificación de dichos requisitos como un refinamiento del sistema que ya ha sido verificado atemporalmente, en nuestro caso, la incorporación del tiempo en un sistema aún no totalmente especificado espacialmente puede ayudar al diseñador a descartar ordenamientos temporales que acabaría descartando la adición de requisitos de tiempo en el sistema.

7.9 Recursión e Instanciación en SCTL-T

En esta sección definimos la recursión e instanciación en SCTL-T. Para ello, en primer lugar definimos la semántica global de una fórmula SCTL-T sobre un grafo semántico MUS-T. Por brevedad denotaremos s al estado extendido (s, γ, ζ, c_e) en la definición 7.6.

7.9.1 Definiciones preliminares

Definición 7.11. **Partición ordenada de cardinalidad 6**. Sea $\mathcal{M}$ un modelo MUS-T y $\mathcal{ST}$ el conjunto de estados de su grafo semántico asociado. En el espacio de estados $\mathcal{ST}$ definimos $\mathcal{P}^6$ como el conjunto de particiones del espacio de estados $\mathcal{ST}$ de cardinalidad 6. Una partición $P \in \mathcal{P}^6$ se define como:

$$P = \{\langle P \rangle_0, \langle P \rangle_{\frac{1}{4}}, \langle P \rangle_{\widehat{\frac{1}{2}}}, \langle P \rangle_{\frac{1}{2}}, \langle P \rangle_{\frac{3}{4}}, \langle P \rangle_1\}$$

Definición 7.12. **Partición asociada a ϕ.** Sea ϕ una fórmula SCTL-T, definimos la partición asociada a ϕ, $P_{\mathcal{M}}(\phi)$, como una partición $\in \mathcal{P}^6$ tal que:

$$\forall\, s \in \mathcal{ST}, s \in \langle P \rangle_i \text{ sii } \vdash (\phi, s) = i$$

[3] $s_i{}' = s$ para $\otimes \in \{\Rightarrow_+, \Rightarrow_-, \Rightarrow\}$, y $\gamma_i{}' = \gamma$ para $\Rightarrow$

[4] Idéntico a 3.

Es decir, asociamos a cada fórmula ϕ, una partición del espacio de estados donde cada subconjunto $\langle P\rangle_i \subseteq \mathcal{ST}$ representa el total de estados con un valor de verdad i.

Definición 7.13. **Inclusión en $\mathcal{P}^6$.** Dados $P, P' \in \mathcal{P}^6$, definimos la inclusión $\subseteq_P$ en el conjunto $\mathcal{P}^6$ como:

$$P \subseteq_P P' \text{ sii } \langle P\rangle_i \subseteq \bigcup_{i\leq j\leq 1} \langle P'\rangle_j$$

donde $\subseteq$ es la relación de inclusión entre conjuntos de estados en el espacio $\mathcal{ST}$ y $\leq$ es el orden parcial en $\mathcal{L}_6$.

Definición 7.14. **Unión e Intersección en $\mathcal{P}^6$.** Dados $P, P' \in \mathcal{P}^6$, definimos la unión $\cup_P$ entre particiones en $\mathcal{P}^6$ ($P \cup_P P'$) como:

$$s \in \langle P \cup_P P'\rangle_j \text{ sii } j = \sup\{i, i' | s \in \langle P\rangle_i \text{ y } s \in \langle P'\rangle_{i'}\}$$

Asimismo, dados $P, P' \in \mathcal{P}^6$, definimos la intersección $\cap_P$ entre particiones en $\mathcal{P}^6$ ($P \cap_P P'$):

$$s \in \langle P \cap_P P'\rangle_j \text{ sii } j = \inf\{i, i' | s \in \langle P\rangle_i \text{ y } s \in \langle P'\rangle_{i'}\}$$

siendo inf y sup, respectivamente, el ínfimo y el supremos en el orden parcial $\leq$ sobre $\mathcal{L}_6$.

El conjunto de particiones $\mathcal{P}^6$ es un retículo bajo la relación de inclusión $\subseteq_P$, donde para $P, P' \in \mathcal{P}^6$, el ínfimo es $P \cap_P P'$, y el supremo es $P \cup_P P'$. Además:

- $P_\perp = \{\langle P\rangle_0 = \mathcal{ST}, \langle P\rangle_{\frac{1}{4}\leq i\leq 1} = \{\}\}$, es un mínimo en $\mathcal{P}^6$.
- $P_\top = \{ \langle P\rangle_{0\leq i\leq \frac{3}{4}} = \{\}, \langle P\rangle_1 = \mathcal{ST}\}$, es un máximo en $\mathcal{P}^6$.

Definición 7.15. **Complemento en $\mathcal{P}^6$.** Definimos el complemento $\overline{P}^P$ entre particiones en $\mathcal{P}^6$ como:

$$s \in \langle \overline{P}^P\rangle_j \text{ sii } s \in \langle P\rangle_{\neg i}.$$

siendo $\neg\, i$ el complemento definido en $\mathcal{L}_6$.

7.9.2 Sintaxis y Semántica de la recursión e instanciación

Sintaxis

Un requisito SCTL-T será un conjunto ordenado de declaraciones:

$$\{I_0 := \phi_0, \cdots, I_n := \phi_n\}$$

Para permitir la instanciación y recursión en las fórmulas SCTL-T, ampliamos la sintaxis de la lógica (sección 7.5) con los siguientes elementos en $\langle\phi_C\rangle$:

$$\begin{array}{l} \langle \phi_C \rangle ::= \ I_R \\ \quad | \quad \ll I_R \gg \\ \quad | \quad [\![I_R]\!] \end{array}$$

siendo $I_R \in \mathcal{I}_R$ un identificador de requisito, y donde:

- I_R expresa la instanciación de la fórmula ϕ, tal que I_R es una variable requisito definida como $I_R := \phi$.
- $\ll I_R \gg$ y $[\![I_R]\!]$ expresan, respectivamente, la recursión mínima y máxima de una variable requisito I_R definida como $I_R := \phi$, siendo la recursión mínima el punto fijo mínimo del transformador definido por la recursión, y análogamente la recursión máxima.

Definición 7.16. **Requisitos cerrados.** Diremos que un requisito SCTL-T $\{I_i := \phi_i\}$ es cerrado sii:

[1] Reemplazando todas las instanciaciones I_j, en las definiciones $I_i := \phi_i$, por la fórmula SCTL-T ϕ_j que la define, las fórmulas $\{\phi_i\}$ son cerradas.

[2] Todos los identificadores de requisito I_i incluidos en $\{\phi_i\}$, mediante instanciación (I_i) o recursión ($[\![I_i]\!]$ y $\ll I_i \gg$), se encuentran definidos en una cláusula $I_i := \phi_i$.

Definición 7.17. **Requisitos bien formados.** Un requisito SCTL-T $\{I_i := \phi_i\}$ se dice bien formado sii:

[1] $\{I_i := \phi_i\}$ es cerrado.

[2] Reemplazando todas las instanciaciones I_j en las definiciones $I_i := \phi_i$, por la fórmula SCTL-T ϕ_j que la define, las fórmulas $\{\phi_i\}$ son bien formadas.

[3] Todos los elementos $I_{n'}$ en una definición $I_n := \phi_n$ se encuentran definidos en una sentencia $I_{n'} := \phi_{n'}$, con $n' > n$.

[4] Todos los elementos $\ll I_{n'} \gg$, $[\![I_{n'}]\!]$ en una definición $I_n := \phi_n$ se encuentran definidos en una sentencia $I_{n'} := \phi_{n'}$, con $n' \leq n$.

[5] No existe alternancia de recursiones mínimas y máximas, es decir, no existe una recursión mínima bajo una recursión máxima o viceversa (*Alternation Free*).

El punto [5] se ha introducido sólo en aras de la simplicidad del algoritmo de *model checking* realizado (nos remitimos a la extensa literatura en *μ-calculus* (Stirling y Walker, 1991; Liu et al., 1998) como punto de partida para soluciones sin este tipo de restricción).

Semántica

Una declaración SCTL-T $I_j := \phi_j$ que incluye una recursión $\ll I_i \gg$ (o $[\![I_i]\!]$) con $i \leq j$ en ϕ_j, define el transformador de predicados $I_i = f(I_i)$, con f el obtenido reemplazando las instanciaciones por su fórmula asociada, y la recursión $\ll I_i \gg$ (o $[\![I_i]\!]$) por la variable identificador de requisito I_i.

Dado el transformador de predicados $f : \mathcal{P}^6 \rightarrow \mathcal{P}^6$, diremos que f es monótono sii:

$$P \subseteq_P P' \text{ implica } f(P) \subseteq_P f(P')$$

Si el transformador es monótono, se garantiza la existencia de un punto fijo mínimo y un punto fijo máximo (Teorema de Tarski-Knaster (Tarski, 1955)), siendo:

- $\bigcap_P \{Z | f(Z) \subseteq_P Z\}$, el punto fijo mínimo.
- $\bigcup_P \{Z | f(Z) \supseteq_P Z\}$, el punto fijo máximo.

Una recursión mínima (máxima) será el punto fijo mínimo (máximo) del transformador de predicados definido por la recursión $I_R = f(I_R)$. Con la incorporación de la instanciación y recursión, la relación de satisfacción de una fórmula $\in \Phi_{SCTL-T}$ se amplía con los siguientes elementos adicionales:

x) $\vdash (I_R, (s, \gamma, \zeta, c_e)) \equiv \vdash (\phi_R, (s, \gamma, \zeta, c_e))$, con ϕ_R la incluida en la definición $I_R := \phi_R$.

xi) $\vdash (\ll I_R \gg, (s, \gamma, \zeta, c_e)) \equiv i \in \mathcal{L}_6 \mid (s, \gamma, \zeta) \in < \{\bigcap_P \{Z | f(Z) \subseteq_P Z\} >_i$

xii) $\vdash ([\![I_R]\!], (s, \gamma, \zeta, c_e)) \equiv i \in \mathcal{L}_6 \mid (s, \gamma, \zeta) \in < \{\bigcup_P \{Z | f(Z) \supseteq_P Z\} >_i$

7.9.3 Monotonicidad de transformadores

Vamos ahora a demostrar que los transformadores en SCTL-T son monótonos. Para ello, basta con demostrar, considerando fórmulas en forma normal positiva (sección 7.7), que los operadores básicos definidos en la lógica definen transformadores monótonos.

Demostramos a continuación la monotonicidad del transformador

$$f(Z) = \phi \ \exists \Rightarrow \bigcirc \ Z$$

las demostraciones para los demás operadores son análogas.

Demostración. $\mathbf{f(Z)} = \boldsymbol{\phi} \ \exists \Rightarrow \bigcirc$ **Z es monótono**. Sea $P \subseteq_P P'$, para demostrar que $f(P) \subseteq_P f(P')$, sea $s \in \mathcal{ST}$:

- Si $s \in \langle f(P) \rangle_1$, entonces $\vdash (\phi, s) = 1$ y $\exists \ s' \in \perp (\bigcirc, \phi)$ tal que $ce(s') = 1$ y $s' \in \ \langle P \rangle_1$.

  Dado que $P \subseteq_P P'$, podemos afirmar que $s' \in \ \langle P' \rangle_1$ y, por tanto, $s \in \ \langle f(P') \rangle_1$.

- Si $s \in \langle f(P)\rangle_{\frac{3}{4}}$, entonces:

  - si $\vdash (\phi, s) \in \{\frac{1}{2}, \frac{3}{4}\}$, $\exists\, s' \in\perp (\bigcirc, \phi)$ tal que $\frac{3}{4} \leq ce(s')$ y $s' \in \langle P\rangle_{\frac{3}{4}} \cup \langle P\rangle_1$.

    Dado que $P \subseteq_P P'$, podemos afirmar que $s' \in \bigcup_{\frac{3}{4}\leq i} \langle P'\rangle_i$ y, por tanto (tabla causal), $s \in \bigcup_{\frac{3}{4}\leq i} \langle f(P')\rangle_i$.

  - si $\vdash (\phi, s) = 1$, $\exists\, s' \in\perp (\bigcirc, \phi)$ tal que $\frac{3}{4} \leq ce(s')$, $s' \in \langle P\rangle_{\frac{3}{4}}$, y $\forall\, (s'' \neq s') \in\perp (\bigcirc, \phi), s'' \in \bigcup_{i\leq \frac{3}{4}} \langle P\rangle_i$, o $ce(s'') \leq \frac{3}{4}$.

    Dado que $P \subseteq_P P'$, podemos afirmar que $s' \in \bigcup_{\frac{3}{4}\leq i} \langle P'\rangle_i$, $s'' \in \bigcup_{i\leq 1} \langle P'\rangle_i$ y, por tanto (tabla causal), $s \in \bigcup_{\frac{3}{4}\leq i\leq 1} \langle f(P')\rangle_i$.

- Los demás casos son análogos y se siguen de la monotonicidad de los operadores $\wedge$, $\vee$, $\rightarrow$.

□

Demostración. **El transformador $f(Z) = \neg Z$ no es monótono.** Sea $P = \{s, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset\} \subseteq P' = \{\emptyset, s, \emptyset, \emptyset, \emptyset, \emptyset\}$ ($P \subseteq_P P'$). Entonces, bajo el transformador $f(Z) = \neg Z$, $f(P) = \{\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, s\}$ y $f(P') = \{\emptyset, \emptyset, \emptyset, \emptyset, s, \emptyset\}$. Por tanto $f(P) \not\subseteq_P f(P')$. □

Dado que bajo la sintaxis SCTL-T las variables requisito no son libres, tal requisito no se puede expresar en SCTL-T, y las fórmulas son, por definición, sintácticamente monótonas.

7.9.4 Dualidad de recursiones mínima y máxima

Demostraremos a continuación la dualidad de las recursiones mínimas y máximas:

- $\neg \ll I_R \gg = [\![\neg I_R]\!]$, donde, siendo $I_R = f(I_R)$ el transformador definido por $\ll I_R \gg$, $[\![\neg I_R]\!]$ define el transformador de predicados $\neg I_R = \neg f(\neg I_R)$.

- $\neg [\![I_R]\!] = \ll \neg I_R \gg$, donde, siendo $I_R = f(I_R)$ el transformador definido por $[\![I_R]\!]$, $\ll \neg I_R \gg$ define el transformador de predicados $\neg I_R = \neg f(\neg I_R)$.

Demostración. $\neg \ll \mathbf{I_R} \gg = [\![\neg \mathbf{I_R}]\!]$. Sea f el transformador de predicados de-

finido por $I_R = f(I_R)$, y $\neg f$ el definido por $\neg I_R = \neg f(\neg I_R)$. Para $Z \in \mathcal{P}^6$:

$$s \in \langle \neg \ll I_R \gg \rangle_i$$
$$s \in \langle \ll I_R \gg \rangle_{\neg i}$$
$$s \in \langle \bigcap_P \{Z | f(Z) \subseteq_P Z\} \rangle_{\neg i}$$
$$s \in \langle \bigcup_P \{\overline{Z}^P | f(Z) \subseteq_P Z\} \rangle_i$$

y como el operador $\neg$ es antimonotónico ($a \leq b$ sii $\neg a \geq \neg b$):

$$s \in \langle \bigcup_P \{\overline{Z}^P | \neg f(\overline{Z}^P) \supseteq_P \overline{Z}^P\} \rangle_i$$
$$s \in \langle [\![\neg I_R]\!] \rangle_i$$

□

Demostración. $\neg [\![\mathbf{I_R}]\!] = \ll \neg \mathbf{I_R} \gg$. Análoga. □

En general, dado que los operadores binarios $\wedge$, $\vee$ y $\rightarrow$ son monótonos, y que tenemos las dualidades:

- $\neg\ (\phi\ \forall \otimes \phi\,') = \phi\ \exists \otimes \neg \phi\,'$.
- $\neg\ (\phi\ \exists \otimes \phi\,') = \phi\ \forall \otimes \neg \phi\,'$.
- $\neg \ll I_R \gg = [\![\neg I_R]\!]$
- $\neg [\![I_R]\!] = \ll \neg I_R \gg$

y dado que la fórmulas son sintácticamente monótonas, cada fórmula en una recursión mínima o máxima es monótona; por tanto, sus transformadores serán monótonos, y garantizamos la existencia de un punto fijo mínimo y un punto fijo máximo (Teorema de Tarski-Knaster (Tarski, 1955)). Además, como veremos en el algoritmo de *model checking* (capítulo 8), si $\mathcal{ST}$ es finito se puede realizar el cómputo de forma iterativa.

7.10 Operadores macro en SCTL-T

En esta sección se ilustra la expresividad y flexibilidad de SCTL-T construyendo a partir de sus operadores básicos un conjunto de operadores macro comúnmente utilizados en otras lógicas ramificadas. La flexibilidad de SCTL-T le permitirá adaptarse a las necesidades de cada ámbito de aplicación específico.

Dado que SCTL-T es una lógica de reloj explícito, los operadores macro definidos a continuación son operadores genéricos que no incluyen referencia al tiempo.

Además, el apartado 7.10.2 de esta sección evalúa la expresividad de SCTL-T comparándola con TCTL (*Timed Computation Tree Logic*), T_μ (*Timed μ-calculus*) y L_ν (*Timed Modal Logic*).

7.10.1 Operadores macro

Por simplicidad, se pueden definir una amplia gama de macros de operadores temporales de uso común en las lógicas ramificadas. Así, bajo los símbolos:

- $\mathcal{G}$ (*it is always Going to be the case*), $\mathcal{F}$ (*at least once in the Future*)
- $\mathcal{H}$ (*it Has always been the case*), $\mathcal{P}$ (*at least once in the Past*)
- $\mathcal{U}$ (*Until*), $\mathcal{S}$ (*Since*)
- $\mathcal{U}_w$ (*Weak Until*), $\mathcal{S}_w$ (*Weak Since*)

y siendo $\mathcal{A}$ la cuantificación universal y $\mathcal{E}$ la cuantificación existencial, definimos las siguientes correspondencias:

Operadores de futuro:

- Finalmente $\mathcal{AF}[\phi]$:

$$R_{\mathcal{AF}} ::= \phi \vee \forall \Rightarrow_+ (\phi \vee \forall \Rightarrow \bigcirc \ll R \gg)$$

- Finalmente $\mathcal{EF}[\phi]$:

$$R_{\mathcal{EF}} ::= \phi \vee \exists \Rightarrow_+ (\phi \vee \exists \Rightarrow \bigcirc \ll R \gg)$$

- Globalmente $\mathcal{AG}[\phi]$:

$$R_{\mathcal{AG}} ::= \phi \wedge \forall \Rightarrow_+ (\phi \wedge \forall \Rightarrow \bigcirc [\![R]\!])$$

- Globalmente $\mathcal{EG}[\phi]$:

$$R_{\mathcal{EG}} ::= \phi \wedge \exists \Rightarrow_+ (\phi \wedge \exists \Rightarrow \bigcirc [\![R]\!])$$

- Hasta $\mathcal{A}[\phi \ \mathcal{U} \ \phi']$:

$$R_{\mathcal{AU}} ::= \phi' \vee (\phi \wedge \forall \Rightarrow_+ (\phi' \vee (\phi \wedge \forall \Rightarrow \bigcirc \ll R \gg)))$$

- Hasta $\mathcal{E}[\phi \ \mathcal{U} \ \phi']$:

$$R_{\mathcal{EU}} ::= \phi' \vee (\phi \wedge \exists \Rightarrow_+ (\phi' \vee (\phi \wedge \exists \Rightarrow \bigcirc \ll R \gg)))$$

- A menos que $\mathcal{A}[\phi \ \mathcal{U}_w \ \phi']$:

$$R_{\mathcal{AU}_w} ::= \phi' \vee (\phi \wedge \forall \Rightarrow_+ (\phi' \vee (\phi \wedge \forall \Rightarrow \bigcirc [\![R]\!])))$$

- A menos que $\mathcal{E}[\phi \ \mathcal{U}_w \ \phi']$:

$$R_{\mathcal{EU}_w} ::= \phi' \vee (\phi \wedge \exists \Rightarrow_+ (\phi' \vee (\phi \wedge \exists \Rightarrow \bigcirc [\![R]\!])))$$

Operadores de pasado:

- Finalmente $\mathcal{AP}[\phi]$:

$$R_{\mathcal{AP}} ::= \phi \vee \forall \Rightarrow_- (\phi \vee \forall \Rightarrow \odot \ll R \gg)$$

- Finalmente $\mathcal{EP}[\phi]$:

$$R_{\mathcal{EP}} ::= \phi \vee \exists \Rightarrow_{-} (\phi \vee \exists \Rightarrow \odot \ll R \gg)$$

- Globalmente $\mathcal{AH}[\phi]$:

$$R_{\mathcal{AH}} ::= \phi \wedge \forall \Rightarrow_{-} (\phi \wedge \forall \Rightarrow \odot [\![R]\!])$$

- Globalmente $\mathcal{EH}[\phi]$:

$$R_{\mathcal{EH}} ::= \phi \wedge \exists \Rightarrow_{-} (\phi \wedge \exists \Rightarrow \odot [\![R]\!])$$

- Desde $\mathcal{A}[\phi \ \mathcal{S} \ \phi']$:

$$R_{\mathcal{AS}} ::= \phi' \vee (\phi \wedge \forall \Rightarrow_{-} (\phi' \vee (\phi \wedge \forall \Rightarrow \odot \ll R \gg)))$$

- Desde $\mathcal{E}[\phi \ \mathcal{S} \ \phi']$:

$$R_{\mathcal{ES}} ::= \phi' \vee (\phi \wedge \exists \Rightarrow_{-} (\phi' \vee (\phi \wedge \exists \Rightarrow \odot \ll R \gg)))$$

- A menos que $\mathcal{A}[\phi \ \mathcal{S}_w \ \phi']$:

$$R_{\mathcal{AS}_w} ::= \phi' \vee (\phi \wedge \forall \Rightarrow_{-} (\phi' \vee (\phi \wedge \forall \Rightarrow \odot [\![R]\!])))$$

- A menos que $\mathcal{E}[\phi \ \mathcal{S}_w \ \phi']$:

$$R_{\mathcal{ES}_w} ::= \phi' \vee (\phi \wedge \exists \Rightarrow_{-} (\phi' \vee (\phi \wedge \exists \Rightarrow \odot [\![R]\!])))$$

Notar que, en esta traducción, las fórmulas $\exists \Rightarrow_{+} \phi$ se interpretan como un paso temporal que lleva a un estado en el que la fórmula ϕ se satisface. Tal interpretación hace que, por ejemplo, el operador $\mathcal{EG}[\phi]$ no se corresponda con la semántica habitual, en la que existe una ejecución en la que ϕ se satisface siempre. El operador $\mathcal{EG}[\phi]$ indica que existe una traza con pasos temporales y discretos, en la que todos los estados alcanzables por dichos pasos satisfacen ϕ. La interpretación continua se puede alcanzar definiendo un operador $\exists \Rightarrow_{+}^{\mathcal{U}}$ con semántica *until*. Así, $\phi \ \exists \Rightarrow_{+}^{\mathcal{U}} \phi'$, se satisface si existe un estado alcanzable por una transición temporal, tal que ϕ' se satisface y, para todos los estados en dicho paso temporal se satisface $\phi \vee \phi'$.

$$\phi \ \exists \Rightarrow_{+}^{\mathcal{U}} \phi' \equiv R ::= \phi' \vee (\phi \wedge \forall \Rightarrow_{+} \ll R \gg)$$

Si bien la inclusión $\forall \Rightarrow_{+}$ puede parecer en contradicción con la semántica existencial del operador definido, es consecuencia de que si bien se ha utilizado una semántica ramificada para el tiempo, éste es determinista.

Así, en la figura 7.11, si a partir de un estado existe un paso temporal satisfaciendo el *until*, todas las transiciones temporales lo satisfarán para algún instante temporal intermedio. Con tal operador, definimos $\mathcal{EG}^{\mathcal{U}}[\phi]$ como:

$$R_{\mathcal{EG}^{\mathcal{U}}} ::= \phi \wedge \exists \Rightarrow_{+}^{\mathcal{U}} (\phi \wedge \exists \Rightarrow \bigcirc [\![R_{\mathcal{EG}^{\mathcal{U}}}]\!])$$

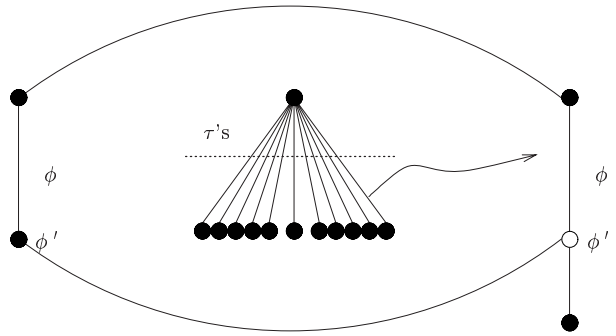


Figura 7.11. Tiempo determinista

7.10.2 Expresividad

La lógica SCTL-T puede ser comparada, en cuanto a expresividad, con distintas lógicas de similares características. Restringiendo el conjunto de valores de verdad a un conjunto booleano (SCTL-T$_{bool}$) y como se sigue de la traducción en 7.10.1, SCTL-T$_{bool}$ permite expresar todas las propiedades TCTL. Sin embargo, dado que SCTL-T, a diferencia de TCTL, permite distinguir de forma explícita las transiciones continuas y discretas en el modelo, una propiedad del tipo:

$$a \ \ \forall\Rightarrow_{+} \ \ b$$

no es expresable en TCTL. Podemos afirmar, por tanto, que $\mathcal{L}_{SCTL-T_{bool}} > \mathcal{L}_{TCTL}$.

Dicha distinción explícita en SCTL-T hace que sea equiparable a L_ν (Laroussinie et al., 1995) [5]. En L_ν, bajo una interpretación modal, se incluyen los operadores:

- $< a >$, $[a]$: denotando la cuantificación universal y existencial sobre transiciones discretas a, y, por tanto, equiparables a $a \ \ \forall\Rightarrow\bigcirc$ y $a \ \ \exists\Rightarrow\bigcirc$.
- $\exists\ \phi$, $\forall\ \phi$: denotando la cuantificación existencial y universal sobre transiciones temporales, y, por tanto, equiparables $\emptyset \ \ \forall\Rightarrow_{+}$ y $\emptyset \ \ \exists\Rightarrow\bigcirc$.

En lo que se refiere a T$_\mu$, se puede afirmar que $\mathcal{L}_{SCTL-T_{bool}} < \mathcal{L}_{\mathrm{T}_\mu}$, dado que SCTL-T no permite la alternancia de puntos fijos ni la utilización de variables requisito libres.

7.11 Conclusiones

En este capítulo se ha definido la lógica de tiempo real SCTL-T, tomando su semántica en modelos MUS-T. Como se sigue de la definición en la sección 6.6.6, podemos decir que para un modelo MUS-T no determinista:

[5] Aunque L_ν sólo incluye recursión máxima.

- Una subfórmula causal $\phi_1 \exists \otimes \phi_2$, tal que el grafo de satisfacción de ϕ_1 es 1, podrá cumplirse (1) en el modelo. Sin embargo, nunca podrá incumplirse, dado que siempre es posible construir un modelo MUS-T por evolución del modelo MUS-T actual, cumpliendo la cuantificación existencial en la subfórmula.

- Una subfórmula causal $\phi_1 \forall \otimes \phi_2$, tal que el grafo de satisfacción de ϕ_1 es 1, podrá incumplirse (0) en el modelo. Sin embargo, nunca podrá cumplirse, dado que siempre es posible construir un modelo MUS-T por evolución del modelo MUS-T actual, incumpliendo la cuantificación universal en la subfórmula.

Consideraciones similares a las anteriores, se pueden realizar para el resto de grados de satisfacción de una fórmula SCTL-T sobre un modelo no determinista.

Aunque desde un punto de vista formal se persigue la definición de lógicas lo más expresivas posible, desde un punto de vista práctico es suficiente con conseguir una lógica que sea capaz de expresar los criterios de corrección del ámbito de aplicación específico. Así, si bien la expresividad es una característica deseable, ha de ser ponderada por la facilidad de expresión de los criterios de corrección usuales en la lógica bajo consideración. Si nos movemos al campo de la concisión o naturalidad, se adivina que la lógica SCTL-T es sintácticamente más pesada que TCTL. Siguiendo a (Stirling y Bradfield, 2001), la sola inclusión de los puntos fijos explícitos hace, para una gran mayoría, la lógica difícilmente comprensible.

Las razones anteriores, aunque de peso a la hora de definir lógicas "demasiado" expresivas, pueden ser rebatidas en parte:

- La definición de operadores macros dependientes del ámbito de aplicación específico pueden ocultar la expresividad "innecesaria" en cada caso.

- La definición de una semántica lo más "fina" posible permite la utilización de una misma base formal para objetivos no tan exigentes.

Sin embargo, y aunque es posible ocultar la complejidad sintáctica en las fórmulas SCTL-T proporcionando la naturalidad de otras lógicas, la flexibilidad que ello permite no es sin costo, resultando en algoritmos, que si bien de un orden de complejidad idéntico a los de otras lógicas, se interpreta sobre estructuras con un número de estados mayor. Remitimos a la traducción a *Timed Automaton* en la sección 6.9.

Apéndices del capítulo

7.A Almacenamiento de Requisitos

Un requisito SCTL-T:

$$\{I_0 = \phi_0, \cdots, I_n = \phi_n\}$$

se almacenará en una estructura de fórmulas SCTL-T indexada con los identificadores I_R:

```
class Requisito {
    list<string> identificadores;
    dictionary<string,list<constructor_sctlt>> declaraciones;
    list<string> relojes_modelo;
    list<string> relojes_futuro;
    list<string> relojes_pasado;
}
```

donde cada elemento I_i en requisito almacenará la fórmula ϕ_i en notación inversa, mediante una lista de constructor_sctlt, siendo constructor_sctlt:

```
constructor_sctlt:=
    Efreeze (reloj)
    | Epredicado (predicado)
    | Eaccion (accion)
    | Ecuantificador (∀ | ∃)
    | Eoperador_temporal (⇒ ◯ | ⇒ ⊙ | ⇒+ | ⇒- | ⇒)
    | Eoperador_logico (∧ | ∨ | ¬)
    | ErecursionMin (I_R)
    | ErecursionMax (I_R)
    | Einstanciacion (I_R)
```

7.B Algoritmo de requisitos bien formados y acciones de aplicabilidad

El algoritmo de detección de requisitos bien formados (algoritmo 7.1) calcula, a partir de una estructura de tipo Requisito, si el requisito es bien formado, en cuyo caso calcula el conjunto de relojes de modelo ($\mathcal{C}$), de futuro ($\mathcal{E}^f$), y de pasado ($\mathcal{E}^p$).

El pseudocódigo en 7.2 muestra el cómputo de las acciones de aplicabilidad para una fórmula f.

Algoritmo 7.1 Detección de requisitos bien formados de una estructura Requisito

```
RBF(string ir, int n, dir op, const tipos relojes)
{
    let f=declaraciones.inf(ir);
    case f[n]
        Efreeze (r): relojes+= (reloj,op);
        Epredicado (predicado):
            foreach r ∈ predicado do
                if (r ∉ relojes) relojes_modelo.push(reloj);
                else if (relojes(r).inf=ERROR) ERROR_NBF;
                    else if (relojes(r).inf=F) relojes_futuro.push(reloj);
                    else if (relojes(r).inf=P) relojes_pasado.push(reloj);
            end foreach
        Eaccion (accion)= n+=1;
        Ecuantificador (c): n+=1;
            case f[n]:
                ⇒ ◯ | ⇒+:op=F;
                ⇒ ⊙ | ⇒−:op=P;
                ⇒:op=?
            end case
            n+=1; RBF(R,is,n,op);
            foreach r ∈ relojes do
                if (op != ?) AND (relojes(r).inf != op)
                relojes(r).inf=ERROR;
            end foreach ;
            n+=1; RBF(R,is,n,?,relojes);
        Eoperador_logico (ol):
            case ol:
                AND | OR:
                    n+=1;RBF(R,id,n,op,relojes);
                    n+=1;RBF(R,id,n,op,relojes);
                NOT:
                    n+=1;RBF()
            end case
        ErecursionMin (I_R) | ErecursionMax (I_R):
            if identificadores.rank(I_R) > identificadores.rank(ir) ERROR_NBF;
        Einstanciacion (I_R):
            if identificadores.rank(I_R) ≤ identificadores.rank(ir) ERROR_NBF;
            else n=0; RBF(R,I_R,0,op,relojes);
    end case
}
```

Algoritmo 7.2 Cálculo de acciones de aplicabilidad para fórmula f en FNP

```
list<string> acciones_aplicabilidad (list<constructor_sctlt> f,int n)
{
    let acciones={};
    case (f[n])
        Eaccion (accion):
            case accion:
                ∅, true: acciones.push(A);
                    return acciones;
                false: acciones.clear();
                    return acciones;
                default: acciones.push(accion);
            n+=1;
        Ecuantificador (Æ):
            n+=2;
            acciones.push(acciones_aplicabilidad(f,n));
            acciones_aplicabilidad(n);
        Eoperador_logico (ol):
            case ol
                NOT: n+=2;
                AND | OR: n+=1;
                acciones.push(acciones_aplicabilidad(f,n));
                acciones.push(acciones_aplicabilidad(f,n));
            end case
        Efreeze (reloj): n+=1;
            acciones.push(acciones_aplicabilidad(f,n));
        else : n=n+1;
    end case
    if acciones.empty() then acciones.push(A);
    return acciones;
}
```

PARTE IV

Mecanismos de análisis y síntesis en el proceso

CAPÍTULO 8

Model Checking SCTL-T sobre MUS-T

En este capítulo se presenta un algoritmo de model checking para propiedades especificadas en SCTL-T sobre modelos MUS-T. Este algoritmo se basa en el concepto de grafo de regiones introducido en (Alur et al., 1993a). El grafo de regiones es una abstracción finita, basada en la equivalencia de regiones, de un modelo de estados denso. Bajo una equivalencia similar se define el grafo RMUS-T, grafo cociente del modelo MUS-T que, adicionalmente, distingue las distintas condiciones de especificación de los estados del modelo subyacente y permite verificar distancias temporales en el pasado.

8.1 Introducción

En cualquier instante de tiempo, el comportamiento futuro de un modelo MUS-T está totalmente determinado por su estado de control y las interpretaciones de todos los relojes. El conjunto de comportamientos que un modelo MUS-T puede exhibir se obtiene del grafo denso de estados (s, γ) que da semántica al modelo (sección 6.6). Sin embargo, a la hora de diseñar un algoritmo de *model checking* para llevar a cabo la verificación de una propiedad SCTL-T sobre un modelo MUS-T hay que partir de una premisa fundamental: el número de estados a explorar ha de ser necesariamente finito. En consecuencia, la solución al problema de verificación vía *model checking* pasa por obtener una representación finita del modelo. El problema de verificación de modelos con un número de estados infinito (o finito, pero de un tamaño inabordable) no es exclusivo del modelado de sistemas de tiempo real. De hecho, en los mecanismos utilizados en *model checking* con tiempo subyace la aplicación de abstracciones, ya utilizadas en formalismos sin tiempo (Cousot y Cousot, 1977).

Una abstracción es, en general, una relación entre un espacio de estados concretos y un espacio de estados abstracto. Para su aplicación al problema de *model*

checking, la clave radica en encontrar un espacio de estados abstracto, de forma que los distintos estados concretos en un estado abstracto no sean distinguibles desde el punto de vista de la verificación.

Ilustramos la idea de las abstracciones con el siguiente ejemplo [1]. Supongamos dos procesos concurrentes que deben acceder a un recurso compartido. La regulación de acceso se realiza mediante una variable entera n. Uno de los procesos accede al recurso cuando n es par y el otro cuando n es impar. Además, cuando los procesos abandonan el recurso compartido, actualizan el valor de la variable n de una forma conocida.

```
wait(par(n)=true);        wait(par(n)=false);
acceso;                   acceso;
n=n+1;                    n=n+1;
```

La verificación mediante *model checking* de la exclusión mutua en el sistema pasa por verificar un conjunto de estados de tamaño infinito (originado en la variable n). Pero, sin duda, no todos los estados son desde un punto de vista funcional distintos; el conjunto de posibles interpretaciones de la variable n puede dividirse en el conjunto abstracto de los pares y el conjunto abstracto de los impares. Además, dado que la forma de actualización de la variable n es conocida, sabemos la forma en que la interpretación abstracta del sistema evoluciona de un conjunto abstracto a otro.

Nos referimos ahora a un modelo MUS-T $\mathcal{M}$ cualquiera, en el que puntos importantes coinciden con el ejemplo anterior:

- El número de estados del grafo semántico $\mathcal{S}_{\mathcal{M}}$ es infinito.
- No todos los estados son distinguibles. Dado que las ejecuciones del programa dependen de las restricciones de tiempo impuestas en las guardas e invariantes del modelo, sólo son distinguibles aquellos estados distinguibles por la gramática definida para dichas restricciones.
- La forma en que las interpretaciones de los relojes cambian es conocida: todos los relojes evolucionan al mismo ritmo, y son eventualmente reiniciados en ciertas transiciones del modelo.

Aunque la idea es sencilla, el problema de encontrar una abstracción correcta no es para nada trivial. La primera solución al problema de *model checking* con semántica de tiempo denso la encontramos en los trabajos de (Alur et al., 1993a), en donde se presenta un algoritmo de *model checking* para la lógica de tiempo real TCTL sobre un autómata temporizado con relojes de valores reales. El avance crucial radica en que el espacio de estados infinito, resultante de la asignación de valores a los relojes del sistema, se puede particionar en un espacio cociente finito mediante una abstracción definida sobre los valores de los relojes en el autómata. De esta forma, el dominio de interpretaciones de los relojes se abstrae a un conjunto finito de clases de equivalencia (con un número de elementos infinito) a los que se denomina **regiones de tiempo**, tales que las interpretaciones en la misma región inducen estados que satisfacen las mismas propiedades lógicas en

[1] El ejemplo es una variante del que se puede encontrar en (Dams et al., 1997).

TCTL. Posteriormente, dicho resultado fue extendido a la lógica T_μ (Henzinger et al., 1992b), de la cual TCTL se puede considerar un fragmento. Además del *model checking* sobre TCTL y T_μ, la construcción de un espacio cociente finito ha abierto la resolución de numerosos problemas para modelos de tiempo denso: relaciones de orden y equivalencia entre modelos, vacuidad, alcanzabilidad y síntesis de controladores.

En este capítulo introduciremos, en primer lugar, el concepto de regiones de tiempo (sección 8.2); seguidamente, explicaremos las bases teóricas para la construcción del grafo de regiones RMUS-T (sección 8.3); y por último, el algoritmo de verificación que, aplicado al grafo RMUS-T, puede reducirse al caso no temporizado de verificación de una fórmula SCTL sobre un modelo MUS (sección 8.4).

8.2 Regiones de tiempo (*clock regions*)

La idea subyacente bajo el concepto de regiones nace de la aplicación de dos abstracciones sobre el conjunto de interpretaciones de los relojes definidos en un modelo.

Una primera abstracción se basa en que dados dos estados con idéntico estado de control e interpretaciones γ y γ', si las interpretaciones γ y γ' coinciden en la parte entera de cada uno de los valores de todos los relojes y también en el orden de las partes fraccionarias de ellos, entonces, las ejecuciones que parten de dichos estados serán similares. Las partes enteras de los valores de los relojes determinan cuándo se cumple o no una restricción del modelo en concreto, mientras el orden de las partes fraccionarias de los relojes permite decidir qué reloj cambiará antes su parte entera. Esta primera abstracción nos lleva a un espacio abstracto en el que el valor exacto de los relojes no es considerado: el espacio abstracto sólo registra las partes enteras de los relojes y el orden de sus partes fraccionarias. Sin embargo, el espacio así obtenido sigue siendo un espacio de estados infinito.

Una segunda abstracción permite acotar el espacio de estados observando que, aunque las partes enteras de los valores de los relojes pueden ser arbitrariamente grandes, una vez superan un cierto valor conocido, el valor de dicha parte entera no afecta a las evoluciones futuras.

Sea $\mathcal{M}$ un modelo MUS-T, se denota $\Psi_{\mathcal{M}}$ al conjunto de restricciones formado por las guardas e invariantes de $\mathcal{M}$ definidas según la gramática en la definición 6.4 [2]. Así, por ejemplo, sea $\mathcal{M}$ un modelo MUS-T con

$$\Psi_{\mathcal{M}} = \{x = 2, y > 1, y > 0, x \leq 1\}$$

Una vez el valor de x en γ supera la constante 2, su valor concreto es irrelevante, dado que nunca va a ser comparado con una constante mayor que 2 (igualmente para el reloj y y la constante entera 1).

La aplicación de la primera de las abstracciones sobre el espacio de interpretaciones de los relojes produciría la partición en la figura 8.1(a). La aplicación de la

[2] La definición original en (Alur et al., 1993a) no incluye la diferencia entre relojes, pero el mecanismo para la solución del problema es el mismo.

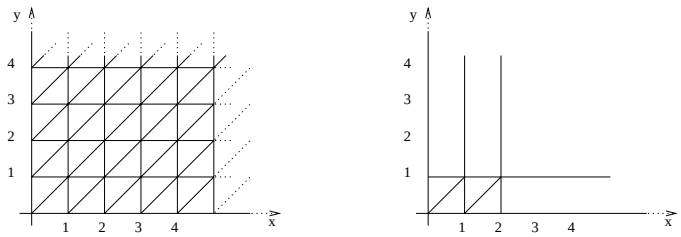


Figura 8.1. Partición del espacio de estados de las interpretaciones de los relojes, sin constantes máximas 8.1(a) y con constantes máximas 8.1(b).

segunda abstracción acota dicha partición, produciendo un espacio abstracto de dimensión finita (figura 8.1(b)).

Esta idea se formaliza definiendo una relación de equivalencia sobre el espacio de interpretaciones de los relojes. Para cualquier $t \in \Re_+$, $frac(t)$ denota la parte fraccionaria de t y $ent(t)$ la parte entera de t. Para cada reloj x en $\mathcal{C}$, y para cada par de relojes x, y en $\mathcal{C}$, sea m_x el mayor entero no negativo en alguna restricción $x \prec c$ en el conjunto $\Psi_{\mathcal{M}}$, y m_{xy} el mayor entero no negativo que interviene en alguna restricción $x - y \prec c$ en el conjunto $\Psi_{\mathcal{M}}$.

La relación de equivalencia $\cong_t$ bajo $\Psi_{\mathcal{M}}$ se define sobre el conjunto de interpretaciones $\Re_+^{\#\mathcal{C}}$. Dos interpretaciones son equivalentes, $\gamma \cong_t \gamma'$, si se cumplen las siguientes condiciones [3]:

- Para todo $x \in \mathcal{C}$, o $ent(\gamma(x)) = ent(\gamma'(x))$, o tanto $ent(\gamma(x))$ como $ent(\gamma'(x))$ son estrictamente mayores que m_x.
- Para todo $x, y \in \mathcal{C}$, o $ent(\gamma(x) - \gamma(y)) = ent(\gamma'(x) - \gamma'(y))$, o tanto $\gamma(x) - \gamma(y)$ como $\gamma'(x) - \gamma'(y)$ son estrictamente mayores que m_{xy}.

En base a esta relación de equivalencia, se definen las regiones de tiempo $\mathcal{RT}$ inducidas por un modelo $\mathcal{M}$ como las clases de equivalencia de las interpretaciones de los relojes bajo la relación $\cong_t$, con constantes m obtenidas de $\Psi_{\mathcal{M}}$.

Así, en la figura 8.1(b), correspondiente a un modelo con $m_x = 2$, $m_y = 1$ y $m_{xy} = m_{yx} = 0$, obtenemos un espacio abstracto de 28 regiones de tiempo: 6 puntos (por ejemplo, $(0, 1)$); 14 segmentos (por ejemplo, $0 < x = y < 1$) y 8 zonas (por ejemplo, $0 < x < y < 1$). La partición es tal que, para cualquier restricción ψ en $\Psi_{\mathcal{M}}$, o bien todas las interpretaciones γ de una región de tiempo satisfacen ψ, o bien ninguna la satisface.

Definición 8.1. **Región de tiempo.** Cada región de tiempo $\mathcal{RT}$ se puede representar especificando:

[3] La relación de equivalencia es ligeramente distinta a la original en (Alur et al., 1993a) debido a la incorporación de diferencias entre relojes.

- Para cada reloj x, una restricción sobre x de entre las del conjunto:

$$\{x = n_x, n_x - 1 < x < n_x, x > m_x\} \text{ con } n_x = 0, 1, \cdots, m_x$$

  Un reloj x se dice acotado si $n_x - 1 < x < n_x$, o $x = n_x$. En otro caso, se dice no acotado ($x > m_x$).

- Para todo par de relojes x, y tales que $n_x - 1 < x < n_x$ y $n_y - 1 < y < n_y$ aparezcan en la definición de la región para algún n_x y n_y, si $frac(x)$ es menor, igual o mayor que $frac(y)$, o, equivalentemente, si $x - y$ es menor, igual o mayor que $n_x - n_y$.

- Para todo par de relojes x, y tales que x es un reloj no acotado ($x > m_x$), e y es un reloj acotado ($n_y - 1 < y < n_y$ o $y = n_y$), una restricción de entre las del conjunto:

$$\{x - y = n_{xy}, n_{xy} - 1 < x - y < n_{xy}, x - y > m_{xy}\}$$

  con $n_{xy} = (m_x - n_y), \cdots, m_{xy}$.

- Para todo par de relojes x, y tales que tanto x como y son relojes no acotados, una restricción de entre las del conjunto:

$$\{x - y = n_{xy}, n_{xy} - 1 < x - y < n_{xy}, x - y > m_{xy}\}$$

  con $n_{xy} = 0, \cdots, m_{xy}$.

Una región de tiempo es abierta si no contiene ninguna restricción del tipo $x = c$; en otro caso, se dirá cerrada. Una región de tiempo se dice final sii, para todos los relojes x, las restricciones de la región son del tipo $x > m_x$. Además, una región de tiempo se dice inicial sii, existe algún reloj x tal que $x = 0$. El número de regiones estará acotado por $n! \cdot 2^n \cdot \prod_{x \in \mathcal{C}}(2m_x + 2)$, donde n es el número de relojes en el sistema (Alur et al., 1993a).

Siendo $\mathcal{RT}$, $\mathcal{RT}'$ dos regiones de tiempo, $\mathcal{RT}'$ es sucesor (en tiempo) de $\mathcal{RT}$ ($RTsuc_t(\mathcal{RT})$), sii para todo $\gamma \in \mathcal{RT}$, existe un $\tau \in \Re_+$ tal que $\gamma + \tau \in \mathcal{RT}'$, y $\gamma + \tau' \in \mathcal{RT} \cup \mathcal{RT}'$ para todo $\tau' < \tau$. De forma inversa, se define el operador predecesor en tiempo $RTpre_t(\mathcal{RT})$: $\mathcal{RT}'$ es predecesor (en tiempo) de $\mathcal{RT}$, sii para todo $\gamma \in \mathcal{RT}'$, existe un $\tau \in \Re_+$ tal que $\gamma + \tau \in \mathcal{RT}$, y $\gamma + \tau' \in \mathcal{RT}' \cup \mathcal{RT}$ para todo $\tau' < \tau$.

Observando la figura 8.1(b), la región de tiempo sucesora de una región de tiempo $\mathcal{RT}$ es aquella región de tiempo que se encuentra primero a partir de una diagonal $x - y = c$ que parte de algún punto de la región de tiempo original en dirección positiva. De igual forma para la región de tiempo predecesora, pero con dirección negativa. Una región de tiempo final no tiene sucesor y una región de tiempo inicial no tiene predecesor en tiempo.

En el apéndice 8.A se detalla la implementación realizada para el almacenamiento de regiones de tiempo, así como las operaciones $RTpre_t$ y $RTsuc_t$ sobre regiones de tiempo.

8.3 Grafo de regiones RMUS-T

El análisis de modelos con tiempo basado en la relación de equivalencia definida pasa por la construcción de un grafo denominado grafo de regiones. El grafo de regiones se construye extendiendo la relación de equivalencia en tiempo $\cong_t$, definida sobre el espacio de interpretaciones de los relojes, a una relación de equivalencia $\cong$ sobre el espacio de estados (s, γ) del grafo semántico. En dicha relación, dos estados serán equivalentes si sus localizaciones de control son idénticas y las interpretaciones de los relojes son equivalentes bajo la relación de equivalencia sobre regiones de tiempo:

$$(s, \gamma) \cong (s', \gamma') \text{ sii } s = s' \text{ y } \gamma \cong_t \gamma'$$

Basándonos en la relación de equivalencia $\cong$, construimos un grafo abstracto RMUS-T, cociente del grafo semántico definido en 6.6.3. El grafo RMUS-T tendrá un conjunto finito de vértices a los que denominamos regiones, que incluyen una región de tiempo e información discreta del modelo, y un número finito de transiciones. Las transiciones en el grafo RMUS-T son una abstracción de las transiciones $\mathcal{T} \subseteq (S \times \Re_+^{\#\mathcal{C}}) \times (A \cup \Re_+) \times \mathcal{L}_3 \times (S \times \Re_+^{\#\mathcal{C}})$, definidas en la sección 6.6.2, a $(S \times \mathcal{RT}) \times (A \cup \{t\}) \times \mathcal{L}_3 \times (S \times \mathcal{RT})$, siendo $\mathcal{RT}$ las regiones en la definición 8.2 y t una etiqueta de paso temporal.

Dado que tanto el número de estados de control como el de regiones de tiempo es finito, también lo será el grafo de regiones. A continuación se introducen las características de los estados (sección 8.3.1) y transiciones (sección 8.3.2) del modelo abstracto RMUS-T.

8.3.1 Estados interpretados del grafo de regiones

A diferencia de un modelo MUS-T, en el grafo RMUS-T los vértices contendrán estados de control interpretados bajo una región de tiempo. Un estado interpretado bajo el contexto de una región de tiempo es un estado de control en el que la información que proporciona la región de tiempo ha permitido seleccionar para cada acción a la condición de especificación que rige en dicha región. Dado que $\{g_{p_i}(s,a), g_{np}(s,a), g^P_{s_j}(s,a), g^T_s(s,a)\}$ son guardas disjuntas que cubren $\Re_+^{\#\mathcal{C}}$ y que la granularidad de las regiones permite distinguir este tipo de guardas, siempre regirá una y sólo una guarda.

Un estado de control s interpretado bajo la región de tiempo $\mathcal{RT}$ ($s^{\mathcal{RT}}$) define el conjunto de acciones posibles (A_P), no posibles (A_{NP}) y subespecificadas (A_S) en el estado s para la región de tiempo $\mathcal{RT}$. En dicho estado, $s^{\mathcal{RT}}$, las guardas de las acciones del estado de control s han desaparecido y, además, se ha seleccionado para cada acción del alfabeto A su condición de especificación actual: posible, no posible o subespecificada.

Sea un estado s del modelo MUS-T y una región de tiempo $\mathcal{RT}$ bajo $\cong_t$. Los conjuntos A_P, A_{NP} y A_S, caracterizando el estado interpretado $s^{\mathcal{RT}}$, se obtienen aplicando la función interpreta(s,RT) (algoritmo 8.1), donde la operación DBM representa la región de tiempo como un DBM (apéndice 8.A).

Ejemplo 8.1. Así, por ejemplo, para un modelo MUS-T con una sola acción a y un

Algoritmo 8.1 interpreta(s,RT)

```
interpreta(s,RT)
{ /* con s estado de control, y RT región de tiempo */
    A_P:={}; A_NP:={}; A_S:={}

    foreach a ∈ A
        if (DBM(RT) ⊆ DBM(g_pi(s,a))) then A_P:=A_P ∪ {a}
        else if (DBM(RT) ⊆ DBM(g_np(s,a))) then A_NP:=A_NP ∪ {a}
            else A_S:=A_S ∪ {a}
        end else
    end foreach
    return A_P,A_NP,A_S
}
```

solo reloj x, en un estado de control s_1 con $g_p(s_1, a) = (x = 3)$ como única guarda posible, $g_{np}(s_1, a) = (x < 3)$, y $g_s(s_1, a) = (x > 3)$, podremos tener regiones en el grafo RMUS-T del tipo:

- $s_1^{\{x=3\}}$, con $A_P = \{a\}$, $A_{NP} = \{\}$, $A_S = \{\}$.
- $s_1^{\{2<x<3\}}$, con $A_P = \{\}$, $A_{NP} = \{a\}$, $A_S = \{\}$.
- $s_1^{\{3<x<4\}}$, con $A_P = \{\}$, $A_{NP} = \{\}$, $A_S = \{a\}$.

□

Con todo lo expuesto, un vértice o región $\mathcal{R}$ del grafo cociente RMUS-T es una tupla $< s^{\mathcal{RT}}, \mathcal{RT} >$, donde $\mathcal{RT}$ es una región de tiempo del espacio $\Re_+^{\#\mathcal{C}}$ de las interpretaciones de los relojes y $s^{\mathcal{RT}}$ es un estado de control de un modelo MUS-T interpretado bajo dicha región $\mathcal{RT}$, definiendo, por tanto, los conjuntos $A_P(s^{\mathcal{RT}})$, $A_{NP}(s^{\mathcal{RT}})$, $A_S(s^{\mathcal{RT}}) \subseteq A$.

El estado sumidero subespecificado, s_s, es tal que $\{A_S(s_s^{\mathcal{RT}}) = A, \forall\ \mathcal{RT}\}$. Igualmente, el estado sumidero no posible, s_{np}, es tal que $\{A_{NP}(s_{np}^{\mathcal{RT}}) = A, \forall\ \mathcal{RT}\}$.

En adelante notaremos, por simplicidad, al par $< s^{\mathcal{RT}}, \mathcal{RT} >$ como $< s, \mathcal{RT} >$. Como veremos, los conjuntos $\{A_P(s^{\mathcal{RT}}), A_{NP}(s^{\mathcal{RT}}), A_S(s^{\mathcal{RT}})\}$ definen la condición de especificación c_A de una acción a en una región $< s, \mathcal{RT} >$.

8.3.2 Transiciones

Mientras en el grafo MUS-T $\mathcal{M}$ tenemos acciones y guardas, y una misma acción puede aparecer en un mismo estado como posible y como no posible, en el grafo RMUS-T no existen guardas y las transiciones posibles ($\rightarrow$), no posibles ($\nrightarrow$), o subespecificadas ($\rightsquigarrow$) se etiquetan en el conjunto de acciones A o con una etiqueta t que indica el progreso del tiempo. En consecuencia, la evolución de una región a otra podrá responder a un paso temporal t (que modifica la región de tiempo), o a una acción ($\in A$) que supone una transición discreta (de duración 0) de un estado de control a otro en el modelo MUS-T.

Sea una región $\mathcal{R} =< s, \mathcal{RT} >$ para un modelo MUS-T $\mathcal{M}$. Las transiciones que parten de dicha región se definen como:

Transiciones discretas con etiqueta $a \in A$:

- Posible (para $t \in T$ con $CE(t) = 1$). Para $t = (s, < a, g_p(s,a), \lambda >, s')$, existe una transición discreta posible:

$$< s, \mathcal{RT} > \xrightarrow{a} < s', R_\lambda(\mathcal{RT}) >$$ [4]

  sii $\mathcal{RT} \subseteq \mathcal{P}(g_p(s,a))$, siendo $\mathcal{P}(g_p(s,a))$ el poliedro definido por la guarda $g_p(s,a)$ (definición A.1 en el apéndice A).

- Parcialmente subespecificada (para $t \in T$ con $CE(t) = \frac{1}{2}$). Para $t = (s, < a, g_s^P(s,a), \lambda >, s')$, existe una transición discreta subespecificada:

$$< s, \mathcal{RT} > \overset{a}{\rightsquigarrow} < s', R_\lambda(\mathcal{RT}) >$$

  sii $\mathcal{RT} \subseteq \mathcal{P}(g_s^P(s,a))$.

- Totalmente subespecificada. Eventualmente, puede existir una transición discreta totalmente subespecificada:

$$< s, \mathcal{RT} > \overset{a}{\rightsquigarrow} < s_s, \mathcal{RT} >$$

  sii $\mathcal{RT} \subseteq \mathcal{P}(g_s^T(s,a))$.

- No posible (para $t \in T$ con $CE(t) = 0$). Con $t = (s, < a, g_{np}(s,a), \{\} >, s_{np})$, no existe una a-transición discreta

$$< s, \mathcal{RT} > \overset{a}{\nrightarrow} \text{ (por } < s, \mathcal{RT} > \overset{a}{\nrightarrow} < s_{np}, \mathcal{RT} > \text{)}$$

  sii $\mathcal{RT} \subseteq \mathcal{P}(g_{np}(s,a))$.

Las transiciones discretas, según las anteriores reglas, definen el operador $Rsuc_d(\mathcal{R}, a)$, sucesor discreto de una región $\mathcal{R}$ mediante una acción a. El operador $Rpre_d(\mathcal{R}, a)$ de predecesores discretos se define de forma inversa.

Transiciones temporales con etiqueta t:

- Si $\mathcal{RT}$ es una región de tiempo final: No existe sucesor temporal.
- Si $\mathcal{RT}$ es una región de tiempo no final, puede existir una única transición temporal, que será:
  - Posible:

$$< s, \mathcal{RT} > \xrightarrow{t} < s, RTsuc_t(\mathcal{RT}) >$$

    sii $\mathcal{RT} \subseteq (\mathcal{P}(I_a(s)))$ y $RTsuc_t(\mathcal{RT}) \subseteq (\mathcal{P}(I_a(s)))$.

[4] siendo $R_\lambda(\mathcal{RT})$ la operación sobre DBMs en el apéndice A

– Subespecificada:

$$< s, \mathcal{RT} > \overset{t}{\rightsquigarrow} < s, RTsuc_t(\mathcal{RT}) >$$

sii o bien $\mathcal{RT} \subseteq (\mathcal{P}(I_s(s)))$ y $RTsuc_t(\mathcal{RT}) \nsubseteq (\mathcal{P}(I_p(s)))$, o bien $\mathcal{RT} \subseteq (\mathcal{P}(I_a(s)))$ y $RTsuc_t(\mathcal{RT}) \subseteq (\mathcal{P}(I_s(s)))$.

– No posible:

$$< s, \mathcal{RT} > \overset{t}{\nrightarrow} \text{ (por } < s, \mathcal{RT} > \overset{t}{\nrightarrow} < s_{np}, RTsuc_t(\mathcal{RT}) > \text{)}$$

sii $\mathcal{RT} \subseteq (\mathcal{P}(I_p(s)))$ o $RTsuc_t(\mathcal{RT}) \subseteq (\mathcal{P}(I_p(s)))$.

Las transiciones de tiempo, según las reglas anteriores, definen el operador $Rsuc_t(\mathcal{R})$, sucesor temporal de una región $\mathcal{R}$. El operador $Rpre_t(\mathcal{R})$ de predecesor temporal se define de forma inversa.

8.3.3 Construcción del grafo de regiones RMUS-T

Dado un modelo MUS-T $\mathcal{M}$, su grafo RMUS-T asociado, $\mathcal{R}_{\mathcal{M}}$, es una tripla $< \mathcal{SR}_0, \mathcal{SR}, \mathcal{TR} >$, donde $\mathcal{SR}_0$ es el conjunto de regiones iniciales; $\mathcal{SR}$ el conjunto de regiones; y $\mathcal{TR}$ el conjunto de transiciones, $\mathcal{R} \rightarrow \mathcal{R}'$, con una etiqueta $\in \{(A \cup t) \times \mathcal{L}_3\}$ que asocia a cada transición $\in \mathcal{TR}$ una etiqueta de acción discreta $\in A$ o una etiqueta de acción temporal t y una condición de especificación en $\{0, \frac{1}{2}, 1\}$. El algoritmo, para el cómputo de $\mathcal{R}_{\mathcal{M}}$, construye el grafo a partir de las regiones iniciales $< s_0, \mathcal{RT}^0 > \in \mathcal{SR}_0$, donde $s_0 \in S_0$, y $\mathcal{RT}^0$ es la región de tiempo $x = 0$ para todo $x \in \mathcal{C}$.

Ejemplo 8.2. En la figura 8.2 se muestra un pequeño ejemplo de un modelo MUS-T para un interruptor, y su correspondiente grafo de regiones RMUS-T en la figura 8.3. En el grafo RMUS-T de la figura 8.3, las transiciones posibles se representan en línea continua y las subespecificadas en línea discontinua.

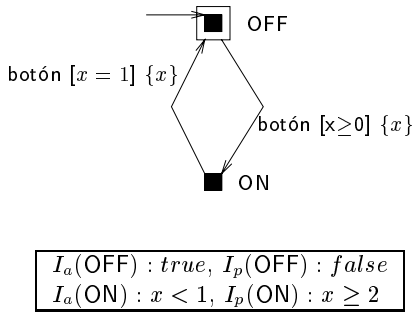


Figura 8.2. Modelo MUS-T de un interruptor

□

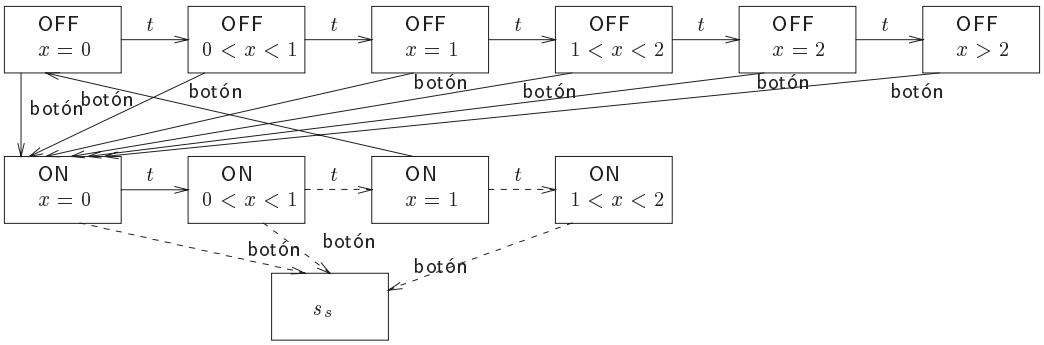


Figura 8.3. Modelo RMUS-T del modelo de la figura 8.2. La etiqueta t denota una transición temporal

8.4 *Model Checking* SCTL/MUS-T

El algoritmo de *model checking* desarrollado calcula el valor de verdad de un requisito:

$$\{I_0 := \phi_0, \cdots, I_n := \phi_n\}$$

en una región del grafo de regiones $\mathcal{R}$. Dada la relación de satisfacción definida para SCTL-T, el algoritmo de *model checking* operará sobre triplas $< s, \mathcal{RT}, c_e >$, donde c_e es la condición de especificación bajo la que $< s, \mathcal{RT} >$ es accesible. Denotaremos por $\mathcal{R}$ a la región ampliada con la condición de accesibilidad $< s, \mathcal{RT}, c_e >$, y $ce(\mathcal{R})$ a la condición de especificación de la región $\mathcal{R}$.

Definimos el grado de satisfacción, GS, de una fórmula $\phi \in \Phi_{SCTL-T}$ en una región $\mathcal{R} \in \mathcal{SR}$ por:

$$GS : (\Phi_{SCTL-T}, \mathcal{SR} \times \mathcal{L}_3) \rightarrow \mathcal{L}_6 = \{0, \frac{1}{4}, \widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$$

Las relaciones de satisfacción, definidas en 7.6.1 para un requisito SCTL-T, se traducen a relaciones de satisfacción sobre el grafo RMUS-T como sigue:

$$\models_{\mathcal{M},s_0} (I_R, \mathcal{M}) \equiv \ GS(I_R, < s_0, \mathcal{RT}^0, 1 >).$$

$$\begin{cases} \models_{\forall} (I_R, s) \ \equiv \ \bigwedge_{<s,\mathcal{R}>} GS(I_R, < s, \mathcal{R}, 1 >) \\ \models_{\exists} (I_R, s) \ \equiv \ \bigvee_{<s,\mathcal{R}>} GS(I_R, < s, \mathcal{R}, 1 >) \end{cases}$$

siendo $< s, \mathcal{RT} > \in \mathcal{SR}$, cualquier región del grafo de regiones con estado de control s.

Para un modelo MUS-T $\mathcal{M}$, trasladamos las definiciones en la sección 7.9.1, para particiones de cardinalidad 6 en el espacio $\mathcal{ST}$, a particiones del espacio de regiones $\mathcal{SR}$. Es decir, para cualquier fórmula SCTL-T ϕ, definimos la partición $P_{\mathcal{M}}(\phi)$ del espacio $\mathcal{SR}$ asociada a ϕ, donde regiones en el elemento $\langle P \rangle_i$ de la

partición se corresponden con las regiones del grafo en las que la fórmula ϕ tiene un valor de verdad $i \in \mathcal{L}_6$.

En una estrategia global para el *model checking*, la evaluación del valor de verdad de una fórmula ϕ en una región $\mathcal{R}$ calcula la partición $P_{\mathcal{M}}(\phi)$ del espacio de regiones asociada a ϕ y devuelve el valor de verdad i tal que $\mathcal{R} \in \langle P \rangle_i$. Las estrategias globales llevan a computaciones innecesarias en el espacio de regiones, dado que, para computar el valor de verdad en $\mathcal{R}$, no es necesario explorar normalmente todo el grafo de regiones. Por contra, los algoritmos locales trabajan bajo demanda en la estructura de la fórmula, sólo evaluando las partes del grafo necesarias para conocer el valor de verdad en $\mathcal{R}$. Además, las estrategias locales permitirían la computación del grafo de regiones RMUS-T al vuelo, como en el algoritmo desarrollado en (Sokolsky y Smolka, 1995).

Como veremos, el algoritmo desarrollado, aunque computando el grafo de regiones a priori, sigue una estrategia local, lo que reduce la exploración del grafo de regiones RMUS-T al fragmento necesario para computar el grado de satisfacción de la fórmula. Además, una estrategia local permite el cómputo de contraejemplos.

En las siguientes secciones se presentan las operaciones en el grafo RMUS-T utilizadas en el algoritmo, que se detalla en la sección 8.4.6.

8.4.1 Operador estados de aplicabilidad sobre regiones

Redefinimos el operador estados de aplicabilidad $\perp$ $((\otimes, P), (s, \gamma))$ (sección 7.6.5) al operador de estados de aplicabilidad sobre regiones $\perp$ $((\otimes, P), < s, \mathcal{RT} >)$. De forma similar a como se hizo en la sección 7.6.5, definimos la condición de accesibilidad de una región $\mathcal{R}$ como la condición de especificación de la transición con la que se alcanza $\mathcal{R}$.

Así, dependiendo del operador temporal:

- $\Rightarrow \bigcirc$ operará sobre el conjunto de regiones $< s', \mathcal{RT}', c_e' >$ que se pueden alcanzar a partir de la región actual $< s, \mathcal{RT} >$ mediante un paso discreto (con condición de especificación c_e') etiquetado con una acción a que aparezca como posible en la premisa más simple del requisito.
- $\Rightarrow \bigodot$ operará sobre el conjunto de regiones $< s', \mathcal{RT}', c_e' >$ que pueden evolucionar a la región $< s, \mathcal{RT} >$ mediante una transición discreta (con condición de especificación c_e') cualquiera.
- $\Rightarrow_+$ operará sobre la región actual y sobre el conjunto de regiones $< s, \mathcal{RT}', c_e' >$ que se pueden alcanzar a partir de la región actual $< s, \mathcal{RT} >$ mediante una transición temporal, con la condición de especificación c_e' que determinen las invariantes asociadas al estado s.
- $\Rightarrow_-$ operará sobre la región actual y sobre el conjunto de regiones $< s, \mathcal{RT}', c_e' >$ que pueden evolucionar a la región $< s, \mathcal{RT} >$ mediante una transición temporal, con la condición de especificación c_e' que determinen las invariantes asociadas al estado s.

El cálculo de las regiones que forman el conjunto de aplicabilidad según el operador $\perp$ $((\otimes, P), < s, \mathcal{RT} >)$ se muestra en el algoritmo 8.2, donde R$=< s, \mathcal{RT} >$,

Algoritmo 8.2 $\perp$ $((\otimes, P), (s, \mathcal{RT})$: EA(OP,P,R). ce(t) indica la condición de especificación de la transición temporal t y ce(d) la condición de especificación de la transición discreta

```
EA(OP,P,R)
{
   let CE:=1;
   case OP
      ⇒+:
         EA:=(R,1);
         if R ⊆ I_a then CE:=1;
         else if R ⊆ I_s then CE:=1/2;
         else if R ⊆ I_p then CE:=0;
         if CE ≠ 1 then EA:= EA + (R,CE);
         while (R':=Rsuc_t(R')) ≠ {}) do EA:= EA + (R', CE ∧ ce(t));
         return EA;
      ⇒−:
         EA:=(R,1);
         if R ⊆ I_a then CE:=1;
         else if R ⊆ I_s then CE:=1/2;
         else if R ⊆ I_p then CE:=0;
         if CE ≠ 1 then EA:= EA + (R,CE);
         while (R'=Rpre_t(R')) ≠ {}) do EA:= EA + (R', CE ∧ ce(t));
         return EA;
      ⇒:
         EA:=(R,1); return EA;
      ◯:
         let AA = AA(P)
         EA:={};
         foreach a ∈ AA do EA:= EA + (Rsuc_d(a,R),ce(d));
         return EA;
      ⊙:
         EA:={};
         foreach a do EA:= EA + (Rpre_d(a,R),ce(d));
         return EA;
   end case ;
}
```

OP=$\otimes$, P=P, y AA son las acciones de aplicabilidad calculadas a partir de la premisa P (algoritmo 7.2).

Ejemplo 8.3. En las siguientes figuras mostramos ejemplos del operador estados de aplicabilidad sobre regiones. Para una fórmula causal SCTL-T, $\phi_1 Æ \otimes \phi_2$, en la que el conjunto de acciones de aplicabilidad es $\{a, b\}$:

- En la figura 8.4 se muestran las regiones de aplicabilidad $\perp((\bigcirc, P), < s, \mathcal{RT} >)$ en la región $< s, \mathcal{RT}_1 >$.

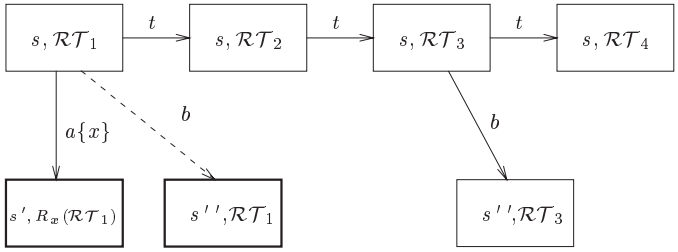


Figura 8.4. Regiones de aplicabilidad para el operador $\Rightarrow \bigcirc$ en $< s, \mathcal{RT}_1 >$. Con $ce = 1$ para $< s', R_x(\mathcal{RT}_1) >$ y $ce = \frac{1}{2}$ para $< s'', \mathcal{RT}_1 >$

- En la figura 8.5 se muestran las regiones de aplicabilidad $\perp((\odot, P), < s, \mathcal{RT} >)$ en la región $< s, \mathcal{RT}_1 >$.

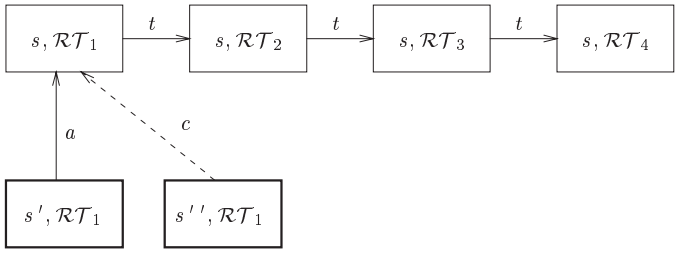


Figura 8.5. Regiones de aplicabilidad para el operador $\Rightarrow \odot$ en $< s, \mathcal{RT}_1 >$. Con $ce = 1$ para $< s', \mathcal{RT}_1 >$ y $ce = \frac{1}{2}$ para $< s'', \mathcal{RT}_1 >$

- En la figura 8.6 se muestran las regiones de aplicabilidad $\perp((\Rightarrow_+, P), < s, \mathcal{RT} >)$ en la región $< s, \mathcal{RT}_1 >$ y $\perp((\Rightarrow_-, P), < s, \mathcal{RT} >)$ en la región $< s, \mathcal{RT}_4 >$.

□

8.4.2 Relojes de especificación

La verificación de una fórmula SCTL-T ϕ utilizando el grafo RMUS-T se realizará mediante la introducción en el sistema de tantos relojes adicionales como operadores de tipo y. aparezcan en la fórmula ϕ. Intuitivamente, dichos relojes

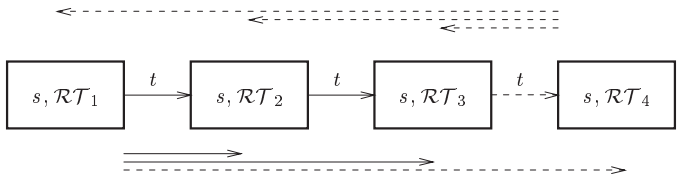


Figura 8.6. Regiones de aplicabilidad para el operador $\Rightarrow_+$ en $< s, \mathcal{RT}_1 >$, y para el operador $\Rightarrow_-$ en $< s, \mathcal{RT}_4 >$. Las condiciones de especificación de cada una de las regiones de aplicabilidad se sigue de la condición de especificación del camino de arcos temporales que lleva a dichas regiones.

nos permiten llevar la cuenta del tiempo transcurrido entre los pares de estados en los que se aplican los elementos de la fórmula $y.$, y algún $\psi_y \in \{y \prec c, x - y \prec c\}$.

En concreto, sea $\mathcal{M}$ un modelo MUS-T y $\mathcal{C}$ su conjunto de relojes. Para una fórmula SCTL-T ϕ, sea $\mathcal{E} = \{y_1, \cdots, y_m\}$ el conjunto de relojes de especificación incluidos en la fórmula. El modelo aumentado $\mathcal{M}^{\mathcal{E}}$ tendrá la misma estructura de $\mathcal{M}$, pero con un conjunto de relojes $\mathcal{C} \cup \mathcal{E}$ y las constantes máximas $\{m_x\}$ y $\{m_{xy}\}$ calculadas a partir del conjunto $\Psi^{\mathcal{E}}_{\mathcal{M}}$, que no es más que el conjunto $\Psi_{\mathcal{M}}$ extendido con los predicados de tiempo incluidos en la fórmula ϕ.

Los relojes de especificación $\mathcal{E}$ tienen un valor irrelevante en la construcción del grafo abstracto RMUS-T, y sólo se manipulan en el algoritmo de verificación de la fórmula ϕ. De hecho, un grafo RMUS-T $\mathcal{R}^{\mathcal{E}}_{\mathcal{M}}$, con un conjunto de regiones extendidas $\mathcal{R}^e$, puede ser utilizado para la verificación de cualquier fórmula SCTL-T sobre el modelo $\mathcal{M}$ cuyas constantes máximas $\{m_x\}$ y $\{m_{xy}\}$ coincidan con las utilizadas para el cálculo de $\mathcal{R}^{\mathcal{E}}_{\mathcal{M}}$.

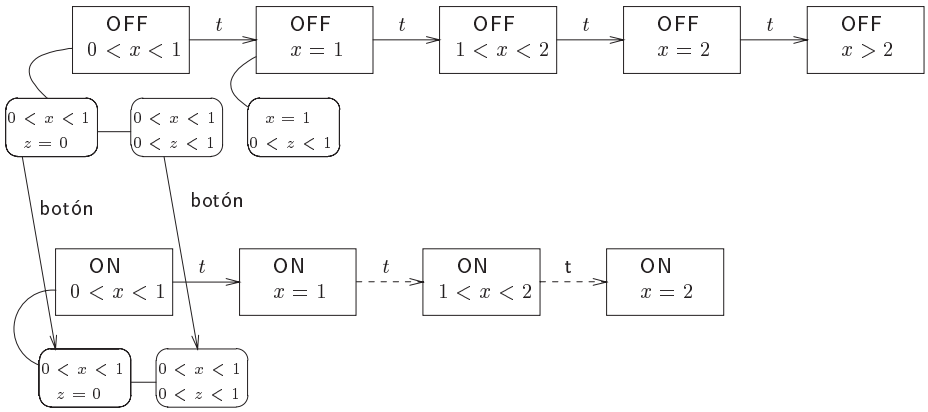


Figura 8.7. Los relojes de especificación introducen regiones de tiempo ampliadas en las regiones del grafo RMUS-T

Ejemplo 8.4. En la figura 8.7 se muestra un fragmento del grafo de regiones de la figura 8.3, extendido con un reloj de especificación z ligado a la región $<$ OFF, $(0 < x < 1) >$. En la figura 8.7 podemos observar que, dado que los relojes de especificación no intervienen en las guardas ni en las invariantes de los estados de control del modelo MUS-T, su inclusión sólo produce versiones aumentadas de

las regiones de tiempo y transiciones de tiempo, siempre posibles, que modifican el valor de dichos relojes. □

Como habíamos mencionado en la definición de la sintaxis SCTL-T, los relojes de especificación se autocaracterizan como relojes de pasado o como relojes de futuro. De esta forma, los predicados de tiempo sobre los relojes de especificación sólo incluyen constantes enteras no negativas, que en el caso de los relojes de pasado miden distancias temporales en el pasado, y en el de relojes de futuro miden distancias temporales en el futuro (tanto el pasado como el futuro incluyen el presente). Sin embargo, por uniformidad, trataremos los relojes de pasado como relojes que toman sus valores en el dominio $\Re_-$.

Ejemplo 8.5. En el conjunto de regiones de tiempo de la figura 8.8(a) se presenta dicha perspectiva de los relojes de pasado, donde x es un reloj definido en el modelo MUS-T, e y es un reloj de especificación de pasado, ligado por un operador y. en un región cuya región de tiempo asociada es $(1 < x < 2)$. El verdadero valor de las regiones de tiempo se obtiene considerando el valor absoluto del reloj y, como se muestra en la figura 8.8(b). □

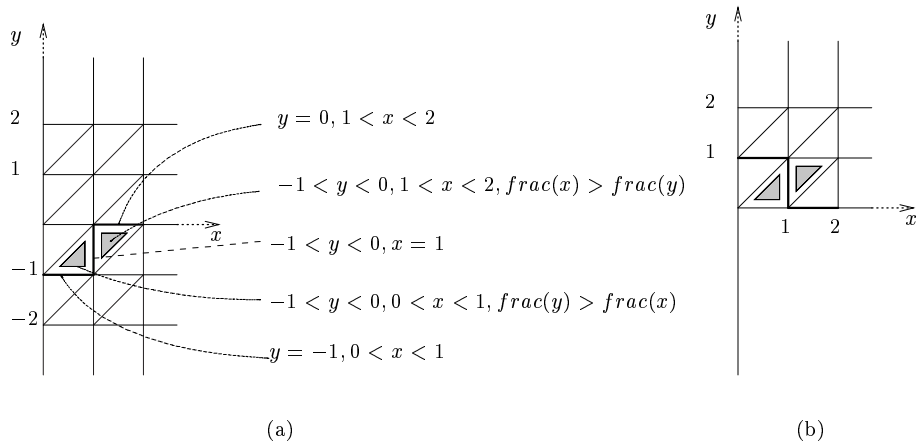


Figura 8.8. Regiones con relojes de especificación de pasado

8.4.3 Grado de satisfacción de proposiciones atómicas

El grado de satisfacción de las constantes de estado $\theta \in \Theta$, predicados de tiempo $\psi \in \Psi$ y acciones del alfabeto $a \in A$, es exacto bajo la equivalencia que define el grafo RMUS-T.

Definición 8.2. **Grado de satisfacción de constantes de estado GSC**$(\theta, \mathcal{R})$. El grado de satisfacción de las constantes $\theta \in \Theta = \{true, false, \emptyset\}$ se realiza según la tabla 7.4 para una condición de especificación $ce(\mathcal{R})$. Dado que los relojes de especificación no influyen en los comportamientos temporizados que exhibe el

modelo, el grado de satisfacción de constantes proposicionales de estado no necesita la evaluación en la región extendida.

Definición 8.3. **Grado de satisfacción de predicados de tiempo GSP**$(\boldsymbol{\psi}, \mathcal{R})$. Para el cálculo del grado de satisfacción de un predicado de tiempo, y dado que las zonas de tiempo son poliedros convexos, representamos la región de tiempo como un DBM (apéndice de capítulo 8.A) y utilizamos las operaciones sobre DBMs recogidas en el apéndice A.

Siendo $\mathcal{R} =< s, \mathcal{RT}^e, c_e >$ una región, posiblemente extendida con relojes de especificación, y $\psi \in \Psi$:

$$GSP(\psi, \mathcal{R}) = (DBM(\mathcal{RT}^e) \subseteq DBM(\psi))$$

Definición 8.4. **Grado de satisfacción de acciones GSA**$(\mathbf{a}, \mathcal{R})$. En base al pseudocódigo 8.1, definiendo los conjuntos $\{A_P, A_{NP}, A_S\}$, se puede trasladar el cómputo de la condición de especificación de una acción, c_A (definición 7.8), al grafo de regiones. Definimos el grado de satisfacción de una acción a en una región $\mathcal{R} =< s, \mathcal{RT}^e, c_e >$ como:

$$GSA(a, \mathcal{R}) = c_A(a, \mathcal{R}) = c_A(a, s^{\mathcal{RT}^e}) = \begin{cases} 0 \text{ si } a \in A_{NP}(s^{\mathcal{RT}^e}) \\ 1 \text{ si } a \in A_P(s^{\mathcal{RT}^e}) \\ \frac{1}{2} \text{ si } a \in A_S(s^{\mathcal{RT}^e}) \end{cases}$$

8.4.4 Cómputo de recursiones mínimas y máximas

El cómputo global de los puntos fijos asigna a cada región en $\mathcal{SR}$ un valor de verdad $\in \{0, \frac{1}{4}, \widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$, formando una partición en $\mathcal{P}^6$. Dado que los transformadores de la recursión en SCTL-T son monótonos y el grafo cociente RMUS-T es finito, los puntos fijos en las recursiones mínimas y máximas (definiendo un transformador de predicados f) se pueden calcular de forma iterativa, partiendo del menor y mayor elemento en las particiones ordenadas de cardinalidad 6. Por tanto:

- Punto fijo mínimo: $P_{i+1} = f(P_i)$, con $P_0 = P_\perp$.
- Punto fijo máximo: $P_{i+1} = f(P_i)$, con $P_0 = P_\top$.

donde

$$P_\perp = \{\langle P \rangle_0 = \mathcal{SR}, \underset{\frac{1}{4} \leq i \leq 1}{\langle P \rangle_i} = \{\}\}$$
$$P_\top = \{ \underset{0 \leq i \leq \frac{3}{4}}{\langle P \rangle_i} = \{\}, \langle P \rangle_1 = \mathcal{SR}\}$$

son, respectivamente, el mínimo y el máximo, bajo la inclusión $\subseteq_P$, en el conjunto de particiones de cardinalidad 6 en el espacio de regiones $\mathcal{SR}$.

Cuando $P_{i+1} = P_i$, la iteración del cómputo de punto fijo termina, siendo P_i la partición del grafo de regiones para el grado de satisfacción del punto fijo mínimo si $P_0 = P_\perp$, o del punto fijo máximo si $P_0 = P_\top$.

En una estrategia local para el cómputo de un punto fijo $I_R = f(I_R)$, el algoritmo evalúa localmente las fórmulas definidas en el transformador f, anotando los estados en los que el requisito I_R ya ha sido evaluado. Cuando se detecta un ciclo en la evaluación de I_R, el algoritmo finaliza la recursión devolviendo, para ese estado, un grado de satisfacción 0 en el caso de que se trate de una recursión $\ll I_R \gg$, y 1 en el caso de una recursión $[\![I_R]\!]$.

Como fue definido en la sección 7.9.2, la sintaxis SCTL-T no permite alternancia en puntos fijos, es decir, no se permite la existencia de recursiones mínimas bajo una recursión máxima o viceversa. Tal restricción se ha incluido sólo en aras de la simplicidad del algoritmo implementado. Dado que todos los puntos fijos evaluados en un mismo cómputo local son del mismo tipo, todos los elementos en el ciclo corresponden al mismo tipo de recursión. La existencia de alternancia en puntos fijos obligaría a recomputar el punto fijo más interno para el cálculo del punto fijo más externo en una alternancia. Como punto de partida a soluciones locales con alternancia en puntos fijos, nos remitimos a los grafos de dependencia en (Liu et al., 1998).

8.4.5 Grado de satisfacción de una fórmula SCTL-T

El grado de satisfacción de una fórmula SCTL-T sobre una región $\mathcal{R}^e$, extendida con un conjunto de relojes de especificación $\mathcal{E}$, se calcula por inducción estructural, de forma idéntica a como se define en la sección 7.8. Sea $\mathcal{M}$ un modelo MUS-T y ϕ una fórmula SCTL-T, el grado de satisfacción GS de ϕ sobre una región $\mathcal{R}^e =< s, \mathcal{RT}^e, c_e >$ del grafo de regiones $\mathcal{R}^{\mathcal{E}}_{\mathcal{M}}$ es:

i) $GS(\psi, < s, \mathcal{RT}^e, c_e >) \equiv \mathrm{GSP}(\psi, < s, \mathcal{RT}^e, c_e >)$

ii) $GS(a, < s, \mathcal{RT}^e, c_e >) \equiv \mathrm{GSA}(a, < s, \mathcal{RT}^e, c_e >)$

iii) $GS(\theta, < s, \mathcal{RT}^e, c_e >) \equiv GSC(\theta, < s, \mathcal{RT}^e, c_e >)$

iv) $GS(y.\phi, < s, \mathcal{RT}^e, c_e >) \equiv GS(\phi, < s, \mathcal{RT}^e_{y=0}, c_e >)$

v) $GS(\neg\phi, < s, \mathcal{RT}^e, c_e >) \equiv \neg(GS(\phi, < s, \mathcal{R}^e, c_e >))$

vi) $GS(\phi_1 \vee \phi_2, < s, \mathcal{RT}^e, c_e >) \equiv$

$$(GS(\phi_1, < s, \mathcal{RT}^e, c_e >) \vee GS(\phi_2, < s, \mathcal{RT}^e, c_e >))$$

vii) $GS(\phi_1 \wedge \phi_2, < s, \mathcal{RT}^e, c_e >) \equiv$

$$(GS(\phi_1, < s, \mathcal{RT}^e, c_e >) \wedge GS(\phi_2, < s, \mathcal{RT}^e, c_e >))$$

viii) $GS(\phi_1 \forall \otimes \phi_2, < s, \mathcal{RT}^e, c_e >) \equiv$

$$\rightarrow \left(GS(\phi_1, < s, \mathcal{RT}^e, c_e >), \bigwedge_{\mathcal{R}_a} (\neg ce(\mathcal{R}_a) \vee GS(\phi_2, \mathcal{R}_a)) \right)$$

con $\mathcal{R}_a \in \perp ((\otimes, \phi_1), < s, \mathcal{RT}^e >)$

ix) $GS(\phi_1 \exists \otimes \phi_2, < s, \mathcal{RT}^e, c_e >) \equiv$

$$\rightarrow \left(GS(\phi_1, < s, \mathcal{RT}^e, c_e >), \bigvee_{\mathcal{R}_a} (ce(\mathcal{R}_a) \wedge GS(\phi_2, \mathcal{R}_a)) \right)$$

con $\mathcal{R}_a \in \perp ((\otimes, \phi_1), < s, \mathcal{RT}^e >)$

x) $GS(I_R, < s, \mathcal{RT}^e, c_e >) \equiv GS(\phi_R, < s, \mathcal{RT}^e, c_e >)$, con ϕ_R la incluida en la definición $I_R := \phi_R$.

xi) $GS(\ll I_R \gg, < s, \mathcal{RT}^e, c_e >) \equiv i \in \mathcal{L}_6 \mid (s, \mathcal{RT}^e) \in < \{\bigcap_P \{Z | f(Z) \subseteq_P Z\} >_i$, punto fijo mínimo del transformador f definido por $I_R = f(I_R)$.

xii) $GS([\![I_R]\!], < s, \mathcal{RT}^e, c_e >) \equiv i \in \mathcal{L}_6 \mid (s, \mathcal{RT}^e) \in < \{\bigcup_P \{Z | f(Z) \supseteq_P Z\} >_i$, punto fijo máximo del transformador f definido por $I_R = f(I_R)$.

8.4.6 Algoritmo de verificación SCTL-T sobre RMUS-T

El algoritmo de *model checking* 8.3 muestra el pseudocódigo del proceso de verificación de una fórmula ϕ, en notación inversa, sobre una región del grafo R=$< s, \mathcal{RT}^e, c_e >$. El algoritmo se invoca con índice n=0 sobre un requisito ir.

El algoritmo utiliza el cómputo local de recursiones mínimas y máximas presentado en la sección 8.4.4; los relojes de especificación en la fórmula se manejan según se detalla en la sección 8.4.2; la aparición de regiones extendidas de especificación introduce algunas peculiaridades en los operadores definidos que se detallan en el apéndice de capítulo 8.A.

8.5 Conclusiones

Debemos resaltar que, desafortunadamente, el tamaño del grafo de regiones es exponencial en el número de relojes en el modelo y en el tamaño de las constantes de tiempo utilizadas tanto en el modelo como en el requisito a verificar. A pesar de que la introducción del concepto de regiones para la construcción de un grafo cociente finito permite la verificación algorítmica de modelos de estados con relojes densos, los algoritmos basados en la construcción explícita de dicho cociente no resultan demasiado eficientes en la práctica. De hecho, en la literatura especializada no se ha podido encontrar ninguna herramienta de verificación formal basada estrictamente en el grafo de regiones.

En el capítulo 10 retomaremos este problema, presentando una solución al problema de *model checking* mediante la computación de un dominio de estados abstracto "menos fino" que el obtenido bajo la equivalencia de regiones. Para ello, en la sección 10.6, presentamos la particularización de bisimulación de abstracción de tiempo fuerte para el modelo MUS-T. Dicha bisimulación nos permite definir un algoritmo de minimización que, a partir de una partición finita del conjunto de estados concretos, permite computar el mínimo espacio abstracto en el que realizar el *model checking* de fórmulas SCTL-T, aliviando, de esta forma, el problema de explosión del espacio de estados en el grafo RMUS-T.

Algoritmo 8.3 Algoritmo de Verificación MCGR(R,req,ir,n), con R=$< s, \mathcal{RT}^e, c_e >$.

```
MCGR(region R, Requisito req, string ir, int n)
{
/* R: region, req: estructura de requisitos */
/* ir: identificador de requisito y n: indice del vector de un requisito*/
   case req[id][n]
      Eaccion(a): n:=n+1; return GSA(a,R)
      Eaccion(true) : n:=n+1; return GSC(true,R);
      Eaccion(false) : n:=n+1; return GSC(false,R);
      Eaccion(∅) : n:=n+1; return 1;
      Epredicado(ψ): n:=n+1; return GSP(ψ,R);
      Eoperador_logico(¬): n:=n+1; return ¬ MCGR(R,req,ir,n);
      Eoperador_logico(∧): n:=n+1; return MCGR(R,req,ir,n) ∧ MCGR(R,req,ir,n);
      Eoperador_logico(∨): n:=n+1; return MCGR(R,req,ir,n) ∨ MCGR(R,req,ir,n);
      Ecuantificador(Æ):
         if Æ = ∀ then GSc:=1; else GSc:= 0;
         OT:= F[n+1]; n:= n+2;
         if (OT=⇒ ◯) then AA:= acciones-aplicabilidad(F,n)
         else AA:= {};
         RA:= EA(R,OT,AA);
         GSp:= MCGR(R,req,ir,n); naux:=n;
         foreach R′ ∈ RA do
            n:=naux;
            if (Æ = ∀) GSc:= GSc ∧ ( ¬ce(R′) ∨ MCGR(R,req,ir,n));
            else GSc:= GSc ∨ ( ce(R′) ∧ MCGR(R,req,ir,n));
         end foreach
         return causal(GSp,GSc);
      ErecursionMin(I_R): n:=n+1;
         if R.Evaluado(I_R) then return 0
         else n′:=0; return MCGR(R,req,I_R,n′);
      ErecursionMax(I_R): n:=n+1;
         if R.Evaluado(I_R) then return 1
         else n′:=0; return MCGR(R,req,I_R,n′);
      Einstanciación(I_R): n:=n+1;
         n′:=0; return MCGR(R,req,I_R,n′);
      Efreeze(r): n:=n+1; R_{r=0}:=R.extiende(r);
         return MCGR(R_{r=0},req,ir,n);
   end case ;
}
```

Desde el punto de vista de la legibilidad de las fórmulas, volvemos a hacer hincapié, como ya habíamos adelantado en el capítulo 7, en la conveniencia de utilizar macros de operadores temporales que eviten la recursión explícita en las fórmulas, trasladando la reescritura en función de los operadores SCTL-T a los algoritmos de verificación.

Apéndices del capítulo

8.A Implementación

8.A.1 Representación de regiones

Una región de tiempo $\mathcal{RT}$ especificada con las restricciones recogidas en la definición 8.1, puede representarse por un DBM, lo que proporciona un mecanismo eficiente (ver apéndice A) para las operaciones del algoritmo de *model checking* sobre el grafo RMUS-T. Sin embargo, durante la construcción del grafo de regiones utilizamos un almacenamiento transitorio que permite una obtención sencilla de los sucesores temporales a medida que se construye el grafo. Una región de tiempo será una estructura conteniendo la siguiente información:

- Un vector cotas que almacena para cada reloj definido en el modelo:
  - (abierta,n), para aquellos relojes $n < x < n + 1$.
  - (cerrada,n), para aquellos relojes $x = n$.
  - (sin,RNA), para aquellos relojes $x > m_x$, siendo RNA la lista con las restricciones de la diferencia $x - y$.
- Una lista frac de grupos de relojes con (abierta,n), ordenada según la magnitud de sus partes fraccionarias (relojes en el mismo grupo tienen idénticas partes fraccionarias).

Por uniformidad, y aunque tanto los relojes de especificación como los relojes de modelo se interpretan $\in \Re_+$, los relojes de especificación de pasado ($\mathcal{E}^f$) se almacenan tomando valores en en el domino $\Re_-$, permitiendo que los progresos y retrocesos en tiempo pueden ser gestionados de forma simétrica en la librería de DBMs.

A partir de la estructura anterior, la obtención del DBM representando una región de tiempo se muestra en el algoritmo 8.4.

8.A.2 Predecesor y sucesor en tiempo

El operador $RTsuc_t$, aplicado a una región de tiempo $\mathcal{RT}$, se calcula según el pseudocódigo en el algoritmo 8.5, donde test-final calcula la siguiente condición:

$$\underset{x \in \mathcal{C} \cup \mathcal{E}^f}{\forall} \mathsf{RT.cotas}(x) = (\mathsf{sin,RNA})$$

El operador $RTpre_t$, aplicado a una región de tiempo $\mathcal{RT}$, se calcula según el pseudocódigo en el algoritmo 8.6, donde test-inicial calcula la siguiente condición:

$$\underset{x \in \mathcal{C}}{\exists} \mathsf{RT.cotas}(x) = (\mathsf{cerrada,0})$$

Algoritmo 8.4 DBM(RT): Algoritmo para el cómputo de la representación en DBM de una región de tiempo

```
DBM(RT){
   [DBM]_{0,0}:=(0,≤);
   foreach i ∈ 1 ··· #(C ∪ E) do
      [DBM]_{i,i}:=(0,≤)
      case RT.cotas(i)
         (abierta,n): [DBM]_{0,i}:=(-n,<);[DBM]_{i,0}:=(n+1,<);
         (cerrada,n): [DBM]_{0,i}:=(-n,≤);[DBM]_{i,0}:=(n,≤);
         (sin,RNA): [DBM]_{0,i}:=(-m_i,<);[DBM]_{i,0}:=(∞,<);
   end foreach ;
   foreach i ∈ 1 ··· #(C ∪ E) do
   foreach j ∈ 1 ··· i-1 do
         if (RT.cotas(i)=(abierta,n_i)) AND (RT.cotas(j)=(abierta,n_j)) then
            case RT.frac(i) ∼ RT.frac(j)
               <: [DBM]_{i,j}:=(n_i-n_j,<);[DBM]_{j,i}:=(-(n_i-n_j-1),<);
               >: [DBM]_{i,j}:=(n_i-n_j + 1,<);[DBM]_{j,i}:=(-(n_i-n_j),<);
               =: [DBM]_{i,j}:=(n_i-n_j,≤);[DBM]_{j,i}:=(-(n_i-n_j),≤);
         else if (RT.cotas(i)=(sin,RNA)) then
            case RNA[j]
               (abierta,n): [DBM]_{j,i}:=(-n,<);[DBM]_{i,j}:=(n+1,<);
               (cerrada,n): [DBM]_{j,i}:=(-n,≤);[DBM]_{i,j}:=(n,≤);
               (sin,{}): [DBM]_{j,i}:=(-m_{ij},<);[DBM]_{i,j}:=(∞,<);
         end if
   end foreach ;
   end foreach ;
   forma-canonica(DBM);
   return DBM;}
```

Algoritmo 8.5 Operación $RTsuc_t$ sobre una región $\mathcal{RT}$

```
RTsuc_t(RT){
    if (test-final == false) then
        if ∃ r in C ∪ E^f | (RT.cotas(r).first==cerrada) then
            let minfrac={};
            foreach r in C ∪ E^f | (RT.cotas(r).first==cerrada) do
                if (RT.cotas(r).scnd()== m_r) then RT.cotas(r):= (sin,calculaRNA(r));
                else
                    RT.cotas(r):= (abierta,RT.cotas(r).scnd()+1); minfrac+=r;
                end else ;
            end foreach
            if minfrac ≠ {} then frac.push(minfrac); end if
            return RT;
        end if
        foreach r in RT.frac.pop() do RT.cotas.r:=(cerrada,n+1); end foreach
        return RT;
    end if
    else return {}
}
```

Algoritmo 8.6 Operación $RTpre_t$ sobre una región $\mathcal{RT}$

```
RTpre_t(RT){
    if (test-inicial == false) do
        if ∃ r in C ∪ E^p | (RT.cotas(r).first == cerrada) then
            let maxfrac:={}
            foreach r in C ∪ E^p | (RT.cotas(r).first == cerrada) do
                if (r in E^p AND (RT.cotas(r).scnd()== -m_r)) then RT.cotas(r):= (sin,calculaRNA(r));
                else
                    RT.cotas(r):= (abierta,RT.cotas(r).scnd()-1); maxfrac+=r;
                end else ;
            end foreach
            if (maxfrac ! = {}) then frac.pushback(maxfrac); end if
            return RT;
        end if
        let menores:={}
        foreach r in C | (RT.cotas(r).first == (sin,RNA)) do
            if (menores=={}) then menores:=r;
            else
                if (r < menores[0]) then menores:=r;
                if (r = menores[0]) then menores+=r;
            end if
        end foreach
        if (menores ! = {}) then
            foreach r in menores do RT.cotas(r):=(cerrada,m_r);
            return RT;
        end if
        foreach r in RT.frac.popback() do
            RT.cotas.r:=(cerrada,RT.cotas(r).scnd()-1);
        end foreach
        return RT;
    end if
    else return {}
}
```

CAPÍTULO 9

Síntesis

En este capítulo se aborda el problema de síntesis incremental de modelos MUS-T, como parte de las distintas iteraciones en el proceso software propuesto. La síntesis automática de modelos a partir de especificaciones en lógica está directamente relacionada con la resolución del problema de satisfactibilidad. Ante los resultados negativos que, en cuanto a decidibilidad, arroja el problema de satisfactibilidad genérico, optamos por un enfoque próximo a satisfactibilidad acotada (Laroussinie et al., 1995). La satisfactibilidad acotada restringe el conjunto de modelos generados a aquellos con un número de relojes y cota máxima conocidos. De esta forma, es posible trasladar el problema de satisfactibilidad al grafo de regiones, abstracción finita del modelo, que permitiría aplicar un método tableau similar al utilizado en las lógicas temporales proposicionales. Dadas las características imperativas del modelo MUS-T (localizadas en el manejo de relojes), ampliamos la lógica SCTL-T con cláusulas que permitan la incorporación y reinicio explícito de relojes. Con la lógica ampliada, lógica ejecutable, realizamos la síntesis basada en el concepto de futuro imperativo (Barringer et al., 1995).

9.1 Introducción

La aplicación a una amplia gama de problemas explica, en gran parte, el éxito del tratamiento automatizado para sistemas reactivos sin tiempo, sustentado en la resolución eficiente del *model checking* y la existencia de procedimientos de decisión para la síntesis. En (Emerson, 1996) se resumen los principales hitos en el tratamiento automatizado de sistemas reactivos.

Es de esperar, por tanto, que parte o todo este éxito sea trasladable a los sistemas reactivos de tiempo real. Sin embargo, si la solución al problema de *model checking* para estos sistemas es relativamente reciente, aunque cargada de resultados teóricos y herramientas prácticas que avalan su madurez, el problema de satisfactibilidad se mueve todavía en un marco no totalmente estable.

A principios de los noventa, y al tiempo que se empezaban a dar los primeros pasos en *model checking*, se atisba que el problema de satisfactibilidad no iba a ser tan fácilmente trasladable a los sistemas reactivos con tiempo. Ya en (Alur et al., 1993a; Henzinger et al., 1992b) se afirma la naturaleza no decidible del problema de satisfactibilidad (finita) para TCTL y T_μ, a pesar de la naturaleza decidible del problema de *model checking* en ambos casos.

El estudio detallado sobre bases formales para sistemas de tiempo real en (Alur y Henzinger, 1992b) muestra que es posible conseguir fragmentos de lógicas de tiempo real con satisfactibilidad decidible, a costa de sacrificar la puntualidad (sección 2.2.2) en las fórmulas. Bajo esta concesión sintáctica, la lógica lineal de tiempo denso MITL (Alur et al., 1996), presentada por primera vez en 1991, aporta un procedimiento de decisión basado en *tableau* para la construcción de un autómata temporizado. Sólo recientemente, (La Torre y Napoli, 2000) han llegado a un resultado similar para la lógica de tiempo ramificado TCTL: prohibiendo la puntualidad en las fórmulas, se establece un procedimiento de decisión para la síntesis de un *Timed Graph*, modelo de la fórmula.

Sin embargo, la exigencia de no puntualidad ha puesto de relieve algunas deficiencias ya conocidas sobre el modelo de autómata temporizado:

- Es posible, bajo el paradigma de autómata temporizado, verificar mediante *model checking* lógicas con precisión infinita, pero no es posible decidir la satisfactibilidad de una fórmula con puntualidad.

- El autómata temporizado no es cerrado bajo complementación, lo que impide la verificación homogénea basada en autómata.

En un intento de conseguir modelos basados en autómata con tiempo que sean cerrados bajo todas la operaciones, en (Alur et al., 1999) se propone un nuevo modelo, *Event-Clock timed Automaton* (ECA). ECA restringe los relojes en el modelo a *event-clocks*, es decir, relojes asociados a los eventos que miden distancias entre eventos iguales en el modelo. En (Henzinger et al., 1998) se pone de relieve la siguiente conexión: las lógicas temporales con relojes de eventos definen el mismo conjunto de lenguajes que las lógicas temporales con limitación de puntualidad. Sobre una variante de ECA, la lógica SCL –*State Clock Logic* (Raskin y Schobbens, 1997)– se caracteriza como de satisfactibilidad finita decidible utilizando una noción de puntualidad relajada.

A pesar de sus buenas características, ECA no es tan expresivo como TA [1]. Si, por contra, añadimos a un ECA la operación proyección (cambio de relojes en las transiciones), obtenemos el autómata temporizado. Como ya hemos mencionado, su mayor expresividad, avalada por el tratamiento automatizado de varios casos reales, ha sido la razón fundamental que nos ha decantado por una extensión basada en *Timed Automaton*.

[1] Todo ECA puede ser traducido a un TA; en (Alur et al., 1999) se demuestra ECA $\subset$ NDTA (*Timed Automaton* no determinista).

9.2 Crecimiento de un modelo MUS-T

Nos referimos en esta sección al crecimiento, en nivel de especificación, de un sistema modelado por un grafo MUS-T, propio de las distintas fases en el proceso de diseño. Aunque entendido de forma intuitiva, sin duda, decir que un modelo MUS-T (intrínsecamente con un número de estados infinito) crece, puede resultar poco correcto. Entenderemos el crecimiento en un grafo MUS-T bajo la definición de simulación de abstracción tiempo especificada en la sección 10.4, tal preorden establece una relación de orden entre modelos MUS-T obtenidos por evolución en el proceso de diseño.

Informalmente, conforme a la dualidad tiempo-control en los modelos MUS-T, y considerando el grafo denso que da semántica al modelo, se puede decir que un modelo MUS-T puede "crecer" en nivel de especificación si:

- Una transición (discreta o temporal) subespecificada pasa a ser posible o no posible. Adicionalmente, si la condición de especificación pasa a ser posible, el estado siguiente pasaría a ser un estado alcanzable, con lo cual, además, el modelo habrá crecido en funcionalidad, aumentando el conjunto de comportamientos especificados.
- Se añade un nuevo reloj en el modelo. Este caso se puede considerar de manera similar al caso anterior si suponemos que un modelo, en realidad, se compone de un número arbitrario (aunque finito) de relojes que, de no utilizarse en el modelo, se encuentran subespecificados.

9.3 Esquema de síntesis incremental en SCTL/MUS-T

Restringiéndose a un conjunto de verdad booleano, las fórmulas TCTL se pueden traducir a fórmulas SCTL-T (ver sección 7.10). Podemos por lo tanto afirmar, basándonos en los resultados en (Alur et al., 1993a), que el problema de satisfactibilidad es no decidible para SCTL-T, incluso si restringimos los modelos a los generados por grafos MUS-T.

En general, el problema de satisfactibilidad, y consecuentemente el de síntesis, para lógicas temporales depende en gran medida del modelo utilizado en la definición del problema y del conjunto de fórmulas permitidas en el proceso de síntesis. Es a menudo necesario realizar ya sea una restricción semántica, o una restricción sintáctica del problema.

En cuanto al **modelo** utilizado, una restricción semántica ya ha sido apuntada: el establecimiento del problema de satisfactibilidad sobre modelos generados por grafos temporizados (satisfactibilidad finita). Si bien las fórmulas SCTL-T se interpretan sobre grafos densos multivalorados, restringimos la síntesis a aquellos grafos densos generados por la sintaxis definida para modelos MUS-T (satisfactibilidad finita). Desgraciadamente, dicha restricción mantiene la satisfactibilidad en la categoría de los problemas no decidibles. Sin embargo, podemos acotar todavía

más el problema: consideremos un grafo temporizado con un número de relojes $\mathcal{C}$ y una constante máxima en los predicados de tiempo en el modelo m; en este caso, el problema se torna decidible, basta con pensarlo como un método *tableau* de una lógica proposicional con predicados de tiempo sobre un espacio de estados finito (el definido en la equivalencia de regiones). Esta forma limitada de satisfactibilidad, introducida en (Laroussinie et al., 1995) para L_ν (*Timed Modal Logic*) bajo el nombre *bounded satisfiability*, resuelve la síntesis sólo en parte. Si bien es posible sintetizar un modelo con dichas características a partir de la especificación lógica, no es posible decidir, a partir de la especificación, el número de relojes necesarios en el modelo, dejando como única salida la utilización de algún tipo de heurístico que guíe dicha elección a priori. Este problema se resuelve en (Andersen et al., 1995) mediante la definición de *Timed Agents*, que no son más que autómatas temporizados con un solo reloj. Es decir, se define la satisfactibilidad finita sobre modelos con un único reloj.

Adicionalmente, se restringe el universo de síntesis a grafos semánticos generados por modelos MUS-T deterministas, dado que, como veremos, el conjunto de fórmulas admitidas para síntesis (de sólo conjunción y universalmente cuantificadas) siempre dan lugar a este tipo de comportamientos en un modelo MUS-T.

Por otro lado, en cuanto a la **sintaxis** considerada, es bien cierto que en las lógicas temporales se adivina una relación, en distinta medida, entre la complejidad sintáctica de las fórmulas lógicas y la complejidad computacional de sus procedimientos de decisión (tanto para verificación como para síntesis). En este sentido, se han desarrollado un cierto número de lógicas limitadas (*restricted*), algunas pensadas para ámbitos de aplicación específicos, pero en general destinadas a permitir procedimientos de decisión eficientes para la síntesis. Sin duda, es delicado obtener una lógica restringida, con procedimientos de decisión eficientes y al mismo tiempo útil. Algunos requisitos en (Emerson, 1996) apuntan a la renuncia de la sintaxis completa por bien de la eficiencia: la restricción de la sintaxis a $\bigwedge$ *aserto* ya que, desde la visión de la especificación formal, $\wedge$ se considera un operado más importante que $\vee$ (usualmente se desea que un programa cumpla un conjunto de criterios); o la limitación de la profundidad de anidamientos de operadores temporales, ya que tales anidamientos son raramente utilizados en la práctica.

La utilización de lógicas limitadas en aras de la eficiencia, ya aplicada a lógicas sin tiempo como CTL (en su versión SCTL), se hace más urgente en el caso de los sistemas temporizados, mucho más costosos en sus procedimientos de decisión.

Basándonos en los dos conceptos presentados, definimos un esquema de síntesis en SCTL/MUS-T que, por un lado, traslada el problema de síntesis al grafo cociente RMUS-T del modelo y, por otro lado, restringe el conjunto de fórmulas SCTL-T sintetizables, utilizando una lógica ejecutable bajo la definición de un conjunto de reglas de síntesis. Como se observa en la figura 9.1, el proceso de síntesis de un grafo semántico generado por un modelo MUS-T se traslada a una abstracción exacta y finita, el grafo de regiones RMUS-T. Sin embargo, dado que es necesario abstraer al usuario de los mecanismos de la metodología, las modificaciones realizadas han de ser traducidas a modificaciones sobre el modelo visual MUS-T.

En las siguientes secciones, justificamos las restricciones realizadas en SCTL-T

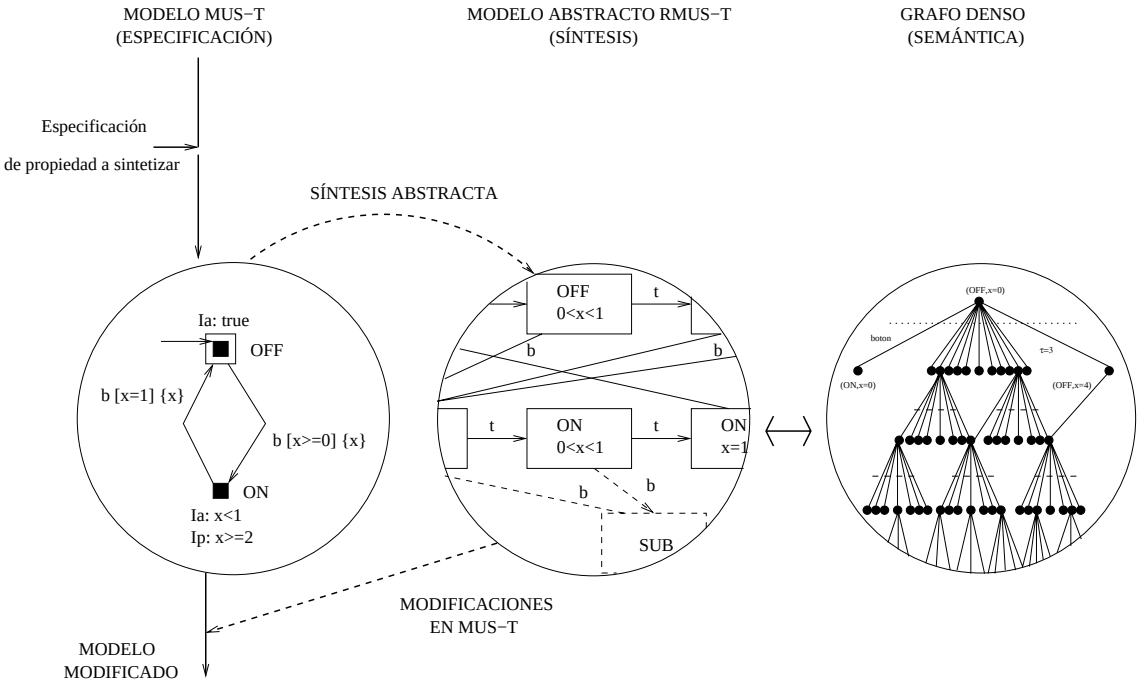


Figura 9.1. Esquema del método de síntesis

para la síntesis incremental. Posteriormente, detallaremos el algoritmo de síntesis sobre RMUS-T y la traducción de modificaciones a MUS-T.

9.4 Consideraciones previas

Dado que el objetivo es la síntesis de un modelo MUS-T a partir de propiedades SCTL-T, es necesario abstraer en lo posible al usuario de los mecanismos utilizados para tal fin (grafo RMUS-T), pero sin caer en un alejamiento excesivo que dificulte la toma de decisiones por parte del diseñador. Por ello, debemos tener en cuenta que:

- La presencia explícita de variables reloj en el modelo y la posibilidad de reiniciar un conjunto de relojes, características principalmente asociadas con los lenguajes imperativos, distinguen al autómata temporizado de los modelos de sistemas reactivos sin tiempo ni datos.

- Un fórmula SCTL-T de futuro puede intentar sintetizarse, o bien modificando las guardas e invariantes del modelo sobre los relojes existentes, o bien introduciendo un nuevo reloj. Sólo con la información en la fórmula es difícil adivinar la intención del usuario. Aunque podríamos utilizar una aproximación en dos pasos (con el número de relojes actual o, si no, ampliado), se debe tener en cuenta que los relojes en un modelo generalmente representan temporizadores en el mundo real que se desea modelar. El usuario, a me-

nudo, es consciente de la necesidad de un reloj concreto que relacione dos comportamientos específicos.

- Cuando es necesario añadir un nuevo reloj en el modelo, dicho reloj sólo puede ser reiniciado en transiciones discretas, reflejando la medida del tiempo ligada a la ocurrencia de eventos propia del comportamiento dirigido por eventos en el sistema real.

- Los relojes en el modelo MUS-T son relojes de futuro, con lo cual las fórmulas de pasado no deben introducir relojes adicionales, ya que dichos relojes serían relojes hacia atrás, no considerados en el modelo [2]. Como veremos, esto restringe el conjunto de operadores temporales sobre los relojes que no sean relojes de especificación en la fórmula a sintetizar.

- El proceso de síntesis se basa en un conjunto de reglas de síntesis sin libertad de elección (no existe $\vee$, ni $\exists$); esta decisión obedece a que, en la síntesis de un componente, se desea que el modelo sintetizado cumpla todas y cada una de las especificaciones de un conjunto, que definen su comportamiento. En la fase de verificación del proceso propuesto, el usuario verifica requisitos SCTL-T genéricos y, mediante los contraejemplos proporcionados, se podría realizar un ajuste fino dirigido por el usuario, aunque guiado por la metodología.

- El proceso de síntesis incremental puede implicar un gran aumento del número de relojes, por lo que resulta necesario aplicar un algoritmo de reducción del número de relojes.

- La especificación y síntesis basada en SCTL-T es más propensa a la introducción de bloqueos o violaciones de los requisitos de avance (difícilmente detectables en la especificación lógica) que la especificación directa en MUS-T, por lo que también es conveniente hacer una detección de corrección mínima en el proceso de síntesis incremental.

Por ello, hemos optado por realizar tanto una adecuación sintáctica como una adecuación semántica del lenguaje de especificación (SCTL-T) para el problema de síntesis de componentes reactivos temporizados:

a) Realizar una extensión sintáctica en SCTL-T que permita un manejo imperativo de los relojes en el modelo, es decir, la posibilidad de explicitar en una fórmula SCTL-T nuevos relojes en el modelo, y el reinicio de relojes en transiciones discretas. Dicha visión se acerca a la utilizada en (Lin y Yi, 2000) para la definición de un lenguaje de procesos similar a CCS, pero con un prefijo de acción con reinicio explícito de los relojes. $a(\bar{x}).t$: el proceso acepta una acción a, pone los relojes $\bar{x}$ a cero, y pasa a comportarse como el proceso t.

b) Redefinir la semántica de las fórmulas SCTL-T con un paradigma de división de la parte declarativa y la parte imperativa, utilizado en metodologías formales como METATEM (Barringer et al., 1995). Además, y tal como se

[2] La introducción de relojes de pasado es parte de las variantes *Two-way Automaton* (Alur y Henzinger, 1992a) y *Event Clock Automaton* (Alur et al., 1999) del autómata temporizado.

observa en la tabla 9.1 (Emerson, 1996), utilizamos una postura común en las lógicas limitadas para síntesis: considerar las cláusulas lógicas como invariantes del modelo, introduciendo la recursión implícitamente en las reglas de síntesis.

c) Utilizar un algoritmo de reducción de relojes como parte de las iteraciones. Sin embargo, como ya hemos mencionado, el usuario puede persistir en modelar el sistema con un número de relojes mayor que el estrictamente necesario.

Simplified CTL
iniciales: $P \vee \cdots \vee P'$
invariantes: $AG(Q \vee \cdots \vee Q')$
finalidades: $AG(P \Rightarrow AF(R \vee \cdots \vee R'))$
garantías: $AG(P \Rightarrow A((Q \vee \cdots \vee Q')U(R \vee \cdots \vee R')))$
sucesores: $AG(P \Rightarrow AX(Q \vee \cdots \vee Q') \wedge EX(R \vee \cdots \vee R') \wedge EX(S \vee \cdots \vee S'))$
Restricted Linear Temporal Logic
invariantes: $AG(Q \vee \cdots \vee Q')$
finalidades: $AG(P \Rightarrow AF(R \vee \cdots \vee R'))$
sucesores: $AG(P \Rightarrow AX(Q \vee \cdots \vee Q'))$
Restricted Initialized Linear Temporal Logic
invariantes: $AG(Q \vee \cdots \vee Q')$
finalidades: $AG(P \Rightarrow AF(R \vee \cdots \vee R'))$
sucesores: $AG(P \Rightarrow AX(Q \vee \cdots \vee Q'))$
iniciales: $P \vee \cdots \vee P'$

Tabla 9.1. Catálogo de restricciones en lógica temporal

9.5 Reglas de síntesis

La definición de las reglas de síntesis en la metodología propuesta se basa en la idea de "futuro imperativo" (METATEM (Barringer et al., 1995)), que se propone como alternativa a la visión declarativa de la lógica utilizada en el campo de la verificación formal. METATEM propone una metodología formal basada en lógica temporal que combina ambas visiones (declarativa e imperativa):

antecedente (sobre el pasado) $\Rightarrow$ *consecuencia* (sobre el presente y el futuro)

donde el antecedente supone una visión declarativa y la consecuencia una visión imperativa. El término "futuro imperativo" deriva de la intuición de que para poder construir el siguiente estado en el modelo, comprobamos condiciones en estados previos y, si es necesario, aplicamos las restricciones imperativas para la construcción del estado actual o de estados futuros. Esta misma idea, como se sigue de la tabla 9.1, se encuentra implícita en la definición de lógicas limitadas destinadas a la síntesis formal.

Una **regla de síntesis**, en la metodología propuesta, será una fórmula SCTL-T que especifique una parte declarativa (premisa) de pasado no estricto, y una parte imperativa (consecuencia) de futuro no estricto, relacionadas por un operador temporal de ámbito universal y de caracterización temporal futuro no estricto ($\Rightarrow$,$\Rightarrow_+$, $\Rightarrow \bigcirc$). Se trata, pues, de una regla causal SCTL-T en la que la premisa se considera la especificación del contexto de síntesis y la consecuencia es el nuevo comportamiento de sólo futuro. Debido a la sobrecarga de semántica:

$$\textit{premisa declarativa } \{\forall\} \Rrightarrow \{_+, \bigcirc\} \textit{ consecuencia imperativa}$$

reemplazamos los operadores temporales de síntesis por $\Rrightarrow \{_+, \bigcirc\}$.

La parte declarativa de la regla de síntesis será cualquier fórmula SCTL-T, de pasado no estricto, más la etiqueta *ini*, indicando la configuración inicial en el modelo.

Definición 9.1. **Regla de síntesis.** Una regla de síntesis $\mathcal{RS}$ sigue la gramática:

$\langle \mathcal{RS} \rangle ::= \; ini \otimes_s \phi_{fs} \mid \phi_p \otimes_s \phi_{fs}$

$\langle \otimes_s \textit{:operador de síntesis} \rangle ::= \Rrightarrow \mid \; \forall \Rrightarrow \bigcirc \; \mid \; \forall \Rrightarrow_+$

donde ϕ_p es una fórmula SCTL-T de pasado no estricto y ϕ_{fs} es una fórmula SCTL-T de futuro no estricto, cuya forma normal positiva sigue la gramática:

$$\begin{array}{ll} \langle \phi_{fs} \rangle ::= & \psi \in \Psi \\ \mid & a, \neg a \text{ con } a \in A \\ \mid & \theta \in \Theta \\ \mid & y.\phi_{fs} \text{ con } y \in \mathcal{E} \\ \mid & a\{y\} \text{ con } a \in A \text{ y } y \in \mathcal{C} \cup \mathcal{C}' \\ \mid & \phi_{fs} \wedge \phi_{fs} \\ \mid & \phi_{fs} \forall \otimes_{fs} \phi_{fs} \end{array}$$

con $\otimes_{fs} \in \{\Rightarrow \bigcirc, \Rightarrow_+, \Rightarrow\}$ y $\mathcal{C}'$ un conjunto genérico de relojes a incorporar en el modelo. Por simplicidad, se sugiere la macro $\forall \Rightarrow \overset{\forall \rightarrow}{\bigcirc}$ (definición 7.1), y su correspondiente sobrecargada $\forall \Rrightarrow \overset{\forall \rightarrow}{\bigcirc}$.

Como se sigue del conjunto $\langle \phi_{fs} \rangle$, el algoritmo se reduce a la síntesis de comportamientos deterministas. Notar que dicho conjunto se corresponde con el fragmento CTL utilizado con aplicaciones de síntesis en (Emerson y Clarke, 1982) [3] y en METATEM, dado que bajo su semántica de lógica lineal, existe una semántica de cuantificación universal.

Siguiendo las pautas, ya utilizadas en los sistemas reactivos sin tiempo, para la definición de lógicas orientadas a la síntesis, consideramos dos únicos ámbitos de aplicación de las reglas de síntesis:

- Ámbito inicial: Aplicado a las reglas de síntesis $ini \otimes_s \phi_{fs}$, permite la especificación de las características del estado inicial en el modelo.

[3] si eliminamos la cuantificación en el número de procesos

- Ámbito invariante: Aplicado a las reglas de síntesis $\phi_p \otimes_s \phi_{fs}$, la regla se aplica a todos los estados del modelo, sintetizándose la parte imperativa sólo en aquellos estados (s, γ), tales que $\vdash (\phi_p, (s, \gamma, 1)) = 1$.

El conjunto de fórmulas SCTL-T restringido para síntesis coincide con el conjunto de fórmulas DSF (*Deterministic Safety formulas*) en (Merz, 1995). La restricción se basa en considerar que, para la síntesis de una propiedad de viveza arbitraria, cualquier ejecución finita puede extenderse a una infinita satisfaciendo dicha propiedad. Consecuentemente, las propiedades de viveza no pueden guiar a un intérprete durante la construcción del modelo; por contra, dichas propiedades sólo se pueden sintetizar mediante estrategias explícitas expresadas como propiedades de seguridad, o postergar su síntesis a una fase posterior de ajuste fino, en el que la metodología proporciona un conjunto de estrategias o sugerencias para la propiedad.

9.6 Relojes en las reglas de síntesis

Sin duda, el problema más complejo a la hora de realizar una síntesis incremental a partir de SCTL-T proviene de las características imperativas en el modelo MUS-T: los relojes en el modelo y su reinicio en transiciones discretas. Si bien la fórmula se interpreta sobre un grafo denso de estados con transiciones temporales y discretas, no es posible, en el modelo MUS-T, reiniciar un reloj en una transición temporal. Además, el número de relojes en el modelo es una característica imperativa no incluida en una especificación orientada a propiedad.

Al margen de dicha restricción, los relojes existentes en una regla de síntesis pueden obedecer, al menos, a tres intenciones del usuario:

- **Relojes declarativos**. Son relojes de especificación "observadores", no existentes en el modelo; se utilizan en una regla de síntesis para forzar distancias temporales en los comportamientos del modelo. Las restricciones de tiempo que imponen han de ser satisfechas por los relojes existentes en el modelo.
- **Relojes de modelo**: Son relojes existentes en el modelo actual del sistema, y pueden aparecer en una regla de síntesis mediante la que se especifican nuevas guardas o invariantes en las que intervienen dichos relojes.
- **Relojes imperativos**: Son relojes nuevos en el modelo. Obedecen a la intención de introducir un nuevo reloj comúnmente asociado a un nuevo comportamiento temporizado.

Como ya hemos mencionado, el reinicio de relojes explícito en las transiciones discretas del modelo no permite la aplicación genérica del operador $x.$, sólo pudiéndose ligar libremente los relojes declarativos. Tomando en consideración la necesidad de permitir la introducción de nuevos relojes en el modelo y el reinicio de relojes ya existentes, introducimos una variante sintáctica de dicho operador. De esta forma, el ligado de un reloj en una regla de síntesis puede obedecer a dos variantes de distinta naturaleza:

- Ligado declarativo ($x.$): Liga un reloj declarativo a un contexto temporal.

- Ligado imperativo ($a\{x\}$): Reinicia un reloj (imperativo o de modelo) en una transición discreta a.

El uso que una regla de síntesis puede hacer sobre los distintos relojes son las siguientes:

- Relojes de modelo: Los relojes de modelo utilizan una forma de ligado imperativo opcional, dado que son relojes ya existentes en el modelo:
  - Si se incorpora el ligado del reloj en la fórmula, ésta ha de ser cerrada y monodireccional (definición 7.3), refiriéndose los puntos de chequeo del reloj a distancias temporales medidas desde el punto de ligado.
  - Si se utiliza una forma relajada sin ligado, los puntos de chequeo pueden ocurrir libremente en la fórmula:
    - Si el punto de chequeo pertenece a la parte declarativa, permite determinar el contexto de síntesis a partir de las guardas e invariantes ya existentes en el modelo.
    - Si el punto de chequeo pertenece a la parte imperativa, fuerza un nuevo comportamiento temporizado, medido a partir de los puntos de ligado ya existentes en el modelo.
- Relojes declarativos: Los relojes declarativos son sintáctica y semánticamente idénticos a los relojes de especificación en SCTL-T y, por tanto, las reglas de síntesis que incluyan dichos relojes, han de ser cerradas y monodireccionales. Dependiendo de la situación del punto de chequeo:
  - En la parte declarativa: determinan el contexto de síntesis.
  - En la parte imperativa: determinan las características temporizadas del nuevo comportamiento.
- Relojes imperativos: Los relojes nuevos utilizan una forma de ligado imperativo obligatoria. Si bien el ligado puede ocurrir tanto en la parte declarativa como en la parte imperativa, el punto de chequeo sólo puede ocurrir en la parte imperativa de la regla, especificando el nuevo comportamiento temporizado. Su uso debe llevar a reglas de síntesis cerradas y monodireccionales.

Reloj	Uso	Punto de Ligado	Punto de Chequeo	Tipo de Ligado
Declarativo	D	SI	SI	$x.$: Necesario
	I	SI	SI	
Modelo	D	SI	SI	$\{x\}$: Opcional
	I	SI	SI	
Imperativo	D	SI	NO	$\{x\}$: Necesario
	I	SI	SI	

Tabla 9.2. Tipos de Relojes, D: parte declarativa de la regla, I: parte imperativa de la regla

La tabla 9.2 resume la utilización de los relojes en las reglas de síntesis, imponiendo las restricciones de buena formación: cerradas en las reglas con ligado necesario (imperativos y declarativos) y monodireccionales

Notar que, para una mayor simplicidad en la expresión de las reglas de síntesis, se ha permitido el ligado imperativo en la parte declarativa de la regla, posibilitando la incorporación de un reloj en una transición discreta ya existente que forma parte del contexto de síntesis.

En resumen, podemos decir que el ligado de relojes en la parte declarativa obedece o bien a relojes imperativos y de modelo, cuyo comportamiento temporizado se especifica en la parte imperativa, o bien a relojes de especificación "observadores" (declarativos), que miden distancias temporales necesarias para determinar el contexto de síntesis.

9.7 Formas básicas de síntesis en una región

Como ya hemos mencionado, las reglas de síntesis se materializarán sobre regiones $\mathcal{R}$ del grafo RMUS-T del modelo actual. Para una región genérica $\mathcal{R} =< s, \mathcal{RT}, c_e >$, ampliada con su condición de accesibilidad, denotaremos $ce(\mathcal{R})$ a la condición de especificación c_e.

Una regla de síntesis $\mathcal{RS}$ será una fórmula SCTL-T con el siguiente patrón:

$$\mathcal{RS} := \phi_p \otimes_s \phi^1_{fs} \wedge \cdots \wedge \phi^n_{fs}$$

especificando un conjunto de comportamientos de futuro no estricto $\{\phi^1_{fs} \cdots \phi^n_{fs}\}$ a sintetizar.

La parte declarativa ϕ_p de la regla identifica el conjunto de regiones objeto de síntesis $\{\mathcal{R}_o\}$, siendo $\mathcal{R}_o$ una región tal que $GS(\phi_p, \mathcal{R}_o) = 1$ o bien la región inicial (si $\phi_p = ini$). La parte imperativa $\phi^1_{fs} \wedge \cdots \wedge \phi^n_{fs}$ se materializa en aquellas regiones $\mathcal{R}_a \in \perp ((\otimes_s, \phi_p), \mathcal{R}_o)$ en función del operador de síntesis $\otimes_s$ y las acciones de aplicabilidad determinadas por ϕ_p.

Los comportamientos a sintetizar se basan en dos actualizaciones básicas: el cambio de la condición de especificación de las transiciones discretas y temporales subespecificadas. Para una región genérica $\mathcal{R}$:

- El paso de la condición de especificación de la transición temporal $\mathcal{R} \overset{t}{\rightsquigarrow} \mathcal{R}'$ de $\frac{1}{2}$ a una condición 0 (no posible) o 1 (posible).
  - El cambio $\frac{1}{2} \rightarrowtail 0$ supone la eliminación de la transición temporal. El cambio realizado conlleva la modificación de la invariante de parada $I_p(s)$, siendo s el estado de control al que pertenece $\mathcal{R}$.
  - El cambio $\frac{1}{2} \rightarrowtail 1$ supone la especificación posible de la transición temporal. El cambio realizado conlleva la modificación de la invariante de avance $I_a(s)$, siendo s el estado de control al que pertenece $\mathcal{R}$.
  - Los cambios $1 \rightarrowtail 0$ y $0 \rightarrowtail 1$ se dicen de síntesis no factible.
- El paso de la condición de especificación de una acción a de $c_A(a, \mathcal{R}) = \frac{1}{2}$ a una condición 0 (no posible) ó 1 (posible).
  - El cambio $c_A(a, \mathcal{R}) = \frac{1}{2} \rightarrowtail 0$ prohíbe la acción a en la región $\mathcal{R}$, es decir, $a \in A_{NP}(\mathcal{R})$. El cambio conlleva la eliminación de la transición

etiquetada en a y, por tanto, la modificación de $g_{np}(s,a)$ donde s es el estado de control al que pertenece la región $\mathcal{R}$.

- El cambio $c_A(a,\mathcal{R}) = \frac{1}{2} \rightarrowtail 1$ supone $a \in A_P(\mathcal{R})$ e, implícitamente, la especificación posible de la transición etiquetada en a. El cambio conlleva la modificación de $g_p(s,a)$ (y eventualmente $g_s^P(s,a)$) donde s es el estado de control al que pertenece la región $\mathcal{R}$. Dado que el modelo es determinista, si existe una transición $\overset{a}{\rightsquigarrow}$ con estado destino $s' \neq s_s$, esa será la transición elegida. Si, por contra, la acción se encontraba totalmente subespecificada, el estado destino será s_s.
- Los cambios $c_A(a,\mathcal{R}) = 1 \rightarrowtail 0$ y $c_A(a,\mathcal{R}) = 0 \rightarrowtail 1$ se dicen de síntesis no factible.

9.8 Significado de la parte imperativa

La semántica de la parte imperativa de una regla de síntesis en una región objeto $\mathcal{R}_o$ se reduce a imponer $GS(\phi^i_{fs}, \mathcal{R}_a) = 1$ en cada uno de los elementos ϕ^i_{fs} de la conjunción imperativa, para cada una de las regiones de aplicabilidad $\mathcal{R}_a$.

Definición 9.2. **Síntesis factible de un elemento.** Diremos un elemento ϕ^i_{fs} de la conjunción imperativa es de síntesis factible si $GS(\phi^i_{fs}, \mathcal{R}_a) \geq \frac{1}{2}$, es decir, es posible forzar $GS(\phi^i_{fs}, \mathcal{R}_a) = 1$.

Definición 9.3. **Síntesis con éxito de un elemento.** Diremos que una regla de síntesis tiene éxito en una región objeto $\mathcal{R}_o$, sii para todo $\mathcal{R}_a$ y para todos sus elementos ϕ^i_{fs}, o bien ϕ^i_{fs} es de síntesis factible o bien ϕ^i_{fs} es no aplicable, es decir, $GS(\phi^i_{fs}, \mathcal{R}_a) = \widehat{\frac{1}{2}}$.

9.8.1 Síntesis de proposiciones atómicas

Sea ϕ un elemento atómico en la parte imperativa de la regla de síntesis aplicada a $\mathcal{R}_o$, y $\mathcal{R}_a$ una región de aplicabilidad de dicha regla:

- $\phi = a \in A$: Impone para la acción a un grado de satisfacción $GSA(a,\mathcal{R}_a) = 1$. La síntesis es factible si:
  - $c_A(a,\mathcal{R}_a) = 1$.
  - $c_A(a,\mathcal{R}_a) = \frac{1}{2}$, en cuyo caso cambiamos la condición de especificación de a: $c_A(a,\mathcal{R}_a) = \frac{1}{2} \rightarrowtail 1$.
- $\phi = \neg\, a \in A$: Impone para la acción a un grado de satisfacción $GSA(a,\mathcal{R}_a) = 0$. La síntesis es factible si:
  - $c_A(a,\mathcal{R}_a) = 0$.
  - $c_A(a,\mathcal{R}_a) = \frac{1}{2}$, en cuyo caso, cambiamos la condición de especificación de a: $c_A(a,\mathcal{R}_a) = \frac{1}{2} \rightarrowtail 0$.

- $\phi = \theta \in \{\emptyset, true, false\}$. Mientras la ausencia de proposición ($\emptyset$) no impone ninguna forma básica, las constantes:
  - $true$: fuerza $GSC(true, \mathcal{R}_a) = 1$. La síntesis es factible si:
    - $ce(\mathcal{R}_a) = 1$.
    - $ce(\mathcal{R}_a) = \frac{1}{2}$, en cuyo caso realizamos el cambio $ce(\mathcal{R}_a) = \frac{1}{2} \rightarrowtail 1$. Para ello, cambiamos la condición de especificación de la transición o transiciones que llevan a $\mathcal{R}_a$ a posible.
  - $false$: fuerza $GSC(false, \mathcal{R}_a) = 1$. La síntesis es factible si $ce(\mathcal{R}_a) = \frac{1}{2}$, en cuyo caso realizamos el cambio $ce(\mathcal{R}_a) = \frac{1}{2} \rightarrowtail 0$. Para ello, si $ce(\mathcal{R}_a)$ se refiere a una transición discreta, cambiamos su condición de especificación a no posible. Si $ce(\mathcal{R}_a)$ se refiere a una transición temporal, cambiaremos la condición de especificación de la transición temporal entrante en el grafo a no posible.
- $\phi = \psi \in \Psi$, fuerza $GSP(\psi, \mathcal{R}_a) = 1$. La síntesis sólo será factible si $\psi \supseteq \mathcal{RT}_a$, siendo $\mathcal{RT}_a$ la región de tiempo que define $\mathcal{R}_a$.

9.8.2 Síntesis de fórmulas causales

Sea $\phi = \phi_{fs} \forall \otimes_{fs} \phi_{fs}{}'$ un elemento causal en la parte imperativa de la regla de síntesis aplicada a $\mathcal{R}_o$, y $\mathcal{R}_a$ una región de aplicabilidad de dicha regla. Dado que:

$$GS(\phi_{fs} \forall \otimes_{fs} \phi_{fs}{}', \mathcal{R}_a) = \rightarrow \left(GS(\phi_{fs}, \mathcal{R}_a), \bigwedge_{\mathcal{R}'} (\neg\ ce(\mathcal{R}')) \vee GS(\phi_{fs}{}', \mathcal{R}') \right)$$

con $\mathcal{R}'$ las regiones de aplicabilidad, la síntesis procederá como sigue:

- si $GS(\phi_{fs}, \mathcal{R}_a) \leq \widehat{\frac{1}{2}}$, ϕ_{fs} es no aplicable en $\mathcal{R}_a$.
- si $GS(\phi_{fs}, \mathcal{R}_a) \geq \frac{1}{2}$, ϕ_{fs} es aplicable:
  - Procedemos a la síntesis de la premisa ϕ_{fs} en $\mathcal{R}_a$ (siempre factible).
  - Procedemos a la síntesis de la consecuencia en $\{\mathcal{R}'\}$. Para cada $\mathcal{R}'$:
    - Si $ce(\mathcal{R}') = 1$, es necesario forzar $GS(\phi_{fs}{}', \mathcal{R}') = 1$. Se procede a la síntesis de $\phi_{fs}{}'$ en $\mathcal{R}'$.
    - Si $ce(\mathcal{R}') = \frac{1}{2}$, para forzar $(\neg\ ce(\mathcal{R}')) \vee GS(\phi_{fs}{}', \mathcal{R}') = 1$, se presentan las siguientes opciones:
      - Se descarta el cambio $ce(\mathcal{R}') = \frac{1}{2} \rightarrowtail 0$, dado que siempre obtendríamos una solución trivial sin estados sucesores.
      - Por tanto, es necesario forzar $GS(\phi_{fs}{}', \mathcal{R}') = 1$, procediendo a la síntesis de $\phi_{fs}{}'$ en $\mathcal{R}'$. Adicionalmente, sería correcto el cambio $ce(\mathcal{R}') = \frac{1}{2} \rightarrowtail 1$; sin embargo, tal opción siempre llevaría a modelos máximos, en los que todas las evoluciones futuras son posibles.

Como se sigue de la síntesis de un elemento causal, las transiciones no modifican su condición de especificación. El cambio de la condición de especificación de las transiciones ha de ser explícito, mediante una proposición de tipo a o $\neg a$, o las constantes $true$ o $false$.

9.8.3 Síntesis de elementos $a\{x\}$

En esta sección nos centramos en las actualizaciones necesarias en el modelo originadas en un ligado $a\{x\}$. Como ya hemos mencionado, dicho ligado imperativo o de modelo supone la síntesis de a más la incorporación de un reloj x en el conjunto de reinicio λ asociado a la acción a. La síntesis de a sigue la reglas presentadas con anterioridad, la síntesis de $\{x\}$ sólo será factible si el valor de dicho reloj no influye en los comportamientos futuros del sistema. Más formalmente definimos a continuación el conjunto de relojes activos asociado a un estado s, siendo dichos relojes los que influyen en la evolución futura del sistema:

Definición 9.4. **Relojes locales activos**. Dado un estado $s \in S$ de un modelo MUS-T $\mathcal{M}$, definimos el conjunto de relojes activos localmente como la función $ActL(s)$:

$$ActL(s) = \{x \mid x \in I(s) \text{ o } x \in g(s,a)\} \text{ para algún } a \in A$$

siendo:

$$I(s) = \{I_a(s), I_p(s)\}$$
$$g(s,a) = \{g_p(s,a), g_{np}(s,a), g_s^P(s,a)\}$$

A partir del conjunto de relojes activos localmente se calcula el conjunto de relojes activos en un estado $Act(s)$ como:

$$Act(s) := ActL(s) \cup \bigcup_{s' \in suc(s)} Act(s')$$

siendo $s' \in suc(s)$ el conjunto de estados s' tales que $(s, < a, g_p(s,a), \lambda >, s')$ o $(s, < a, g_s^P(s,a), \lambda >, s')$ pertenece al conjunto de transiciones T del modelo $\mathcal{M}$ considerado.

Sea $\phi = a\{x\}$ la fórmula a sintetizar en una región $\mathcal{R}_a$. La síntesis será factible si $GSA(a, \mathcal{R}_a) \in \{\frac{1}{2}, 1\}$ y $\{x\} \notin Act(s)$, siendo s el estado de control al que pertenece la región $\mathcal{R}_a$. Si la síntesis es factible:

- Se fuerza $c_A(a, \mathcal{R}_a) = 1$.
- Se introduce el reloj x en el conjunto de reinicio de a. Dado que se trata de un reloj inactivo, los comportamientos futuros serán idénticos.

La síntesis de un elemento $a\{x\}$ supone un mayor ámbito de incidencia en el modelo RMUS-T que el resto de los elementos definidos:

- Si $x \in \mathcal{C}$, se trata de un reloj de modelo (ya existente) que va a ser añadido en el conjunto de reinicio de la transición discreta a, $\lambda(\mathcal{R}, a)$. Tal incorporación supone la actualización de las regiones de tiempo alcanzables mediante la transición discreta a. Sin embargo, dado que se trata de un reloj inactivo, tal actualización no modificará la estructura del subárbol con origen en $\mathcal{R}_a$.

- Si $x \notin \mathcal{C}$, se trata de un nuevo reloj (reloj imperativo). Su incorporación es idéntica al caso anterior, dado que es siempre un reloj inactivo. Esta incorporación supone un cambio global en el sistema, dado que las interpretaciones de los relojes del modelo pasan de un dominio $\Re_+^{\#\mathcal{C}}$ a un dominio $\Re_+^{\#\mathcal{C}+1}$. Sin embargo, dado que es un nuevo reloj, su consideración sólo es necesaria para la síntesis de comportamientos futuros.

9.9 Ejemplos de síntesis

9.9.1 Formas básicas

Observemos el grafo RMUS-T simple en la figura 9.2(a). Los cambios básicos que dicho grafo admite en la región $< \mathsf{ON}, (1 < x < 2) >$ se presentan en los siguientes ejemplos.

Ejemplo 9.1. En 9.2(b), forzamos un grado se satisfacción 1 para la constante $true$, siendo $< \mathsf{ON}, (1 < x < 2) >$ una región de aplicabilidad de un operador $\forall \Rightarrow_+$ en la región $(\mathsf{ON}, x = 0)$. Dicho cambio supone modificar la condición de especificación de la transición $< \mathsf{ON}, x = 0 > \overset{t}{\rightsquigarrow} < \mathsf{ON}, (1 < x < 2) >$, haciéndola posible. El modelo MUS-T resultante del cambio será idéntico al original, salvo $I_a(\mathsf{ON}) := I_a(\mathsf{ON}) \cup (1 < x < 2)$. □

Ejemplo 9.2. En 9.2(c), cambiamos el grado de satisfacción de la acción botón a 1. Este cambio implica el cambio de la condición de especificación de la acción botón a posible. El modelo MUS-T resultante del cambio será idéntico al original, salvo $g_p(\mathsf{ON}, \mathsf{botón}) := g_p(\mathsf{ON}, \mathsf{botón}) \cup (1 < x < 2)$. □

Ejemplo 9.3. En 9.2(d), forzamos un grado se satisfacción 1 para la constante $false$, siendo $< \mathsf{ON}, (1 < x < 2) >$ una región de aplicabilidad de un operador $\forall \Rightarrow_+$ en la región $< \mathsf{ON}, x = 0 >$. Dicho cambio supone deshabilitar la transición temporal t que lleva a ese estado (factible). El modelo MUS-T resultante del cambio será idéntico al original, salvo $I_p(\mathsf{ON}) := I_p(\mathsf{ON}) \cup (1 < x < 2)$. □

Ejemplo 9.4. En 9.2(e), cambiamos el grado de satisfacción de la acción botón a 0. Este cambio implica el cambio de la condición de especificación de la acción botón a no posible. El modelo MUS-T resultante del cambio será idéntico al original, salvo $g_{np}(\mathsf{ON}, \mathsf{botón}) := g_{np}(\mathsf{ON}, \mathsf{botón}) \cup (1 < x < 2)$. □

Las anteriores son las formas básicas de síntesis en la metodología propuesta, que, como vemos, se traducen al cambio de la condición de especificación de las transiciones temporales (mediante las constantes $true$ y $false$) y discretas (genéricas, mediante las constantes $true$ y $false$, y etiquetadas, con a o $\neg\ a$).

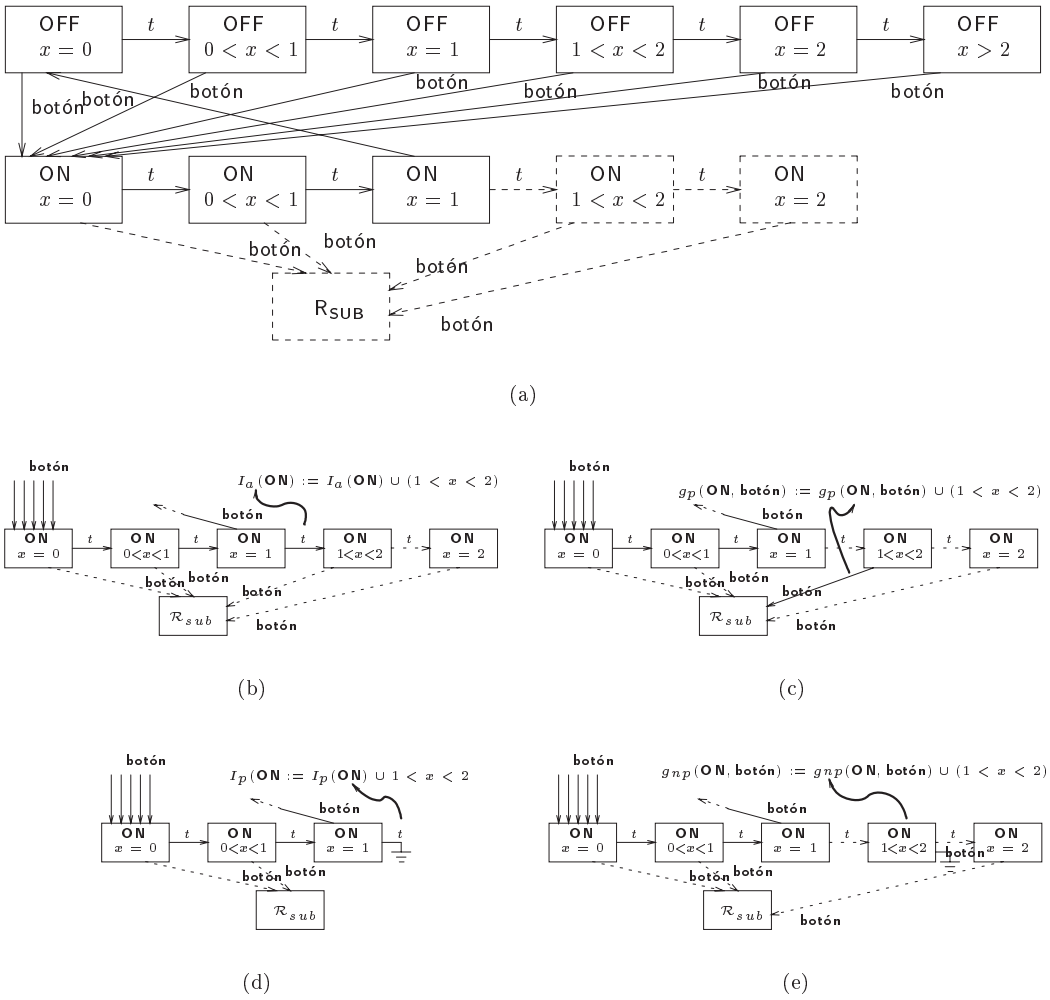


Figura 9.2. Cambios en la condición de especificación

9.9.2 Reglas de síntesis

A continuación se muestran ejemplos de reglas de síntesis simples, en las que la constante $at_{\mathcal{R}}$ se refiere a algún criterio que selecciona la región.

EJEMPLO 9.5. **Estado inicial**: $\{RS_0 := ini \Rrightarrow (p_1 \wedge \cdots \wedge p_n)\}$. Supone la síntesis de la conjunción $(p_1 \wedge \cdots \wedge p_n)$ en la región $< s_0, \mathcal{R}_0, 1 >$, siendo $\mathcal{R}_0$ la región que contiene como única interpretación la inicial. □

EJEMPLO 9.6. **Fórmulas sucesoras**. Cláusulas que conllevan la evaluación de la parte declarativa en una región, y la síntesis de la parte imperativa en regiones alcanzables mediante una transición discreta:

- En la figura 9.3(a), se muestra $\{RS_1 := a \ \forall\Rrightarrow\bigcirc \ (a \wedge b)\}$: Inmediatamente después de una a-transición, la acción a y la acción b son posibles. Esto se traduce en que, en la región sucesora por una a-transición, se especifican a y b como posibles, siendo su estado sucesor subespecificado.

- En la figura 9.3(b), se muestra $\{RS_2 := at_{\mathcal{R}} \ \forall\Rrightarrow\bigcirc \ a \wedge b\}$, donde $at_{\mathcal{R}}$ es algún criterio que selecciona la región mediante una fórmula sin acciones simples (las acciones de aplicabilidad es el total del alfabeto, $AA = A$).

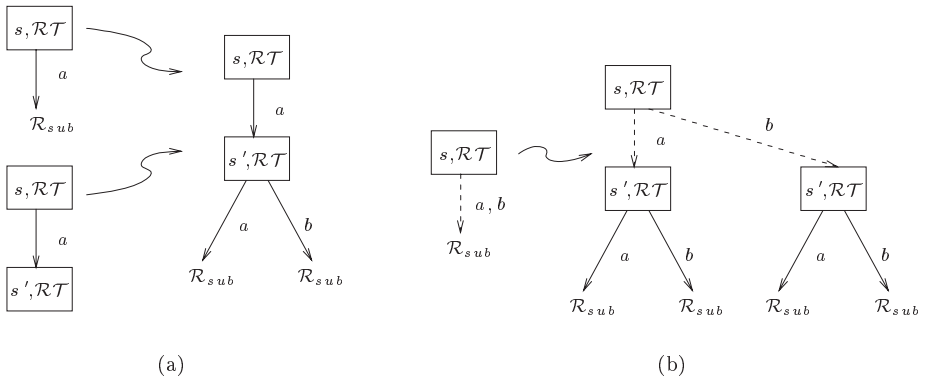


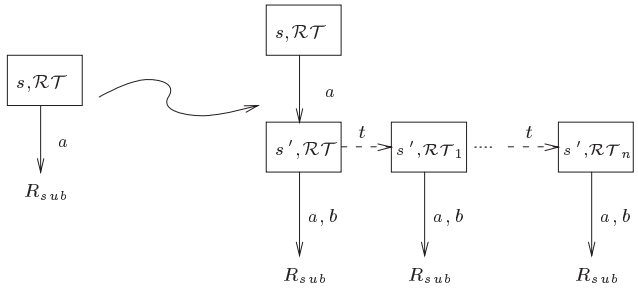
Figura 9.3. Síntesis de $\{RS_1 := a \ \forall\Rrightarrow\bigcirc \ (b \wedge c)\}$ (subfigura 9.3(a)) y $\{RS_2 := at_{\mathcal{R}} \ \forall\Rrightarrow\bigcirc \ b \wedge c\}$ (subfigura 9.3(b))

□

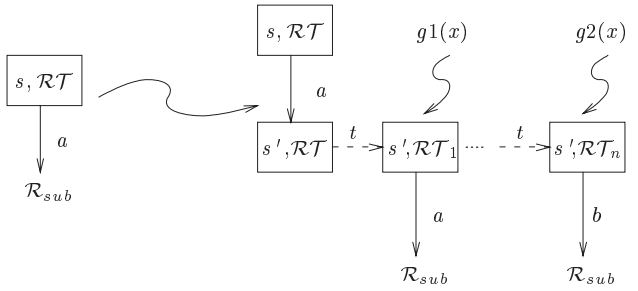
EJEMPLO 9.7. **Fórmulas sucesoras diferidas**. Suponen una evolución discreta y todas las configuraciones de tiempo alcanzables después del cambio de estado de control:

- En la figura 9.4(a), $\{RS_3 := a \ \forall\Rrightarrow\overset{\forall\rightarrow}{\bigcirc} \ a \wedge b\}$: Si desde una región podemos evolucionar mediante un evento discreto a, en la región a-sucesora, a y b son posibles en cualquier contexto temporal futuro.

- En la figura 9.4(b), $\{RS_4 := x.a \ \forall\Rightarrow\overset{\forall\rightarrow}{\bigcirc} \ (g_1(x) \Rightarrow a \wedge g_2(x) \Rightarrow b)\}$: Si desde una región podemos evolucionar mediante un evento a, en la región a-sucesora, cuando se cumpla la guarda de tiempo g_1, es posible a, y cuando se cumpla g_2, es posible b.



(a)



(b)

Figura 9.4. Síntesis de $\{RS_3 := a \ \forall\Rightarrow\overset{\forall\rightarrow}{\bigcirc} \ a \wedge b\}$ (subfigura 9.4(a)) y $\{RS_4 := x.a \ \forall\Rightarrow\overset{\forall\rightarrow}{\bigcirc} \ (g_1(x) \Rightarrow a \wedge g_2(x) \Rightarrow b)\}$ (subfigura 9.4(b))

□

Ejemplo 9.8. **Especificación de invariantes**. La utilización de las constantes $true$ y $false$ permite la especificación de las invariantes de avance y de parada:

- En la figura 9.5(a), $\{RS_5 := at_{\mathcal{R}} \ \forall\Rightarrow_+ \ (g(x) \Rightarrow true)\}$, donde x es un reloj de modelo, permite determinar la invariante de avance.

- En la figura 9.5(b), $\{RS_6 := at_{\mathcal{R}} \ \forall\Rightarrow_+ \ (g(x) \Rightarrow false)\}$ donde x es un reloj de modelo, permite determinar la invariante de parada.

- De forma similar, utilizando un reloj declarativo obtendríamos la regla de especificación de avance $\{RS_7 := x.at_{\mathcal{R}} \ \forall\Rightarrow_+ \ (g(x) \Rightarrow false)\}$ y la regla de especificación de parada $\{RS_8 := x.at_{\mathcal{R}} \ \forall\Rightarrow_+ \ (g(x) \Rightarrow true)\}$.

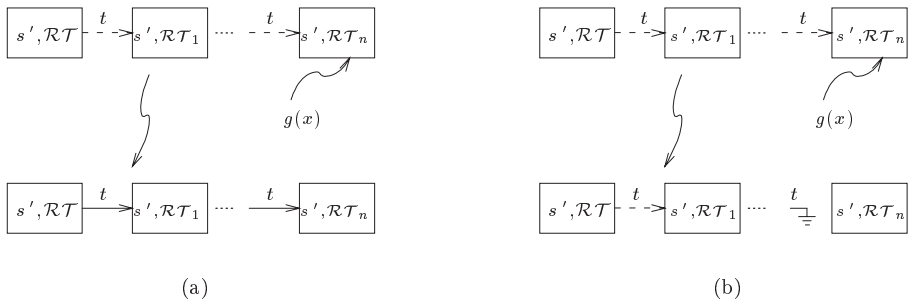


Figura 9.5. Cambio de condición de especificación de transiciones temporales: $\{RS_5 := at_{\mathcal{R}}\ \forall\Rightarrow_+\ (g(x) \Rightarrow true)\}$ (subfigura 9.5(a)) y $\{RS_6 := at_{\mathcal{R}}\ \forall\Rightarrow_+\ (g(x) \Rightarrow false)\}$ (subfigura 9.5(b))

□

Ejemplo 9.9. **Reinicio de un reloj en una transición discreta:** $a\{x\}$. En la figura 9.6(a) podemos ver el caso de un reloj de modelo y en la figura 9.6(b) el caso de un reloj imperativo. Cuando el reloj en el reinicio es un reloj modelo se modifican las regiones de tiempo por evoluciones futuras; por contra, cuando se trata de un reloj imperativo aparecen nuevas regiones de tiempo bajo las mismas condiciones de especificación en la estructura original. □

9.10 Selección de un estado de control sucesor discreto

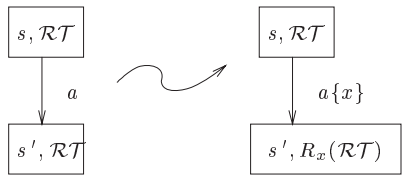
La síntesis en una región $\mathcal{R}$ de un elemento del tipo:

$$\phi^i_{fs} := \phi_{fs} \otimes_s \phi_{fs}{}'$$

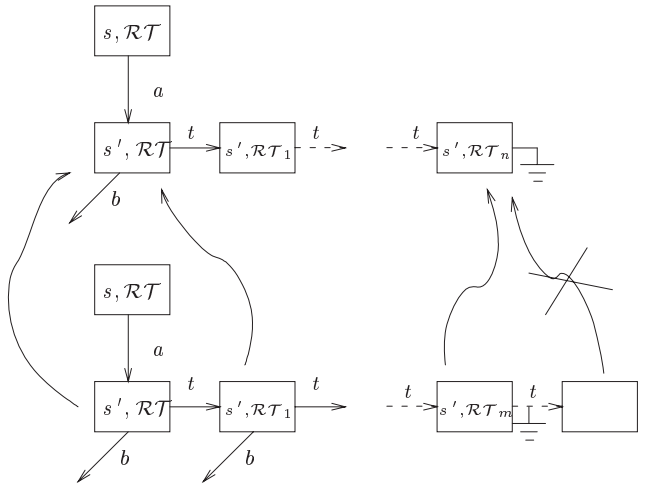
impone, fundamentalmente, la síntesis en regiones sucesoras de la región $\mathcal{R}$, ya sea mediante transiciones temporales ($\Rightarrow_+$) o mediante transiciones discretas ($\Rightarrow \bigcirc$). Si se trata de una transición temporal, los estados sucesores, de existir, serán únicos y presentes en la abstracción. Si, por contra, se trata de una fórmula sobre los sucesores discretos, puede ser necesario la creación de una nueva región a-sucesora de la actual.

Sea una fórmula $\phi^i_{fs} := \phi_{fs}\ \forall\Rightarrow\bigcirc\ \phi_{fs}{}'$ (factible) a sintetizar en una región $\mathcal{R}$, y sea AA el conjunto de acciones de aplicabilidad determinadas por la premisa ϕ_{fs}. Para todo $a \in AA$:

- Si existe una transición $\mathcal{R} \overset{a}{\rightsquigarrow} \mathcal{R}'$, por el criterio determinista, la síntesis debe ser realizada en $\mathcal{R}'$.
- Si $\mathcal{R}$ está totalmente subespecificada en la acción a ($\mathcal{R} \overset{a}{\rightsquigarrow} \mathcal{R}_s$), tenemos dos opciones:



(a)



(b)

Figura 9.6. Síntesis de un elemento $a\{x\}$ para el caso de reloj de modelo (subfigura 9.6(a)) y para el caso de reloj imperativo (subfigura 9.6(b))

– Si el modelo es determinista en control impone que, la síntesis en una región a-sucesora de una región $\mathcal{R}$ tal que $\mathcal{R} \overset{a}{\rightsquigarrow} \mathcal{R}_s$ y existe $\mathcal{R}'$ cuyo estado de control s coincide con el estado de control de $\mathcal{R}$, tal que $\mathcal{R}' \overset{a}{\rightsquigarrow} \mathcal{R}''$, debe realizarse en una región $\mathcal{R}''$ cuyo estado de control coincida con el de $\mathcal{R}'$.

– Si el criterio de determinismo en control no ha encontrado la región sucesora discreta o no se impone determinismo en control, es necesario crear una nueva región sucesora en la que sintetizar $\phi_{fs}{}'$. Evidentemente, con esta solución simple la síntesis del conjunto de invariantes que definen las reglas de síntesis del modelo podría llevar a un grafo de regiones RMUS-T con un número de regiones no finito y, consecuentemente, a un modelo MUS-T igualmente infinito. Es, por tanto, estrictamente necesario, cada vez que se ha de sintetizar en una región sucesora discreta no existente, hacer una exploración del conjunto de regiones en busca de una región compatible con la región sucesora a sintetizar (ver siguiente sección).

9.10.1 Criterios de compatibilidad entre regiones

Esta sección se refiere a la búsqueda de una región compatible para síntesis en los casos en que dicha región no ha sido determinada, ni por el criterio de determinismo, ni por el criterio de determinismo de control. Sea $\phi_{fs}{}'$ la parte de la regla de síntesis $\phi_{fs}\forall \Rightarrow \bigcirc\phi_{fs}{}'$ a sintetizar en una región a-sucesora de la región $\mathcal{R}$. Sea $\mathcal{R}^{Req}$ una región a-sucesora de $\mathcal{R}$ obtenida mediante una transición discreta a a un estado de control s^{Req} ficticio, en la que se sintetiza $\phi_{fs}{}'$. Como se muestra en la figura 9.7, la búsqueda de una región compatible se reduce a encontrar una región $\mathcal{R}^{RC}$ en el modelo RMUS-T $\mathcal{R}^{\mathcal{M}}$ cuya relación con $\mathcal{R}^{Req}$ cumpla uno de los criterios de compatibilidad que se definen a continuación.

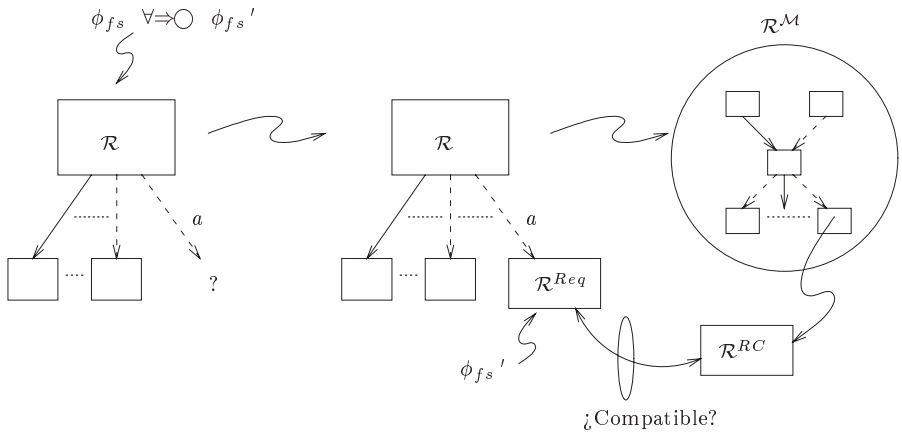


Figura 9.7. Mecanismo de búsqueda de una región compatible para la síntesis de una regla de síntesis sucesora discreta

Para $\mathcal{R}$ una región del grafo, definimos $in(\mathcal{R})$ como el conjunto de transiciones $t \in \mathcal{TR}$ entrantes, $out(\mathcal{R})$ como el conjunto de transiciones $t \in \mathcal{TR}$ salientes, y

$RT(\mathcal{R})$ la región de tiempo asociada a $\mathcal{R}$. Para cada transición t denotamos $ce(t)$ a la condición de especificación de t, y $label(t)$ a la etiqueta discreta $\in A$ o la etiqueta temporal t de la transición en el grafo.

Definición 9.5. **Regiones Compatibles Totalmente(TIPO RCT)**: Para una región $\mathcal{R}^{Req}$, son regiones RCT, aquellas regiones $\mathcal{R}^{RCT} \in \mathcal{SR}$ en el grafo RMUS-T, que no imponen pérdida de subespecificación ni en el modelo ni en el requisito, además de garantizar la preservación de los operadores de pasado:

- $RT(\mathcal{R}^{Req}) = RT(\mathcal{R}^{RCT})$
- $\forall t \in out(\mathcal{R}^{Req})$ existe $t' \in out(\mathcal{R}^{RCT})$, tal que $t = t'$ y las regiones destino son también RCT.
- $\forall t \in in(\mathcal{R}^{Req})$ existe $t' \in in(\mathcal{R}^{RCT})$, tal que $t = t'$ y las regiones origen son también RCT.
- $\forall t \in in(\mathcal{R}^{RCT})$ existe $t' \in in(\mathcal{R}^{Req})$, tal que $t = t'$ y las regiones origen son también RCT.

Como veremos, en el capítulo 10, tal criterio se corresponde con una bisimulación *backward-forward*. Si bien la definición no es la común en una bisimulación, es equivalente si tenemos en cuenta que el modelo es determinista.

Definición 9.6. **Regiones Compatibles Totalmente para Futuro (TIPO RCTF)**: Para una región $\mathcal{R}^{Req}$, son regiones RCTF aquellas regiones $\mathcal{R}^{RCTF} \in \mathcal{SR}$ que no suponen pérdida de subespecificación ni en el modelo ni en el requisito:

- $RT(\mathcal{R}^{Req}) = RT(\mathcal{R}^{RCTF})$
- $\forall t \in out(\mathcal{R}^{Req})$ existe $t' \in out(\mathcal{R}^{RCTF})$, tal que $t = t'$ y las regiones destino son también RCTF.

Como veremos, en el capítulo 10, tal criterio se corresponde con una bisimulación *forward*.

Definición 9.7. **Regiones Compatibles sin pérdida de subespecificación en Modelo (TIPO RCM)**: Para una región $\mathcal{R}^{Req}$, son regiones compatibles RCM aquellas cuya región de tiempo coincide con la original y cuyas condiciones de especificación (especificadas) contienen a las condiciones de especificación en el requisito:

- $RT(\mathcal{R}^{Req}) = RT(\mathcal{R}^{RCM})$
- $\forall\ t \in out(\mathcal{R}^{Req})$, existe $t' \in out(\mathcal{R}^{RCM})$, tal que $label(t) = label(t')$, $ce(t) \leq_c ce(t')$ y las regiones destino son también RCM.

Como veremos, en el capítulo 10, tal criterio se corresponde con una simulación especificada tal que el modelo simula el requisito.

Definición 9.8. **Regiones compatibles sin pérdida de subespecificación en Requisito (TIPO RCR)**: Para una región $\mathcal{R}^{Req}$, son regiones compatibles RCR, aquellas regiones cuya región de tiempo es idéntica a la original y cuyas condiciones de especificación (especificadas) están contenidas en las condiciones de especificación en el modelo:

- $RT(\mathcal{R}^{RCR}) = RT(\mathcal{R}^{Req})$
- $\forall\ t \in out(\mathcal{R}^{RCR})$ existe $t' \in out(\mathcal{R}^{Req})$, tal que $label(t) = label(t')$, $ce(t) \leq_c ce(t')$ y las regiones destino son también RCR.

Como veremos en el capítulo 10 tal criterio se corresponde con una simulación especificada tal que el requisito simule el modelo.

Como se demostrará en el capítulo 10, un criterio RCT asegura siempre la corrección máxima de la síntesis, es decir, para cualquier regla de síntesis aplicada al modelo, su condición de aplicable y su condición de éxito serían idénticas si no hubiésemos reutilizado un estado compatible RCT. Por tanto, la síntesis no factible de una regla de síntesis sobre un modelo generado con un criterio RCT asegura que el nuevo requisito contradice un requisito anterior.

Los demás criterios se pueden clasificar como sigue:

- El criterio RCTF, por no preservar las fórmulas de pasado, puede introducir falsos aplicables y falsos no aplicables. Para el caso del cuantificador $\forall$ sobre el operador de pasado, si la $\mathcal{RS}$ es aplicable, también lo sería en el caso de no haber utilizado el criterio RCTF. Inversamente, para el caso de cuantificador $\exists$ sobre el operador de pasado, si $\mathcal{RS}$ es no aplicable, tampoco lo sería en el caso de no haber utilizado el criterio RCTF. En general, podemos decir que el criterio RCTF no preserva la propiedad de aplicable de una regla de síntesis si ésta incluye operadores de pasado, sin embargo, preserva la propiedad de éxito de la síntesis.
- Por último los criterios más débiles, RCR y RCM, además de falsos aplicables y no aplicables, pueden, para el caso de una regla aplicable, producir falsas síntesis con éxito y falsas síntesis sin éxito. Sin embargo, si la síntesis tiene éxito, nos aseguramos de que el modelo sintetizado contiene todos los comportamientos especificados.

La elección utilizada en el algoritmo de síntesis SCTL (García Duque, 2000) es RCM, dando prioridad al modelo sobre el requisito, en un intento de que las modificaciones en el modelo sean las mínimas posibles, aunque esto implique sobreespecificar algunos elementos que en el requisito se encuentran subespecificados. Sin embargo, un criterio RCM no garantiza la corrección RCT, dado que reutilizamos regiones que pueden incorporar características a mayores de las impuestas en el requisito.

Todos los criterios arriba presentados suponen que la región de tiempo del estado a reutilizar es idéntica a la obtenida por síntesis. Tal criterio garantiza la preservación del grado de satisfacción de un predicado de tiempo que incluya relojes de modelo. Sin embargo, se trata de una suposición demasiado fuerte,

cuando, en definitiva, el objetivo de las reglas de síntesis es especificar distancias temporales entre comportamientos y no medidas absolutas.

Definición 9.9. **Regiones compatibles con reinicio implícito de relojes de modelo (TIPO RCXReset)**: Para cualquier criterio de compatibilidad (RCT, RCTF, RCM, RCR), se define el criterio correspondiente (RCTReset, RCTFReset, RCMReset, RCRReset) que relaja la igualdad de la región de tiempo permitiendo el reinicio de relojes de modelo en la transición discreta:

- $R_\lambda(RT(\mathcal{R}^{Req})) = \mathcal{R}^{RCX}$

9.11 Recomputaciones necesarias y notificación de conflicto

La corrección de la síntesis incremental del modelo se basa en que para cualquier conjunto de reglas de síntesis $\mathcal{RS}$ no contradictorias, el modelo obtenido mediante síntesis incremental satisfaga todas las reglas de síntesis en $\mathcal{RS}$. Para asegurar dicha corrección, hay que tener en consideración lo siguiente:

- El procedimiento de síntesis aplica las reglas de síntesis en las distintas regiones del grafo, preservando el orden temporal de ejecución. La separación de pasado declarativo (criterio de selección) y futuro imperativo (modificación del modelo) garantiza la corrección de dicho mecanismo: dado que el criterio de selección sólo depende del pasado y las modificaciones impuestas afectan sólo al futuro, nunca tendremos que reevaluar la parte declarativa de las reglas de síntesis en estados ya visitados.

- El crecimiento de un modelo MUS-T, es decir, el cambio de condición de un elemento subespecificado a posible o a no posible, puede implicar la recomputación de una regla de síntesis previa. Sin embargo, sólo será necesario la evaluación de la parte declarativa de reglas previas aún no sintetizadas en la región en la que se produce el cambio. En ese caso, una vez sintetizada la regla de síntesis actual, se procederá a la síntesis de las reglas (que han pasado a ser aplicables) procedentes de fases de diseño anteriores.

- En caso de no poder realizarse dicha síntesis se notificará el conflicto. La notificación del conflicto implica la identificación de los requisitos contradictorios con el objetivo de permitir al diseñador decidir si desistir en la síntesis del nuevo requisito o eliminar del conjunto de síntesis algún requisito previo. Para tal fin, anotamos el modelo MUS-T con un conjunto de expresiones simbólicas que almacenen la historia de evolución de cada uno de los elementos. Si el criterio es RCT o RCTF sólo se notificará un conflicto si los requisitos son contradictorios. En otro caso, la notificación de conflicto puede ayudar al diseñador a seleccionar otro criterio en que la síntesis sea factible.

9.11.1 Notificación de conflicto

Sea ω un elemento de un modelo MUS-T, siendo $\omega \in A \cup \{true, false\}$ y s un estado de control.

DEFINICIÓN 9.10. **Historia de síntesis**. La historia de síntesis de un elemento w en un estado s es una disyunción de predicados de tiempo ψ^i pertenecientes a un conjunto de requisitos $\{I_R^i\}$.

$$H(s,\omega) : \{\psi^1[I_{R_1}^1 \wedge \cdots \wedge I_{R_j}^1] \vee \cdots \vee \psi^i[I_{R_1}^i \wedge \cdots \wedge I_{R_j}^i,]\}$$

donde :

- $H(s,a)$: denota la historia de evolución de $g_p(s,a)$.
- $H(s,\neg a)$: respectivamente $g_{np}(s,a)$.
- $H(s,true)$: respectivamente $I_a(s)$.
- $H(s,false)$: respectivamente $I_p(s)$.

Cuando el proceso de síntesis se encuentra un cambio no factible:

- $c_A(a,\mathcal{R}_a) = 0 \rightarrowtail 1$, se selecciona el conjunto $\{I_{R_j}^i\} \in H(s,\neg a)$ tal que $\mathcal{R}_a \subseteq \psi^i$.
- $c_A(a,\mathcal{R}_a) = 1 \rightarrowtail 0$ se selecciona el conjunto $\{I_{R_j}^i\} \in H(s,a)$ tal que $\mathcal{R}_a \subseteq \psi^i$.
- $ce(\mathcal{R}_a) = 0 \rightarrowtail 1$ se selecciona el conjunto $\{I_{R_j}^i\} \in H(s,false)$ tal que $\mathcal{R}_a \subseteq \psi^i$.
- $ce(\mathcal{R}_a) = 1 \rightarrowtail 0$ se selecciona el conjunto $\{I_{R_j}^i\} \in H(s,true)$ tal que $\mathcal{R}_a \subseteq \psi^i$.

9.12 Algoritmo de Síntesis sobre MUS-T

9.12.1 Esbozo del Algoritmo

Sea $\mathcal{M}$ un componente MUS-T sintetizado a partir del conjunto de reglas de síntesis previas. La síntesis de un nuevo modelo MUS-T $\mathcal{M}'$ resultante de la incorporación de una nueva regla de síntesis $\mathcal{RS}$, implica las siguientes tareas:

- Evaluación de las cotas máximas de relojes imperativos, declarativos o de modelo en la regla $\mathcal{RS}$ y en el modelo $\mathcal{M}$.
- Cálculo del grafo RMUS-T del modelo actual en base a las cotas máximas anteriores.
- Síntesis de $\mathcal{RS}$:

- Si se trata de una regla *ini*, sin parte declarativa, aplicamos el algoritmo SI (algoritmo 9.1) a la región inicial $< s_0^{\mathcal{R}^0}, \mathcal{R}^0, 1 >$.
- Si se trata de una regla genérica $\phi_p \otimes_s \phi_{fs}$, aplicamos el algoritmo de síntesis SI (algoritmo 9.1) en un orden *depth-first* del grafo RMUS-T, partiendo de $< s_0^{\mathcal{R}^0}, \mathcal{R}^0, 1 >$.

9.12.2 Aplicación de una regla de síntesis en una región RMUS-T

Dada una regla de síntesis $\mathcal{RS} := \phi_p \otimes_s \phi_{fs}$ y $\mathcal{R}$ una región del grafo RMUS-T, el algoritmo de síntesis procede como sigue:

[1] Evaluación de la parte declarativa (si $\phi_p = ini$, entonces $\phi_p = \emptyset$):

- Si $GS(\phi_p, (\mathcal{R}, 1)) = 1$, sintetizar la parte imperativa ([2]).
- Si no, fin de síntesis.

[2] Evaluación y síntesis de la parte imperativa $\phi_{fs} = \phi_{fs}^1 \wedge \cdots \wedge \phi_{fs}^n$ en las regiones de aplicabilidad $\{\mathcal{R}_a\}$ determinadas por el operador de síntesis $\otimes_s$. Para todo ϕ_{fs}^i y $\mathcal{R}_a$:

- Si $GS(\phi_{fs}^i, \mathcal{R}_a) \in \{0, \frac{1}{4}\}$, imposible satisfacer la parte imperativa.
- Si $GS(\phi_{fs}^i, \mathcal{R}_a) = \widehat{\frac{1}{2}}$, no aplicable.
- Si $GS(\phi_{fs}^i, \mathcal{R}_a) \in \{\frac{1}{2}, \frac{3}{4}\}$, se procede a la síntesis ([3]).
- Si $GS(\phi_{fs}^i, \mathcal{R}_a) = 1$, $\mathcal{R}_a$ ya satisface la parte imperativa.

[3] Síntesis de la parte imperativa: La síntesis de la parte imperativa opera según lo descrito en la sección 9.8.

Como se sigue de la sección 9.6, se ha permitido la introducción de relojes imperativos en la parte declarativa de las reglas de síntesis, lo que permite añadir un reloj a una transición discreta ya existente en el modelo. Conforme a la definición de esta clase de relojes el algoritmo `MCGR(R,F,n)`, descrito en el pseudocódigo 8.3, se modifica incorporando la posibilidad de introducción de un nuevo reloj en el grafo.

El algoritmo 9.1 muestra, en pseudocódigo, la solución propuesta. Como se puede observar, el algoritmo siempre sintetiza (deshaciendo los cambios si la síntesis no es posible) y, por tanto, devuelve todos los conflictos posibles en la regla.

9.13 Traducción RMUS-T a MUS-T

Una vez sintetizado el requisito sobre el modelo RMUS-T actual, obtenemos un modelo RMUS-T modificado. En esta sección se resume la obtención del nuevo modelo MUS-T a partir del grafo RMUS-T modificado. La traducción se limita al cálculo de las guardas e invariantes en el modelo MUS-T. Para cada estado de control s en el grafo de regiones RMUS-T, se calcula:

Algoritmo 9.1 Algoritmo Síntesis Imperativa SI($< s, \mathcal{RT}, c_e >$,f,n), donde SIRC realiza la síntesis con la búsqueda de una región compatible

```
SI(region R=<s,RT,ce>, list<constructor_sctlt> f, int n){
    reset:=0;
    case f[n]
        Ereset(r): n:=n+1; reset:=1;
        Eaccion(a): n:=n+1;
            if a ∈ A_NP(R) then return (ERROR,notifica_conflicto(s,a));
            else
                A_P(R):=A_P(R) ∪ a;
                if (reset==1) then
                    if (r ∈ Act(s)) then return (ERROR,ACTIVO)
                    else λ(R,a)=λ(R,a) ∪ r; actualiza_futuro(R,a); return (OK);
                end if
            end else ;
        Eaccion(true) : n:=n+1; if (ce=1/2) then actualiza_transicion(true);
            return (OK);
        Eaccion(false) : n:=n+1;
            if (ce=1/2) then actualiza_transicion(false);
            else if (ce=1) then return (ERROR,notifica_conflicto(s,false));
            return (OK)
        Epredicado(ψ): n:=n+1;
            if (GSP(ψ,R)== 0) return (ERROR,{})
            else return (OK)
        Eaccion(∅) : n:=n+1; return (OK);
        Eoperador_logico(¬): n:=n+1; a:=f[n]; n:=n+1;
            if (a ∈ A_P(R)) then return (ERROR,notifica_conflicto(s,¬ a));
            else A_NP(R):=A_NP(R) ∪ a; return (OK);
        Eoperador_logico(∧): naux:=n; n=:n+1;
            Res1 := SI(R,f,n)); Res2 := SI(R,f,n));
            if ((Res1=OK) AND (Res2=OK)) then return (OK);
            else Deshacer(R,f,naux); return (Res1+Res2)
            end else
        Ecuantificador(Æ): /* Æ = ∀ */
            OT:= f[n+1]; naux:=n; n:= n+2;
            if ((GSP := MCGR(R,f,n)) ≤ 1̂/2) then return (OK);
            ResP := SI(R,f,n);
            if (OT ! = ⇒) then Marca(R); ResC := OK;
            else ResC:= SI(R,f,n);
            if (OT == ⇒+) then
                EA= EA(R,OT,{}); nC:=n;
                foreach R′ ∈ EA do
                    n:=nC; ResC := ResC + SI(R′,f,n);
            else /* ⇒ ◯ */
                AA= acciones-aplicabilidad(f,naux);C:=n;
                foreach a ∈ AA do
                    n:=nC;
                    if (R.Rsuc_d(a) ! = s_s) ResC:= ResC + SI(R.Rsuc_d(a),f,n)
                    else ResC := ResC + SIRC(R,f,n,a)
            end else ;
            if (ResP+ResC=ERROR) Deshacer(R,f,naux);
            return (ResP+ResC);
        end else ;
    end case }
```

- $I_a(s) = \bigcup_i \mathcal{R}_i$, tal que $\xrightarrow{t} < s^{\mathcal{R}_i}, \mathcal{R}_i >$ o $< s^{\mathcal{R}_i}, \mathcal{R}_i > \xrightarrow{t}$
- $I_p(s) = \bigcup_i \mathcal{R}_i$, tal que $\not\xrightarrow{t} < s^{\mathcal{R}_i}, \mathcal{R}_i >$
- Para todo $a \in A$:
  - $\begin{cases} g_p(s,a) = \bigcup_i \mathcal{R}_i, \text{ tal que } < s^{\mathcal{R}_i}, \mathcal{R}_i > \xrightarrow{a} \\ g_{np}(s,a) = \bigcup_i \mathcal{R}_i, \text{ tal que } < s^{\mathcal{R}_i}, \mathcal{R}_i > \not\xrightarrow{a} \end{cases}$
  - $\lambda(s,a) = \bigcup_i \{x_i\}$, tal que $< s^{\mathcal{R}}, \mathcal{R} > \xrightarrow{a\{x_i\}}$

La obtención de los predicados de tiempo a partir de las regiones, aunque directa, es costosa si queremos ofrecer al usuario predicados de tiempo legibles. La obtención se basa en un cálculo simbólico mediante DBMs (sección 9.A.1). Una vez sintetizado el nuevo requisito sobre RMUS-T:

- Se computa el nuevo grafo MUS-T según la traducción arriba descrita.
- Se aplica el algoritmo de reducción del número de relojes (apéndice de capítulo 9.A).
- Se aplica el algoritmo de detección de *deadlocks* y *timelocks* (capítulo 11), informando al usuario de los puntos conflictivos. Usualmente, la presencia de un *deadlock* revela que una invariante de avance es demasiado permisiva y la presencia de un *timelock* la existencia de un ciclo en el modelo con duración temporal cero.

9.14 Conclusiones

En la metodología SCTL/MUS-T se proponen dos soluciones complementarias al problema de síntesis de modelos MUS-T a partir de especificaciones en lógica: una técnica de síntesis "restringida" sin decisión (la que se ha presentado en este capítulo), y otra de síntesis "genérica", que mediante una técnica basada en *model checking* permita el cómputo de sugerencias para la síntesis del conjunto completo SCTL.

Como hemos mencionado en este capítulo, el algoritmo de síntesis garantiza que para cualquier modificación en el modelo, sólo es necesario reevaluar la parte declarativa de reglas de síntesis previamente aplicadas. En la exposición no se ha considerado el caso en el que en una fase de diseño se añade un nuevo evento al conjunto de acciones A. Para incluir dicho cambio, es suficiente con considerar el modelo $\mathcal{M}$ sobre el que se realiza la síntesis con un conjunto de acciones $A \cup \{a_s\}$, donde a_s es una acción subespecificada. La incorporación de una nueva acción a en el modelo sólo implicará una copia de las transiciones etiquetadas con a_s a transiciones etiquetadas con a.

Apéndices del capítulo

9.A Implementación

Existen tres puntos importantes referentes a la implementación del método de síntesis propuesto:

- La implementación de una librería de DBMs de dimensión variable que permita la incorporación sencilla de nuevos relojes en el modelo.
- La obtención de un predicado de tiempo a partir de un conjunto de regiones (traducción RMUS-T a MUS-T), que se detalla en la sección 9.A.1.
- La reducción del número de relojes en el modelo, que se ha realizado utilizando el algoritmo propuesto en (Daws y Yovine, 1996). En la sección 3.3.1 se puede encontrar una breve exposición de dicha técnica.

9.A.1 Obtención de los predicados MUS-T

Dado que toda región de tiempo es un poliedro convexo, consideramos las zonas de tiempo en la unión $\bigcup_i \mathcal{R}_i$ bajo la siguiente expresión (deducible de DBM).

$$\bigwedge_{0\leq i\leq n, 0\leq j\leq i-1} \left(x_i - x_j \prec_{snd([D]_{i,j})} fst([D]_{i,j}) \right) \wedge \left(x_j - x_i \prec_{snd([D]_{j,i})} fst([D]_{j,i}) \right)$$

Denominamos restricción i, j (r_{ij}) al doble par (i, j) y (j, i) definiendo el límite inferior y superior de $x_i - x_j$, siendo una variedad lineal bidimensional del espacio $\Re_+^{\#\mathcal{C}}$ de dimensión $\#\mathcal{C}$ (tenemos $(n + 1 * n)/2$ restricciones). De esta forma, reescribimos la expresión de poliedro convexo (y como tal de toda región de tiempo) como:

$$\bigwedge_{0\leq i\leq n, 0\leq j\leq i-1} r_{ij} \tag{9.1}$$

Una condición suficiente para que la disyunción de dos poliedros convexos $\mathcal{P}$ y $\mathcal{P}'$, según la expresión 9.1, sea un poliedro convexo es la siguiente:

i) $\exists\ i^d, j^d$ tal que $r_{i^d j^d} \neq r'_{i^d j^d}$ y $\forall\ i \neq i^d, j \neq j^d$ se cumple $r_{ij} = r'_{ij}$.

ii) $r_{i^d j^d}$ y $r'_{i^d j^d}$ se puedan unir, es decir, son expresables por una nueva restricción $r^{\vee}_{i^d j^d}$.

Siendo:

$$\begin{aligned} r_{i^d j^d} &: \quad inf \sim_{inf} x_{i^d} - x_{j^d} \sim_{sup} sup \\ r'_{i^d j^d} &: \quad inf' \sim_{inf'} x_{i^d} - x_{j^d} \sim_{sup'} sup' \end{aligned}$$

con $\sim \in \{<, \leq\}$, entonces:

i) Si $sup' < inf$ o $inf' > sup \rightarrow$ no se pueden unir.

ii) Si $sup' = inf, \rightarrow$ se pueden unir si $\sim_{sup'} = \leq$ o $\sim_{inf} = \leq$:

$$r^{\vee}_{i^d j^d} = inf' \sim_{inf'} x_{i^d} - x_{j^d} \sim_{sup} sup$$

iii) Si $inf' = sup, \rightarrow$ se pueden unir si $\sim_{inf'} = \leq$ o $\sim_{sup} = \leq$:

$$r^{\vee}_{i^d j^d} = inf \sim_{inf} x_{i^d} - x_{j^d} \sim_{sup'} sup'$$

iv) Si $sup' > inf, \rightarrow$ se pueden unir , y $r^{\vee}_{i^d j^d} =$

$$min(inf, inf') \sim_{min(inf,inf')} x_{i^d} - x_{j^d} \sim_{max(sup,sup')} max(sup, sup')$$

v) Si $inf' > sup, \rightarrow$ se pueden unir , y $r^{\vee}_{i^d j^d} =$

$$min(inf, inf') \sim_{min(inf,inf')} x_{i^d} - x_{j^d} \sim_{max(sup,sup')} max(sup, sup')$$

La obtención de un predicado de tiempo reducido, a partir del conjunto $\bigcup_i \mathcal{R}_i$ obtenido en la traducción (sección 9.13), parte del conjunto de regiones $\mathcal{R}_i$ almacenados en una lista de DBMs. El algoritmo hace una exploración exhaustiva de la lista en busca de DBMs que se puedan unir, actualizando la nueva unión en la lista. Cuando no existan en la lista DBMs que se puedan unir, la reducción se acaba.

La obtención del predicado de tiempo reducido es especialmente costosa. Para aliviar en parte dicha complejidad, se realiza el cálculo aplicando el algoritmo de reducción de redundancia en (Larsen et al., 1997a) a los DBMS representando las regiones de tiempo.

CAPÍTULO 10

Minimización y corrección del algoritmo de síntesis

Por ser los métodos de decisión basados en el grafo de regiones RMUS-T muy poco eficientes, en este capítulo se proporciona una algoritmo de minimización que, dada una fórmula o una regla SCTL-T, calcula una abstracción mínima en la que poder llevar a cabo la verificación o la síntesis. Además, se define la relación de "evolución" en modelos MUS-T obtenidos por síntesis incremental, y se demuestra que el proceso de síntesis incremental respeta el nivel de conocimiento de fórmulas SCTL-T genéricas.

10.1 Introducción

Diferentes equivalencias y preórdenes se utilizan comúnmente en el ámbito del tratamiento formal de sistemas concurrentes con dos objetivos principales: la obtención de abstracciones que permitan procedimientos de decisión más eficientes, y la búsqueda de una relación de implementación entre modelos.

En lo que se refiere al objetivo de esta tesis, y dado que la manipulación del modelo MUS-T es la base de la metodología, es necesario la búsqueda de una abstracción que permita la obtención de algoritmos de *model checking* y de síntesis más eficientes. Con tal objetivo, en este capítulo se define una bisimulación de abstracción de tiempo fuerte para MUS-T ($\mathrm{STaB}_{\text{MUS-T}}$), que preserva las propiedades SCTL-T (sección 10.3). Tal bisimulación permite obtener abstracciones mínimas en las que llevar a cabo la verificación y la síntesis.

Tanto las relaciones de equivalencia como las relaciones de preorden permiten comparar distintas descripciones de un sistema dado. Típicamente, siendo $\mathcal{M}_s$ el modelo más abstracto de especificación de un sistema y $\mathcal{M}_i$ el modelo detallado de su implementación, es posible comprobar si $\mathcal{M}_i$ es en efecto una implementación de $\mathcal{M}_s$, mediante una relación de equivalencia o preorden apropiado. De especial

interés para el proceso *software* propuesto y, en concreto, para la corrección del algoritmo de síntesis, será la búsqueda de una relación de orden entre modelos MUS-T generados en las distintas fases del proceso *software*. En la sección 10.4 se define la simulación de abstracción de tiempo especificada (ETaS) que relaciona modelos MUS-T obtenidos por síntesis incremental en la metodología propuesta.

Por otro lado, es necesario aportar una garantía de corrección sobre el procedimiento de síntesis definido, es decir, garantizar que el modelo sintetizado es realmente un refinamiento o una versión con menor nivel de subespecificación que el modelo original (sección 10.4.2). Dado el orden parcial en nivel de conocimiento definido en la sección 7.4.4, se prueba que el algoritmo de síntesis preserva dicho orden parcial para cualquier fórmula SCTL-T con criterio RCT (sección 10.4.3).

10.2 Definiciones preliminares

10.2.1 Simulaciones y Bisimulaciones

En esta sección, recapitulamos las definiciones de simulación y bisimulación utilizadas habitualmente para establecer relaciones de equivalencia y preórdenes entre estados de una estructura o entre estructuras.

Definimos una estructura como una tripla $E =< S_0, S, T >$, siendo S el conjunto de vértices de la estructura, S_0 el conjunto de vértices iniciales y T el conjunto de sus arcos etiquetados por un elemento en un alfabeto A.

Sea A^* el conjunto de secuencias finitas del alfabeto A. Dado $\omega = \omega_1 \cdots \omega_n \in A^*$ y $s, s' \in S$, diremos que $s \overset{\omega}{\rightarrow} s'$ es una transición generalizada sii:

$$\exists\ s_1, \cdots, s_{n-1} \in S \text{ tal que } s \overset{\omega_1}{\rightarrow} s_1 \cdots \overset{\omega_{n-1}}{\rightarrow} s_{n-1} \overset{\omega_n}{\rightarrow} s'$$

Sea Ω un conjunto de transiciones generalizadas y E una estructura. Una relación binaria $\sqsubseteq$ en $S \times S$ es una simulación en E bajo Ω sii, para cualquier $s_1, s_2 \in S$, y para cualquier $\omega \in \Omega$:

$$\text{si } s_1 \sqsubseteq s_2, \forall\ s_1{}' \mid s_1 \overset{\omega}{\rightarrow} s_1{}' \in T, \exists\ s_2{}' | s_2 \overset{\omega}{\rightarrow} s_2{}' \text{ y } s_1{}' \sqsubseteq s_2{}'$$

Una relación binaria $\cong$ en $S \times S$ es una bisimulación en E bajo Ω sii, para cualquier $s_1, s_2 \in S$, y para cualquier $\omega \in \Omega$:

$$\begin{aligned} &\text{si } s_1 \cong s_2 \\ &\qquad \forall\ s_1{}' \mid s_1 \overset{\omega}{\rightarrow} s_1{}' \in T, \exists\ s_2{}' | s_2 \overset{\omega}{\rightarrow} s_2{}' \text{ y } s_1{}' \cong s_2{}' \\ &\qquad \forall\ s_2{}' \mid s_2 \overset{\omega}{\rightarrow} s_2{}' \in T, \exists\ s_1{}' | s_1 \overset{\omega}{\rightarrow} s_1{}' \text{ y } s_1{}' \cong s_2{}' \end{aligned} \tag{10.1}$$

Las definiciones de simulación y bisimulación, dadas sobre un dominio de estados S, se extienden a relaciones en el dominio de estructuras $\mathcal{E}$, entre estructuras E y E', considerando relaciones binarias en $S \times S'$, y definiendo:

- Simulación: $E \sqsubseteq E'$, sii $\forall\ s_0 \in S_0, \exists\ s_0{}' \in S_0{}'$, tal que $s_0 \sqsubseteq s_0{}'$. Diremos que E' simula E.

- Bisimulación: $E \cong E'$, sii $\forall\ s_0 \in S_0, \exists\ s_0{}' \in S_0{}'$, tal que $s_0 \cong s_0{}'$, y $\forall\ s_0{}' \in S_0{}', \exists\ s_0 \in S_0$, tal que $s_0{}' \cong s_0$. Diremos que E y E' son bisimilares o equivalentes bajo bisimulación.

Una bisimulación en un dominio $\mathcal{D}$ es una relación de equivalencia (reflexiva, simétrica y transitiva), y, por tanto, subordina una partición de $\mathcal{D}$ en clases de equivalencia bajo $\cong$. Siendo $\cong$ una bisimulación en S, $\cong$ subordina una partición de S en clases de equivalencia $[s]$. Siendo $\cong$ una bisimulación en el espacio de estructuras $\mathcal{E}$, $\cong$ subordina una partición del espacio de estructuras en clases de equivalencia $[E]$, donde estructuras en la misma clase de equivalencia son bisimilares.

Una simulación es un preorden (reflexiva y transitiva). Considerando $\sqsubseteq$ una simulación en el dominio de las estructuras $\mathcal{E}$, diremos que dos estructuras $E, E' \in \mathcal{E}$ son similares ($\sim_{\sqsubseteq}$) si $E \sqsubseteq E'$ y $E' \sqsubseteq E$. La relación $\sim_{\sqsubseteq}$ entre estructuras es una relación de equivalencia en el dominio de las estructuras $\mathcal{E}$ (la equivalencia asociada a $\sqsubseteq$), produciendo una partición en clases de equivalencia $[E]$ de estructuras similares. En el conjunto cociente, $\mathcal{E}/\sim_{\sqsubseteq}$, se define una relación de orden: $[E] \sqsubseteq [E']$ sii $E \sqsubseteq E'$. Resulta, pues, que el preorden en $\mathcal{E}$ subordina una relación de orden en $\mathcal{E}/\sim_{\sqsubseteq}$ bajo la equivalencia de similaridad.

Dada una relación binaria definiendo una bisimulación $\cong$ (simulación $\sqsubseteq$), diremos que la bisimulación (simulación) preserva un conjunto de proposiciones atómicas Φ_A sii $\forall\ s, s'$ tal que $s \cong s'$ ($s \sqsubseteq s'$), y $\forall\ \phi_A \in \Phi_A$, se cumple $\vdash (\phi_A, s) = \vdash (\phi_A, s')$.

10.2.2 Propiedades de las simulaciones y bisimulaciones

Correspondencia de caminos

Dada una estructura E, definimos un camino π como un secuencia de vértices $s_1, s_2 \cdots$, tales que $\forall\ i > 1$, existe un $t \in T$ y un $w_i \in A$, tal que que $s_{i-1} \stackrel{w_i}{\rightarrow} s_i$. Igualmente definimos un camino π bajo Ω como una secuencia de vértices $s_1, s_2 \cdots$, tales que $\forall\ i > 1$, existe un $\omega \in \Omega$, tal que que $s_{i-1} \stackrel{w}{\rightarrow} s_i$

Dada una bisimulación $\cong$ (simulación $\sqsubseteq$), diremos que dos caminos $\pi = s_1, s_2 \cdots$ y $\pi' = s_1{}', s_2{}' \cdots$ se corresponden, sii para todo $i = 1 \cdots$, $s_i \cong s_i{}'$ ($s_i \sqsubseteq s_i{}'$).

Sea $\cong$ una bisimulación en el dominio $S \times S'$ bajo Ω, y sean s, s' tales que $s \cong s'$. Para todo camino π con origen en s ($s_1 = s$) bajo Ω, existe un camino π' con origen en s' bajo Ω, tal que π y π' se corresponden, y para todo camino π' con origen en s', existe un camino π con origen en s tal que π y π' se corresponden.

Sea $\sqsubseteq$ una bisimulación en el dominio $S \times S'$ bajo Ω, y sean s, s' tales que $s \sqsubseteq s'$. Para todo camino π con origen en s bajo Ω, existe un camino π' con origen en s' bajo Ω tal que π y π' se corresponden.

Por tanto, informalmente, mientras la bisimulación garantiza que dos estructuras E y E' tienen los mismos comportamientos, la simulación sólo garantiza que todo comportamiento de E es también un comportamiento de E'.

Estabilidad

Diremos que una partición (recubrimiento) $\mathcal{P}$ en un dominio S es post-estable con respecto al conjunto de transiciones generalizadas Ω, sii, para todo $\omega \in \Omega$, dados dos elementos P_i y P_j de $\mathcal{P}$:

- o bien $P_i \subseteq suc_\omega(P_j)$
- o bien $suc_\omega(P_j) \cap P_i = \{\}$

siendo $suc_\omega(P_j)$ el sucesor abstracto que computa para todo $s \in P_j$ el conjunto de estados $\{s'\}$ tales que $s \xrightarrow{\omega} s'$. Es decir, o bien todos los estados en P_i son sucesores por ω de algún estado en P_j, o ninguno lo es.

Diremos que la partición (recubrimiento) $\mathcal{P}$ es pre-estable con respecto a transiciones generalizadas Ω, sii, para todo $\omega \in \Omega$, dados dos elementos P_i y P_j de $\mathcal{P}$:

- o bien $P_i \subseteq pre_\omega(P_j)$
- o bien $P_i \cap pre_\omega(P_j) = \{\}$

siendo $pre_\omega(P_j)$ el predecesor abstracto que computa para todo $s \in P_j$ el conjunto de estados $\{s'\}$ tales que $s' \xrightarrow{\omega} s$. Es decir, o bien todos los estados en P_i son predecesores por ω de algún estado en P_j, o ninguno lo es.

Una partición (recubrimiento) del espacio S es estable respecto al conjunto de transiciones generalizadas Ω, sii es pre-estable y post-estable.

Por definición de bisimulación y de partición pre-estable, se sigue que: una bisimulación en el espacio de estados S bajo transiciones generalizadas en el conjunto Ω define una partición pre-estable con respecto a Ω, y viceversa.

Informalmente, la pre-estabilidad garantiza que la estructura ramificada de las evoluciones en el espacio de estados (en el futuro) se traslada al dominio de los elementos de la partición (recubrimiento). Inversamente, para la post-estabilidad y la estructura ramificada en el pasado. En consecuencia, si además los estados en un elemento preservan un conjunto de proposiciones atómicas de interés, se posibilita el cómputo del *model checking* o la síntesis sobre los elementos de la partición (recubrimiento), con la consiguiente reducción del espacio a explorar.

Grafo cociente (de recubrimiento) para una relación entre estados

Sea $\cong$ una bisimulación en la estructura E y $\mathcal{P}_\cong$ la partición del espacio de estados bajo $\cong$. El grafo cociente de E con respecto a la partición $\mathcal{P}_\cong$ es el grafo $E_\cong$, cuyos nodos son las clases de equivalencia $P_i \in \mathcal{P}_\cong$, y cuyas transiciones son:

$$P_i \xrightarrow{w} P_j \text{ si } \exists\, s \in P_i, s' \in P_j \text{ tal que } s \xrightarrow{w} s'$$

Una estructura E y el grafo cociente $E_\cong$ bajo una bisimulación $\cong$ son bisimilares.

Sea $\sqsubseteq$ una simulación en la estructura E y $\mathcal{P}_{\sqsubseteq}$ el recubrimiento del espacio de estados bajo $\sqsubseteq$. El grafo de recubrimiento de E con respecto al recubrimiento $\mathcal{P}_{\sqsubseteq}$ es el grafo $E_{\sqsubseteq}$, cuyos nodos son los subconjuntos $P_i \in \mathcal{P}$, y cuyas transiciones son:

$$P_i \xrightarrow{w} P_j \text{ si } \exists\ s \in P_i, s' \in P_j \text{ tal que } s \xrightarrow{w} s'$$

El grafo de recubrimiento $E_{\sqsubseteq}$ bajo una simulación $\sqsubseteq$ simula la estructura E.

Informalmente, la relación de bisimulación (simulación) entre una estructura E y su grafo cociente (de recubrimiento) obtenido por bisimulación (simulación) permitirá trasladar el cómputo del *model checking* o la síntesis al grafo cociente (de recubrimiento).

10.2.3 Particularización para MUS-T

Las distintas equivalencias y preórdenes definidos a continuación aplican los conceptos de simulación y bisimulación para distintas elecciones del conjunto de transiciones generalizadas Ω. Sea un modelo MUS-T $\mathcal{M}$ con un conjunto de acciones A. Para su grafo semántico $\mathcal{S}_{\mathcal{M}}$, sea el conjunto de transiciones $\mathcal{T} \subseteq (S \times \Re_+^{\#\mathcal{C}}) \times (A \cup \Re_+) \times \mathcal{L}_3 \times (S \times \Re_+^{\#\mathcal{C}})$ (sección 6.6.3).

Para una secuencia de acciones $\in ((A \cup \Re_+) \times \mathcal{L}_3)^*$, definimos la transición generalizada $(\omega, l) = (\omega_1, l_1) \cdots (\omega_n, l_n)$, para $s, s' \in S$, denotada por $s \overset{(\omega,l)}{\multimap} s'$ sii:

$$\exists\ s_1, \cdots, s_{n-1} \in S \text{ tal que } s \overset{(\omega_1,l_1)}{\multimap} s_1 \cdots \overset{(\omega_{n-1},l_{n-1})}{\multimap} s_{n-1} \overset{(\omega_n,l_n)}{\multimap} s'$$
$$\text{y } l = \bigwedge_{i=1\cdots n} l_i$$

Definimos la etiqueta especial t, y las transiciones generalizadas temporales (t, l) con $l \in \mathcal{L}_3$, que denotaremos $s \overset{(t,l)}{\multimap} s'$, como aquellas transiciones generalizadas $\in (\Re_+ \times \mathcal{L}_3)^*$ y, por tanto, $(t, l) = (\tau_1, l_1) \cdots (\tau_n, l_n)$, con $\tau_i \in \Re_+$ y $l_i \in \mathcal{L}_3$, tales que:

$$\exists\ s_1, \cdots, s_{n-1} \in S \text{ tal que } s \overset{(\tau_1,l_1)}{\multimap} s_1 \cdots \overset{(\tau_{n-1},l_{n-1})}{\multimap} s_{n-1} \overset{(\tau_n,l_n)}{\multimap} s'$$
$$\text{y } l = \bigwedge_{i=1\cdots n} l_i$$

10.3 Bisimulación de abstracción de tiempo fuerte (STaB$_{\text{MUS-T}}$)

Las simulaciones y bisimulaciones definidas en el espacio de estados de un modelo MUS-T tienen como objetivo la consecución de una abstracción finita del grafo semántico con fines de análisis o síntesis incremental del modelo MUS-T subyacente. Definiremos una bisimulación de abstracción de tiempo fuerte que ignora el valor exacto de los pasos temporales introduciendo una transición especial

(t, l), y sólo "distingue" las transiciones discretas en un conjunto A^S. La bisimulación de abstracción de tiempo fuerte para MUS-T particulariza la bisimulación de abstracción de tiempo comúnmente utilizada en los sistemas con tiempo para las condiciones de especificación en $\mathcal{L}_3$.

Dado un modelo MUS-T $\mathcal{M}$, se denomina bisimulación de abstracción de tiempo fuerte $\mathbf{STaB_{MUS\text{-}T}}$ (*Strong Time Abstracting Bisimulation*) a la bisimulación $\cong_S \subseteq \mathcal{ST} \times \mathcal{ST}$ en los estados s (por (s, γ)) de $\mathcal{M}$, obtenida bajo el conjunto de transiciones generalizadas $\Omega^S = \{(A^S \times \mathcal{L}_3) \cup (d \times \mathcal{L}_3) \cup (t \times \mathcal{L}_3)\}$. Siendo $A^S \subseteq A$ el conjunto de etiquetas de acción que determinan las acciones atómicas a considerar en la bisimulación y d la etiqueta de transición discreta para aquellas acciones $a \notin A^S$.

Por tanto, y particularizando la definición de bisimulación (relación 10.1) para transiciones $\omega \in \Omega^S$:

$$
\begin{aligned}
& s_1 \cong_S s_2 \text{ sii} \\
& \qquad \forall\, s_1{}' \mid s_1 \overset{\omega}{\rightarrow} s_1{}' \in \mathcal{T}, \exists\, s_2{}' \mid s_2 \overset{\omega}{\rightarrow} s_2{}' \text{ y } s_1{}' \cong_S s_2{}' \\
& \qquad \forall\, s_1{}' \mid s_1 \overset{\omega}{\rightsquigarrow} s_1{}' \in \mathcal{T}, \exists\, s_2{}' \mid s_2 \overset{\omega}{\rightsquigarrow} s_2{}' \text{ y } s_1{}' \cong_S s_2{}' \\
& \qquad \text{si } s_1 \overset{\omega}{\nrightarrow}, \text{ entonces } s_2 \overset{\omega}{\nrightarrow} \\
& \qquad \text{las propiedades anteriores se cumplen si intercambiamos } s_1 \text{ y } s_2
\end{aligned}
$$

donde $\overset{w}{\rightarrow}$ denota $\overset{(w,1)}{\multimap}$; $\overset{w}{\nrightarrow}$ denota $\overset{(w,0)}{\multimap}$; y $\overset{w}{\rightsquigarrow}$ es equivalente $\overset{(w,\frac{1}{2})}{\multimap}$.

Por tanto, dos estados son bisimilares si pueden realizar, bajo idéntica condición de especificación, las mismas transiciones temporales y discretas (con la misma etiqueta en el caso $a \in A^S$), llevando a estados que también son bisimilares. La relación de equivalencia sobre la que se define la bisimulación depende del propósito de su aplicación. Se trata de una bisimulación de abstracción de tiempo porque las transiciones temporales se abstraen de la cantidad de tiempo que transcurre entre un estado y otro, sólo quedándose con un paso temporal arbitrario.

Dado un modelo MUS-T $\mathcal{M}$ y una bisimulación $\mathrm{STaB}_{\text{MUS-T}} \cong_S$ en $\mathcal{M}$, denotaremos $\mathcal{M}_{\cong_S}$ al grafo cociente bajo $\cong_S$, y $\mathcal{P}_{\cong_S}$ a la partición del espacio de estados que induce la relación de equivalencia $\cong_S$.

Bisimilaridad de lo estados sumideros

Dada la definición de estados sumideros de la sección 6.5.2, podemos afirmar:

$$
\begin{aligned}
& (s_{np}, \gamma) \cong_S (s_{np}, \gamma') \\
& (s_s, \gamma) \cong_S (s_s, \gamma')
\end{aligned}
$$

La bisimilaridad de los estados sumideros no posibles justifica la tercera condición de la definición.

El grafo RMUS-T es fruto de una bisimulación $\mathsf{STaB}_{\mathsf{MUS\text{-}T}}$

El grafo de regiones RMUS-T definido en el capítulo 8 es un grafo cociente obtenido por una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$, la obtenida mediante la equivalencia de regiones. La bisimulación obtenida mediante la equivalencia de regiones preserva todas las proposiciones atómicas $\theta \in \Theta$, $a \in A$ y Ψ_m, siendo Ψ_m el conjunto de predicados de tiempo con las constantes máximas m_x y m_{xy} utilizadas en la definición de la equivalencia de regiones (sección 8.2).

10.3.1 Propiedades de STaB

Como hemos mencionado, la bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ es una particularización para modelos MUS-T de la bisimulación de abstracción de tiempo fuerte (STaB) típicamente utilizada en modelos con tiempo denso. A continuación presentamos dos lemas para una bisimulación STaB, que son trasladables a las bisimulaciones $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$.

LEMA 10.1. *(Lema 3.35 de (Alur, 1991)) Toda ejecución en $\mathcal{M}$ se corresponde con un único camino en el cociente $\mathcal{M}_{\cong_S}$. De forma inversa, dado un camino en el cociente con origen en la clase de equivalencia $\mathcal{P}_1$, para todo $s \in \mathcal{P}_1$ existe una ejecución con origen en s que se corresponde con dicho camino.*

Este lema demuestra la bisimilaridad entre un modelo $\mathcal{M}$ y un grafo cociente obtenido por una bisimulación $\mathcal{M}_{\cong_S}$, siendo, por tanto, el equivalente a su correspondiente en el caso no temporizado.

LEMA 10.2. *(Lema 5.6 de (Tripakis, 1998))*

- *Cualquier transición $\xrightarrow{(t,l)}$ atraviesa un conjunto de clases en la partición $\mathcal{P}_{\cong_S}$ finito y único.*

- *Si $s_1 \cong_S s_2$, para cualquier transición temporal $s_1 \xrightarrow{(\tau,l)} s_1{}'$, existe una transición temporal $s_2 \xrightarrow{(\tau',l)} s_2{}'$, tal que $s_1{}' \cong_S s_2{}'$ y las dos transiciones atraviesan el mismo conjunto de clases en $\mathcal{P}_{\cong_S}$.*

Este lema garantiza que el grafo cociente es determinista con respecto al tiempo, característica que era de esperar dado que el tiempo es determinista en la ejecución del modelo.

Las demostraciones de los lemas anteriores se basan en que, dado que el cociente por una bisimulación STaB es finito (el grafo de regiones es finito), si dos caminos con origen en estados bisimilares atraviesan un conjunto distinto de clases, se podría construir un cociente no finito. En consecuencia, dado que existe una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ para MUS-T con cociente finito (RMUS-T), la demostración para modelos MUS-T y la bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ definida sería análoga.

10.3.2 Preservación de fórmulas SCTL-T

Sea $\cong_S$ una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ preservando un conjunto de proposiciones atómicas $Props \subseteq \{A \cup \Psi\}$, es decir:

$$\forall\ a \in A \mid a \in Props, a \in A^S$$
$$\forall\ \psi \in \Psi \mid \psi \in Props, s \cong_S s' \text{ sii } \vdash (\psi, s) = \vdash (\psi, s')$$

Probaremos a continuación que, dada una fórmula SCTL-T ϕ incluyendo el conjunto de proposiciones atómicas $Props$ y $\cong_S$ una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ preservando $Props$, entonces $\forall\ s \cong_S s'$, se cumple $\vdash (\phi, s) = \vdash (\phi, s')$. En primer lugar, probaremos que $\cong_S$ preserva el conjunto de fórmulas SCTL-T de futuro no estricto. Sin embargo, la preservación de fórmulas de pasado conlleva la definición de una bisimulación "hacia atrás" en el espacio de estados.

DEMOSTRACIÓN. **Preservación de fórmulas SCTL-T de futuro no estricto**. La demostración se realiza por inducción en la estructura de la fórmula:

- $\theta \in \Theta$: Trivial, dado que por la definición de bisimulación STaB, si $s \cong_S s'$ las transiciones con etiqueta w se realizan bajo la misma condición de especificación.
- $a \in A$: Trivial dado que $a \in A^S$.
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg\ \phi_1$: Triviales por la hipótesis de inducción.
- $\phi_1 \Rightarrow \phi_2$: Trivial dado que $s \cong_S s'$ y, por tanto, $\vdash (\phi_1, s) = \vdash (\phi_1, s')$ y $\vdash (\phi_2, s) = \vdash (\phi_2, s')$.
- $\phi_1\ \forall\Rightarrow\bigcirc\ \phi_2$: Por la hipótesis de inducción $\vdash (\phi_1, s) = \vdash (\phi_1, s')$. Dado AA el conjunto de acciones de aplicabilidad en ϕ_1, para todo $a \in AA$, si $s \xrightarrow{(a,l)} s_1$, existe $s_1{}'$ tal que $s' \xrightarrow{(a,l)} s_1{}'$, y $s_1 \cong_S s'_1$, e inversamente para s'. Por tanto, los estados de aplicabilidad son bisimilares bajo (a, l), y $\vdash (\phi_1\ \forall\Rightarrow\bigcirc\ \phi_2, s) = \vdash (\phi_1\ \forall\Rightarrow\bigcirc\ \phi_2, s')$. (Donde si para $a \in AA$, a no pertenece a A^s la etiqueta será d).
- $\phi_1\ \exists\Rightarrow\bigcirc\ \phi_2$: Análoga.
- $\phi_1\ \forall\Rightarrow_+\ \phi_2$: Por la hipótesis de inducción $\vdash (\phi_1, s) = \vdash (\phi_1, s')$. Dado $\{s_i\} = \perp ((\Rightarrow_+, \phi_1), s)$, supongamos $\bigwedge_i (\neg\ ce(s_i)) \vee \vdash (\phi_2, s_i) = l$; entonces existe, una transición $s \xrightarrow{(t,ce(s_k))} s_k$, tal que $\neg\ ce(s_k) \vee \vdash (\phi_2, s_k) = l$, y para toda transición $s \xrightarrow{(t,ce(s_j))} s_j$, $\neg\ ce(s_j) \vee \vdash (\phi_2, s_k) \geq l$. Dado que, $s \cong_S s'$, existe una transición generalizada $s' \xrightarrow{(t,ce(s_k{}'))} s_k{}'$, tal que $s_k \cong_S s_k{}'$ y, por tanto, $\neg\ ce(s_k{}') \vee \vdash (\phi_2, s_k{}') = l$. Además, si existiese un transición $s' \xrightarrow{(t,ce(s_j{}'))} s_j{}'$, tal que $\neg\ ce(s_j{}') \vee \vdash (\phi_2, s_j{}') = l' < l$, tendría que existir una transición $s \xrightarrow{(t,ce(s_j))} s_j$ tal que $\neg\ ce(s_j) \vee \vdash (\phi_2, s_j) = l' < l$ (contradicción). Por tanto, $\vdash (\phi_1\ \forall\Rightarrow_+\ \phi_2, s) = \vdash (\phi_1\ \forall\Rightarrow_+\ \phi_2, s')$.

- $\phi_1 \ \exists\Rightarrow_+ \ \phi_2$: Análoga.

□

Demostración. **Preservación de fórmulas SCTL-T con operadores de pasado**. La preservación de las fórmulas que incluyen operadores temporales de pasado impone que, dados dos estados bisimilares, sus estados predecesores sean también bisimilares (bisimulación *backward-forward*). Así, imponemos para la fórmulas SCTL-T de pasado las siguientes propiedades adicionales en $s_1 \cong_S s_2$:

- $\forall\ s_1{}' \mid s_1{}' \overset{\omega}{\rightarrow} s_1 \in \mathcal{T}, \exists\ s_2{}' \mid s_2{}' \overset{\omega}{\rightarrow} s_2$ y $s_1{}' \cong_S s_2{}'$
- $\forall\ s_1{}' \mid s_1{}' \overset{\omega}{\rightsquigarrow} s_1 \in \mathcal{T}, \exists\ s_2{}' \mid s_2{}' \overset{\omega}{\rightsquigarrow} s_2$ y $s_1{}' \cong_S s_2{}'$

Las propiedades anteriores se cumplen si intercambiamos s_1 y s_2.

Dado que la relación de equivalencia $\cong_S$, definiendo una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$, no induce necesariamente una partición post-estable del espacio de estados, una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ no siempre preserva el grado de satisfacción de fórmulas SCTL-T con operadores de pasado. Dado que el grafo RMUS-T respeta tal característica, dada una partición $\mathcal{P}_{\cong_S}$, es posible encontrar un refinamiento de tal partición de naturaleza post-estable, imponiendo las propiedades anteriores. Con tal propiedad la demostración de la preservación de propiedades SCTL-T de pasado es análoga a las de futuro no estricto.

□

Demostración. **Preservación de fórmulas SCTL-T**. Lo que resta para la demostración de la preservación del conjunto completo SCTL-T son las fórmulas que incluyen operadores de ligado.

Sea una fórmula SCTL-T incluyendo un conjunto de relojes $\mathcal{E} = \{x\}$ ligados, y un conjunto de predicados de tiempo $\Psi^{\mathcal{E}}_{\mathcal{M}} = \{\psi\}$ sobre dichos relojes y sobre el conjunto de relojes del modelo $\mathcal{M}$. Basta considerar un modelo $\mathcal{M}^{\mathcal{E}}$ con el conjunto de relojes original más el conjunto de relojes $\mathcal{E}$, y una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ sobre el modelo $\mathcal{M}^{\mathcal{E}}$ tal que la partición $\mathcal{P}_{\cong_S}$ preserve además el conjunto de proposiciones:

$$\begin{aligned} &\forall\ x \in \mathcal{E}, x = 0 \\ &\forall\ \psi \in \Psi^{\mathcal{E}}_{\mathcal{M}}, \psi \text{ y } \neg\psi \end{aligned} \tag{10.2}$$

Tal bisimulación existe, dado que es posible conseguir un grafo RMUS-T con dichas características. □

10.3.3 *Model Checking* y síntesis SCTL-T

La bisimulación definida es la comúnmente utilizada en el análisis de autómatas temporizados, simplemente particularizada para las tres condiciones de especificación en el modelo MUS-T. Sin embargo, dada una fórmula $\phi \in \Phi_{SCTL-T}$, sería deseable obtener una bisimulación de abstracción de tiempo fuerte del grafo $\mathcal{M}$

preservando sólo las proposiciones atómicas que determina el valor de verdad de ϕ. Para ello, en la sección 10.6, se detalla un algoritmo de minimización que, a partir de una partición del espacio de estados que preserva todas las proposiciones atómicas en una fórmula ϕ, calcula un refinamiento pre-estable en el que realizar el *model checking* y la síntesis. Los algoritmos de *model checking* y de síntesis sólo necesitarán post-estabilizar la partición para la verificación de operadores temporales de pasado.

10.4 Simulación de abstracción de tiempo especificada ($\text{ETaS}_{\text{MUS-T}}$)

En la metodología propuesta, sería deseable poder comprobar si, dados dos modelos MUS-T $\mathcal{M}_1$ y $\mathcal{M}_2$ generados en fases consecutivas del proceso de diseño, existe una relación incremental entre ellos, que notaremos por $\sqsubseteq_E$. Informalmente, podemos decir que un modelo $\mathcal{M}_2$ es un modelo MUS-T más detallado o con menor cantidad de subespecificación que $\mathcal{M}_1$ ($\mathcal{M}_1 \sqsubseteq_E \mathcal{M}_2$), si contiene, al menos, todos los comportamientos temporizados posibles y no posibles del modelo $\mathcal{M}_1$, pudiendo contener otros comportamientos temporizados adicionales. Asimismo, la relación de orden definida permite comprobar si se trata de dos modelos equivalentes bajo subespecificación, ya que bastaría con comprobar si $\mathcal{M}_1 \sqsubseteq_E \mathcal{M}_2$ y $\mathcal{M}_2 \sqsubseteq_E \mathcal{M}_1$.

El preorden $\sqsubseteq_E$ define, bajo similaridad, un orden parcial en el dominio de modelos MUS-T, relacionando modelos MUS-T obtenidos por síntesis incremental en el proceso *software*.

Dado un modelo MUS-T $\mathcal{M}$, se denomina simulación de abstracción de tiempo especificada $\text{ETaS}_{\text{MUS-T}}$ (*spEcified Time Abstracting Simulation*) a la relación binaria $\sqsubseteq_E \subseteq \mathcal{ST} \times \mathcal{ST}$ en los estados s (por (s, γ)) de $\mathcal{M}$, bajo el conjunto de acciones generalizadas $\Omega^E = \{(A \times \{0,1\}) \cup (t \times \{0,1\})\}$ tal que:

$$
\begin{aligned}
& s_1 \sqsubseteq_E s_2 \text{ sii} \\
& \qquad \forall\ s_1{}' \mid s_1 \xrightarrow{\omega} s_1{}' \in \mathcal{T}, \exists\ s_2{}' \mid s_2 \xrightarrow{\omega} s_2{}' \text{ y } s_1{}' \sqsubseteq_E s_2{}' \\
& \qquad \text{si } s_1 \overset{\omega}{\nrightarrow}, \text{ entonces } s_2 \overset{\omega}{\nrightarrow}
\end{aligned}
$$

La simulación de abstracción de tiempo especificada permite establecer un orden parcial en los modelos generados mediante la síntesis incremental en la metodología. Formalmente, dados dos modelos MUS-T $\mathcal{M}_1$ y $\mathcal{M}_2$, se dice que $\mathcal{M}_2$ simula bajo especificación $\mathcal{M}_1$ sii existe una simulación de abstracción de tiempo especificada $\sqsubseteq_E \subseteq \mathcal{ST}_1 \times \mathcal{ST}_2$ en los estados de $\mathcal{M}_1$ y $\mathcal{M}_2$ tal que:

- $\forall\ (s_{0_1}, \gamma_1^0) \in \mathcal{ST}_1, \exists\ (s_{0_2}, \gamma_2^0) \in \mathcal{ST}_2$ tal que $(s_{0_1}, \gamma_1^0) \sqsubseteq_E (s_{0_2}, \gamma_2^0)$
- $\sqsubseteq_E$ preserva el conjunto de predicados $\Psi_{\mathcal{M}}$ con constantes máximas m_x, m_{xy} definidas por el modelo $\mathcal{M}_1$

10.4.1 ETaS$_{\text{MUS-T}}$ es más débil que STaB$_{\text{MUS-T}}$

De la definición de bisimulación STaB$_{\text{MUS-T}}$, se sigue que dados dos modelos $\mathcal{M}_1$ y $\mathcal{M}_2$, si $\mathcal{M}_1 \cong_S \mathcal{M}_2$ entonces $\mathcal{M}_1 \sqsubseteq_E \mathcal{M}_2$, y, por tanto, $\sqsubseteq_E \subseteq \cong_S$. Probaremos que $\cong_S \not\subseteq \sqsubseteq_E$ con el ejemplo de la figura 10.1. Con 10.1(a) $\mathcal{M}_1$ y 10.1(b) $\mathcal{M}_2$, vemos que $\mathcal{M}_1 \sqsubseteq_E \mathcal{M}_2$ y, sin embargo, $\mathcal{M}_1 \not\cong_S \mathcal{M}_2$. Por tanto, el cómputo de la relación $\sqsubseteq_E$ se puede realizar a partir de $\cong_S$.

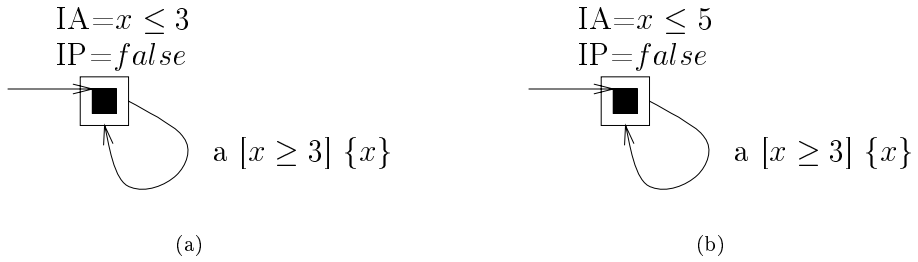


Figura 10.1. STaB es más fuerte que ETaS. Para $\mathcal{M}_1$ en la figura 10.1(a) y $\mathcal{M}_2$ en la figura 10.1(b): $\mathcal{M}_1 \sqsubseteq_E \mathcal{M}_2$ y, sin embargo, $\mathcal{M}_1 \not\cong_S \mathcal{M}_2$

10.4.2 Corrección del algoritmo de síntesis

Demostraremos en esta sección que, dado un modelo $\mathcal{M}$ y una regla de síntesis $\mathcal{RS}$, el modelo $\mathcal{M}^s$, sintetizado mediante la aplicación del algoritmo de síntesis, es tal que $\mathcal{M} \sqsubseteq_E \mathcal{M}^s$. Para ello basta con considerar que, dado $\mathcal{M}^s$ idéntico a $\mathcal{M}$, y, por tanto, $\mathcal{M} \sqsubseteq_E \mathcal{M}^s$, los distintos pasos del algoritmo de síntesis preservan dicha relación.

Sean $\mathcal{S}$ el conjunto de estados del grafo semántico de $\mathcal{S}_{\mathcal{M}}$ y $\mathcal{S}^s$ sus idénticos en $\mathcal{S}_{\mathcal{M}^s}$. Sea s un estado tal que $\vdash (\mathcal{RS}, s) = 1$ y, por tanto, $\vdash (\mathcal{RS}, s^s) = 1$. Los cambios realizados en el algoritmo de síntesis serán:

- El cambio de la condición de especificación de una transición discreta o temporal. Es decir, si $s \sqsubseteq_E s^s$ y, dado el conjunto de transiciones salientes $t \in \mathcal{ST}$ con origen en s, el algoritmo de síntesis actualiza transiciones t con $\mathcal{CE}(t) = \frac{1}{2}$ a transiciones t con $\mathcal{CE}(t) = \{0, 1\}$ y, por tanto, respeta la relación.
- El reinicio de un reloj imperativo o de modelo en una transición discreta con destino en s'. Dado que en el algoritmo sólo se permite tal reinicio si el reloj es inactivo, el valor de dicho reloj es tal que su modificación (por reinicio) respeta las condiciones de especificación del subgrafo con origen en s'.

10.4.3 Preservación del nivel de conocimiento

Al igual que en la sección anterior, nos basamos en que los cambios realizados sólo alteran la condición de especificación de las transiciones discretas y temporales. Por tanto, la demostración se basa en considerar dos grafos semánticos con

estructura idéntica hacia delante para el modelo inicial $\mathcal{M}$ y para el modelo sintetizado $\mathcal{M}^s$. Dado que los modelos sintetizados son deterministas, el modelo $\mathcal{M}^s$ tiene estructura idéntica simplemente considerando que, para un estado (s, γ), el cambio de una transición subespecificada con destino (s', γ') a no posible introduce un estado (s_{np}, γ').

Sea el orden en nivel de conocimiento, $\leq_c$, definido en la sección 7.4.4, $\mathcal{M}$ un modelo MUS-T y $\mathcal{M}^s$ el obtenido a partir $\mathcal{M}$ mediante la aplicación de una regla de síntesis $\mathcal{RS}$ con criterio RCT, probaremos que, para cualquier fórmula ϕ y dos estados s y s^s que se corresponden en la estructura, tales que s^s no es un estado s_{np}, se cumple:

$$\vdash (\phi, s) \leq_c \vdash (\phi, s^s)$$

Dado el orden $\leq_c$ en nivel de conocimiento, la demostración se basa en la monotonicidad de los operadores lógicos $\wedge$ $\vee$ y del operador causal $\rightarrow$ respecto al orden $\leq_c$. Probaremos, en primer lugar, la preservación de fórmulas SCTL-T de futuro no estricto. En segundo lugar se esbozará la base de la demostración para fórmulas SCTL-T de pasado.

Demostración. **Preservación del nivel de conocimiento para fórmulas de futuro no estricto** Dada una fórmula ϕ de futuro no estricto, probaremos que, si los estados s y s^s se corresponden en las estructuras idénticas (generadas con criterio RCT o RCTF), $\vdash (\phi, s) \leq_c \vdash (\phi, s^s)$. La demostración se realiza por inducción estructural:

- $a \in A$: Si s y s^s se corresponden en la estructura, los cambios realizados por el algoritmo de síntesis respetan el nivel de conocimiento y, por tanto, $\vdash (a, s) \ \leq_c \vdash (a, s^s)$.
- $\psi \in \Psi$: Dado que los estados se corresponden en la estructura, se cumple $\vdash (\psi, s) = \vdash (\psi, s^s)$.
- $\theta \in \Theta$: Dado que el valor de verdad de las proposiciones constantes sólo depende de la condición de especificación de la transición, y esta no decrece en el algoritmo de síntesis, $\vdash (\theta, s) \leq \vdash (\theta, s^s)$.
- $\phi_1 \Rightarrow \phi_2$: Por la hipótesis de inducción

  $$\vdash (\phi_1, s) \leq_c \vdash (\phi_1, s^s)$$
  $$\vdash (\phi_2, s) \leq_c \vdash (\phi_2, s^s)$$

  y, dado que la operación $\rightarrow$ es monótona, $\vdash (\phi_1 \Rightarrow \phi_2, s) \leq_c \vdash (\phi_1 \Rightarrow \phi_2, s^s)$.
- $\phi_1 \forall \otimes \phi_2$: El grado de satisfacción en s es

  $$\vdash (\phi_1 \forall \otimes \phi_2, s) = \rightarrow (\vdash (\phi_1, s), \bigwedge_{\{s'\}} (\neg\ ce(s') \vee \vdash (\phi_2, s'))$$

  consideremos un estado de aplicabilidad s' cualquiera y su elemento $\neg\ ce(s') \vee \ \vdash (\phi_2, s')$ en la conjunción anterior:

  - Si $ce(s') = \frac{1}{2}$ y, por tanto, $\neg\ ce(s') = \frac{1}{2}$, los cambios realizados pueden ser:

- Si $ce(s\,') \rightarrowtail 1$ entonces $\neg\ ce(s^s\,') = 0$ y, dado que, por la hipótesis de inducción, $\vdash (\phi_2, s) \leq_c \vdash (\phi_2, s^s\,')$, no decrece.
- Si $ce(s\,') \rightarrowtail 0$ entonces $\neg\ ce(s^s\,') = 1$, el resultado de la conjunción es 1 y no decrece.
- Si $ce(s\,')$ no cambia, dado que, por la hipótesis de inducción, el nivel de conocimiento para ϕ_2 cumple $\vdash (\phi_2, s\,') \leq_c \vdash (\phi_2, s^s\,')$, no decrece.

– Si $ce(s\,') = 0$, el resultado será siempre 1 y, por tanto, dado que la condición de especificación $ce(s\,')$ no puede cambiar en el algoritmo de síntesis, el resultado permanece idéntico.

– Si $ce(s\,') = 1$, $\neg\ ce(s\,') = 0$ y, por tanto, el resultado de la conjunción será $\vdash (\phi_2, s\,')$. Dado que, por la hipótesis de inducción, se cumple $\vdash (\phi_2, s\,') \leq_c \vdash (\phi_2, s^s\,')$, el resultado no decrece.

Dado que todos los elementos en la conjunción respetan el orden $\leq_c$, y $\wedge$ es monótono en $\leq_c$, podemos afirmar que $\vdash (\phi_1 \forall \otimes \phi_2, s) \leq_c \vdash (\phi_1 \forall \otimes \phi_2, s^s)$.

- $\phi_1 \exists \otimes \phi_2$: Análoga.

□

Además, los criterios RCTReset y RCTFReset preservarán el nivel de conocimiento de cualquier fórmula de futuro no estricto que no incluya predicados de tiempo sobre relojes de modelo.

Preservación del nivel de conocimiento para fórmulas de pasado

La preservación para el caso de operadores de pasado sólo se cumple si utilizamos un criterio RCT. La demostración se basa en considerar que el criterio RCT impone una bisimulación especificada *backward* y, por tanto, las demostraciones son simétricas. La imposición de una bisimulación especificada hacia atrás arranca del hecho de que el pasado es no determinista. En el caso de criterio RCTReset se mantendrá la preservación salvo para aquellas fórmulas que incluyan predicados de tiempo sobre relojes de modelo.

10.5 Relación de abstracción de tiempo total posible ($TP_{MUS\text{-}T}$)

Como hemos mencionado, una de las características que garantizan una corrección mínima de un modelo MUS-T será la ausencia de bloqueos. La ausencia de bloqueos en tiempo, formalizada en (Henzinger et al., 1992b) bajo el denominación "*nonzeno*", garantiza que el tiempo puede progresar sin una cota superior. Sin embargo, una bisimulación $\mathrm{STaB}_{\mathrm{MUS\text{-}T}}$ no respeta la característica "*nonzeno*" de un sistema. Con tal objetivo, y para permitir la ejecución simbólica de un modelo MUS-T en el capítulo 11 se presenta el grafo de simulación posible.

Definimos a continuación la relación de abstracción de tiempo total posible ($\mathrm{TP}_{\mathrm{MUS\text{-}T}}$), abstracción base para el cómputo del grafo de simulación posible.

Dicha abstracción permite obtener modelos abstractos que sólo tengan en consideración los estados en $\mathcal{ST}$ que son accesibles mediante transiciones posibles, reflejando los comportamientos especificados en un modelo MUS-T. Es decir, la relación sólo considera el conjunto de transiciones generalizadas en:

$$\Omega^{TP} = \{(t,1) \cup (A \times 1)(t,1)\}$$

Dado un modelo MUS-T $\mathcal{M}$, se denomina relación de abstracción de tiempo total posible ($\mathrm{TP}_{\text{MUS-T}}$) a la relación binaria $\sqsubseteq_{TP} \subseteq \mathcal{ST} \times \mathcal{ST}$ en los estados s (por (s,γ)) de $\mathcal{M}$, bajo el conjunto de acciones generalizadas Ω^{TP} tal que:

- $\forall\ s \in \mathcal{ST}$, si $s \xrightarrow{\tau} s'$ entonces $s \sqsubseteq_{TP} s'$.
- $\forall\ s_1, s_2 \in \mathcal{ST} \mid s_1 \sqsubseteq_{TP} s_2$, si $s_1 \xrightarrow{a}\xrightarrow{\tau} s_1{}'$ y $s_2 \xrightarrow{a}\xrightarrow{\tau} s_2{}'$ entonces $s_1{}' \sqsubseteq_{TP} s_2{}'$.

10.6 Análisis basado en minimización

Un problema crucial en la verificación basada en *model checking* es la generación del espacio de estados en el que realizar la exploración exhaustiva en busca de la veracidad de la fórmula. En el caso de la verificación con tiempo, este problema se traduce en la generación del grafo de regiones. El tamaño elevado del espacio de estados del sistema no sólo se origina en la complejidad intrínseca de los modelos, o en el número de relojes definidos, sino también en que el grafo de regiones contiene una gran cantidad de estados que, dependiendo de la fórmula a verificar, mantienen algún tipo de equivalencia. Es decir, una fórmula ϕ, normalmente, no distingue todas las regiones en RMUS-T.

En lo que se refiere a la explosión del espacio de estados originada en las interpretaciones reales de los relojes introducidos en los modelos basados en el paradigma del autómata temporizado, se han aportado fundamentalmente tres categorías de soluciones (sección 3.3.1): la utilización de representantes discretos de las regiones (Bozga et al., 1997); el cómputo simbólico de la verificación (Henzinger et al., 1992b; Bouajjani et al., 1997); y la utilización de particiones menos finas del espacio de estados (Tripakis y Yovine, 1996; Godskesen et al., 1993). Además, existen numerosas técnicas de menor entidad que, en su mayoría, tratan de aplicar conceptos ya utilizados con éxito en el tratamiento de sistemas concurrentes y que suponen variaciones de menor entidad que las anteriores.

Si bien el análisis mediante representantes trata de llevar el problema de un dominio denso a un dominio discreto que se pueda beneficiar de los *model checkers* sin tiempo, se considera probado que su aplicación al tratamiento de sistemas de tiempo real no es adecuada en todos los casos (Asarin et al., 1998a). En cuanto a los algoritmos simbólicos, conviene decir que todos los *model checkers* sobre tiempo denso son, en mayor o menor medida, simbólicos, dado que el conjunto de estados es infinito. Se suele hablar de algoritmos simbólicos para referirse a aquellos algoritmos que no se basan en la exploración de un grafo cociente, sino en puntos fijos sobre predicados lógicos. Por tanto, los algoritmos simbólicos no acaban hasta que converja el punto fijo, produciendo, además, una diagnosis más compleja, dado que los contraejemplos son generados mediante fórmulas características.

Aunque los algoritmos simbólicos evitan la explosión del espacio de estados, no resultan adecuados para nuestros propósitos. La idea de permitir un desarrollo iterativo e incremental, en el que el usuario va variando los requisitos exigidos al sistema, no se basa únicamente en sucesivas fases de especificación y verificación, sino que ha de ir acompañada de una modificación del sistema guiada por la metodología. Para tal propósito, el tipo de diagnosis ofrecida por el *model checking* simbólico, mediante una fórmula característica, no resulta de aplicación directa.

Por tanto, hemos considerado más adecuado la utilización de una abstracción más fuerte que la alcanzada con la utilización de la equivalencia de regiones. Dicha abstracción nos debe permitir construir, a partir de un modelo MUS-T y de un requisito SCTL-T, el grafo cociente mínimo que permita la verificación y síntesis exacta del requisito. Si bien la utilización de este tipo de abstracciones podría llevar, en el peor caso, a un grafo cociente tan grande como el grafo de regiones, en la práctica su tamaño es, normalmente, mucho menor.

10.6.1 Esbozo de los algoritmos de minimización

La búsqueda de una bisimulación para la resolución de un problema concreto se torna compleja si consideramos que, por un lado, el conjunto de estados concretos es infinito; y, por otro, la bisimulación deseada ha de respetar una serie de propiedades para que sea aplicable a un tipo de análisis en concreto. La obtención de una bisimulación que, no siendo la mayor, respete un conjunto de propiedades deseables se ha realizado en los modelos sin tiempo explotando la naturaleza pre-estable de las bisimulaciones (algoritmos de minimización). Una vez demostrada la naturaleza decidible del problema de *model checking* sobre tiempo denso, dichos algoritmos han encontrado sus equivalentes en los formalismos con tiempo.

El problema que abordan los algoritmos de minimización es el cómputo del grafo de estados mínimo para un proceso sin construir a priori el total del espacio de estados. Un grafo de estados "mínimo", lo será, desde el punto de vista de una bisimulación definida en los estados del proceso.

Este problema ha sido resuelto en el caso de los modelos de estados sin tiempo (Bouajjani et al., 1992; Lee y Yannakakis, 1992) con el objetivo de reducir el espacio de estados originado en la composición de sistemas concurrentes, y el análisis de espacios de estados infinitos (sistemas parametrizados y modelos con datos).

Informalmente, el mecanismo de minimización trabaja como sigue. Supongamos una partición del espacio de estados preservando un conjunto de proposiciones atómicas $Props$. Entonces, cualquier refinamiento de dicha partición preservará $Props$. En base a este hecho, la minimización trata de refinar los elementos de la partición hasta conseguir una partición en clases de equivalencia bajo una bisimulación $\cong$. El criterio que determina que una partición del espacio de estados sea una bisimulación se basa en la siguiente proposición:

"Una bisimulación induce una partición pre-estable del espacio de estados y viceversa"

Recordando la definición de pre-estabilidad, una partición $\mathcal{P}$ es pre-estable con

respecto a un conjunto de transiciones generalizadas Ω, sii, para todo $\omega \in \Omega$, dados dos elementos P_i y P_j de $\mathcal{P}$:

- o bien $P_i \subseteq pre_\omega(P_j)$
- o bien $P_i \cap pre_\omega(P_j) = \{\}$

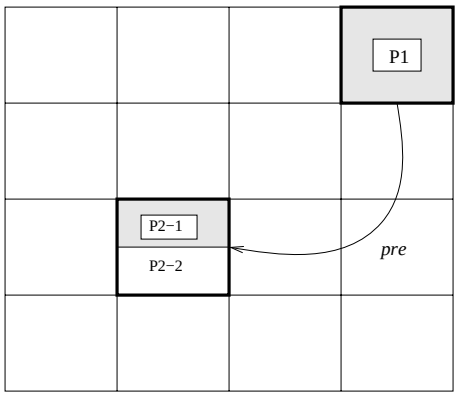


Figura 10.2. Esbozo del mecanismo de minimización

Sea, por ejemplo, la partición en la figura 10.2 representando un espacio de estados ficticio. La comprobación de pre-estabilidad entre los elementos de la partición P_1 y P_2 computa $pre_w(P_1)$ mediante el operador abstracto pre de predecesores por una acción generalizada w. Si $P_2 \cap pre_\omega(P_1) = \{\}$, se garantiza la pre-estabilidad; si $P_2 \cap pre_\omega(P_1) \neq P_2$, podemos concluir que $P_2 \not\subseteq pre_\omega(P_1)$. En ese caso, se refina P_2 en dos zonas $P_{2-1} = P_2 \cap pre_\omega(P_1)$ y $P_{2-2} = P_2 - pre_\omega(P_1)$, garantizando que tanto P_{2-1} como P_{2-2} son pre-estables con respecto a P_1.

Una vez obtenido un refinamiento pre-estable de la partición inicial, garantizamos que dicha partición es un grafo cociente bajo la bisimulación $\cong$ y, además, respeta el conjunto de proposiciones $Props$. Mientras que el conjunto $Props$ determina las proposiciones atómicas preservadas en la abstracción, el tipo de relación computada determina el conjunto de operadores temporales preservados.

En el ámbito de los modelos con tiempo denso, la verificación formal de propiedades por *model checking* basada en el grafo de regiones, tal como fue introducido originalmente, utiliza el grafo cociente obtenido mediante la equivalencia de regiones. Como ya hemos mencionado, dicha equivalencia induce una bisimulación de abstracción de tiempo fuerte. Sin embargo, no es necesariamente la equivalencia (induciendo una bisimulación) más fuerte que preserva la veracidad de un problema concreto. Es decir, no es necesariamente la que produce una partición menos fina (con menor numero de clases de equivalencia).

Una vez demostrada la existencia de un grafo cociente bajo bisimulación de abstracción de tiempo fuerte (el grafo de regiones), la técnica de minimización se adapta (Alur et al., 1992; Tripakis y Yovine, 1996) simplemente tomando en consideración los dos tipos de transiciones en la semántica de dichos modelos: transiciones discretas y transiciones temporales.

10.6.2 Particularización para MUS-T

En el caso de una bisimulación $\mathrm{STaB}_{\text{MUS-T}}$ ($\cong_S$) sobre un modelo MUS-T, el algoritmo de minimización ha de refinar la partición inicial con respecto al conjunto de transiciones generalizadas Ω^S. Se propone un algoritmo que, al igual que el presentado en (Bouajjani et al., 1992), sólo tiene en consideración las clases accesibles de la partición, que son aquellas que contienen, por lo menos, un estado accesible mediante transiciones posibles o subespecificadas. Serán alcanzables todos aquellos (s, γ) tales que:

$$\exists\ \{(w_i, l_i)\}_{0\cdots n} \in \Omega \text{ tal que } l_i \in \{1, \frac{1}{2}\} \text{ y } (s_0, \gamma^0) \overset{w_0, l_0}{\multimap} \cdots \overset{w_n, l_n}{\multimap} (s, \gamma)$$

Por tanto, para una transición generalizada $\omega \in \Omega$, las transiciones a tener en cuenta serán $(w, 1)$ y $(w, \frac{1}{2})$. El grafo cociente obtenido contendrá una partición parcial del espacio de estados $S \times \Re_+^{\#\mathcal{C}}$.

10.6.3 Manipulación de los elementos de la partición

Cada elemento P_i en la partición $\mathcal{P}$ se puede representar de forma genérica como un conjunto de zonas $[s_1, Z_1], \cdots, [s_n, Z_n]$, donde cada $s_i \in S$ es un estado de control en S, y cada $Z_i \in \mathcal{P}^{\mathcal{C}}$ será un poliedro (no necesariamente convexo) representado por una lista de DBMs (apéndice A).

Siendo $P_i = [s_i^1, Z_i^1], \cdots, [s_i^n, Z_i^n]$ y $P_j = [s_j^1, Z_j^1], \cdots, [s_j^m, Z_j^m]$, se definen a continuación las operaciones abstractas sobre zonas:

- $P_i \cap P_j$, se define como:

  $$\forall\ i = 1 \cdots n, \text{ si } \exists\ [s_j, Z_j] \in P_j \mid s_i = s_j \text{ entonces } [s_i, Z_i \cap Z_j] \in P_i \cap P_j$$

- Si $P_i \cap P_j \neq \{\}$, entonces $P_i - P_j$ se define como:

  $$\forall\ i = 1 \cdots n, \text{ si } \exists\ [s_j, Z_j] \in P_j \mid s_i = s_j \text{ entonces } [s_i, Z_i - Z_j] \in P_i - P_j$$

  si $P_i \cap P_j = \{\}$, entonces $P_i - P_j = P_i$

- $suc_w(P_i)$ y $pre_w(P_i)$: los sucesores y predecesores abstractos de un elemento P_i por una acción generalizada $w \in \Omega^S$.

  Estos operadores se pueden reducir a predecesores y sucesores abstractos sobe zonas $[s, Z]$:

  $$suc_w(P_i) = suc_w([s_1, Z_1]), \cdots, suc_w([s_n, Z_n])$$
  $$pre_w(P_i) = pre_w([s_1, Z_1]), \cdots, pre_w([s_n, Z_n])$$

  donde w puede ser una transición discreta $a \in A^S$, una transición discreta genérica d, o un cierre de transiciones temporales τ (o bien todas posibles, o bien todas subespecificadas).

  Si bien el cómputo de una partición inducida por una bisimulación $\mathrm{STaB}_{\text{MUS-T}}$ no impone la definición de los sucesores abstractos, dichos operadores son necesarios para llevar a cabo post-estabilización que garantice la preservación de operadores de pasado.

Sucesores y predecesores temporales

Sea $pre_t([s,Z],1)$ (resp. $pre_t([s,Z],\frac{1}{2})$) el operador que computa los predecesores en tiempo de $[s,Z]$ mediante una transición temporal generalizada con condición de especificación 1 (resp. $\frac{1}{2}$). Se define el operador abstracto de predecesores temporales, $pre_t([s,Z],c_e)$, como:

$$\begin{cases}\{(s,\gamma) \mid \exists\ \tau \in \Re_+, \gamma+\tau \in Z,\ \text{y}\ \forall\ 0 \leq \tau' \leq \tau, I_a(s)(\gamma+\tau') = 1\}, c_e = 1\\ \{(s,\gamma) \mid \exists\ \tau \in \Re_+, \gamma+\tau \in Z,\ \text{y}\ \forall\ 0 \leq \tau' \leq \tau, I_s(s)(\gamma+\tau') = 1\}, c_e = \frac{1}{2}\end{cases}$$

y el operador abstracto de sucesores temporales, $suc_t([s,Z],c_e)$, como:

$$\begin{cases}\{(s,\gamma+\tau) \mid \exists\ \tau \in \Re_+, \gamma \in Z,\ \text{y}\ \forall\ 0 \leq \tau' \leq \tau, I_a(s)(\gamma+\tau') = 1\}, c_e = 1\\ \{(s,\gamma+\tau) \mid \exists\ \tau \in \Re_+, \gamma \in Z,\ \text{y}\ \forall\ 0 \leq \tau' \leq \tau, I_s(s)(\gamma+\tau') = 1\}, c_e = \frac{1}{2}\end{cases}$$

En resumen, estos operadores calculan los sucesores (predecesores) temporales incluyendo sólo pasos temporales posibles o subespecificados. Para el cálculo de dichos operadores utilizamos la operación *until* recogida en el apéndice A:

$$until(Z,Z') = \{\gamma \in Z' \mid \exists\ \gamma' \in Z \text{ tal que } \gamma'+\tau = \gamma \text{ y } \forall \tau' \leq \tau, \gamma+\tau' \in Z'\}$$

y, por tanto:

- $pre_t([s,Z],\frac{1}{2}) = until(I_s(s),Z)$
- $pre_t([s,Z],1) = until(I_a(s),Z)$

El cálculo para los sucesores temporales es análogo utilizando el operador *since* recogido en el apéndice A.

Predecesor discreto

Para un estado de control s y una acción $a \in A^S$, sea:

$$dest_p(s,a) = \{t = (s_i, < a, g_{p_i}(s_i,a), \lambda_i >, s) \mid CE(t) = 1\}$$

y:

$$dest_s(s,a) = \{t = (s_i, < a, g^P_{s_i}(s_i,a), \lambda_i >, s) \mid CE(t) = \frac{1}{2}\}$$

El predecesor abstracto discreto $pre_d([s,Z],a,1)$ (resp. $pre_d([s,Z],a,\frac{1}{2})$) computará el conjunto de estados predecesores de algún estado en $[s,Z]$, mediante una transición discreta con condición de especificación 1 (resp. $\frac{1}{2}$):

$$pre_d([s,Z],a,1) = \{[s_i,Z_i] \mid t \in dest_p(s,a) \text{ y } Z_i = R^{-1}_{\lambda_i}(Z \cap g_{p_i}(s_i,a))\}$$

$$pre_d([s,Z],a,\frac{1}{2}) = \{[s_i,Z_i] \mid t \in dest_s(s,a) \text{ y } Z_i = R^{-1}_{\lambda_i}(Z \cap g^P_{s_i}(s_i,a))\}$$

Para la acción discreta genérica d:

$$pre_d([s,Z],d,c_e) = \bigcup_{a \notin A^S} pre_d([s,Z],a,c_e)$$

Sucesor discreto

Para un estado de control s y una acción $a \in A^S$, sea:

$$orig_p(s,a) = \{t = (s, < a, g_{p_i}(s,a), \lambda_i >, s_i) \mid CE(t) = 1\}$$

y:

$$orig_s(s,a) = \{t = (s, < a, g^P_{s_i}(s,a), \lambda_i >, s_i) \mid CE(t) = \frac{1}{2}\}$$
$$\bigcup \{t = (s, < a, g^T_s(s,a), \{\} >, s_s)\}$$

Los sucesores abstractos discretos, con condición de especificación posible o subespecificada, serán [1]:

$$suc_d([s,Z],a,1) = \{[s_i, Z_i] \mid t \in orig_p(s) \text{ y } Z_i = R_{\lambda_i}(Z \cap g_{p_i}(s,a))\}$$
$$suc_d([s,Z],a,\frac{1}{2}) = \{[s_i, Z_i] \mid t \in orig_s(s) \text{ y } Z_i = R_{\lambda_i}(Z \cap g_{s_i}(s,a))\}$$

Para la acción discreta genérica d:

$$suc_d([s,Z],d,c_e) = \bigcup_{a \notin A^S} suc_d([s,Z],a,c_e)$$

10.6.4 Algoritmo de minimización

El algoritmo 10.1 muestra el pseudocódigo del algoritmo de minimización Minimiza($\mathcal{M}$,$\mathcal{P}_0$), donde $\mathcal{M}$ es el modelo MUS-T y $\mathcal{P}_0$ es la partición parcial inicial. Dicha partición (parcial) debe ser tal que todos los estados (s,γ) accesibles en el modelo estén incluidos en algún elemento de la partición $\mathcal{P}_0$ que contendrá, por tanto, un elemento con estado de control s_s.

Debemos resaltar que la pre-estabilización con respecto a transiciones totalmente subespecificadas es implícita a la comprobación de estabilidad de los elementos de la partición con respecto a los elementos que contengan s_s. Para ese caso, $dest_s(s_s)$ será el conjunto de transiciones discretas totalmente subespecificadas.

10.6.5 Post-estabilización para el cálculo de operadores de pasado

Siendo P_i un elemento en la partición, el cálculo del valor de verdad de un operador de pasado en P_i implica la post-estabilización de P_i. Por tanto:

- Para el cálculo del valor de verdad de un operador $\Rightarrow_-$ en P_i, y dados $\{P_j\}$ los predecesores en el grafo cociente por transiciones t:
  - Si $suc_t(P_j, c_e) \cap P_i = \{\}$ se garantiza la post-estabilidad.

[1] Con $g_{s_i} \in \{g^P_{s_i}, g^T_s\}$ y $s_i = s_s$ cuando $g_{s_i} = g^T_s$

Algoritmo 10.1 Algoritmo de Minimización

```
Minimiza(M,P0)
{
   P:=P0
   Accesibles:={(s,Z) ∈ P0 | s=s0 y γ^0 ∈ Z}; Estables:={}

   while Accesibles - Estables ≠ {} do
      let P1=(s,Z) in Accesibles - Estables;

      /* Refinar con respecto a las transiciones de tiempo */
      if (∃ P2 ∈ P | P1 ∩ pre_t(P2,1) ∉ {P1,{}}) then
         Preestabiliza(P,Accesibles,Estables,P1,P2,t,{},1);
         continue;
      if (∃ P2 ∈ P | P1 ∩ pre_t(P2,1/2) ∉ {P1,{}}) then
         Preestabiliza(P,Accesibles,Estables,P1,P2,t,{},1/2);
         continue;

      /* Refinar con respecto a las transiciones discretas*/
      if (∃ P2 ∈ P, a ∈ A | P1 ∩ pre_d(P2,a,1) ∉ {P1,{}}) then
         Preestabiliza(P,Accesibles,Estables,P1,P2,d,a,1);
         continue;
      if (∃ P2 ∈ P | P1 ∩ pre_d(P2,a,1/2) ∉ {P1,{}}) then
         Preestabiliza(P,Accesibles,Estables,P1,P2,d,a,1/2)
         continue;

      /* es estable */
      Estables:= Estables ∪ P1;
      Accesibles:= Accesibles ∪ {P ∈ P | P1 ∩ pre_t(P,ce) ≠ 0}
                    ∪ {P ∈ P | P1 ∩ pre_d(P,a,ce) ≠ 0 para a ∈ A};
   end while
}

Preestabiliza(P,Accesibles,Estables,P1,P2,tipo,a,ce)
{
   P_P1 :={ P1 ∩ pre_tipo(P2,a,ce), P1 − pre_tipo(P2,a,ce)};
   Accesibles:= (Accesibles − {P1}) ∪ { P ∈ P_P1 | (s0,γ^0) ∈ P};
   Estables:= Estables − {P ∈ P | P ∩ pre_tipo(P1,a,ce) ≠ {}}
                    − {P ∈ P | P ∩ pre_tipo(P1,a,ce) ≠ {}};
   P:= (P − P1) ∪ P_P1;
}
```

  - Si $suc_t(P_j, c_e) \cap P_i = P_i$ se garantiza la post-estabilidad.
  - En otro caso, se refina P_i como $P_i \cap suc_t(P_j, c_e)$ y $P_i - suc_t(P_j, c_e)$.

- Para el cálculo del valor de verdad de un operador $\Rightarrow \bigodot$ en la zona P_i, se post-estabiliza la zona con respecto a transiciones discretas $\in A^s \cup \{d\}$.

10.6.6 Verificación y síntesis basada en minimización para SCTL-T

Sea un modelo MUS-T $\mathcal{M}$ y ϕ una fórmula a verificar o sintetizar. A partir de ϕ se define:

- $\mathcal{E}^f$ y $\mathcal{E}^p$, respectivamente los relojes de especificación de futuro y de pasado en ϕ.
- A^ϕ el conjunto de acciones $a \in A$ en ϕ.
- $\Psi^\phi_{\mathcal{C}}$, $\Psi^\phi_{\mathcal{E}^f}$ y $\Psi^\phi_{\mathcal{E}^p}$ las restricciones de tiempo en ϕ, respectivamente sobre relojes de modelo, relojes de especificación de futuro y relojes de especificación de pasado.

Sea $Props$ el conjunto de proposiciones atómicas A^ϕ, $\Psi^\phi_{\mathcal{C}}$, y, adicionalmente, las proposiciones en la expresión 10.2 para los relojes de especificación ($\mathcal{E}^f$ y $\mathcal{E}^p$) y sus predicados de tiempo ($\Psi^\phi_{\mathcal{E}^f}$ y $\Psi^\phi_{\mathcal{E}^p}$). Es decir, $Props$ es el conjunto de proposiciones atómicas en la fórmula ϕ a verificar.

El algoritmo que se propone para la verificación y síntesis de fórmulas SCTL-T realiza el cómputo del grafo cociente en dos pasos:

- **Paso I**: En primer lugar, calcula el grafo cociente mínimo (según $\text{STaB}_{\text{MUS-T}}$) a partir de una partición inicial preservando las proposiciones atómicas $Props$ correspondientes a elementos del modelo, es decir, $a \in A^\phi$ y $\Psi^\phi_{\mathcal{C}}$.
- **Paso II**: Se aplica el algoritmo de *model checking* o de síntesis, que extiende el grafo abstracto con relojes de especificación "al vuelo", y post-estabiliza la partición para el cálculo del grado de satisfacción de los operadores de pasado.

10.7 Conclusiones

En este capítulo, hemos definido las distintas relaciones que intervienen en la metodología propuesta:

- **Bisimulación de abstracción de tiempo fuerte**: permitiendo la verificación y síntesis de fórmulas SCTL-T en un modelo abstracto de menor tamaño que el obtenido bajo la equivalencia de regiones (capítulo 8).

- **Simulación de abstracción de tiempo especificada**: definiendo una relación de orden en el conjunto de modelos MUS-T generados a lo largo del proceso *software*, siendo la base para la corrección del algoritmo de síntesis. Además, el algoritmo de síntesis es tal que preserva el nivel de conocimiento de cualquier fórmula SCTL-T si se prescribe un criterio RCT. Tal resultado sugiere que, bajo un criterio RCT, la relación entre modelos generados por síntesis responde a una bisimulación especificada *backward-forward*.

- **Relación de abstracción de tiempo total posible**: siendo la abstracción que define el grafo de simulación posible en modelos MUS-T recogido en el capítulo 11. El grafo de simulación posible es la base, en la metodología SCTL/MUS-T, para la validación, mediante la simulación simbólica del modelo, y para aportar una garantía de corrección mínima sobre el modelo, mediante la detección de bloqueos en tiempo y en control.

CAPÍTULO 11

Grafo de simulación: animación y garantía de avance

En este capítulo se introduce el grafo de simulación del modelo MUS-T, utilizado en herramientas como KRONOS como un mecanismo menos costoso que el grafo de regiones para la resolución de problemas de alcanzabilidad. El grafo de simulación para MUS-T es la base, en la metodología propuesta, para la animación y para el chequeo de una corrección mínima en modelos MUS-T.

11.1 Introducción

La validación propuesta en la metodología SCTL/MUS-T se realizará a través de un animador que permita la ejecución simbólica del modelo. De esta forma, el usuario puede asegurarse de que la especificación captura el comportamiento deseado en el sistema, proporcionándose una forma intuitiva de validar la especificación. Además:

- Ejecutando simbólicamente el modelo a partir de un escenario de entrada (contraejemplo), el usuario puede llegar a comprender las razones del incumplimiento de propiedades específicas.
- Ejecutando simbólicamente el modelo aumentado con alguna sugerencia formal sobre la posible modificación del sistema, el usuario tiene una base de conocimiento para elegir una u otra posibilidad.

Para tal fin, se propone la utilización del grafo de simulación posible para MUS-T que se presenta en este capítulo. El grafo de simulación posible es la particularización del grafo de simulación, formalizado en (Daws y Tripakis, 1998), que es utilizado en la herramienta KRONOS (sección 3.4) para la verificación de propiedades resolubles por alcanzabilidad.

Si bien, el grafo de simulación posible representa el conjunto de comportamientos especificados al nivel de diseño actual, también se debe permitir la exploración las partes subespecificadas del modelo. Mediante la inspección de las partes subespecificadas, el diseñador puede valorar distintas alternativas de diseño en el modelo.

A pesar de que la animación simbólica del modelo por medio de su especificación formal puede aportar al diseñador ciertas garantías sobre la corrección del modelo, no se trata de un método sistemático. Una forma sistemática de aportar una garantía de corrección mínima consistiría en comprobar si el modelo está libre de bloqueos (*deadlock*). La ausencia de bloqueos en un sistema es un requisito mínimo ya que, si no se cumple, el modelo probablemente será incorrecto (suponemos que el sistema real nunca se para). Dado que un sistema temporizado puede evolucionar mediante pasos discretos y mediante pasos temporales, se definen dos tipos de avance (*progress*) y, consecuentemente, dos tipos de bloqueo en un sistema con tiempo. A partir del grafo de simulación posible, el animador debe comprobar tales fuentes de error.

En este capítulo se presenta el grafo de simulación posible para MUS-T, así como los operadores necesarios para permitir la inspección de las partes subespecificadas del modelo. Además del cómputo del grafo de simulación posible, se han desarrollado los algoritmos de detección de bloqueos operando sobre dicho grafo.

11.2 Abstracción en el grafo de simulación

El grafo de simulación posible de un modelo MUS-T es una abstracción del conjunto de estados $\mathcal{ST}$ de su grafo semántico basada en la relación $\mathrm{TP}_{\text{MUS-T}}$ definida en la sección 10.5. Tal abstracción sólo considera los estados alcanzables por transiciones discretas y temporales posibles ($t \in \mathcal{T}$ tal que $\mathcal{CE}(t) = 1$), y se abstrae totalmente de las transiciones temporales: el tiempo transcurre en el estado abstracto y sólo existen transiciones discretas.

11.3 Operadores de simulación

En base a la relación de abstracción expuesta en la sección 10.5, definiremos, en primer lugar, un conjunto de operadores abstractos o de simulación. Un operador de simulación, operando sobre un conjunto de estados, debe considerar todos aquellos estados alcanzables, a partir de los primeros, mediante la concatenación de una transición discreta $\xrightarrow{a}$ y una transición temporal $\xrightarrow{\tau}$. En base a las distintas condiciones de especificación, en las transiciones $\xrightarrow{a}$ y $\xrightarrow{\tau}$ se define un conjunto de operadores abstractos que permitirán el cómputo del grafo de simulación posible y el cómputo de las partes subespecificadas en base a dicha abstracción. Asimismo, los operadores de simulación definidos articulan la detección de bloqueos propuesta y permiten la distinción entre bloqueos reales y bloqueos transitorios, que pueden ser subsanados en fases posteriores del proceso *software*.

Sea un modelo MUS-T $\mathcal{M} =< S_0, S, T, I, A, \mathcal{C} >$ bajo el conjunto de verdad $\mathcal{L}_3$. Para $t \in T$ con $CE(t) \in \{\frac{1}{2}, 1\}$ definimos dos operadores sucesores abstractos

que computan los estados alcanzables mediante una transición discreta $t \in T$ más una transición temporal de τ arbitrario que podrá ser o bien posible o bien subespecificada.

DEFINICIÓN 11.1. **Sucesor abstracto con avance temporal posible.** Dada una zona de tiempo $Z \in \mathcal{P}^{\mathcal{C}}$ y una transición $(s, a^t, s\,') \in T$ con $CE(t) = \{\frac{1}{2}, 1\}$, definimos el operador $AsucTP([s, Z], a^t, s\,')$ que calcula sucesores abstractos con avance temporal posible (con a^t la acción temporizada $< a, g_x(s, a), \lambda >$) como:

$$AsucTP([s, Z], a^t, s\,') = \begin{cases} [s\,', since(R_\lambda(Z \cap g_x(s, a)), I_a(s\,'))] \text{ si } \neq \{\} \\ [s\,', (R_\lambda(Z \cap g_x(s, a)))] \text{ en otro caso} \end{cases}$$

donde $\cap$, R_λ y $since$ son los operadores sobre poliedros definidos en el apéndice A y, por simplicidad, $g_x(s, a)$ se refiere al poliedro $\in \mathcal{P}^{\mathcal{C}}$ definido por la restricción $g_x(s, a)$.

DEFINICIÓN 11.2. **Sucesor abstracto con avance temporal subespecificado.** Dada una zona de tiempo $Z \in \mathcal{P}^{\mathcal{C}}$ y una transición $(s, a^t, s\,') \in T$ con $CE(t) = \{\frac{1}{2}, 1\}$, definimos el operador $AsucTS([s, Z], a^t, s\,')$ para el cómputo sucesores abstractos con avance temporal subespecificado (con a^t la acción temporizada $< a, g_x(s, a), \lambda >$) como:

$$AsucTS([s, Z], a^t, s\,') = \begin{cases} [s\,', since(R_\lambda(Z \cap g_x(s, a)), I_s(s\,'))] \text{ si } \neq \{\} \\ [s\,', (R_\lambda(Z \cap g_x(s, a)))] \text{ en otro caso} \end{cases}$$

DEFINICIÓN 11.3. **Sucesor abstracto posible.** Dada una zona de tiempo $Z \in \mathcal{P}^{\mathcal{C}}$ y un arco posible $(s, < a, g_p(s, a), \lambda >, s\,') \in T$, definimos el operador $AsucP([s, Z], a^t, s\,')$ de sucesores abstractos (con a^t la acción temporizada $< a, g_p(s, a), \lambda >$) como:

$$AsucP([s, Z], a^t, s\,') = AsucTP([s, Z], a^t, s\,')$$

El estado abstracto $AsucP([s, Z], a^t, s\,')$ representa el conjunto de estados concretos alcanzables a partir de algún estado (s, γ), con $\gamma \in Z$, mediante una transición discreta posible a y una transición temporal τ arbitraria que satisfaga continuamente la invariante de avance de $s\,'$.

Dado que el conjunto de estados abstractos obtenidos mediante la abstracción $\mathrm{TP}_{\text{MUS-T}}$ puede ser infinito, se compone la abstracción $\mathrm{TP}_{\text{MUS-T}}$ con la abstracción de extrapolación (Daws y Tripakis, 1998). La idea de la extrapolación es semejante a la utilizada en la definición del grafo de regiones (sección 8.2). Si k es la mayor constante apareciendo en las restricciones de tiempo que definen las guardas o las invariantes de un modelo MUS-T, el valor exacto de una interpretación para un reloj $x \in \mathcal{C}$ es irrelevante una vez que $\gamma(x) > k$. Por tanto, redefinimos los operadores:

$$\begin{aligned} AsuccP([s, Z], a^t, s\,') &= cierre(AsucP([s, Z], a^t, s\,'), k) \\ AsuccTP([s, Z], a^t, s\,') &= cierre(AsucTP([s, Z], a^t, s\,'), k) \\ AsuccTS([s, Z], a^t, s\,') &= cierre(AsucTS([s, Z], a^t, s\,'), k) \end{aligned} \tag{11.1}$$

siendo *cierre* el operador sobre poliedros definido en el apéndice A.

11.4 Grafo de simulación posible para MUS-T

El grafo de simulación posible para MUS-T se construye en base a la relación de abstracción $TP_{\text{MUS-T}}$ definida en la sección 10.5. En consecuencia, sólo incluirá estados del modelo MUS-T alcanzables mediante transiciones con condición de especificación posible, de ahí su denominación.

Dado el modelo MUS-T $\mathcal{M}$, definimos su grafo de simulación posible $\mathcal{M}^{sim}$ como la tripla $< S_0^{sim}, S^{sim}, T^{sim} >$, con S_0^{sim} el conjunto de estados abstractos iniciales, S^{sim} el conjunto de estados abstractos, y $T^{sim} \subseteq S^{sim} \times A \times S^{sim}$ el conjunto de arcos etiquetados en el alfabeto A. El grafo $\mathcal{M}^{sim}$ se construye bajo las siguientes reglas:

- $\forall s_0 \in S_0, s_0^{sim} \in S_0^{sim}$, donde:

$$s_0^{sim} = \begin{cases} [s_0, since(Z^0, I_a(s_0))] \text{ si } \neq \{\} \\ [s_0, Z^0] \text{ en otro caso} \end{cases}$$

  siendo la zona Z^0 aquella que incluye como única interpretación γ^0.

- $\forall\ s^{sim} = [s, Z] \in S^{sim}$ y $\forall\ (s, a^t =< a, g_p(s,a), \lambda >, s\,') \in T$ con CE(t)=1, si $AsuccP(s^{sim}, a^t, s\,') \neq \{\}$, entonces:
  - $s^{sim\,\prime} = AsuccP(s^{sim}, a^t, s\,') \in S^{sim}$
  - $(s^{sim}, a, s^{sim\,\prime}) \in T^{sim}$

Dado que la extrapolación garantiza un número de estados abstractos finito bajo la abstracción $\text{TP}_{\text{MUS-T}}$, el grafo de simulación para MUS-T es finito.

11.5 Propiedades del grafo de simulación

Gran parte del éxito del grafo de simulación en su aplicación al análisis de modelos con tiempo se basa en la doble simulación implícita en su construcción. Informalmente, mientras una simulación entre dos estructuras E y $E\,'$ garantiza que todos los comportamientos en E tienen un comportamiento en $E\,'$ que se le corresponde, una doble simulación de E a $E\,'$ y de $E\,'$ a E, garantiza, además, que todos los comportamientos en $E\,'$ tienen su correspondiente en E.

Formalmente, sea $\mathcal{M}$ un modelo MUS-T y $\mathcal{M}^{sim}$ su grafo de simulación posible. Entre $\mathcal{S}_{\mathcal{M}}$ (grafo semántico de $\mathcal{M}$) y su grafo de simulación $\mathcal{M}^{sim}$, se establecen dos simulaciones:

- Simulación $\sqsubseteq_1 \subseteq (\mathcal{ST} \times S^{sim})$: para todo $s \in \mathcal{ST}$

$$\text{si } s \xrightarrow{a}\xrightarrow{\tau} s\,' \text{ y } s \in s^{sim}, \text{ entonces } s^{sim} \xrightarrow{a} s^{sim\,\prime}$$

- Simulación $\sqsubseteq_2 \subseteq (S^{sim} \times \mathcal{ST})$: para todo $s^{sim} \in S^{sim}$

$$\text{si } s^{sim} \xrightarrow{a} s^{sim\,\prime}, \text{ entonces } \exists\, s \in s^{sim} \mid s \xrightarrow{a}\xrightarrow{\tau} s\,' \text{ con } s\,' \in s^{sim\,\prime}$$

El operador $AsuccP$ (expresión 11.1) computa el estado abstracto sucesor por un arco posible $< a, g_p(s,a), \lambda >$, donde:

[1] En $AsuccP([s,Z], a^t, s\,')$ se encuentran todos los estados concretos tales que $(s,\gamma) \xrightarrow{a}\xrightarrow{\tau} (s\,',\gamma\,')$, con $\gamma \in Z$ y $\xrightarrow{a}$ la a-transición posible definida por a^t.

[2] Todos los estados concretos en $[s\,',Z\,'] = AsuccP([s,Z], a^t, s\,')$ tienen su correspondiente en $[s,Z]$, es decir, para cualquier estado $(s\,',\gamma\,') \in [s\,',Z\,']$ existe un estado $(s,\gamma) \in [s,Z]$ tal que $(s,\gamma) \xrightarrow{a}\xrightarrow{\tau} (s\,',\gamma\,')$.

Las propiedades [1] y [2] (figura 11.1(a)) garantizan que el grafo de simulación es post-estable respecto a transiciones posibles diferidas.

La doble simulación implícita en la definición garantiza que todos los estados en el grafo de simulación posible son alcanzables por transiciones posibles en la ejecución del modelo original. Inversamente, cualquier estado alcanzable por transiciones posibles, en la ejecución del modelo original, pertenece a algún estado abstracto en el grafo de simulación.

11.5.1 Pre-estabilización del grafo de simulación

Dado que el grafo de simulación es post-estable, nos aseguramos de que todos los estados concretos en $AsuccP([s,Z], a^t, s\,')$ provienen de algún estado en $[s,Z]$ mediante una transición discreta y, posiblemente, una temporal. Por contra, la simulación no es pre-estable, es decir, pueden existir estados concretos en $[s,Z]$ que no tengan un a sucesor en $AsuccP([s,Z], a^t, s\,')$. Definimos los operadores predecesores abstractos de forma inversa a los operadores sucesores. Como veremos, dichos operadores permiten pre-estabilizar el grafo de simulación en la búsqueda de bloqueos.

Definición 11.4. **Predecesor abstracto con retroceso temporal posible.** Dada una zona de tiempo $Z \in \mathcal{P}^{\mathcal{C}}$ y una transición $(s\,', a^t, s) \in T$ con $CE(t) = \{\frac{1}{2}, 1\}$, definimos el operador $ApreTP([s,Z], a^t, s\,')$ de predecesores abstractos con retroceso temporal posible (con a^t la acción temporizada $< a, g_x(s\,', a), \lambda >$) por:

$$ApreTP([s,Z], a^t) = \begin{cases} [s\,', R_\lambda^{-1}(until(Z, I_a(s\,'))) \cap g_x(s\,', a)] \text{ si } \neq \{\} \\ [s\,', R_\lambda^{-1}(Z) \cap g_x(s\,', a)] \text{ en otro caso} \end{cases}$$

Definición 11.5. **Predecesor abstracto con retroceso temporal subespecificado.** Dada una zona de tiempo $Z \in \mathcal{P}^{\mathcal{C}}$ y una transición $(s\,', a^t, s) \in T$ con $CE(t) = \{\frac{1}{2}, 1\}$, definimos el operador $ApreTS([s,Z], a^t, s\,')$ de predecesores abstractos con retroceso temporal subespecificado (con a^t la acción temporizada $< a, g_x(s\,', a), \lambda >$) por:

$$ApreTS([s,Z], a^t, s\,') = \begin{cases} [s\,', R_\lambda^{-1}(until(Z, I_s(s\,'))) \cap g_x(s\,', a)] \text{ si } \neq \{\} \\ [s\,', R_\lambda^{-1}(Z) \cap g_x(s\,', a)] \text{ en otro caso} \end{cases}$$

y, por tanto, el predecesor abstracto posible por:

$$ApreP([s,Z],a^t,s') = ApreTP([s,Z],a^t,s')$$

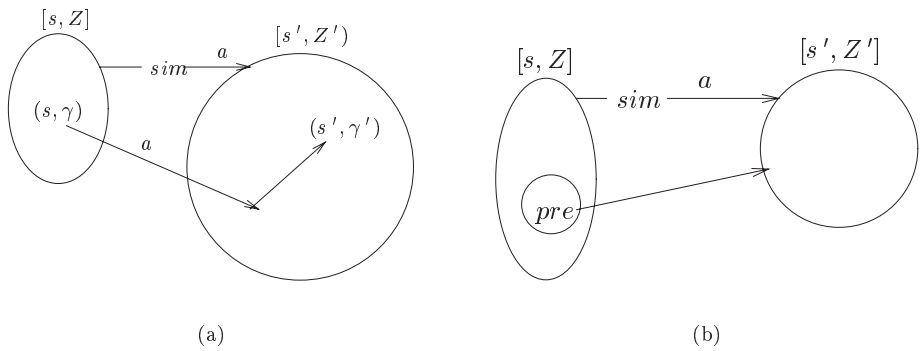


Figura 11.1. Operador $AsuccP$ 11.1(a) y operador $Apre$ 11.1(b)

La idea de la pre-estabilización se muestra en la figura 11.1. El operador $ApreP$ sobre un estado del grafo de simulación $[s', Z']$, con respecto a una transición $(s, a^t =< a, g_p(s,a), \lambda >, s') \in T$ con $CE(T) = 1$, calcula el conjunto de estados concretos en el estado $[s, Z]$ tales que existe una evolución $\xrightarrow{a}\xrightarrow{\tau}$ a un estado en $[s', Z']$.

11.5.2 Corrección mínima del modelo

Los requisitos de avance en el autómata temporizado fueron introducidos originalmente en (Henzinger et al., 1992b) mediante la definición del concepto *nonzenoness*. Informalmente, un modelo será *nonzeno* si permite avanzar el tiempo sin una cota superior.

DEFINICIÓN 11.6. **Traza *zeno* (Henzinger et al., 1992b).** Sea la semántica lineal del modelo MUS-T definida en la sección 6.6.3. Una traza infinita con origen en el estado (s_1, γ_1) de un modelo MUS-T:

$$(s_1,\gamma_1) \overset{(\omega_1,l_1)}{\multimap} (s_2,\gamma_2) \cdots \overset{(\omega_{n-1},l_{n-1})}{\multimap} (s_{n-1},\gamma_{n-1}) \overset{(\omega_n,l_n)}{\multimap} (s_n,\gamma_n) \cdots$$
$$\text{y } l = \bigwedge_{i=1\cdots n} l_i$$

es *zeno* sii $\exists\ \tau \in \Re_+$, tal que para todo i, $\gamma_i < \tau$. En otro caso, se dirá *nonzeno*.

Un estado (s, γ) es *nonzeno* si existe una traza *nonzeno* con $(s_1, \gamma_1) = (s, \gamma)$ y $l = 1$. En otro caso, se dirá de calidad *nonzeno* subespecificada si existe una traza *nonzeno* con $l = \frac{1}{2}$. Si no existen trazas *nonzeno*, el estado (s, γ) es un estado *zeno*.

Por tanto, un modelo MUS-T es *nonzeno* si todos los estados (s, γ) alcanzables por transiciones posibles son *nonzeno*. En otro caso, el modelo podrá contener estados alcanzables *zeno*, o con calidad *zeno* subespecificada.

Para un autómata temporizado, la comprobación *nonzeno* se puede caracterizar mediante la fórmula TCTL $\phi^{nz} = \forall\ \Box(\exists\ \Diamond_{\geq 1} true)$. Es decir, en cualquier estado del modelo, es siempre posible avanzar el tiempo una unidad. Por definición, si un autómata temporizado es *nonzeno* satisface ϕ^{nz}, e inversamente, si el autómata temporizado satisface ϕ^{nz}, siempre es posible, a partir de cualquier estado, construir una traza infinita con pasos temporales 1, obteniendo una traza *nonzeno*.

Posteriormente, en (Tripakis, 1998) se hace una distinción explícita entre los bloqueos de avance discreto (*deadlocks*) y los bloqueos de avance temporal (*timelocks*) en el autómata temporizado.

El requisito de evolución discreta o ausencia de *deadlock* establece que no existen estados en los que no es posible realizar ninguna transición discreta, incluso después de un paso temporal.

El requisito de evolución temporal o ausencia de *timelock* establece que el tiempo debe avanzar sin la existencia de ningún límite superior. En un modelo libre de *deadlock*, la evolución temporal se traduce en la ausencia de estados *timelock*, es decir, estados a partir de los cuales todas las posibles ejecuciones infinitas no permiten el avance del tiempo.

Si bien la detección de *deadlocks* puede realizarse mediante una abstracción obtenida por $\mathrm{STaB}_{\text{MUS-T}}$, tal bisimulación no respeta la calidad *nonzeno* del sistema, dado que se trata de una bisimulación que se abstrae de la parte cuantitativa de los pasos temporales. Aun así, siempre sería posible, mediante un grafo cociente obtenido para la verificación de la fórmula ϕ^{nz}, su utilización para la detección de *timelocks*.

Sin embargo, dado que el grafo de simulación preserva el conjunto de estados alcanzables en un modelo, incluimos la detección de *timelocks* y *deadlocks* como un método de decisión asociado al animador en la metodología. Así, presentaremos los algoritmos de detección de *deadlocks* y *timelocks* sobre el grafo de simulación, siguiendo la ideas presentadas en (Tripakis, 1999).

En las siguientes secciones, se presentan los algoritmos que permiten la detección de *timelocks* y *deadlocks* en un modelo MUS-T. Se presentan dos versiones: la que opera sobre el grafo de regiones y la que opera sobre el grafo de simulación. Por último, citar que, aunque la computación sobre el grafo de simulación supone un ahorro importante respecto al grafo de regiones, el tamaño de dicho grafo puede ser elevado si jugamos con la composición de modelos MUS-T. En (Tripakis, 1999) se dan condiciones suficientes para el tratamiento de los requisitos de avance en el autómata temporizado, que pueden ser computadas de forma estática sobre la estructura del autómata y que serán adaptadas al modelo MUS-T en las siguientes secciones.

11.5.3 Detección de *deadlocks*

- Un estado temporizado (s, γ) en $\mathcal{S}^{\mathcal{M}}$ es *no-deadlock* si existe un $\tau \in \Re_+$ y $a \in A$, tal que:
$$(s, \gamma) \xrightarrow{\tau}\xrightarrow{a} (s', \gamma')$$
donde tanto $\xrightarrow{\tau}$ como $\xrightarrow{a}$ son transiciones posibles.

- Un estado temporizado (s, γ) es *deadlock* si no existe $\tau \in \Re_+$, $a \in A$, tal que:
$$(s, \gamma) \xrightarrow{\tau}\xrightarrow{a} (s', \gamma')$$
donde tanto $\xrightarrow{\tau}$ como $\xrightarrow{a}$ son transiciones posibles, y no existen transiciones con origen en (s, γ) subespecificadas.

- Un estado temporizado (s, γ) es *deadlock* subespecificado (*deadlock-sub*) si no es ni *deadlock* ni *no-deadlock*. Es decir, el estado (s, γ) no garantiza la ausencia de *deadlock* pero, dado que se trata de un estado en el que las evoluciones temporales o discretas no se encuentran totalmente especificadas, podría evolucionar en el proceso de diseño a un estado *no-deadlock* o *deadlock*.

En un modelo MUS-T no totalmente especificado, la característica *no-deadlock* en todos los estados alcanzables del modelo nos garantiza una mínima corrección; la característica *deadlock* nos garantiza que, probablemente, el modelo sea incorrecto, y la característica *deadlock* subespecificada nos indica que hay algún estado que no tiene evolución discreta (diferida) posible, pero, dado que dicho estado no está todavía totalmente especificado podría evolucionar hacia un modelo *no-deadlock*.

Detección sintáctica de *deadlock*

Una condición suficiente sobre la sintaxis de un modelo MUS-T para la ausencia de *deadlock* es la siguiente. Un modelo MUS-T está libre de *deadlock*, si para todos los estados de control $s \in S$ y para todas las transiciones posibles a con origen en s' y destino en s, se cumple:

$$R_\lambda(g_p(s', a)) \ \cup \ since(R_\lambda(g_p(s', a)), I_a(s)) \subseteq \bigcup_{\forall a' \in P(s)} (g_p(s, a') \cup \\ until(g_p(s, a'), I_a(s)))$$

donde la expresión a la izquierda de la inclusión representa una aproximación del conjunto de estados que puede alcanzar s, y la expresión a la derecha los contextos temporales en s que pueden realizar una transición discreta diferida con condición de especificación posible.

Detección de *deadlocks* mediante el grafo de regiones

Sea $\mathcal{R}_{\mathcal{M}}$ el grafo RMUS-T de un modelo $\mathcal{M}$. Los vértices del grafo serán regiones $< s, \mathcal{RT} >$ y las transiciones estarán etiquetadas o bien con las acciones

del alfabeto $a \in A$ o bien con un paso temporal arbitrario t. Además, cada estado $< s, \mathcal{RT} >$ tendrá asociadas un conjunto de acciones no posibles y un conjunto de acciones subespecificadas, ambos disjuntos entre sí y disjuntos con el conjunto de acciones posibles en ese estado que etiqueta sus transiciones.

El algoritmo de detección de *deadlock* mediante el grafo de regiones se muestra en el pseudocódigo 11.1. Para dicho algoritmo, se definen los operadores sobre regiones del grafo RMUS-T: RsucP_d y RsucP_t, de forma que RsucP_d nos devuelve el conjunto de sucesores discretos posibles y RsucP_t el sucesor temporal posible. El operador sobre regiones Rsuc_{sub} comprueba si existe algún sucesor discreto o sucesor discreto diferido subespecificado.

Algoritmo 11.1 Detección de *deadlock* en el grafo de regiones

```
(ResD,ResSD) Busqueda_Deadlock(RMUS-T)
{
   Visitados:={}
   AVisitar:={R0=< s0, RT0 >}

   while AVisitar ≠ {} do
      let R ∈ AVisitar
      if R ∉ Visitados
         if RsucPd(R) ∪ diferido(RsucPt(R)) ≠ {}
            Visitados:= Visitados ∪ {R}
            AVisitar:= AVisitar - R ∪ {RsucPt(R) ∪ RsucPd(R)}
         else
            if Rsucsub(R) then ResSD := ResSD + R
            else ResD := ResD + R
         end if
      else Avisitar:=Avisitar - R
   end while
   return (NO-DEADLOCK,_)
}

diferido(R)
{
   if RsucPd(R) ≠ {} then return RsucPd(R)
   else if (RsucPt(R))={} then return {}
      else diferido(RsucPt(R))
}
```

Detección de *deadlocks* mediante el grafo de simulación

Dado que el grafo de simulación ha sido construido mediante una abstracción exacta con respecto al problema de alcanzabilidad, podemos utilizar dicho grafo para la detección de *deadlocks*. Dado un estado abstracto en el grafo de simulación $[s, Z]$, definimos las condiciones $libre - deadlock$, y $sub - deadlock$ como sigue.

Definición 11.7. La condición $libre - deadlock$ caracteriza el conjunto de estados

concretos con localización s que están libres de *deadlock*:

$$libre - deadlock(s) = \bigcup_{\forall a} g_p(s,a) \ \cup \ until(g_p(s,a), I_a(s))$$

DEFINICIÓN 11.8. La condición $sub - deadlock$ caracteriza el conjunto de estados concretos con localización s cuya calidad *deadlock* está subespecificada:

$$\begin{aligned} sub - deadlock(s) = & \left(\bigcup_{\forall g_s} g_s(s,a) \ \cup \ until(g_s(s,a), I_a(s) \cup I_s(s)) \right) \\ & \cup \left(\bigcup_{\forall g_p} until(g_p(s,a), I_s(s)) \right) \\ & \cup \left(\bigcup_{\forall a} g_s^T(s,a) \ \cup \ until(g_s^T(s,a), I_a(s) \cup I_s(s)) \right) \end{aligned}$$

Algoritmo 11.2 Detección de *deadlock* con grafo de simulación

```
(ResD,ResSD) Busqueda_Deadlock_Sim(s^sim,Visitados)
{
    let [s,Z]=s^sim
    if s^sim ∉ Visitados =[s',Z'] ∈ Visitados | s=s' y Z⊆ Z'
        Visitados.push(s^sim)
        foreach (a^t,s' ∈ P(s)
            (ResD,ResSD)+=:Busqueda_Deadlock_Sim(AsuccP(s^sim,a^t,s'),Visitados)
        end foreach
        if Z ⊆ libre-deadlock(s) then
            return ({},{})
        else
        let Z':= Z - libre-deadlock(s)
            if Z' ⊆ sub-deadlock(s) then
                return ({},{[s,Z'}})
            else return ({[s, Z' - sub-deadlock(s)]},{[s,Z' ∩ sub-deadlock(s)})
    else return ({},{})
}
```

El algoritmo de detección de *deadlocks* (pseudocódigo 11.2) explora la totalidad de estados alcanzables (posibles) en el modelo MUS-T actual (posiblemente incompleto), mediante la construcción del grafo de simulación posible. El algoritmo devuelve un conjunto de estados abstractos tanto para *deadlock* (ResD), como para *deadlock-sub* (ResDS). A diferencia del algoritmo de detección de *timelocks* que veremos a continuación, no es necesario pre-estabilizar el algoritmo, dado que el criterio que determina la condición de *deadlock* es local al estado abstracto.

11.5.4 Detección de *timelocks*

Basándonos en la noción de evolución temporal (*nonzeno*) en (Henzinger et al., 1992b), definimos el nivel de *timelock* de un estado del grafo de regiones:

- Un estado temporizado (s, γ) es *no-timelock* si existe una traza infinita (con transiciones discretas y temporales posibles) partiendo de (s, γ) a (s', γ'), tal que $\gamma' - \gamma \geq 1$.

- Un estado temporizado (s, γ) es *timelock* si, para todas las trazas infinitas (con pasos posibles) partiendo de (s, γ), no existe ningún estado (s', γ') satisfaciendo $\gamma' - \gamma \geq 1$, no existiendo una traza subespecificada.

- En caso contrario, el estado poseerá una calidad de *timelock* subespecificada (*timelock-sub*).

Detección sintáctica de *timelock*

En (Tripakis, 1999) se define, bajo la denominación *strongly nonzeno*, una condición suficiente sobre la sintaxis del modelo para la ausencia de *timelocks*. En base a tal definición, diremos que un modelo MUS-T es *strongly nonzeno* sii para cualquier ciclo $s_1, s_2 \cdots s_n, s_1$ de estados de control con arcos posibles, existe un reloj x y dos estados $s_i < s_j$ tales que:

[1] x es reiniciado en s_i

[2] $g_p^{\rightarrow j} \cap (x < 1) = \{\}$, donde $g_p^{\rightarrow j}$ es la guarda en el modelo MUS-T del arco $s_{j-1} \rightarrow s_j$ en el ciclo.

Intuitivamente, dichas propiedades garantizan que por lo menos una unidad de tiempo transcurre en cada ciclo del modelo.

Detección de *timelock* mediante el grafo de regiones

Para la detección de *timelocks* utilizamos dos algoritmos de búsqueda anidados (ver algoritmo 11.3), uno semejante al utilizado en la detección de *deadlocks* que explora todo el grafo de regiones, y otro que, para cada una de las regiones exploradas, identifica la calidad de *timelock* de la región (*timelock*, *no-timelock* o *timelock-sub*). La calidad de *timelock* de una región es una propiedad cuantitativa, con lo cual utilizamos el mismo mecanismo que el *model checking* de SCTL-T: incorporamos un reloj adicional tl que nos permitirá comprobar la condición de no-*timelock* $\gamma(tl) \geq 1$.

El algoritmo anidado que detecta el *timelock* en un estado $< s, \mathcal{RT} >$ construye el grafo RMUS-T que parte del estado $< s, \mathcal{RT}_{tl=0} >$, siendo $\mathcal{RT}_{tl=0}$ la región obtenida a partir de $\mathcal{RT}$ reiniciando el reloj tl.

Algoritmo 11.3 Detección de *timelock* con grafo de regiones

```
(ResT, ResST) Busqueda_Timelock(RMUS-T)
{
    Visitados:={}
    AVisitar:={s0=< s0, RT0 >}

    while AVisitar ≠ {}do
        let R ∈ AVisitar
        if R ∉ Visitados
            case timelock(R+_{tl=0})
                TL: ResT := ResT + R
                TLSUB : ResST := ResST
                NTL :
            end case
        Avisitar:=Avisitar - R ∪ RsucP_t(R) ∪ RsucP_d(R)
        Visitados:= Visitados ∪ {R}
        end if
        else Avisitar:=Avisitar - {R}
    end while
    return (ResT,ResST)
}
timelock(R+)
{
    Sub+:=false
    Visitados+:={}
    Avisitar+:={R+}

    while AVisitar+ ≠ {} do
        let R ∈ AVisitar+
        if R ∉ Visitados+ then
            if tl ≥ 1 then return NTL
            else
                if Rsuc_sub(R) ≠ {} then Sub+:=true
                AVisitar+ := Avisitar+ - R ∪ RsucP_t(R) ∪ Rsuc_d(R)
                Visitados+ := Visitados+ ∪ {R}
        end if
    end while
    if (Sub+) then return TLSUB
    else return TL
}
```

Detección de *timelock* mediante el grafo de simulación

La detección de *timelocks* mediante el grafo de simulación es algo más compleja que su equivalente para *deadlocks*. La diferencia estriba en que, mientras que en la detección de *deadlocks* existe una condición local al estado abstracto, la detección de *timelocks* impone realizar una búsqueda completa en un grafo de simulación aumentado con un reloj adicional que registra la condición de no-*timelock*.

Dado que debemos garantizar que todos los estados concretos de un estado abstracto son no-*timelock*, debemos pre-estabilizar el algoritmo. Es decir, el hecho de que un estado abstracto $[s, Z]$ alcance otro estado abstracto que garantice el no-*timelock*, no indica que todos los estados concretos en $[s, Z]$ lo hagan.

La estructura (ver algoritmo 11.4) es idéntica a la del algoritmo de detección con el grafo de regiones, pero, a diferencia de éste, se realiza la pre-estabilización de los caminos comprobados.

Para la detección de *timelock* subespecificados es necesaria la búsqueda de trazas subespecificadas a partir de un estado abstracto. Para ello, definimos el operador $AsuccS$, que computa el estado abstracto sucesor por transiciones subespecificadas. Para tal fin, es necesario distinguir en el operador $AsuccS$ si el estado abstracto pertenece al grafo de simulación posible para MUS-T ($c_e = 1$) o, por contra, ya es un estado abstracto correspondiente a estados concretos alcanzables por subespecificación ($c_e = \frac{1}{2}$).

Definición 11.9. **Sucesor abstracto subespecificado.** Para un estado abstracto subespecificado $[s, Z]$ con condición de especificación c_e, y un arco $t = (s, a^t =< a, g_x(s, a), \lambda >, s\,') \in T$ con $CE(t) = \{1, \frac{1}{2}\}$, definimos el operador $AsuccS([s, Z], a^t, s\,', c_e)$ de sucesores abstractos subespecificados:

$$AsuccS([s, Z], a^t, s\,', c_e) =$$
$$\begin{cases} \text{si } c_e = 1 \begin{cases} AsuccTS([s, Z], a^t, s\,') \text{ si } CE(t) = 1 \\ AsuccTP([s, Z], a^t, s\,') \cup AsuccTS([s, Z], a^t, s\,') \text{ si } CE(t) = \frac{1}{2} \end{cases} \\ \text{si } c_e = \frac{1}{2} \begin{cases} AsuccTP(s, Z, a^t, s\,') \cup AsuccTS(s, Z, a^t, s\,') \end{cases} \end{cases}$$

Inversamente, para la pre-estabilización de estados abstractos alcanzables por subespecificación definimos el operador $ApreS$.

Definición 11.10. **Predecesor abstracto subespecificado.** Para un estado abstracto $[s, Z]$ y un arco $t = (s\,', a^t =< a, g_x(s, a), \lambda >, s) \in T$ con $CE(t) = \{1, \frac{1}{2}\}$, definimos el operador $ApreS([s, Z], a^t, s\,', c_e)$ de predecesores abstractos subespecificados:

$$ApreS([s, Z], a^t, s\,') =$$
$$\begin{cases} ApreTS([s, Z], a^t, s\,') \text{ si } CE(t) = 1 \\ ApreTP([s, Z], a^t, s\,') \cup ApreTS(s, Z, a^t, s\,') \text{ si } CE(t) = \frac{1}{2} \end{cases}$$

Algoritmo 11.4 Detección de *timelock* con grafo de simulación

```
(ResT,ResST) Busqueda_Timelock_Sim(s^sim,Visitados)
{
    let [s,Z]=s^sim
    if (∄ s^sim'=[s',Z'] ∈ Visitados | s=s' y Z⊆ Z') then
        Visitados.push(s^sim)
        foreach (a^t,s') ∈ P(s)
            (ResT,ResST)+=: Busqueda_Timelock_Sim(AsuccP(s^sim,a^t,s'),Visitados)
        end foreach
        while ((Comprobado:=Timelock+([s,Z^+|_{tl=0}],{},P,1) ≠ 0 AND Z ≠ {})
            Z:= Z - Comprobado
        end while
        let Z_TL=Z;
        if (Z ≠ {}) then
            while ((Comprobado:=Timelock+([s,Z^+|_{tl=0}],{},S,1) ≠ 0 AND Z ≠ {})
                Z:= Z - Comprobado
            end while
            if (Z ≠ {}) then return ({[s,Z]},{[s,Z_TL - Z]})
            else return ({},{[s,Z_TL]})
        else
            return ({},{})
        end if
    else return ({},{}) }

Timelock+(s^sim+,Visitados_+,tipo,c_e)
{   /* tipo P o S */
    /* Z-tl:= tl ≥ 1 */
    let [s,Z^+]=s^sim+
    if (Z^+ ∩ Z-tl ≠ {}) then return ([s,Z^+ ∩ Z-tl])
    Visitados_+:=Visitados_+ ∪ s^sim+
    /* tipo(s)=P(s) si tipo = P; tipo(s)=P(s) ∪ S(s) si tipo = S*/
    foreach (a^t,s') ∈ tipo(s) do
    /* Asucc_P(s^sim+,a^t,s',c_e)= AsuccP(s^sim+,a^t,s') */
    /* Asucc_S(s^sim+,a^t,s',c_e)= AsuccS(s^sim+,a^t,s',c_e)*/
        s-suc:=Asucc_tipo(s^sim+,a^t,s',c_e);
        if (s-suc ≠ 0 AND s-suc ∉ Visitados_+) then
            if (c_e=1 AND tipo(s)=S) then s-ntl:= Timelock+(s-suc,Visitados_+,tipo,1/2);
            else s-ntl:= Timelock+(s-suc,Visitados_+,tipo,c_e);
            if (s-ntl ≠ {}) then return s^sim+ ∩ Apre_tipo(s-ntl,a^t);
        end if
    end foreach
    return {}
}
```

11.6 Conclusiones

Con la definición de la **relación de abstracción de tiempo total posible** ($\mathrm{TP}_{\text{MUS-T}}$) en la sección 10.5, se ha proporcionado la base para el cómputo del grafo de simulación posible en modelos MUS-T. Dicho grafo de simulación permite la validación en la metodología por medio de la ejecución simbólica de la especificación formal. En base al grafo de simulación y a los operadores abstractos definidos, se proporcionan las bases para una herramienta de animación que permita inspeccionar los comportamientos temporizados aun no especificados en el modelo. Sin embargo, especialmente este tipo de herramientas deben ir acompañadas de capacidades gráficas potentes y fácilmente utilizables.

Asimismo se ha utilizado el grafo de simulación posible para la detección de bloqueos, tanto en tiempo como en control, en el modelo. La herramienta de animación, además de permitir la ejecución simbólica, debe permitir la identificación rápida de estas incorrecciones potenciales.

PARTE V

Caso de estudio y conclusiones

CAPÍTULO 12

Caso de estudio

En este capítulo se presenta el caso de estudio seleccionado para mostrar las ventajas de la metodología propuesta. Dicho caso de estudio ha sido propuesto en (Abrial et al., 1996) como una forma de evaluar distintas metodologías formales en su aplicación a un caso real de tamaño manejable.

12.1 Introducción y motivación del caso de estudio

El caso de estudio abordado en este capítulo, bajo el nombre *steam boiler*, fue propuesto a finales del año 1994 con el objetivo de evaluar distintas metodologías formales para el diseño de un sistema real. El resultado de tal competición, llevada a cabo en Junio de 1995, es recopilado en el libro (Abrial et al., 1996) (recogiendo las 21 mejores soluciones), en el que también se puede encontrar la especificación informal del caso de estudio proporcionada a los competidores. Podemos encontrar más información de la competición en:

http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html

Con el objetivo de la evaluación de distintas técnicas formales se pide a los participantes que propongan una solución a un problema lo bastante grande como para exhibir características de un sistema real, pero lo bastante pequeño como para ser desarrollado en un tiempo razonable.

El caso de estudio se esquematiza en la figura 12.1. Una caldera de producción de vapor dispone de 4 bombas que introducen agua en la caldera, y una salida para el vapor producido. Además, dispone de una unidad de medida del nivel de agua, una unidad de medida del flujo de vapor saliente, cuatro dispositivos que comprueban el funcionamiento correcto de las bombas, un sistema de transmisión de mensajes y un operador humano. El objetivo es el diseño de un controlador para la caldera que, en base a la información proporcionada por los sensores, realice las acciones adecuadas, que deben ir destinadas a mantener el nivel de agua en

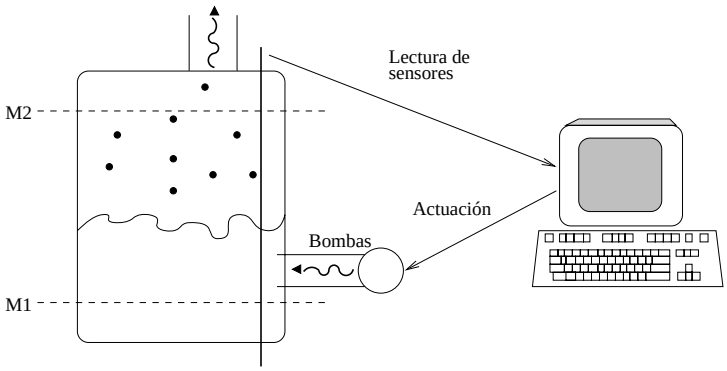


Figura 12.1. Esquema del problema

los límites normales (rendimiento), y a evitar la explosión o el daño de la turbina (seguridad).

La elección del *steam boiler* como caso de estudio en la presente tesis se basa en los siguientes puntos:

- La disponibilidad de soluciones de distinta naturaleza que, por un lado, ha permitido una mejor comprensión del problema y, por otro lado, posibilita la comparación de los resultados obtenidos.
- Las características del caso de estudio que conllevan, de forma natural, un proceso iterativo e incremental que permite cotejar distintas alternativas de diseño.
- Las características de tiempo real del caso de estudio y, al mismo tiempo, su naturaleza crítica.

El caso de estudio, aunque simple, ha requerido un tiempo de desarrollo de los equipos participantes variado, aunque considerable, entre 2 y 9 meses. En base a ello, y por la brevedad de la presentación, hemos realizado las siguientes simplificaciones:

- Se ignora el sistema de transmisión de mensajes, aunque en cualquier caso en la especificación se considera instantáneo y sin error.
- Sólo existe una bomba.
- El ritmo de producción de vapor es constante. Tal suposición puede ser relajada a un ritmo de variación lineal. Con un ritmo de variación lineal es posible obtener un modelo MUS-T del entorno con el cálculo del caso peor y el caso mejor en la variación lineal.

En la especificación se prescribe un ciclo de proceso de 5 segundos, durante el cual el proceso realiza tareas de naturaleza variada. Para la mejor comprensión de la metodología propuesta, se han definido un par de tareas básicas que el proceso debe acometer en cada ciclo.

12.2 Especificación informal

En esta sección se presenta la adaptación de la especificación informal que podemos encontrar en (Abrial et al., 1996). Se obvian todas las partes de la especificación referentes al sistema de transmisión de mensajes. Se supone, por tanto, que el intercambio de mensajes entre el proceso y el sistema físico es instantáneo y sin error. Además, se obvia la fase de inicio para el arranque de la caldera, nos interesa centrarnos en el comportamiento reactivo del sistema durante la fase de operación: entre la la fase de inicio y la parada.

El sistema físico se compone de una caldera con una bomba, un sensor de funcionamiento de la bomba, un sensor de medida de agua y un sensor de medida del flujo de vapor generado en la caldera. Al igual que en la especificación original se impone:

- Un intervalo en el nivel normal de agua $N1 < N2$.
- Una bomba que, cuando se encuentra abierta, introduce agua en la caldera a un ritmo constante P.
- La caldera produce vapor a un ritmo constante W.
- Dos límites mínimo y máximo en el nivel de agua $M1 < M2$. Si cualquiera de los límites se supera de forma continuada durante 5 segundos, existen riesgos de daños graves en la caldera.

El proceso sigue un ciclo de 5 segundos que consta de las siguientes tareas:

- Regulación de los sensores, tarea que tarda entre 1 y 3 unidades de tiempo.
- Generación de informes, siempre que sea posible se actualiza la generación de informes, tarea en la que se invierten 3 segundos.
- Cada cinco segundos el proceso, en función de los datos en los sensores, opera sobre los actuadores. Dado que en la especificación del caso de estudio no se prescribe la duración de estas tareas, se supone que son instantáneas.

Modo normal

El modo normal es el modo de operación estándar en el que el controlador intenta mantener el nivel de agua entre $N1$ y $N2$, con todos los sensores y actuadores funcionando de forma correcta. Tan pronto como el nivel de agua esté por debajo de $N1$ o por encima de $N2$, el proceso de control debe ajustar el nivel abriendo o cerrando la bombas. La decisión de basa en la información recibida de los sensores. Tan pronto como el proceso detecta un fallo en el dispositivo de medida de agua, el proceso pasa a modo rescate. Ante el fallo de cualquier otro dispositivo (bombas o sensor de flujo de vapor), el proceso pasa a modo degradado. Si el nivel de agua corre el riesgo de producir daños en la caldera ($M1$ o $M2$), el proceso entra en el modo parada de emergencia.

Modo degradado

En el modo degradado, el proceso intenta mantener un nivel satisfactorio de agua a pesar de la presencia de fallo en alguna unidad física (bombas o sensor

de flujo de vapor). Se supone, sin embargo, que la unidad de medida del nivel de agua en la caldera está trabajando de forma correcta. La funcionalidad es la misma que en el modo normal. Una vez las unidades con fallo han sido reparadas se vuelve al modo normal. Tan pronto como se detecte un fallo en la unidad de medida de agua, el proceso pasa a modo rescate. Si el nivel de agua corre el riesgo de producir daños en la caldera ($M1$ o $M2$), el proceso entra en el modo parada de emergencia.

Modo de rescate

En el modo rescate el proceso trata de mantener un nivel de agua satisfactorio a pesar del fallo de la unidad de medida de agua. En este modo el nivel de agua se basa en una estimación realizada a partir del sensor de flujo de vapor más el funcionamiento correcto de cada una de las bombas. Tan pronto como se detecte un fallo en la unidad de medida de vapor o en las bombas, el proceso entra en modo parada de emergencia. Además, tan pronto como se repare el sensor de nivel de agua, el proceso pasa a modo normal o a modo degradado.

Modo parada de emergencia

Este modo transfiere el control a los operadores humanos y el proceso acaba. Aunque no se detalla en la especificación recogida en (Abrial et al., 1996), prescribimos dos formas de alcanzar la parada de emergencia. Ante la detección del alcance de los límites $M1$ o $M2$:

- Si se detecta que se ha superado el límite en no más de 3 segundos, se produce una parada *stop*.
- Si se detecta que se ha superado el límite en más de 3 segundos, se produce una parada con disparo de alarma a la que denominaremos $stopAL$.

Si bien en un sistema físico con una sola bomba el modo degradado pierde su objetivo, mantenemos la estructura de la especificación original, lo que permite extender el caso de estudio, con la incorporación de datos, a entornos más generales.

12.3 Arquitectura

12.3.1 Arquitectura del sistema para modo normal

El sistema cerrado, con el proceso controlador funcionando en modo normal, consta de un componente MUS-T, $\mathcal{M}_E$, para el modelado del comportamiento del entorno, y un componente MUS-T, $\mathcal{M}_C^n$, que representa el diseño del controlador en modo normal. El sistema cerrado será, por tanto, la composición (sección 6.8) de ambos modelos sincronizados en un conjunto de acciones de sincronización:

$$\mathcal{M}_E|[A_S]|\mathcal{M}_C^n$$

El conjunto de acciones de sincronización, A_S, contendrá el conjunto de acciones que representan la lectura de valores de los sensores y el envío de órdenes a los actuadores en el sistema físico. Dicho conjunto dependerá de los modos de fallo considerados tanto en el entorno como en el controlador. De esta forma, identificamos dos escenarios: un escenario ideal sin fallos (A_S) y un escenario con fallos en el sensor de nivel de agua y en el funcionamiento correcto de la bomba (A_S^F). Dado que en el presente desarrollo sólo se incluye el modo de operación normal, no consideramos el sensor de flujo de vapor.

Entorno sin fallos

Se identifican el conjunto de acciones de sincronización

$$A_S = \{p_{on}, p_{off}, na, n1_{\downarrow}, n2_{\uparrow}, n_{\leftrightarrow}, m1_{\downarrow}, m2_{\uparrow}\}$$

representando:

- p_{on}, p_{off}, na: apertura, cierre y no acción sobre la bomba, respectivamente.
- $n_{\leftrightarrow}, m1_{\downarrow}, m2_{\uparrow}, n1_{\downarrow}, n2_{\uparrow}$: lecturas del sensor de nivel de agua. Siendo $n_{\leftrightarrow}$ una lectura $> N1$ y $< N2$, $m1_{\downarrow}$ una lectura $< M1$, $m2_{\uparrow}$ una lectura $> M2$, $n1_{\downarrow}$ una lectura $< N1$ y $n2_{\uparrow}$ una lectura $> N2$.

Entorno con fallos

Se identifican el conjunto de acciones de sincronización

$$A_S^F = A_S \cup \{errorB, errorNA\}$$

representando:

- $errorNA$, el evento que se produce cuando se detecta el funcionamiento incorrecto del sensor de nivel de agua.
- $errorB$, el evento que se produce cuando se detecta un error en el funcionamiento de la bomba.

12.3.2 Otros modos de operación

El diseño del controlador funcionando bajo otros modos de operación sería similar. Se propone una estructura que, en base a la especificación de los ordenamientos relativos de los distintos modos de operación, integre una especificación global con los siguientes componentes:

- Componente MUS-T del entorno ($\mathcal{M}_E$ o $\mathcal{M}_{EF}$).
- Componente MUS-T del funcionamiento del controlador bajo cada uno de los modos de operación ($\mathcal{M}_C^n$, $\mathcal{M}_C^d$, $\mathcal{M}_C^r$)
- Componente MUS-T de los cambios de modo de operación especificados, $\mathcal{M}_M$, representado en la figura 12.2.

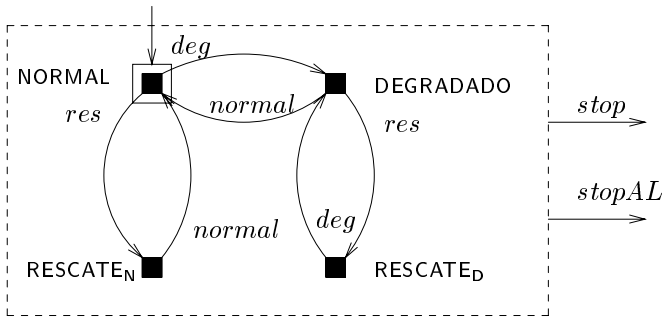


Figura 12.2. Modelo de modos de operación $\mathcal{M}_M$

Por brevedad en la presentación, sólo se incluye el diseño del controlador en modo normal ($\mathcal{M}_C^n$). No obstante, se propone un proceso de diseño global que, tomando como punto de partida el prototipo del controlador en modo normal, permite obtener los prototipos $\mathcal{M}_C^d$ y $\mathcal{M}_C^r$ por evolución del anterior. De esta forma, la especificación global del controlador vendría dada por:

$$\mathcal{M}_M|[A_S^M]|(M_C^n \parallel M_C^d \parallel M_C^r)$$

siendo $\parallel$ la composición con un conjunto de acciones de sincronización vacío, y A_S^M el conjunto de acciones de cambio de modo $\{normal, deg, res\}$.

12.4 Modelado del entorno: totalmente especificado

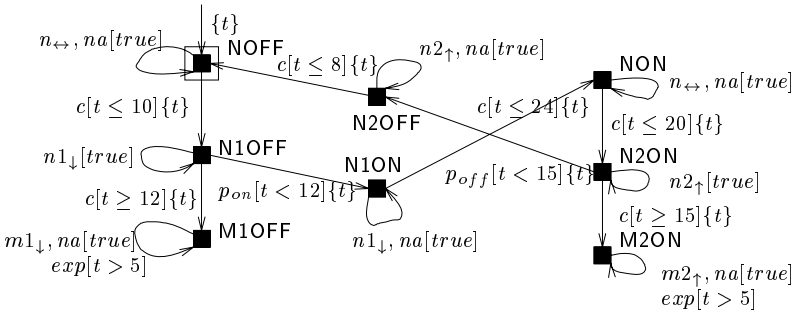
Se ha supuesto que el entorno es un modelo MUS-T totalmente especificado. Si bien tal suposición parece natural, en el modelado del entorno se podría seguir un proceso incremental que permitiese decidir, por ejemplo, qué simplificaciones son realmente adecuadas mediante la validación basada en el grafo de simulación posible del modelo $\mathcal{M}_E$. En cualquier caso, la ventaja principal de un modelo del entorno sin subespecificación es que permite la síntesis de propiedades del sistema cerrado, dado que las operaciones de síntesis realizadas siempre se traducirán en modificaciones del componente no totalmente especificado.

En las figuras 12.3 y 12.4, representando los modelos MUS-T del entorno y del entorno con fallos, sólo se muestran las acciones temporizadas e invariantes con condición de especificación posible. Los contextos temporales para evolución discreta y temporal no mostrados se suponen de condición de especificación no posible.

12.4.1 Modelado del entorno sin fallos

El modelado del entorno se basa en los valores que definen el sistema físico ($\{P, W, M1, M2, N1, N2\}$), es decir, el flujo de agua procedente de la bomba, el ritmo de producción de vapor y los límites en el nivel de agua. Para los valores

ejemplo seleccionados, el modelo, $\mathcal{M}^E$, del entorno sin fallos se muestra en la figura 12.3.



I_a(NOFF) : $t \leq 10$	I_a(NON) : $t \leq 20$
I_a(N1OFF) : $t \leq 12$	I_a(N2ON) : $t \leq 15$
I_a(M1OFF) : $true$	I_a(M2ON) : $true$
I_a(N1ON) : $t \leq 24$	I_a(N2OFF) : $t \leq 8$

Figura 12.3. Modelo del comportamiento de la caldera (sin fallos) $\mathcal{M}^E$

En el modelado del entorno, además de las acciones de sincronización A_S, se identifican las siguientes acciones:

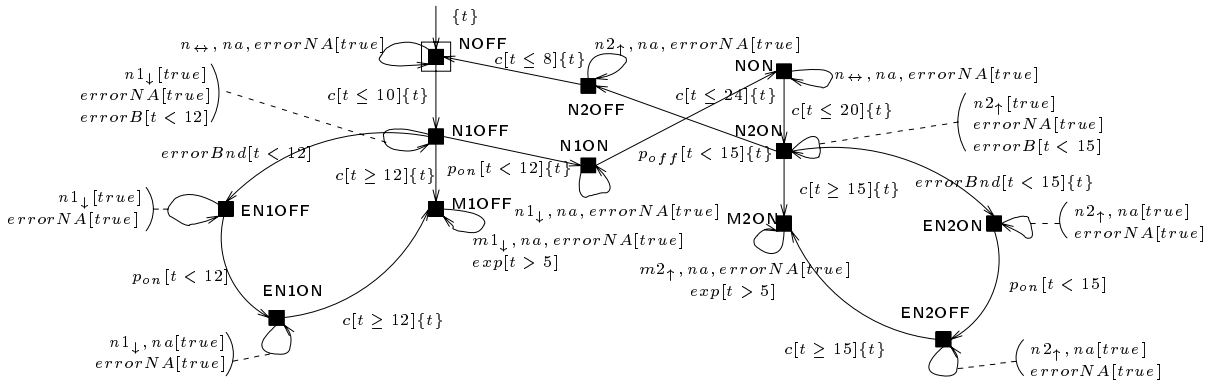
- *exp*, explosión o daño grave en el sistema físico.
- *c*, acción de detección de cambio en los umbrales del nivel de agua.

12.4.2 Modelado del entorno con fallos

En el modelado del entorno con fallos se contemplan los casos en que se produzca un fallo en el sensor de medida del nivel de agua y en el funcionamiento de la bomba:

- El sensor de medida del nivel de agua puede fallar, lo que implica que, en vez de un valor de lectura correcto, proporcione una lectura $errorNA$.
- Para el caso del fallo de la bomba, realizamos la siguiente simplificación: la bomba sólo puede fallar en el momento que se abre o se cierra. Además, el error en la apertura o cierre de la bomba puede ser detectado (mediante el evento $errorB$) o no detectado (mediante un evento interno $errorBnd$). El objetivo será intentar diseñar un controlador que proteja la bomba frente a un error de apertura o cierre no detectado.

Además de las acciones de sincronización, A_S^F, y de las acciones internas en $\mathcal{M}^E$, se identifica la acción interna $errorBnd$ que permite modelar el mal funcionamiento de la bomba en un error no detectado. El modelo del entorno con fallos, $\mathcal{M}^{EF}$, se muestra en la figura 12.4.



I_a(NOFF) : $t \leq 10$	I_a(NON) : $t \leq 20$
I_a(N1OFF) : $t \leq 12$	I_a(N2ON) : $t \leq 15$
I_a(M1OFF) : $true$	I_a(M2ON) : $true$
I_a(N1ON) : $t \leq 24$	I_a(N2OFF) : $t \leq 8$
I_a(EN1OFF) : $t \leq 12$	I_a(EN2ON) : $t \leq 15$
I_a(EN1ON) : $t \leq 12$	I_a(EN2OFF) : $t \leq 15$

Figura 12.4. Modelado del comportamiento de la caldera (con fallos) $\mathcal{M}^{EF}$

12.4.3 Consideraciones sobre el modelo

Por simplicidad se ha optado por un modelado del entorno a un elevado nivel de abstracción. En los modelos propuestos debemos recalcar los siguientes puntos:

- Se han eliminado distintos comportamientos posibles en el entorno, por ejemplo, la apertura (cierre) de la bomba cuando el nivel de agua se encuentra en los límites normales. Dado que el secuenciamiento de la lectura de sensores y la actuación es inmediato, serían comportamientos nunca alcanzados en la ejecución del sistema cerrado.

- Igualmente se ha eliminado el comportamiento del entorno ante eventos p_{on} y p_{off}, una vez alcanzado **M2ON** o **M1OFF**. Se supone, por tanto, que en base a un ciclo de actuación de 5 segundos y al ritmo de producción de agua y de vapor, cualquier actuación llegará demasiado tarde para evitar la parada.

Estas consideraciones nos permiten aportar un modelo simplificado bajo la suposición de un ciclo de operación 5 segundos. Sin embargo, esta suposición se aparta de la flexibilidad obtenida mediante el modelado denso del tiempo. Un modelo sin dicha suposición puede ser obtenido del anterior por evolución (considerándolo no totalmente especificado).

12.4.4 Propiedades del entorno

Si bien, por razones de brevedad, no se ha detallado la síntesis del modelo del entorno, a continuación, se presentan algunas de sus propiedades generales. Asimismo, se presenta la propiedad básica que debe exhibir el entorno para que el controlador sea "realizable".

Propiedades generales del entorno

Comportamiento cualitativo:

- $RE1 := m1_{\downarrow} \Rightarrow \mathcal{AP}[n1_{\downarrow}]$, sólo se puede llegar al nivel $M1$ si antes se pasó el límite $N1$.
- $RE2 := m2_{\uparrow} \Rightarrow \mathcal{AP}[n2_{\uparrow}]$, sólo se puede llegar al nivel $M2$ si antes se pasó el límite $N2$.
- $RE3 := (\emptyset \ \exists\Rightarrow\odot \ \mathcal{E}[\neg p_{off} \ \mathcal{S} \ p_{on}]) \Rightarrow \neg p_{on}$, sólo se puede abrir la bomba si nunca fue abierta o si antes fue cerrada.
- $RE4 := (\emptyset \ \exists\Rightarrow\odot \ \mathcal{E}[\neg p_{on} \ S_w \ p_{off}]) \Rightarrow \neg p_{off}$, sólo se puede cerrar la bomba si antes fue abierta.

Comportamiento deducido de la física del entorno:

- $RE5 := x.m1_{\downarrow} \ \forall\Rightarrow_{+} \ ((m1_{\downarrow} \wedge x \geq 5) \Rightarrow exp)$, si se ha producido una lectura de sensor $M1$ durante 5 segundos, se produce un daño grave exp.
- $RE6 := x.m2_{\uparrow} \ \forall\Rightarrow_{+} \ ((m2_{\uparrow} \wedge x \geq 5) \Rightarrow exp)$, si se ha producido una lectura de sensor $M2$ durante 5 segundos, se produce un daño grave exp.
- $RE7 := x.(n_{\leftrightarrow} \wedge \mathcal{A}[\neg p_{on} \ \mathcal{S}_w \ p_{off}]) \Rightarrow \mathcal{AF}[n1_{\downarrow} \wedge x \leq 10]$, en el intervalo normal, cuando la bomba está cerrada se tardan no más de 10 segundos en alcanzar el nivel $N1$.
- $RE8 := x.(n_{\leftrightarrow} \wedge \mathcal{A}[\neg p_{off} \ \mathcal{S} \ p_{on}]) \Rightarrow \mathcal{AF}[n2_{\uparrow} \wedge x \leq 20]$, en el intervalo normal, cuando la bomba está abierta se tardan no más de 20 segundos en alcanzar el nivel $N2$.
- $RE9 := (n1_{\downarrow} \wedge p_{on}) \Rightarrow (x.p_{on} \ \forall\Rightarrow\bigcirc \ \mathcal{AF}[n_{\leftrightarrow} \wedge x \leq 24])$, una vez abierta la bomba en el intervalo $N1 - M1$, no tarda más de 24 segundos en alcanzar el nivel normal.
- $RE10 := (n2_{\uparrow} \wedge p_{off}) \Rightarrow (x.p_{off} \ \forall\Rightarrow\bigcirc \ \mathcal{AF}[n_{\leftrightarrow} \wedge x \leq 8])$, una vez cerrada la bomba en el intervalo $N2 - M2$, no tarda más de 8 segundos en alcanzar el nivel normal.
- La especificación del resto de comportamientos sería similar.

Comportamientos adicionales para el entorno con modos de fallo:

- $RE11 := n1_{\downarrow} \vee n2_{\uparrow} \vee m1_{\downarrow} \vee m2_{\uparrow} \vee n_{\leftrightarrow} \Rightarrow errorNA$, el sensor puede proporcionar el nivel correcto o bien un evento de error $errorNA$.

- $RE12 := p_{on} \vee p_{off} \Rightarrow errorB$, sólo cuando se actúa sobre la bomba se puede producir error en su funcionamiento.
- En un entorno con fallos, las propiedades $RE9$ y $RE10$ se modifican por:
  - $RE9 := (n1_{\downarrow} \wedge p_{on} \wedge errorB) \Rightarrow (x.p_{on} \Rightarrow \mathcal{AF}[n_{\leftrightarrow} \wedge t \leq 24])$.
  - $RE10 := (n2_{\uparrow} \wedge p_{off} \wedge errorB) \Rightarrow (x.p_{off} \Rightarrow \mathcal{AF}[n_{\leftrightarrow} \wedge t \leq 8])$.

Las propiedades son verificadas en los modelos $\mathcal{M}_E$ o $\mathcal{M}_{EF}$, resultando requisitos que o bien se satisfacen (1) o son no aplicables ($\widehat{\frac{1}{2}}$), consecuencia de considerar el entorno totalmente especificado. Para el cálculo del grado de satisfacción se introduce equidad en el algoritmo de *model checking* en forma de *weak transition fairness*.

El modelo no contiene estados *deadlock*, y aunque no es *strongly nonzeno*, tampoco contiene estados *timelock*.

Propiedad básica del entorno

La propiedad básica que se requiere del entorno es que permita el diseño de un controlador. Por tanto ha de existir, por lo menos, un camino no trivial en el que el entorno no se vea dañado. Como se sigue de los grafos de simulación del entorno (sin fallos) y del entorno (con fallos), el sistema exhibe esta propiedad informal. La especificación formal de esta propiedad básica sería:

$$\mathcal{M}_E \models \mathcal{EG}^{\mathcal{U}}[\neg exp]$$
$$\mathcal{M}_{EF} \models \mathcal{EG}^{\mathcal{U}}[\neg exp]$$

12.5 Propiedades del sistema cerrado

12.5.1 Entorno si fallos

En un entorno sin fallos, el sistema cerrado debe cumplir, en todos sus estados, las propiedades de seguridad y rendimiento a continuación especificadas:

$$RSCS^E := \emptyset \Rightarrow \neg exp$$
$$RSCR^E := \emptyset \Rightarrow \mathcal{AF}[n_{\leftrightarrow}]$$

12.5.2 Entorno con fallos

En un entorno con fallos, el sistema cerrado debe cumplir, en todos sus estados, la propiedades de seguridad y rendimiento siguientes:

$$RSCS^{EF} := \emptyset \Rightarrow \neg exp$$
$$RSCR^{EF} := \emptyset \Rightarrow \mathcal{A}[\mathcal{EF}[n_{\leftrightarrow}]\ \mathcal{U}_w(errorB \vee errorNA)]$$

12.6 Especificación inicial del controlador (sin fallos)

Identificación de acciones en el controlador

Además del conjunto de acciones de sincronización A_S^E, se identifican:

- rs, fr, respectivamente, inicio y terminación de la tarea de regulación de sensores.
- gi, fi, respectivamente, inicio y terminación de la tarea de generación de informes.
- act, entrada en la fase de actuación.
- $stopAL$, parada con alarma.
- $stop$, parada.

Propiedades del controlador

- **Actuaciones correctas**: sólo se puede abrir la bomba si está abierta; sólo se puede cerrar la bomba si está cerrada.

$$RC1 := p_{on} \;\; \forall\Rightarrow\bigcirc \; (\emptyset \Rightarrow \mathcal{A}[\neg p_{on} \; \mathcal{U}_w \; p_{off}])$$
$$RC2 := p_{off} \;\; \forall\Rightarrow\bigcirc \; (\emptyset \Rightarrow \mathcal{A}[\neg p_{off} \; \mathcal{U}_w \; p_{on}])$$

- **Finalización de tareas**: Una vez comenzada la regulación de sensores siempre finaliza en no menos de 1 segundo y no más de 3 segundos. Además, una vez comenzada la generación de informes siempre finaliza en 3 segundos.

$$RC3 := x.rs \;\; \forall\Rightarrow\bigcirc \; (\emptyset \Rightarrow \mathcal{AF}[fr \wedge x \leq 3 \wedge x \geq 1])$$
$$RC4 := x.gi \;\; \forall\Rightarrow\bigcirc \; (\emptyset \Rightarrow \mathcal{AF}[fi \wedge x = 3])$$

- **Ciclo de actuación**: Las actuaciones se realizan en un ciclo de 5 segundos. Además, si no han transcurrido 5 segundos desde la actuación anterior, no es posible la actuación.

$$RC5 := x.act \Rightarrow \mathcal{AF}[x = 5 \Rightarrow act]$$
$$RC6 := x.act \Rightarrow \mathcal{EF}[act \wedge x = 5]$$
$$RC7 := x.act \;\; \forall\Rightarrow\bigcirc \;\; \mathcal{AG}[(x < 5 \Rightarrow \neg act) \vee (x \geq 5)]$$

- **Secuenciamiento de ciclo**: Una vez leídos los valores de los sensores, si el sistema no ha alcanzado los límites $M1$ o $M2$, se inicia un nuevo ciclo

que siempre incluirá la tarea de regulación de sensores y, eventualmente, la generación de informes.

$$RC8 := (n1_{\downarrow} \vee n2_{\uparrow} \vee n_{\leftrightarrow}) \;\; \forall\Rightarrow\bigcirc \;\; (\emptyset \Rightarrow \mathcal{AF}[fr \Rightarrow \mathcal{AF}[act]])$$
$$RC9 := (n1_{\downarrow} \vee n2_{\uparrow} \vee n_{\leftrightarrow}) \;\; \forall\Rightarrow\bigcirc \;\; (\emptyset \Rightarrow \mathcal{EF}[fr \;\; \forall\Rightarrow\bigcirc \;\; (gi \Rightarrow \mathcal{AF}[act])])$$

- **Parada de emergencia**: Si, una vez alcanzados los límites $M1$ y $M2$, la última lectura de sensores $> M1$ y $< M2$ se produjo en no más de 3 segundos desde el instante actual, se produce una parada *stop*. Si por contra dicha lectura corresponde a una distancia temporal mayor de 3 segundos, se produce una parada $stopAL$.

$$\begin{aligned} RC10 := \;\; & (m1_{\downarrow} \vee m2_{\uparrow}) \wedge \\ & (x.\emptyset \Rightarrow \mathcal{EP}[x \leq 3 \wedge (n_{\leftrightarrow} \vee n1_{\downarrow} \vee n2_{\uparrow})]) \;\; \forall\Rightarrow\bigcirc \;\; stop \\ RC11 := \;\; & (m1_{\downarrow} \vee m2_{\uparrow}) \wedge \\ & (x.\emptyset \Rightarrow \mathcal{AP}[x > 3 \vee (\neg n_{\leftrightarrow} \wedge \neg n1_{\downarrow} \wedge \neg n2_{\uparrow})]) \;\; \forall\Rightarrow\bigcirc \;\; stopAL \end{aligned}$$

Escenarios identificados en un ciclo del controlador

El procedimiento de síntesis propuesto se basa en la identificación de escenarios de comportamiento conocidos, expresados mediante reglas de síntesis SCTL-T:

- **Inicio**: Al inicio el proceso empieza la regulación de sensores inmediatamente, siendo la única tarea a realizar en ese momento:

$$RSINI_0 := ini \;\; \forall\overset{\forall\rightarrow}{\Rightarrow}\bigcirc \;\; rs \wedge \neg(fi \vee gi \vee fr)$$
$$RSINI_1 := ini \Rrightarrow (\emptyset \;\; \forall\Rightarrow_{+} \;\; false)$$
$$RSINI := RSINI_0 \wedge RSINI_1$$

- **Fase de regulación**: La regulación de sensores dura entre 1 y 3 unidades de tiempo. Una vez acabada se espera a la fase de actuación:

$$RSREG_0 := rs\{x_s\} \;\; \forall\overset{\forall\rightarrow}{\Rightarrow}\bigcirc \;\; ((x_s \leq 3 \Rightarrow true) \wedge (x_s > 3 \Rightarrow false))$$
$$RSREG_1 := rs\{x_s\} \;\; \forall\overset{\forall\rightarrow}{\Rightarrow}\bigcirc \;\; ((x_s \geq 1 \wedge x_s \leq 3) \Rightarrow fr)$$
$$RSREG := RSREG_0 \wedge RSREG_1$$

- **Entrada en la fase de actuación**: La fase de actuación comienza (*act*) 5 segundos después, o bien desde el inicio de la regulación de sensores, o bien

desde el comienzo de la anterior.

$$
\begin{aligned}
&RSEA_0 := x.\emptyset \Rightarrow \mathcal{EP}[rs \wedge x = 5] \Rrightarrow act \\
&RSEA_1 := x.\emptyset \Rightarrow \mathcal{EP}[rs \wedge x < 5] \Rrightarrow \neg act \\
&RSEA_2 := (x.\emptyset \ \ \exists\Rightarrow\odot \ (\emptyset \Rightarrow \mathcal{EP}[act \wedge x < 5])) \Rrightarrow \neg act \\
&RSEA_3 := x.\emptyset \Rightarrow \mathcal{AP}[act \wedge x = 5] \Rrightarrow act \\
&RSEA := RSEA_0 \wedge RSEA_1 \wedge RSEA_2 \wedge RSEA_3
\end{aligned}
$$

- **Actuación**: En la fase de actuación el proceso comprueba los valores de los sensores ($n1_{\downarrow}$, $n2_{\uparrow}$, $m1_{\downarrow}$, $m2_{\uparrow}$, $n_{\leftrightarrow}$), siendo dicha comprobación instantánea:

$$
\begin{aligned}
&RSACT_0 := act \ \forall\Rrightarrow\bigcirc \ (\emptyset \ \ \forall\Rightarrow_{+} \ \ false) \\
&RSACT_1 := act \ \forall\Rrightarrow\bigcirc \ n1_{\downarrow} \wedge n2_{\uparrow} \wedge m1_{\downarrow} \wedge m2_{\uparrow} \wedge n_{\leftrightarrow} \\
&RSACT_2 := (\emptyset \ \ \forall\Rightarrow\odot \ \ \neg act) \Rrightarrow \neg(n1_{\downarrow} \vee n2_{\uparrow} \vee m1_{\downarrow} \vee m2_{\uparrow} \vee n_{\leftrightarrow}) \\
&RSACT := RSACT_0 \wedge RSACT_1 \wedge RSACT_2
\end{aligned}
$$

En función de la medida del sensor de agua obtenida, procede abriendo o cerrando la bomba p_{on}, p_{off} de forma instantánea:

$$
\begin{aligned}
&RSTACT_0 :: (n1_{\downarrow} \wedge n2_{\uparrow} \wedge m1_{\downarrow} \wedge m2_{\uparrow} \wedge n_{\leftrightarrow}) \Rrightarrow \neg p_{on} \wedge \neg p_{off} \\
&RSTACT_1 := n1_{\downarrow} \ \forall\Rrightarrow\bigcirc \ (na \wedge p_{on} \wedge \neg p_{off}) \\
&RSTACT_2 := n2_{\uparrow} \ \forall\Rrightarrow\bigcirc \ (na \wedge p_{off} \wedge \neg p_{on}) \\
&RSTACT_3 := n_{\leftrightarrow} \ \forall\Rrightarrow\bigcirc \ na \wedge \neg p_{on} \wedge \neg p_{off} \\
&RSTACT_4 := \neg(n1_{\downarrow} \ \wedge \ n2_{\uparrow}) \ \forall\Rrightarrow\overset{\forall\rightarrow}{\bigcirc} \ \neg p_{on} \wedge \neg p_{off} \\
&RSTACT_5 := (\emptyset \ \ \exists\Rightarrow\odot \ \ act) \ \forall\Rrightarrow\bigcirc \ (\neg act \wedge (\emptyset \ \ \forall\Rightarrow_{+} \ \ false)) \\
&RSTACT := RSTACT_0 \wedge RSTACT_1 \wedge RSTACT_2 \wedge RSTACT_3 \\
&\qquad\qquad \wedge RSTACT_4 \wedge RSTACT_5
\end{aligned}
$$

- **Fin de actuación y comienzo de un nuevo ciclo**: Una vez realizadas las actuaciones, y siempre que no se haya detectado los valores límite del nivel de agua, se comienza un nuevo ciclo:

$$RSNC := p_{on} \vee p_{off} \vee na \ \forall\Rrightarrow\bigcirc \ rs$$

Escenarios de generación de informes (comportamiento opcional)

La generación de informes, si se produce, es después de acabar la regulación y tarda en acabar 3 unidades de tiempo.

$$RSGI_0 := (\emptyset \ \exists\Rightarrow\odot \ fr) \vee gi\{x_i\} \ \forall\overset{\forall\rightarrow}{\Rrightarrow}\bigcirc \ (x_i = 3 \Rightarrow fi)$$

$$RSGI_1 := (\emptyset \ \exists\Rightarrow\odot \ fr) \vee gi\} \ \forall\overset{\forall\rightarrow}{\Rrightarrow}\bigcirc \ (x_i \leq 3 \Rightarrow true) \wedge (x_i > 3 \Rightarrow false)$$

$$RSGI := RSGI_0 \wedge RSGI_1$$

12.7 Síntesis del controlador I (sin fallos)

En las siguientes secciones, las figuras de los modelos sintetizados sólo muestran las acciones parcialmente subespecificadas. La acciones temporizadas no representadas serán transiciones totalmente subespecificadas con destino en el estado sumidero s_s.

Se comienza por aplicar las reglas de inicio y de fase de regulación $\{RSINI, RSREG\}$ sintetizando de esta forma el contexto inicial y la fase de regulación. El esquema de síntesis se muestra en la figura 12.5.

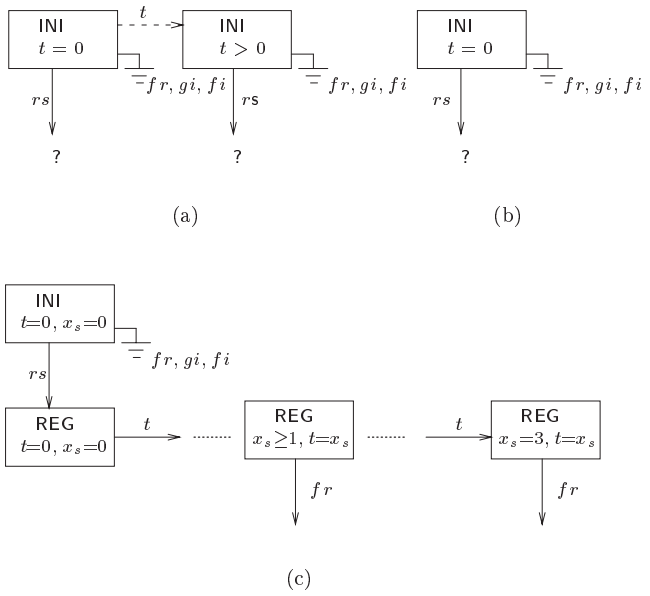


Figura 12.5. Esquema del procedimiento de síntesis para la regla $RINI$ (figuras 12.5(a) y 12.5(b)) y la regla $RSREG$ (figura 12.5(c))

Obtenemos de esta forma el grafo de la figura 12.6. A continuación se procede a identificar el contexto de entrada en la fase de actuación, las reglas en $RSEA$ no son aplicables en ningún contexto temporal. La regla $RSEA_0$, determinando el contexto de la primera actuación, es tal que su parte declarativa:

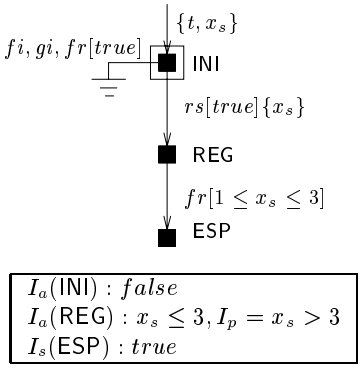


Figura 12.6. Controlador sintetizado mediantes las reglas de síntesis $\{RSINI, RSREG\}$.

- $GS(x.\emptyset \Rightarrow \mathcal{EP}[rs \wedge x = 5], \mathcal{R}) = 0$, para todos los contextos temporales en los estados de control INI y REG, indicando que la regla no será aplicable en ninguna evolución posterior bajo un criterio RCT.

- $GS(x.\emptyset \Rightarrow \mathcal{EP}[rs \wedge x = 5], \mathcal{R}) \in \{0, \frac{1}{2}\}$ en los contextos temporales del estado de control ESP. El valor 0 corresponde a las regiones $<$ ESP$, (1 \leq x_s = t \leq 3), 1 >$ alcanzables mediante la transición discreta posible fr.

Inspeccionamos el grafo de simulación posible del prototipo actual, que se muestra en la figura 12.7. Comprobamos que una distancia temporal en el pasado $x = 5$ sólo puede ser alcanzada en partes todavía subespecificadas del modelo, ya sea mediante transiciones discretas (representadas por d), todavía subespecificadas en los estados INI o REG, o transiciones temporales en REG. Dada la propiedad de finalización de la tarea de regulación $RC3$, el cambio que más concuerda con las expectativas será la habilitación de pasos temporales en el estado de control nombrado como SUB.

Con el objetivo de permitir la aplicabilidad de la regla de entrada en la fase de actuación, incorporamos la propiedad:

$$RSREG_2 := fr \ \forall\overset{\forall\rightarrow}{\Rightarrow}\bigcirc \ true$$
$$RSREG := RSREG_0 \wedge RSREG_1 \wedge RSREG_2$$

en el conjunto de reglas de síntesis de la fase de regulación, permitiendo de esta forma el avance del tiempo en el estado nombrado como ESP. La regla de entrada en la fase de actuación pasa a ser aplicable.

Aplicando la regla $RSEA_0$ obtenemos el modelo MUS-T de la figura 12.8(a), y mediante la regla $RSEA_1$ el de la figura 12.8(b). Adicionalmente, las reglas $RSEA_2$ y $RSEA_3$ serían siempre aplicables en el estado ESP, produciendo eventos act para $t = 10, t = 15 \cdots$. Dado que no es ese el comportamiento esperado del sistema, se decide no aplicar dicha regla.

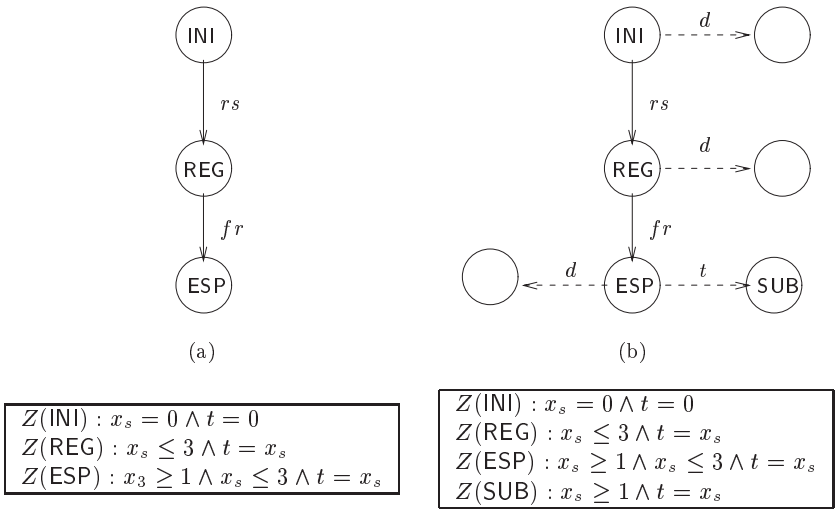


Figura 12.7. Grafo de simulación posible (subfigura 12.7(a)) para el modelo de la figura 12.6, e inspección de las partes subespecificadas (subfigura 12.7(b))

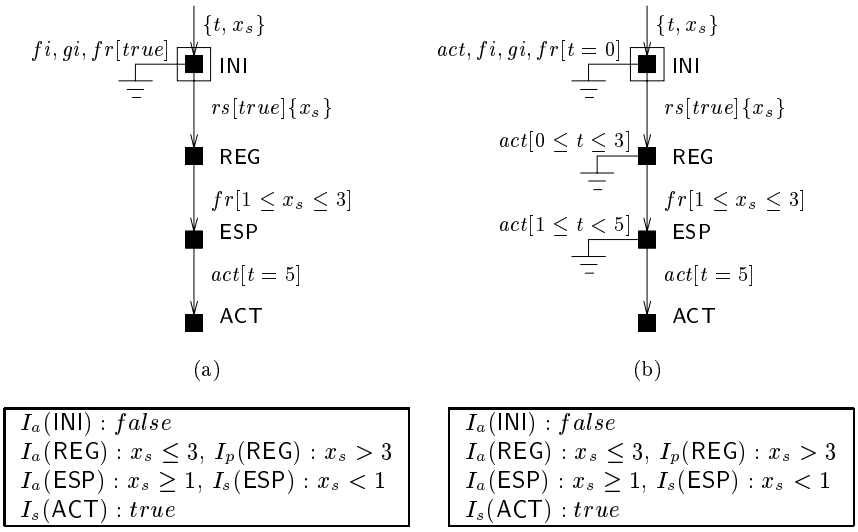


Figura 12.8. Controlador sintetizado mediante la regla $RSEA_0$ (12.8(a)), y mediante las reglas $RSEA_1$ (12.8(b)). La regla $RSEA_2$ no es aplicable.

A continuación, se procede a la síntesis de los escenarios en la fase de actuación $\{RSACT, RSTACT\}$. El modelo sintetizado mediante la aplicación de las reglas $RSACT$ se muestra en la figura 12.9. La síntesis del evento act como posible, hace que la regla de síntesis $RSEA_2$ sea aplicable, sintetizando $\neg act$ en todos los estados sucesores discretos del estado ACT. Sin embargo, en dichos estados sucesores también es aplicable la regla $RSEA_0$ produciendo conflicto con la regla $RSEA_2$. El motivo del conflicto se debe a que la regla $RSEA_0$ permite acciones act consecutivas con una distancia temporal 0, siempre que exista una distancia temporal 5 respecto a rs. Refinamos, por tanto, la propiedad:

$$RSEA_0 := (x.\emptyset \Rightarrow \mathcal{EP}[rs \wedge x = 5]) \wedge \mathcal{A}[\neg act\ \mathcal{S}\ rs] \Rrightarrow act$$

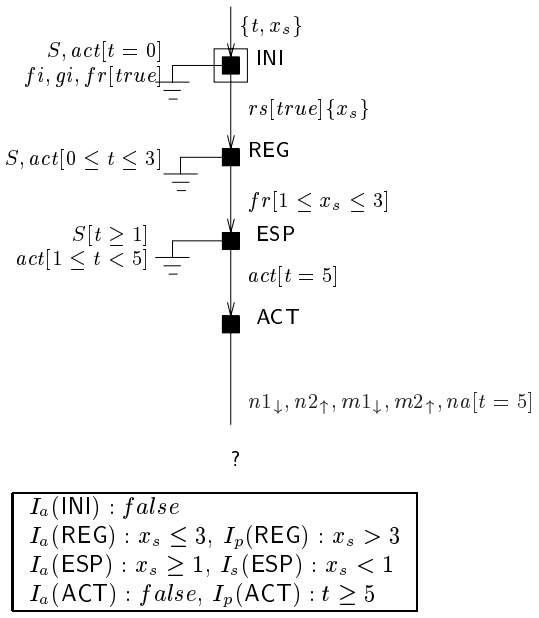
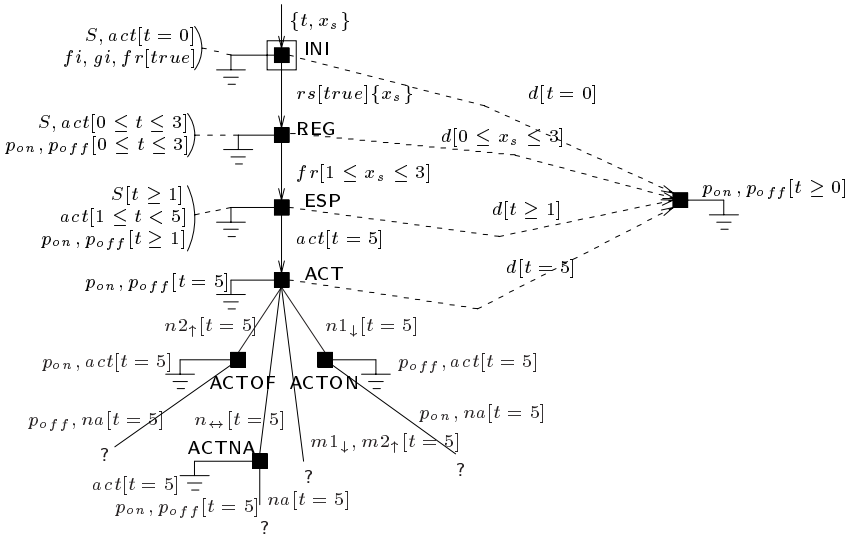


Figura 12.9. Controlador sintetizado mediante las reglas $\{RSACT\}$. S representa el conjunto $\{n1_\downarrow, n2_\uparrow, m1_\downarrow, m2_\uparrow, n_\leftrightarrow\}$

Mediante la aplicación de las reglas $RSTACT$ obtenemos el modelo en la figura 12.10, en el que se aplica un criterio RCM para la regla $RSTACT_4$; las reglas $\{RSTACT_1, RSTACT_2, RSTACT_3\}$ no encuentran, bajo ningún criterio, una región compatible en el prototipo actual.

Hay que notar que en la síntesis se realizan las suposiciones mínimas. Por ejemplo, en el estado ACT la ejecución del modelo sería la misma bajo una acción temporizada $n1_\downarrow[true]$, dado que la invariante de avance es $false$. Sin embargo la síntesis $n1_\downarrow[t = 5]$ es la mínima que satisface la regla de síntesis aplicada. En cuanto a los relojes, y dado que el modelo es sencillo, podemos, sólo visualmente, afirmar que ninguno de los relojes puede ser reducido, dado que: no son iguales, y además los dos están activos al mismo tiempo. Debemos notar que el algoritmo de reducción de relojes utilizado realiza un cómputo sintáctico sobre el grafo MUS-T;



I_a(INI) : $false$	I_a(ACTON) : $false$, I_p(ACTON) : $t \geq 5$
I_a(REG) : $x_s \leq 3$, I_p(REG) : $x_s > 3$	I_a(ACTOFF) : $false$, I_p(ACTOFF) : $t \geq 5$
I_a(ESP) : $x_s \geq 1$, I_s(ESP) : $x_s < 1$	I_a(ACTNA) : $false$, I_p(ACTNA) : $t \geq 5$
I_a(ACT) : $false$, I_p(ACT) : $t \geq 5$	

Figura 12.10. Controlador sintetizado mediante las reglas $\{RSTACT\}$. S representa el conjunto $\{n1_{\downarrow}, n2_{\uparrow}, m1_{\downarrow}, m2_{\uparrow}, n_{\leftrightarrow}\}$

por tanto, aunque el algoritmo de detección de relojes iguales arroja un resultado negativo, en la ejecución del modelo y dado que la invariante de avance del estado INI es $false$, los relojes t y x_s serán iguales.

Verificación de propiedades del controlador

La verificación de las propiedades generales del controlador proporciona los siguientes resultados:

- **Actuaciones correctas**: La propiedad $RC1$ sólo es aplicable en el estado de control $ACTON$, proporcionando el valor $\frac{1}{2}$, el sistema puede llegar a verificar o no la propiedad. De forma idéntica para la regla $RC2$ y el estado $ACTOFF$.

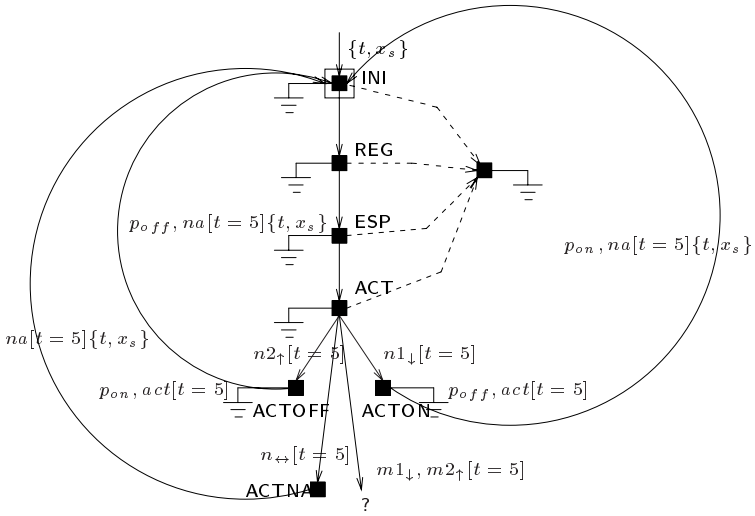
- **Finalización de tareas**: Las propiedad $RC3$ proporciona los valores $\{\widehat{\frac{1}{2}}, \frac{1}{2}\}$ para todos los contextos temporales del modelo, por lo que no se incumple en el prototipo actual. Es decir, siempre que se empieza la fase de regulación, no existen contextos temporales en los que no se acaba ($\neg fr$) en el intervalo especificado. Sin embargo, un modelo totalmente subespecificado arrojaría igualmente un valor $\frac{1}{2}$. Dado que se trata de una propiedad cuantificada universalmente, podemos concluir que la finalización de tareas, en cada uno de las posibles ejecuciones, proporciona un grado de satisfacción $\{\widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$. Refinando la propiedad, podemos decir:

  - $GS(x.rs \;\; \forall\Rightarrow\bigcirc \; (\emptyset \Rightarrow \mathcal{EF}[fr \wedge x \leq 3 \wedge x \geq 1]), \mathcal{R})$ es 1 en el estado INI. Es decir, cuando la regulación empieza existe un camino totalmente especificado que cumple la especificación.
  - El modelo completo mínimo con pérdida de subespecificación proporciona los resultados $\{\widehat{\frac{1}{2}}, 1\}$ para la propiedad $RC3$, es decir el modelo obtenido satisface la propiedad siempre que sea aplicable.

  La propiedad $RC4$ proporciona un valor $\frac{1}{2}$ para todos los contextos temporales, podrá llegar a cumplirse o incumplirse en todos ellos. Además, su valor es siempre $\widehat{\frac{1}{2}}$ para el modelo mínimo obtenido por pérdida de subespecificación, ya que en todos los contextos temporales del prototipo actual la acción gi se encuentra subespecificada. Dado que la generación de informes es un comportamiento opcional, estos resultados no imponen síntesis adicionales.

- **Ciclo de actuación**: Seleccionamos la propiedad $RC6$ como el requisito menos exigente en el conjunto de propiedades bajo el nombre ciclo de actuación. En el prototipo actual, $GS(RC6, \mathcal{R}) \in \{\widehat{\frac{1}{2}}, \frac{1}{2}\}$, es decir, si bien la propiedad no se incumple en el ciclo actual, no nos hemos asegurado de que haya, por lo menos, un ciclo de actuaciones de 5 segundos.

En consecuencia, se opta por sintetizar el comienzo de un nuevo ciclo mediante la aplicación de la regla de síntesis $RSNC$, obteniéndose el modelo en la figura 12.11, para un criterio RCMReset.



I_a(INI) : $false$	I_a(ACTON) : $false$, I_p(ACTON) : $t \geq 5$
I_a(REG) : $x_s \leq 3$, I_p(REG) : $x_s > 3$	I_a(ACTOFF) : $false$, I_p(ACTOFF) : $t \geq 5$
I_a(ESP) : $x_s \geq 1$, I_s(ESP) : $x_s < 1$	I_a(ACTNA) : $false$, I_p(ACTNA) : $t \geq 5$
I_a(ACT) : $false$, I_p(ACT) : $t \geq 5$	

Figura 12.11. Modelo obtenido de la síntesis de la regla $RSNC$ con criterio RCMReset

12.7.1 Corrección mínima del modelo

Para la garantía de una corrección mínima del modelo se utiliza el grafo de simulación posible para MUS-T, en busca de una forma de validar el prototipo obtenido y detectar bloqueos reales o potenciales.

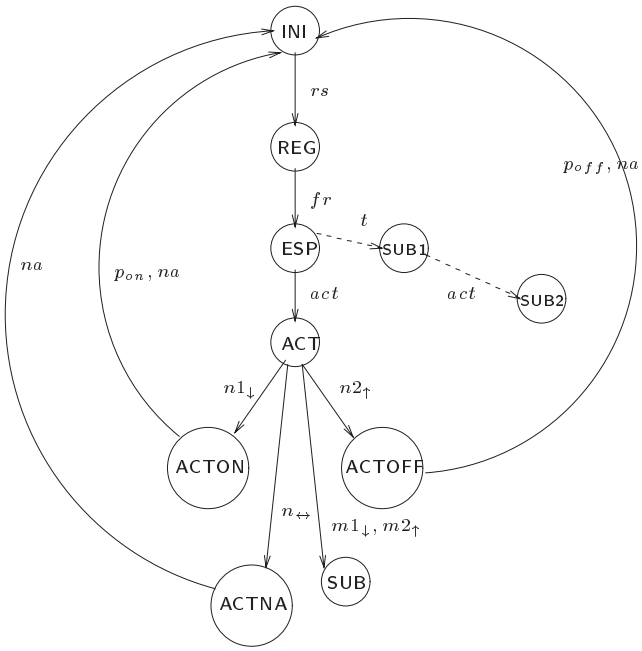
El grafo de simulación posible se presenta en la figura 12.12. La ejecución simbólica del modelo responde a las expectativas del diseñador, sin embargo:

- Hay estados de *deadlock-sub* en el estado ESP
- Hay estados de *timelock-sub*, *deadlock-sub* en el estado SUB.

Si bien el segundo de los errores potenciales era de esperar, dado que el estado SUB es un estado de no evolución posible, el *deadlock* potencial en el estado ESP debe ser subsanado en fases posteriores. Para ello, o bien se debe acotar la invariante de avance, o bien se debe permitir al modelo evolucionar de forma discreta para valores $t > 5$.

Adicionalmente, se verifican las propiedades generales del controlador (RC) en el prototipo obtenido, arrojando los siguientes resultados:

- **Actuaciones correctas**: Las propiedades $\{RC_1, RC_2\}$ se incumplen en contextos temporales del modelo. En la fase de diseño anterior habíamos obtenido un valor $\in \{\widehat{\frac{1}{2}}, \frac{1}{2}\}$ para esta propiedad, por tanto era posible obtener un modelo de evolución posterior cumpliendo la propiedad. El incumplimiento



$Z(\text{INI}) : t = 0 \wedge x_s = t$	$Z(\text{ACT}) : t = 5 \wedge x_s = t$
$Z(\text{REG}) : x_s \leq 3 \wedge t = x_s$	$Z(\text{ACTON}) : t = 5 \wedge x_s = t$
$Z(\text{ESP}) : x_s \geq 1 \wedge t = x_s$	$Z(\text{ACTOFF}) : t = 5 \wedge x_s = t$
$Z(\text{SUB}) : t = 5 \wedge x_s = t$	

Figura 12.12. Grafo de simulación posible para el modelo MUS-T de la figura 12.11

tiene su origen en la selección de un criterio de sólo futuro en la compatibilidad para el estado sucesor por p_{on} y p_{off} (criterio RCMReset para $RSNC$). Sin embargo, no existe ninguna región compatible para un criterio RCT. Sin la aplicación de ningún criterio de compatibilidad obtendríamos una estructura como la representada en la figura 12.13. Sin embargo y dado que en el modelado del entorno la bomba sólo puede abrirse si está cerrada y al contrario, optamos por este modelo más simple. Eliminamos por tanto dichas propiedades del conjunto de propiedades del controlador y las convertimos en propiedades sobre el sistema cerrado.

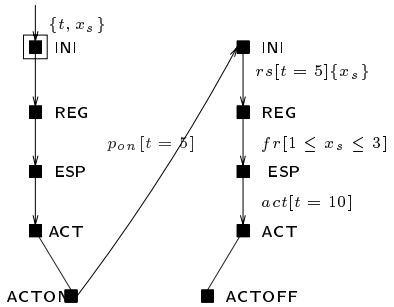


Figura 12.13. Modelo obtenido sin reutilización de región compatible para la regla $RSNC$

- **Finalización de tareas**: Mismos resultados que en la fase anterior.
- **Ciclo de actuación**: La propiedad $RC6$ proporciona los resultados $\{\widehat{\frac{1}{2}}, \frac{1}{2}, 1\}$, y los resultados $\{\widehat{\frac{1}{2}}, 1\}$ con pérdida de subespecificación en el modelo. Sin embargo, esta es la propiedad más débil de las del conjunto ciclo de actuación. La verificación de la propiedad $RC5$, produce los siguientes resultados:

$$GS(RC5, \mathcal{R}) \in \{\widehat{\frac{1}{2}}, \frac{1}{2}\}$$

y para el modelo con pérdida de subespecificación:

$$GS(RC5, \mathcal{R}) \in \{0, \widehat{\frac{1}{2}}\}$$

La propiedad $RC5$ produce un resultado 0 en contextos temporales de ESP en los que $GS(act, \mathcal{R}) = 1$, el origen del no cumplimiento se puede observar en el grafo de simulación posible del modelo (figura 12.12), existen evoluciones partiendo de ESP que permiten distancias temporales mayores que 5 sin un evento *act* con condición de especificación posible. Como se había detectado en la fase anterior, `ESP` es un estado con *deadlock-sub*. Se decide por tanto acotar el tiempo que el modelo puede residir en `ESP` mediante la regla:

$$RSACT_3 := act \ \forall \Rightarrow_+ \ false$$

y consiguientemente $I_a(\mathsf{ESP}) = x_s \geq 1 \wedge t \leq 5$. Con esta nueva incorporación si bien en el prototipo actual $GS(RC5, \mathcal{R}) \in \{\widehat{\frac{1}{2}}, \frac{1}{2}\}$; el prototipo con pérdida de subespecificación proporciona los valores $\{\widehat{\frac{1}{2}}, 1\}$, la propiedad se cumple siempre que es aplicable.

- **Secuenciamiento de ciclo**: Ambas propiedades proporcionan grados de satisfacción en $\{\widehat{\frac{1}{2}}, \frac{1}{2}\}$. Para discernir el motivo del del no cumplimiento cuando es aplicable, evaluamos el modelo con pérdida de subespecificación. En este modelo, si bien $GS(RC8, \mathcal{R}) = 1$, el resultado para $GS(RC9, \mathcal{R})$ es 0: no existe ninguna ejecución en la que se realice el comportamiento opcional de generación de informes. En la siguiente sección se procede, por tanto, a incorporar los escenarios de dicho comportamiento opcional.

12.7.2 Incorporación de la generación de informes

Si se aplica al prototipo actual (figura 12.11) la regla de generación de informes $RSGI$, se obtiene el modelo de la figura 12.14(a), reflejando el comportamiento de esta tarea. Sin embargo, tal comportamiento sigue siendo opcional. La aplicación de $RSGI$ habilita la aplicación de la regla $RSEA_3$ de entrada en la fase de actuación, y las consiguientes $\{RSACT, RSTACT\}$ obteniendo el modelo de la figura 12.14(b) mediante un criterio RCM.

Corrección mínima del modelo

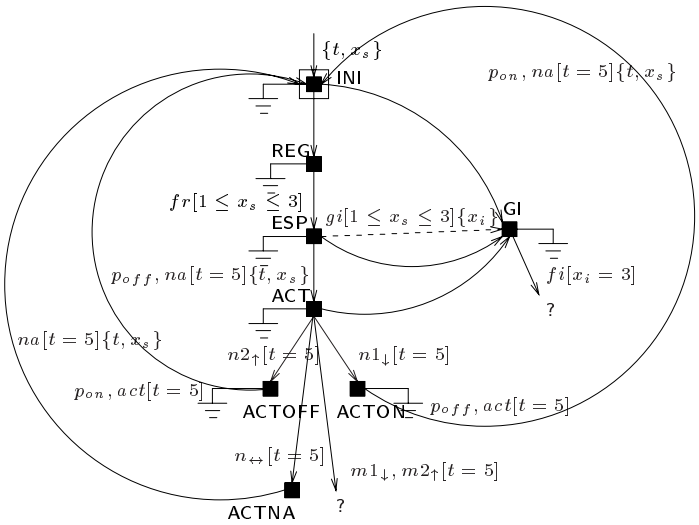
La incorporación de las reglas de entrada en la fase de actuación ocasiona la imposibilidad de cumplimiento de la regla $RC4$ para la finalización de tareas. Mediante la pérdida de subespecificación en el subgrafo con origen en GI, existen contextos temporales en GI tales que $GS(RC4, \mathcal{R}) = \frac{1}{4}$. Es decir, existen ejecuciones en las que no se acaba la generación de informes. La imposibilidad de cumplimiento de $RC4$ en prototipos futuros tiene su origen en comportamientos del prototipo actual en los que la urgencia de actuación (cada 5 segundos) imposibilita la finalización de los informes. Es necesario, por tanto, refinar la generación de informes, sólo habilitándola cuando al sistema le de tiempo tanto a finalizar la tarea (3 segundos), como a pasar a la fase de actuación (5 segundos después de la anterior). Con tal objetivo, incorporamos la siguiente regla:

$$RSGI_2 := (x.\emptyset \Rightarrow \mathcal{AP}[act \wedge x > 2]) \Rrightarrow \neg gi$$

Con la síntesis de ese nuevo escenario obtenemos el fragmento de controlador en la figura 12.15(a). Los contextos temporales que ocasionaban $GS(RC4, \mathcal{R}) = \frac{1}{4}$ son ahora no alcanzables, eliminándose, en consecuencia, la transición por act partiendo del estado de control GI. El grado de satisfacción de RC es $\frac{1}{2}$, consecuencia de que existen acciones discretas subespecificadas que podrían llevar a la no finalización de la generación de informes. Prohibimos por tanto las acciones discretas hasta que no se produzca la finalización de la generación de informes, mediante el escenario:

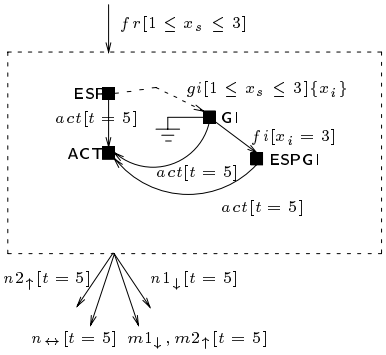
$$RSGI_3 := (\emptyset \ \forall\Rightarrow \odot \ gi) \ \forall\Rrightarrow^{\forall\rightarrow} \bigcirc \ (x_i < 3 \ \forall\Rightarrow \bigcirc \ false)$$

Con este nuevo escenario, el grado de satisfacción de $RC4$ es tal que $GS(RC4, \mathcal{R}) = \frac{3}{4}$, consecuencia de que el comportamiento si bien sigue siendo opcional, si la generación de informes pasa a ser posible en el modelo, se cumplirá



$I_a(\mathsf{GI}) : x_i \leq 3$
$I_p(\mathsf{GI}) : x_i > 3$

(a)



$I_a(\mathsf{GI}) : x_i \leq 3$
$I_p(\mathsf{GI}) : x_i > 3$

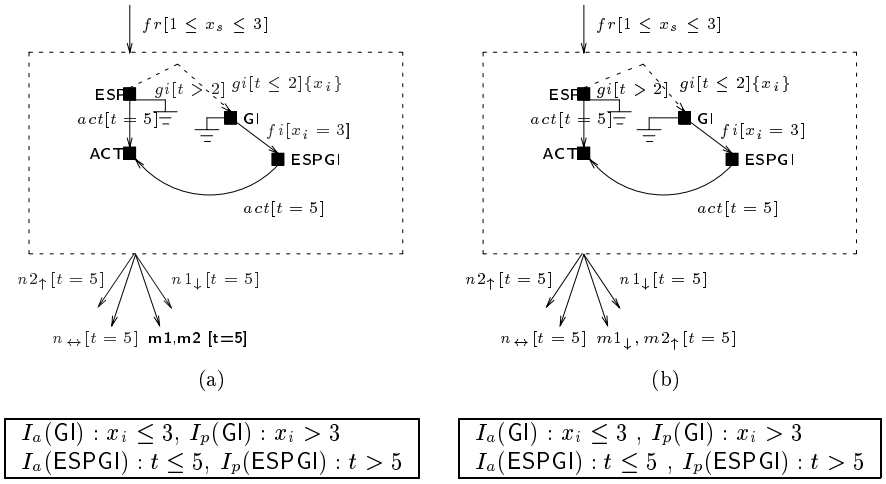
(b)

Figura 12.14. Modelo MUS-T obtenido con los escenarios de generación de informes $RSGI$, más el conjunto de reglas $\{RSEA, RSACT, RSTACT\}$, que pasan a ser aplicables

la propiedad. Habilitamos, por tanto, la generación de informes en este entorno:

$$RSGI_4 := (\emptyset \ \exists\Rightarrow\odot \ fr) \Rrightarrow gi$$

La regla produce conflictos con la regla $RSGI_2$, sólo sintetizándose en los contextos temporales permitidos por dicha regla.



$I_a(\mathsf{GI}) : x_i \leq 3$, $I_p(\mathsf{GI}) : x_i > 3$ $I_a(\mathsf{ESPGI}) : t \leq 5$, $I_p(\mathsf{ESPGI}) : t > 5$

$I_a(\mathsf{GI}) : x_i \leq 3$, $I_p(\mathsf{GI}) : x_i > 3$ $I_a(\mathsf{ESPGI}) : t \leq 5$, $I_p(\mathsf{ESPGI}) : t > 5$

Figura 12.15. Modelo obtenido con los escenarios de generación de informes

Detección de relojes sobrantes

Dado que el reloj x_s sólo está activo en comportamientos disjuntos con los comportamientos en los que está activo x_r, se puede utilizar un sólo reloj x, obteniendo de esta forma un modelo MUS-T con $\mathcal{C} = \{t, x\}$.

12.7.3 Verificación de propiedades sobre el sistema cerrado (sin fallos)

La verificación de propiedades sobre el sistema cerrado, composición de $\mathcal{M}^E$ y $\mathcal{M}^n_C$, se ha realizado mediante el grafo de simulación posible, dado que las propiedades de seguridad y rendimiento no incluyen medidas temporales:

- Dado que el grafo de simulación posible respeta el conjunto de estados alcanzable por evoluciones posibles y el evento *exp* no se produce, el sistema cerrado no incumple la propiedad. Por tanto, el modelo del controlador obtenido por paso de condición de especificación subespecificada a condición de especificación no posible, cumplirá la propiedad de seguridad.
- Para la propiedad de rendimiento podemos comprobar que los estados M1OFF y M2ON no son alcanzables, además, desde cualquier estado abstracto en el grafo de simulación posible, siempre existe un acción posible $n_\leftrightarrow$ posterior.

12.8 Síntesis del controlador II (con fallos)

Abordamos ahora una evolución del controlador desarrollado en la fase anterior para que se comporte de forma adecuada ante un entorno con fallos. En primer lugar, presentamos la especificación informal del comportamiento frente a fallos, para, a continuación, identificar las nuevas acciones en el modelo y los nuevos escenarios a incorporar.

12.8.1 Especificación informal de los modos de fallo

En el modelado del controlador con fallos se introducen los eventos de cambio de modo en la figura 12.2. En función del tipo de error producido en el modo normal, el proceso evolucionará a los modos degradado (evento deg) o rescate (evento res). En el modo normal, como ya habíamos adelantado, se pueden producir dos errores básicos: el mal funcionamiento de la bomba y el fallo del nivel de agua. En caso de mal funcionamiento de la bomba, el proceso evoluciona al modo degradado mediante el evento deg. Por contra, si se detecta un mal funcionamiento en el sensor del nivel de agua se debe evolucionar al modo rescate mediante el evento res. Como también habíamos mencionado, aunque el error en el sensor de nivel de agua es detectado de forma fiable, no sucede así en la detección de error en la bomba. Remitimos a la figura 12.4, en la que se muestran los tipos de errores que se pueden producir en el entorno.

12.8.2 Especificación formal de los modos de fallo

Identificación de nuevas acciones en el controlador

Además de las acciones de sincronización con el entorno con fallos A_S^{EF}, se identifican los eventos de cambio de modo: deg, res; de sincronización con el modelo de modos de operación $\mathcal{M}_M$.

Propiedades del controlador (con fallos)

Además de las propiedades $\{RC\}$[1], ya identificadas, se especifican las propiedades de **cambio de modo**:

- Sólo si hubo un mal funcionamiento de la bomba el sistema pasa al modo de degradado:

$$RC12 := \emptyset \Rightarrow \mathcal{A}[\neg deg\ \mathcal{U}_w\ errorB]$$

- Sólo si se produjo un mal funcionamiento del sensor de nivel de agua el sistema pasa a modo rescate:

$$RC13 := \emptyset \Rightarrow \mathcal{A}[\neg res\ \mathcal{U}_w\ errorNA]$$

[1] Se modifica las propiedades $RC8$ y $RC9$ sólo exigiendo el secuenciamiento si no hubo error ($errorB$ o $errorNA$).

Escenarios identificados en el controlador (con fallos)

- **Detección de modos de fallos**:
  - Cuando se realiza la lectura del valor del sensor de agua, es posible la detección de un error en la lectura:

$$RSMF_0 := (n1_{\downarrow} \vee n2_{\uparrow} \vee m1_{\downarrow} \vee m2_{\downarrow} \vee n_{\leftrightarrow}) \Rrightarrow errorNA$$

  - Cuando se actúa sobre la bomba es posible que se produzca un error de funcionamiento de la bomba:

$$RSMF_1 := (p_{on} \vee p_{off}) \Rrightarrow errorB$$
$$RSMF := RSMF_0 \wedge RSMF_1$$

- **Respuesta a fallos**: La respuesta a los modos de fallos es instantánea, y será el paso a modo degradado en el caso de un error en el sensor de nivel de agua; y el paso a modo rescate en caso de un error en el funcionamiento de la bomba.

$$RSRF_0 := (errorB \vee errorNA)\ \forall\Rrightarrow\overset{\forall\rightarrow}{\bigcirc}\ false$$
$$RSRF_1 := (errorB\ \forall\Rrightarrow\bigcirc\ deg) \wedge (errorNA\ \forall\Rrightarrow\bigcirc\ res)$$
$$RSRF := RSRF_0 \wedge RSRF_1$$

12.8.3 Síntesis del controlador II (con fallos)

Con la síntesis de los escenarios $\{RSMF, RSRF\}$ obtenemos el modelo de controlador de la figura 12.16. En el modelo sintetizado, los estados de control DEG y RES son *timelock-sub* para todos su estados.

Debemos notar que, como se explica en el capítulo 9, mediante la incorporación de la acción subespecificada a_s es posible almacenar la síntesis de comportamientos que serán aplicables con la incorporación de nuevas acciones. Así, por ejemplo, para la regla:

$$RSTACT_5 := (\emptyset\ \exists\Rrightarrow\bigcirc\!\!\!\cdot\ act)\ \forall\Rrightarrow\bigcirc\ (\neg act \wedge (\emptyset\ \forall\Rrightarrow_{+}\ false))$$

se realizará una síntesis como la de la figura 12.17. Todas las especificaciones realizadas en el estado sucesor por a_s se trasladan a los estados sucesores de las nuevas acciones incorporadas en el modelo (en este caso $errorNA$).

Verificación de propiedades sobre el sistema cerrado (con fallos)

La verificación de la propiedad de seguridad sobre el sistema cerrado (mediante el grafo de simulación posible) muestra que los estados M1OFF y M2ON son alcanzables. La habilitación del progreso del tiempo en el controlador para el estado sucesor por $m1_{\downarrow}$ y $m2_{\uparrow}$ supone el riesgo de producir daños graves en la caldera (exp). Por tanto, es necesario la incorporación de la parada de emergencia

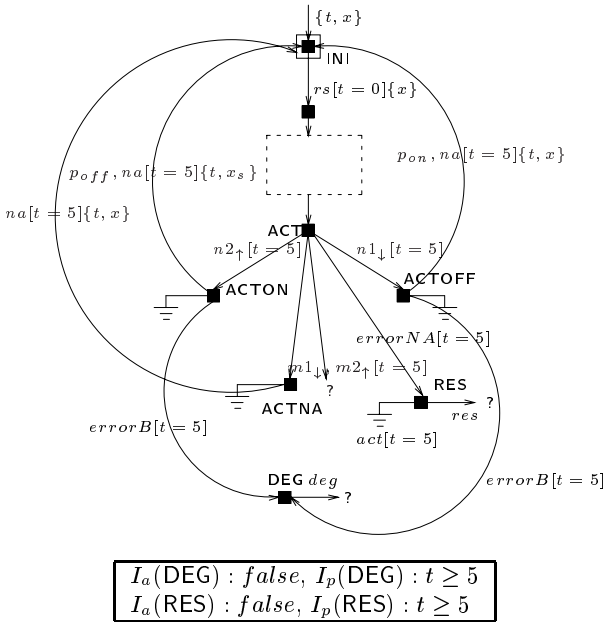


Figura 12.16. Incorporación de modos de fallo mediante las reglas $\{RSMF, RSRF\}$

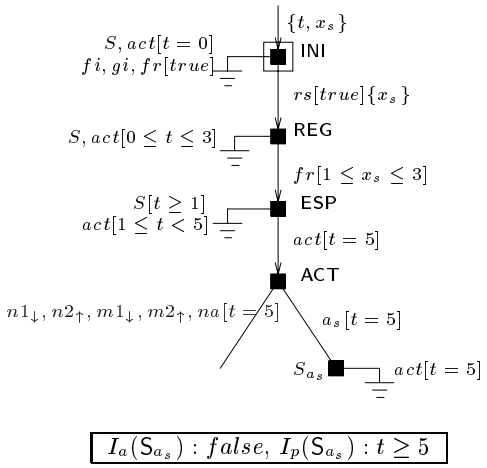


Figura 12.17. Reutilización de reglas aplicables por la incorporación de una nueva acción en el alfabeto

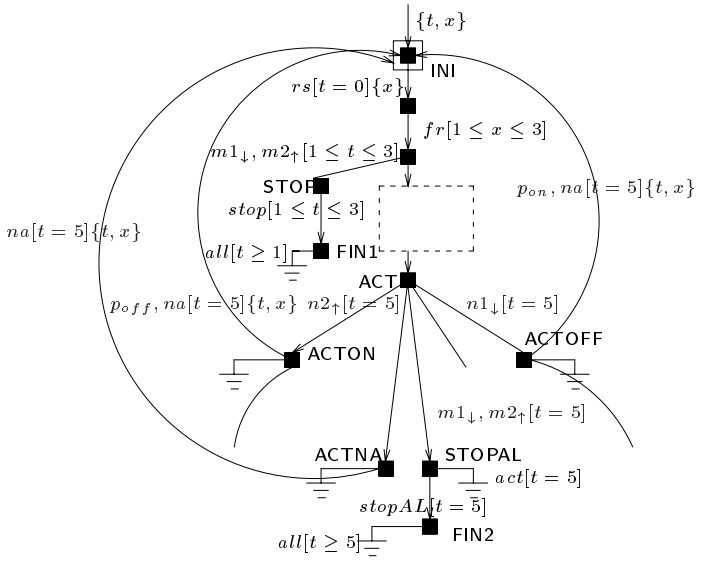
12.8.4 Incorporación de parada de emergencia

Dado que, en base a la especificación informal de la parada de emergencia, se deben detectar los valores límites con un retardo no mayor de 3 segundos, se opta por comprobar $M1$ y $M2$ dos veces en un ciclo del controlador: después de la regulación de sensores y durante la fase de actuación. Se identifican para ello los siguientes escenarios de parada de emergencia:

$$
\begin{aligned}
RSPE_0 &:= (\emptyset \ \ \exists\Rightarrow\bigcirc \ \ fr) \Rrightarrow m1_{\downarrow} \wedge m2_{\uparrow} \\
RSPE_1 &:= (x.m1_{\downarrow} \Rightarrow \mathcal{EP}[n1_{\downarrow} \wedge x \leq 3]) \Rrightarrow (m1_{\downarrow} \ \ \forall\Rightarrow\bigcirc \ \ stop) \\
RSPE_2 &:= (x.m2_{\uparrow} \Rightarrow \mathcal{EP}[n2_{\uparrow} \wedge x \leq 3]) \Rrightarrow (m2_{\uparrow} \ \ \forall\Rightarrow\bigcirc \ \ stop) \\
RSPE_3 &:= (x.m1_{\downarrow} \Rightarrow \mathcal{AP}[\neg n1_{\downarrow} \vee x > 3]) \Rrightarrow (m1_{\downarrow} \ \ \forall\Rightarrow\bigcirc \ \ stopAL) \\
RSPE_4 &:= (x.m2_{\uparrow} \Rightarrow \mathcal{AP}[\neg n2_{\uparrow} \vee x > 3]) \Rrightarrow (m2_{\uparrow} \ \ \forall\Rightarrow\bigcirc \ \ stopAL) \\
RSPE &:= RSPE_0 \wedge RSPE_1 \wedge RSPE_2 \wedge RSPE_3 \wedge RSPE_4
\end{aligned}
$$

y el escenario de comportamiento de la parada y la parada con alarma:

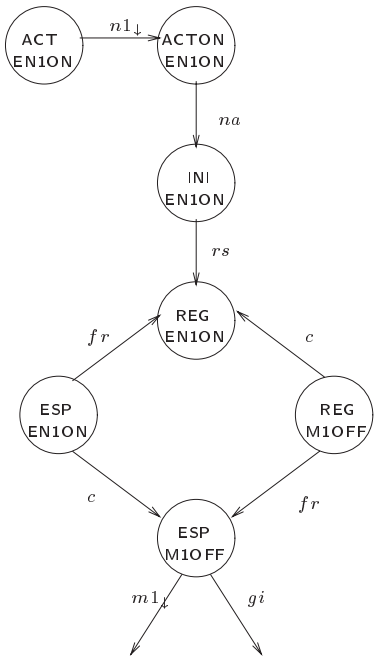
$$RSCPE := stop \vee stopAL \ \ \forall\Rrightarrow^{\forall\rightarrow}\bigcirc \ \ false \wedge (\emptyset \ \ \forall\Rightarrow\bigcirc \ \ false)$$



I_a(STOP) : $false$, I_p(STOP) : $t \geq 1$
I_a(STOPAL) : $false$, I_p(STOPAL) : $t \geq 5$
I_a(FIN1) : $false$, I_p(FIN1) : $t \geq 1$
I_a(FIN2) : $false$, I_p(FIN2) : $t \geq 5$

Figura 12.18. Controlador sintetizado para la parada de emergencia mediante las reglas $RSPE$ y $RSCPE$. La etiqueta *all* se corresponde con el total de acciones identificadas en el modelo

Obtenemos, mediante la aplicación de las reglas $\{RSPE, RSCPE\}$, el modelo en la figura 12.18, prototipo que comprueba el valor de los límites máximos dos veces en cada ciclo. Las reglas $RSPE_3$ y $RSPE_4$ no son aplicables en ningún contexto temporal del estado ESP. Igualmente, las reglas $RSPE_1$ y $RSPE_2$ no son aplicables en ningún contexto temporal del estado ACT. Los estados de control STOP y STOPAL son estados con *timelock-sub*, los estados de control FIN1 y FIN2 son estados con *timelock* y *deadlock*. Sin embargo, dado que se espera que el sistema se pare en este contexto no es necesariamente un error de diseño.



Z(ACT,EN1ON) : $(t = x = 5) \wedge (te = 10)$
Z(ACTON,EN1ON) : $(t = x = 5) \wedge (te = 10)$
Z(INI,EN1ON) : $(t = x = 0) \wedge (10 \leq te \leq 12)$
Z(REG,EN1ON) : $x = (0 \leq t = x \leq 2) \wedge (10 \leq te \leq 12)$
Z(ESP,EN1ON) : $(1 \leq t = x \leq 2) \wedge (11 \leq te \leq 12)$
Z(REG,M1OFF) : $(2 \leq t = x \leq 3) \wedge (0 \leq te \leq 1)$
Z(ESP,M1OFF) : $(2 \leq t = x \leq 5) \wedge (0 \leq te \leq 3)$

Figura 12.19. Fragmento del grafo de simulación posible para la parada de emergencia

Sin embargo, observando el grafo de simulación del sistema cerrado, se puede comprobar que la parada sin alarma (*stop*) es diferible. En la figura 12.19 se muestra un fragmento del grafo de simulación posible en el que el entorno sobrepasa el límite $M1$, en el estado (ESP,M1OFF), pudiendo residir en dicho estado abstracto para valores de tiempo [2]:

$$(2 \leq t = x_s \leq 5) \wedge (0 \leq te \leq 3)$$

y por tanto el controlador podrá postergar la evolución discreta por $m1_{\downarrow}$, y por

[2] donde te representa el reloj t del modelo del entorno.

tanto la parada de emergencia, en favor de la generación de informes. El modelado de las acciones $m1_{\downarrow}$ y $m2_{\uparrow}$ como eventos prioritarios sobre la generación de informes no se puede alcanzar sin una noción explícita de prioridades en el modelo MUS-T.

Sin la consideración de prioridades en MUS-T, una solución de diseño de este tipo siempre podía pasar por alto los eventos $m1_{\downarrow}$ y $m2_{\uparrow}$, no modelando el comportamiento de una interrupción. No obstante, las prioridades pueden ser modeladas mediante la introducción de un componente MUS-T ($\mathcal{M}_P$) de gestión de prioridades, como se muestra en la figura 12.20, donde las acciones $cm1_{\downarrow}$ y $cm2_{\uparrow}$ son las correspondientes al evento c en los modelos $\mathcal{M}^E$ y $\mathcal{M}^{EF}$ para los cambios al umbral $M1$ y $M2$, respectivamente.

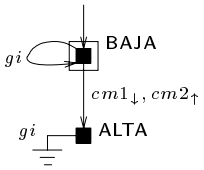


Figura 12.20. Modelo $\mathcal{M}_P$ de gestión de prioridades

El modelo global con la inclusión de prioridades será:

$$\mathcal{M}_{EF}|[cm1_{\downarrow}, cm2_{\uparrow}]|\mathcal{M}_P|[A_S^F]|\mathcal{M}_C^n$$

sin embargo, por simplicidad del caso de estudio, descartamos la distinción realizada en la especificación inicial, considerando un sólo tipo de parada *stop* en la fase de actuación de proceso.

12.8.5 Verificación de propiedades sobre el sistema cerrado (con fallos)

Dadas la propiedades del sistema cerrado en un entorno con fallos (sección 12.5):

$$RSCS^{EF} := \emptyset \Rightarrow \neg exp$$
$$RSCR^{EF} := \emptyset \Rightarrow \mathcal{A}[\mathcal{EF}[n_{\leftrightarrow}]\ \mathcal{U}_w\ (errorB \vee errorNA)]$$

se procede a su verificación mediante el grafo de simulación posible. Si bien el sistema cerrado nunca se ve dañado, existen ejecuciones en las que el sistema no alcanza los límites normales sin que se haya producido un error en la bomba ($errorB$) o en el sensor de agua ($errorNA$). Tales ejecuciones contienen el evento interno $errorBnd$. El objetivo será intentar detectar dichos comportamientos y permitir el paso al modo `rescate` en tal contexto. Realizaremos el estudio en la siguiente sección.

12.8.6 Protección frente a errores de bomba no detectados

Nos centramos en esta sección en aquellas ejecuciones del sistema cerrado que, no habiéndose producido los errores $errorNA$ o $errorB$, no alcanzan los límites

normales en el nivel de agua. Como hemos mencionado, una traza de contraejemplo identificaría el origen del error en aquellas ejecuciones del sistema que incorporan el evento $errorBnd$, indicando que no hemos protegido al controlador frente a este tipo de fallos. Dado que se trata de un evento interno al entorno, la detección debe basarse en las diferencias entre el comportamiento esperado, cuando no hay error, y el comportamiento obtenido en presencia de $errorBnd$. Es decir, si una vez realizada una actuación en la bomba, no se produce la consecuencia deseada, en el caso peor, se ha producido un error no detectado en la bomba.

Es en este punto donde un mecanismo de cálculo de distancias temporales ayudaría al diseñador. Simulamos dicho mecanismo mediante la incorporación de un reloj observador cuando se pone en funcionamiento la bomba, y analizando el grafo de simulación posible obtenido. De esta forma, podemos afirmar que cuando la bomba se abre (p_{on}) y funciona correctamente alcanzaremos un nivel de agua normal ($n_{\leftrightarrow}$) en no más de 24 segundos. Igualmente, cuando la bomba se cierra (p_{off}) y funciona correctamente alcanzaremos un nivel de agua normal ($n_{\leftrightarrow}$) en no más de 8 segundos

Sobre el sistema cerrado especificamos los escenarios que implican que se ha producido un error no detectado en la bomba:

$$
\begin{aligned}
&RSEND_0 := \mathcal{E}[\neg p_{off}\mathcal{S}p_{on}\{x_{ndon}\}] \wedge n1_{\downarrow}\ \forall\Rrightarrow\bigcirc\ (x_{ndon} > 24) \Rightarrow derrorB\\
&RSEND_1 := \mathcal{E}[\neg p_{on}\mathcal{S}p_{off}\{x_{ndoff}\}] \wedge n2_{\uparrow}\ \forall\Rrightarrow\bigcirc\ (x_{ndoff} > 8) \Rightarrow derrorB\\
&RSEND_2 := derrorB\ \ \forall\Rrightarrow\bigcirc\ deg \wedge (\emptyset\ \ \forall\Rightarrow_{+}\ \ false)\\
&RSEND := RSEND_0 \wedge RSEND_1 \wedge RSEND_2
\end{aligned}
$$

Las reglas de síntesis se aplican al sistema cerrado, composición del entorno y del controlador; una vez realizada la síntesis, los cambios obtenidos se traducen al componente no totalmente especificado (controlador), obteniendo modelo de la figura 12.21, donde sólo es necesario el uso de un reloj (x_{nd}). La síntesis se ha realizado sobre un modelo mínimo obtenido para cada una de las reglas $RSEND$.

Para el nuevo controlador especificamos la propiedad de rendimiento como:

$$RSCR^{EF}_{end} := \mathcal{AG}[\neg n_{\leftrightarrow}] \Rightarrow \mathcal{AF}[derrorB \vee errorB \vee errorNA]$$

de forma que si la caldera no puede alcanzar los límites normales, se detecta que se ha producido un error.

Una validación, mediante el grafo de simulación posible del sistema cerrado, permite observar que existen ejecuciones que no pudiendo alcanzar el nivel normal no suponen la ocurrencia de ningún evento de error ($derrorB$, $errorB$ y $errorNA$), consecuencia del siguiente hecho: en base a los tiempos especificados para el entorno, antes de la detección $derrorB$, el sistema ya ha alcanzado los límites máximos $M1$ y $M2$ y se ha parado. Más concretamente, cuando se activa p_{on} en el estado EN1OFF se tarda un tiempo $0 < t \leq 12$ unidades de tiempo en alcanzar el estado M1OFF, tiempo menor que el límite impuesto para la protección frente a errores en la bomba no detectados. Sin embargo, cuando el error de bomba no detectado se produce al cerrar la bomba (p_{off}), los tiempos en la especificación del entorno sí permiten la protección.

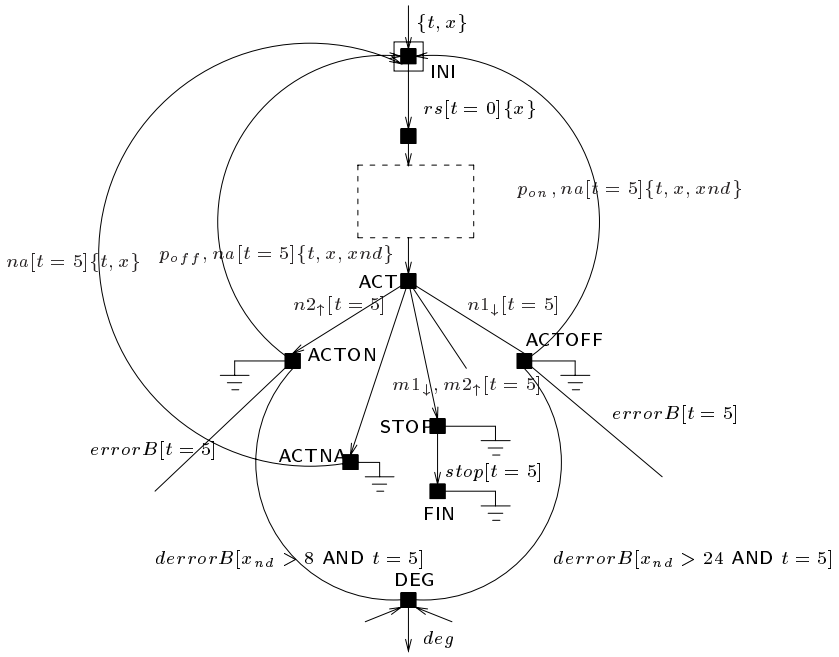


Figura 12.21. Controlador sintetizado para la protección frente a errores no detectados de la bomba mediante las reglas $RSEND$.

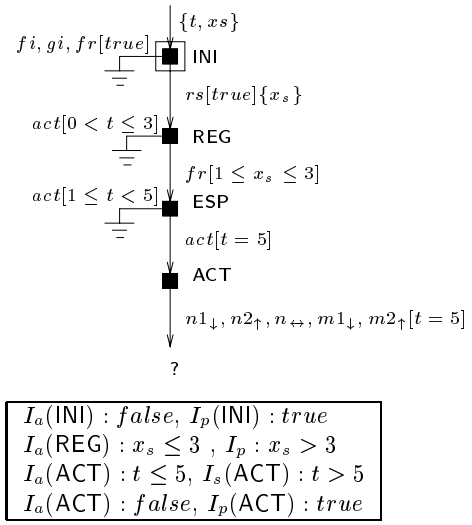
A partir del prototipo inicial desarrollado, el proceso de diseño partiría del modelo generado antes de la incorporación de los modos de fallo para la síntesis de los prototipos correspondientes a los otros modos de operación. Para finalizar, el diseñador debería eliminar la subespecificación en los modelos, ya sea mediante especificaciones adicionales, o mediante la pérdida de subespecificación por paso a no posible.

12.9 Utilización del algoritmo de minimización

En el caso de estudio descrito en este capítulo se han empleado los algoritmos de verificación y síntesis desarrollados (operando sobre el grafo RMUS-T). Sin embargo, cuando se realiza la composición de componentes, entre el controlador y el entorno y cuando la constante de tiempo máxima en la especificación es grande ($t = 24$ para el entorno), el espacio de estados crece, pudiendo en algunos casos resultar inmanejable. Además, un mecanismo que realizara el cómputo de contraejemplos obtendría trazas de tamaño ilegible para el diseñador.

Con el fin de reducir el espacio de estados y habilitar unos métodos de decisión eficientes, en el capítulo 10 se define $\mathrm{STaB}_{\text{MUS-T}}$, así como un algoritmo de minimización en base a tal bisimulación. A continuación presentamos un ejemplo del resultado del algoritmo desarrollado, y el mecanismo de post-estabilización que debe ser incorporado en los algoritmos de verificación y síntesis, para fórmulas de pasado.

Sea el modelo incompleto del controlador sin fallos en la figura 12.22 sobre un alfabeto $A = \{m1_{\downarrow}, m1_{\downarrow}, n2_{\uparrow}, m2_{\uparrow}, n_{\leftrightarrow}, act, rs, fr, gi, fi\}$.



I_a(INI) : $false$, I_p(INI) : $true$
I_a(REG) : $x_s \leq 3$, $I_p : x_s > 3$
I_a(ACT) : $t \leq 5$, I_s(ACT) : $t > 5$
I_a(ACT) : $false$, I_p(ACT) : $true$

Figura 12.22. Modelo de controlador para minimización

Para el modelo de la figura 12.22, el cómputo del grafo RMUS-T, en el que no se representan explícitamente los arcos totalmente subespecificados, produce 21 regiones y 23 transiciones, a pesar de ser un modelo pequeño con sólo 4 estados y cuya constante máxima es 5.

Sea la regla de síntesis:

$$RSACT_4 := (\emptyset \ \exists\Rightarrow \odot \ act) \Rightarrow (n1_{\downarrow} \ \wedge \ n2_{\uparrow} \ \wedge \ m1_{\downarrow} \ \wedge \ m2_{\uparrow} \wedge n_{\leftrightarrow})$$

tal que:

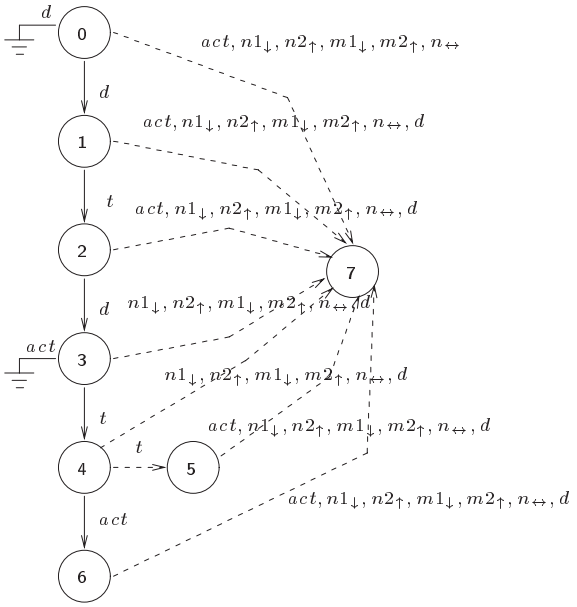
- $\mathcal{E}^f = \{\}$ y $\mathcal{E}^p = \{\}$, dado que no existen relojes de especificación en $RSACT_4$.
- $A^{RSACT_4} = \{act, n1_{\downarrow}, n2_{\uparrow}, m1_{\downarrow}, m2_{\uparrow}, n_{\leftrightarrow}, d\}$, el conjunto de acciones discretas consideradas en la minimización más la acción genérica discreta d.
- $\Psi_{\mathcal{C}}^{RSACT_4} = \{\}$, $\Psi_{\mathcal{E}^f}^{RSACT_4} = \{\}$ y $\Psi_{\mathcal{E}^p}^{RSACT_4} = \{\}$, dado que no existen predicados de tiempo en $RSACT_4$

Dado que no existen predicados de tiempo en $RSACT_4$ invocamos el algoritmo de minimización con la siguiente partición inicial:

$$\mathcal{P}_0 := \{[INI, Z^u], [REG, Z^u], [ESP, Z^u], [ACT, Z^u], [s_s, Z^u]\}$$

donde Z^U es el DBM universal en el dominio $\mathcal{P}^{\mathcal{C}}$ de poliedros sobre el conjunto de relojes $\mathcal{C} = \{t, x_s\}$.

El grafo cociente obtenido, por refinamiento de la partición inicial $\mathcal{P}_0$ se muestra en la figura 12.23, donde sólo se representan los elementos de la partición accesibles. Todos los elementos P_i en el grafo cociente de la figura 12.23 son tal que por lo menos existe un $(s, \gamma) \in P_i$ accesible; además existirá un (s, γ) accesible bajo cada una de las condiciones de especificación de los caminos que llevan a P_i.



$0 \rightarrowtail [\mathsf{INI}, (0 \leq t < 2) \wedge (0 \leq x < \infty]$	$4 \rightarrowtail [\mathsf{ESP}, (t = 5) \wedge (0 \leq x < \infty)]$
$1 \rightarrowtail [\mathsf{REG}, (0 \leq t < 3) \wedge (0 \leq x < 1)]$	$5 \rightarrowtail [\mathsf{ESP}, (5 < t < \infty) \wedge (0 \leq x < \infty)]$
$2 \rightarrowtail [\mathsf{REG}, (0 \leq t < 5) \wedge (1 \leq x \leq 3)]$	$6 \rightarrowtail [\mathsf{ACT}, (0 \leq t < \infty) \wedge (0 \leq x < \infty)]$
$3 \rightarrowtail [\mathsf{ESP}, (0 \leq t < 5) \wedge (0 \leq x < \infty)]$	$7 \rightarrowtail [\mathsf{SUB}, (0 \leq t < \infty) \wedge (0 \leq x < \infty)]$

Figura 12.23. Abstracción del modelo de la figura 12.22 obtenida, por minimización, para la regla $RSACT_4$

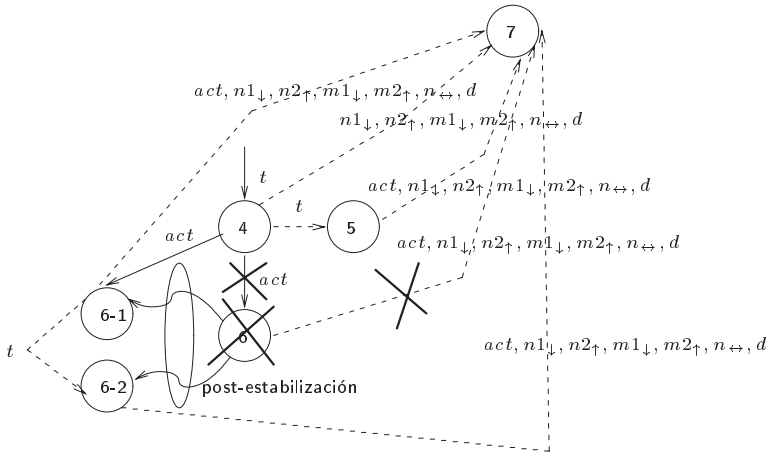
12.9.1 Ejemplo de post-estabilización

Dado que la regla de síntesis $RSACT_4$ contiene operadores de pasado es necesaria una post-estabilización del grafo cociente. La regla sólo será aplicable en estados pertenecientes al elemento $[\mathsf{ACT}, (0 \leq t < \infty) \wedge (0 \leq x < \infty)]$, dado que es el único con una transición entrante posible *act*. Sin embargo y si bien todos los estados accesibles en $[\mathsf{ESP}, (t = 5) \wedge (0 \leq x < INF)]$ pueden evolucionar de forma discreta por *act*, en general, no todos los estados accesibles en $[\mathsf{ACT}, (0 \leq t < \infty) \wedge (0 \leq x < \infty)]$ tienen un predecesor discreto en $[\mathsf{ESP}, (t = 5) \wedge (0 \leq x < INF)]$.

La post-estabilización debe computar, a partir del elemento de la partición $[\mathsf{ESP}, (t = 5) \wedge (0 \leq x < INF)]$, los sucesores por *act* mediante el operador:

$$suc_d([\mathsf{ESP}, (t = 5) \wedge (0 \leq x < INF)], act, 1) := [\mathsf{ACT}, (t = 5) \wedge (0 \leq x < INF)]$$

La post-estabilización en el elemento de la partición 6 produce el grafo cociente en la figura 12.24, en el que es posible realizar la síntesis de la regla.



$$\boxed{\begin{array}{l} 6-1 \rightarrowtail [\mathsf{ACT}, (t = 5) \wedge (0 \leq x < \infty)] \\ 6-2 \rightarrowtail [\mathsf{ACT}, ((0 \leq t < 5) \wedge (0 \leq x < \infty)) \vee ((5 < t < \infty) \wedge (0 \leq x < \infty))] \end{array}}$$

Figura 12.24. Ejemplo de post-estabilización del grafo cociente de la figura 12.23, para la síntesis de la regla $RSACT_4$

12.10 Conclusiones

En primer lugar, recapitulamos brevemente los resultados presentados en (Abrial et al., 1996) para el caso de estudio desarrollado. Las soluciones presentadas se abstraen en su mayoría del tiempo, considerando un sistema lo suficientemente rápido. Sin embargo, se plantean algunas aproximaciones de tiempo discreto y tres enfoques con tiempo denso. La especificación del caso de estudio

prescribe un ciclo de proceso controlador de 5 segundos, permitiendo un enfoque discreto, dado que los sensores y actuadores sólo operan en ese intervalo. Sin embargo, como ya hemos mencionado, el modelado denso permite una mayor flexibilidad. Supongamos, por ejemplo, que el sistema de transmisión de mensajes (en la especificación ideal y sin error) incluyese retardos de 1 segundo entre el controlador y el sistema físico, un enfoque sin tiempo o un enfoque discreto necesitaría una revisión de los modelos realizados a partir de la especificación. La comparativa con las soluciones propuestas con tiempo denso se resume a continuación:

- En el enfoque basado en *Timed I/O Automaton* en (Leeb y Lynch, 1996), no se utiliza ninguna metodología explícita para la captura de requisitos, presentando el diseño en forma de componentes *Timed I/O Automaton*. Si bien realiza el diseño de forma incremental, incorporando bombas y modos de fallo adicionales, en cada incremento no se parte del modelo *Timed I/O Automaton* anterior.

- La propuesta basada en *Hybrid Automaton* (Henzinger y Toi, 1996), si bien alejada de los objetivos de esta tesis, modela tanto el entorno como el controlador como un autómata híbrido con reglas de variación lineal. El diseño realizado no es incremental y, al igual que el anterior, no propone ninguna forma metódica de obtener los modelos *Hybrid Automaton*. La aportación fundamental de esta propuesta es que permite el cálculo de los niveles de agua óptimos a partir de las magnitudes que definen el sistema físico.

- El enfoque basado en *Duration Calculus* y *Real-time Communicating Automaton* (XiaoShan y JuAn, 1996), realiza la captura de requisitos mediante una lógica de intervalo. Si bien la captura de requisitos está formalizada, no se propone una metodología automática para el paso de dicha captura de requisitos, expresada en lógica, al diseño en autómata. Sin embargo, y al igual que el anterior, propone un método que "a mano" permite optimizar el diseño del controlador.

Del caso de estudio realizado en este capítulo se concluye que la característica más sobresaliente de la metodología propuesta en esta tesis es su naturaleza incremental, así como la definición de una forma metódica de la obtención de un prototipo a partir de un conjunto de escenarios identificados en la captura de requisitos. Se ha confirmado que el conjunto SCTL-T restringido para síntesis, es suficientemente expresivo como para una obtención de prototipos MUS-T basada en escenarios. Bajo esta perspectiva, el diseñador puede especificar el comportamiento actual o futuro del modelo en función de condiciones sobre el pasado, reflejando, de esta forma, la intuición del comportamiento de un sistema. En (Somé y Dssouli, 1998), se plantea una solución similar para *Timed Automaton* a partir de escenarios en lenguaje natural. Sin embargo, en dicho trabajo se realiza una síntesis sintáctica de los escenarios sin comprobación de conflictos a priori, necesitando, en consecuencia, un chequeo de consistencia posterior.

Se ha puesto de relieve que las primeras fases de diseño son, por naturaleza, de fuerte interacción con el diseñador: los cambios de requisitos son frecuentes y, a menudo, el diseñador no dispone de un conocimiento profundo del problema. Por brevedad, dicha realidad se ha ocultado, en parte, en el presente capítulo: tanto la especificación de propiedades del controlador como la de los escenarios

ha conllevado distintos trabajos de síntesis, verificación y validación en prototipos alternativos, tanto del entorno como del proceso controlador. Dada la fuerte interacción en estas fases, queda patente la necesidad de disponer de herramientas de tratamiento formal de distinta naturaleza.

En lo referente a la incorporación del tiempo en la metodología, se ha corroborado la ventaja de tener en cuenta el tiempo desde las primeras fases del ciclo de vida. La imposibilidad de proteger el controlador frente a errores no detectados en la bomba se hace, de esta forma, palpable en la captura de requisitos inicial, no siendo retrasada a fases posteriores que aumentaría los costes asociados.

Adicionalmente, resaltamos la ventaja del tipo de arquitectura lógica seleccionada para el modelado del sistema, así como la flexibilidad de la incorporación de un modelo de tiempo denso. Componentes adicionales en el modelo, como un gestor de prioridades o un mecanismo de comunicación de mensajes, pueden ser introducidos en el modelado del sistema con cambios mínimos en la especificación.

Además de la adecuación de la metodología propuesta, como resultado del desarrollo del caso de estudio se han inferido un conjunto de conclusiones que exponemos a continuación, algunas de las cuales sugieren líneas de trabajo futuro:

- Se ha puesto de manifiesto la gran reducción obtenida mediante la aplicación del algoritmo de minimización. Como se sigue del grafo mínimo obtenido y de la estructura de los algoritmos de *model checking* y síntesis, dichos métodos de decisión podrían operar, de forma directa, sobre dicha abstracción. Los métodos de decisión basados en minimización habilitan la aplicación de la metodología a casos de estudio que, bajo la abstracción RMUS-T, se harían inabordables.

- En lo que se refiere a la síntesis de escenarios de emergencia se ha optado por una solución de diseño en la que el proceso controlador comprueba los valores extremos dos veces en un ciclo. La urgencia necesaria en la detección de valores en los límites máximos, supone la inclusión de un mecanismo de gestión de prioridades externo al componente. La ventaja de un mecanismo de prioridades externo reside en la flexibilidad de modelado, dado que el cambio de prioridades entre transiciones discretas sólo implica el cambio del modelo de gestión de prioridades. Aun así, como resaltaremos en la exposición de las líneas de trabajo futuro, se deben investigar las ventajas de una incorporación explícita de prioridades en el modelo MUS-T. Dicha incorporación, lejos de reemplazar un mecanismo basado en sincronización, permitiría el modelado de prioridades cuando éstas son internas al componente.

- Se ha resaltado la utilidad del grafo de simulación posible como una forma de validar los prototipos, así como de adquirir un mayor conocimiento sobre el problema y los requisitos especificados. El caso de estudio propuesto, por su tamaño, permite discernir (por validación) el origen de incumplimiento de una propiedad, o la forma en que una propiedad (no en el conjunto de reglas de síntesis) puede ser realizada. Sin embargo, es necesario adaptar el algoritmo de verificación y el algoritmo de síntesis para aportar información de este tipo. Los algoritmos propuestos se basan en la exploración de una abstracción de los comportamientos del modelo, la generación de contraejemplos y sugerencias es inmediata dado que ambos utilizan una estrategia

local sobre dicha abstracción.

- Se ha resuelto el problema del cómputo de distancias temporales entre el cumplimiento de dos subfórmulas genéricas, mediante la incorporación de un reloj observador en el grafo de simulación posible. De esta forma es posible calcular distancias mínimas y máximas en los estados alcanzables del prototipo actual. Dicho mecanismo puede ser ocultado al diseñador, convirtiéndolo en automático, asociando el reinicio del reloj observador a la subfórmula origen, y haciendo una búsqueda en el grafo de simulación para la subfórmula destino.

- En la síntesis de reglas sobre el sistema cerrado, nos hemos basado en la naturaleza totalmente especificada del entorno, lo que permite traducir los cambios de síntesis al componente no totalmente especificados.

- Por brevedad en la exposición del caso de estudio, se ha propuesto un modelo de caldera con una sola bomba. Sin embargo, la inclusión de bombas adicionales, en un modelo sin datos, llevaría a modelos visualmente poco intuitivos. Retomaremos el problema de inclusión de datos en las líneas de trabajo futuro. Con una metodología con tiempo y datos el desarrollo del caso de estudio completo sería similar al realizado en este capítulo.

CAPÍTULO 13

Conclusiones y líneas de trabajo futuro

El trabajo presentado en esta tesis abarca todos los elementos formales, así como los procedimientos de decisión necesarios para la adecuación de la metodología formal SCTL/MUS al tratamiento formal de sistemas con requisitos de tiempo real críticos. Asimismo, se han estudiado las necesidades que un sistema de tiempo real impone en el modelo de ciclo de vida propuesto. En este capítulo se realiza un breve compendio de las contribuciones realizadas, así como de las conclusiones recabadas durante su elaboración. Algunas de las conclusiones expuestas sugieren, de forma directa, líneas de trabajo futuro.

13.1 Conclusiones

El trabajo de esta tesis arranca con el objetivo de extender la metodología formal SCTL/MUS al tratamiento formal de sistemas de tiempo real críticos. Tal metodología propone una estructura de proceso *software* iterativa e incremental con prototipado, guiada por la formalidad de las especificaciones y de los procedimientos de decisión definidos en el ciclo de vida.

Esta extensión de la metodología formal SCTL/MUS ha supuesto la revisión de las necesidades que un sistema de tiempo real impone en el modelo de ciclo de vida. Una de las conclusiones inferidas es la adecuación de un modelo de ciclo de vida incremental e iterativo para este tipo de sistemas, en los que es especialmente importante estudiar alternativas de diseño y prototipos desde las primeras fases del proceso *software*. Sin embargo, y si bien el modelo de ciclo de vida en la metodología original es del todo adecuado al tratamiento de sistemas de tiempo real críticos, la adaptación de la fase de refinamientos requiere un estudio más amplio que detallaremos en las líneas futuras.

Independientemente de la estructura del modelo de ciclo de vida utilizada, existe un amplio consenso en que, especialmente en los sistemas con características

de tiempo real, el tiempo debe ser tenido en cuenta desde las primeras fases del proceso *software*. En base a tal aseveración se hace del todo necesario la adecuación del lenguaje de requisitos SCTL para permitir la expresividad necesaria que fenómenos como plazos, retardos de propagación, temporizaciones, etc, impone en la captura de requisitos para un sistema de tiempo real.

Adicionalmente, y dada la necesidad de disponer de prototipos que permitan validar el diseño propuesto para el sistema, se hace necesaria la extensión del modelo MUS para la incorporación de comportamientos temporizados.

Con el ánimo de extender tanto el lenguaje de especificación de requisitos SCTL como el modelo operacional MUS, ha sido necesario hacer una revisión del estado del arte en busca de la forma más adecuada de incorporar el dominio temporal en la semántica de ambos formalismos. Tal revisión nos permite afirmar que los modelos densos son más adecuados para composición y refinamiento, además de no imponer la necesidad de seleccionar un cuanto de tiempo a priori. Una semántica densa del tiempo conlleva un modelado más flexible e independiente del entono del modelo (otros componentes, características hardware, etc). Si a tales ventajas en flexibilidad, se suma el hecho de que los procedimientos de decisión sobre dominios densos son cada vez más eficientes, habremos dibujado las principales justificaciones que nos han llevado a la selección de este tipo de dominio.

Incorporación de requisitos de tiempo

Hemos considerado que los requisitos de tiempo en un sistema de tiempo real son, en esencia, requisitos funcionales y, por tanto, deben ser tenidos en cuenta desde las primeras fases del proceso *software*. Con tal objetivo se ha extendido el lenguaje de especificación de requisitos SCTL con una sintaxis y una semántica que permita expresar restricciones de tiempo en los comportamientos esperados del sistema.

La extensión temporizada SCTL-T respeta los principios básicos de la lógica SCTL subyacente: simplicidad, causalidad y naturaleza multivalor. Adicionalmente, se ha aumentado la expresividad de la lógica incluyendo tanto la cuantificación universal como la cuantificación existencial, y se la ha dotado de los operadores de punto fijo mínimo y máximo, permitiendo la definición de una amplia gama de requisitos y operadores macro a partir de los operadores básicos SCTL-T.

Sin embargo, la expresividad no es la única característica deseable en una lógica para un transferencia tecnológica real. Se imponen demandas de facilidad de uso, no consideradas en la presente tesis. La lógica definida se basa en un conjunto de operadores básicos que permiten la definición de una amplia gama de operadores macro. A pesar de ello, el sólo uso de lógica temporal para la captura de requisitos hace que pueda ser considerada difícilmente utilizable en la práctica profesional. En este sentido apuntamos algunas soluciones en las líneas futuras.

Extensión del modelo MUS

Para la extensión del modelo operacional MUS se ha seleccionado el paradigma de autómata temporizado, avalado por el éxito de la resolución de los problemas teóricos fundamentales y su aplicación a casos reales.

Se define el modelo MUS-T, un modelo MUS ampliado con arcos temporales, donde la condición de especificación opera tanto sobre los arcos temporales como sobre los arcos discretos, los únicos considerados en el modelo sin tiempo MUS. Las acciones en un modelo MUS se tornan en acciones temporizadas en un modelo MUS-T, llevando a acciones que son posibles, no posibles o subespecificadas, dependiendo del contexto temporal en el que se encuentre la ejecución del modelo. La incorporación de una condición de especificación multivalorada en el progreso del tiempo ha llevado a la inclusión de invariantes en los estados del modelo, identificando cuándo el tiempo puede progresar, cuándo el tiempo no puede progresar, y cuando el progreso del tiempo se encuentra subespecificado en el prototipo actual del sistema a diseñar.

Como hemos mencionado a lo largo de esta memoria, el autómata temporizado no es cerrado bajo complementación. Si bien existen otros modelos de tiempo denso que, aunque menos expresivos, son cerrados bajo todas la operaciones booleanas, tal deficiencia sólo tiene incidencia en la verificación homogénea basada en autómata, por lo que este hecho no presenta ninguna limitación en nuestra metodología ya que todos sus procedimientos de decisión están basados en verificación heterogénea.

Análisis de la corrección del diseño

La corrección del diseño realizado puede ser observada a distintos niveles de cumplimiento y a distintos niveles de abstracción. En este sentido, se han aportado distintos procedimientos de decisión que permiten aumentar la confianza del diseñador en el prototipo obtenido.

Como de corrección mínima consideramos la definición de prototipos que no contengan bloqueos en su ejecución. En este sentido hemos adaptado las definiciones de bloqueo en tiempo y bloqueo en control comúnmente utilizadas en este tipo de sistemas. Como es intrínseco en un modelo de ciclo de vida incremental, la detección de tales bloqueos puede representar solamente una incorrección potencial, dado que, en un modelo incompleto, pueden existir elementos todavía no especificados que permitan evolucionar al sistema a un modelo sin bloqueos.

La detección de bloqueos se ha realizado sobre el grafo de simulación posible para MUS-T. Tal grafo de simulación preserva el conjunto de estados alcanzables mediante pasos discretos y temporales con condición de especificación posible. Es decir, representa el conjunto de estados alcanzables al nivel de especificación actual. Sin embargo, para posibilitar la evaluación de alternativas de diseño es necesario permitir la inspección del conjunto de comportamientos aún subespecificados en el modelo. A pesar de haber aportado las bases y algoritmos necesarios para una herramienta de simulación simbólica, es necesario dotar a esta herramienta de una interfaz que, además de facilidad de uso, aporte unas capacidades gráficas potentes.

Adicionalmente a una corrección mínima del prototipo diseñado, y a la validación mediante su ejecución simbólica, se proporciona un algoritmo de *model checking* que permite conocer el grado de satisfacción de propiedades genéricas. Tal grado de satisfacción permite al diseñador conocer lo alejado o próximo que se encuentra el modelo de satisfacer la propiedad especificada.

El algoritmo de *model checking* desarrollado opera sobre el grafo de regiones RMUS-T, computado mediante la aplicación de la equivalencia de regiones en el espacio de estados del modelo MUS-T. El algoritmo sigue una estrategia local para el cómputo de recursiones mínimas y máximas, lo que, además de evitar la exploración completa del espacio de regiones, permite la generación de contraejemplos. Sin embargo, y dado que el algoritmo sólo ha sido desarrollado como un medio de validar la metodología propuesta, el cómputo local no va acompañado de un cómputo al vuelo del grafo de regiones. Retomaremos estos puntos en la exposición de las líneas de trabajo futuro.

Síntesis incremental en el modelo propuesto

La síntesis de prototipos a partir de requisitos expresados en lógica temporal es un campo largamente estudiado en la teoría de la computación. Sin embargo, en los sistemas de tiempo real, tropieza con la naturaleza no decidible de la síntesis cuando la semántica en el lenguaje de especificación de requisitos incluye la puntualidad como una propiedad. En lo que se refiere a la síntesis, nos hemos acercado al problema desde el punto de vista de las lógicas ejecutables, incluyendo una perspectiva de síntesis que, en base a la ejecución pasada del sistema, construye el futuro esperado según la regla de síntesis especificada.

Con el objetivo de proporcionar un mecanismo de síntesis que sea, en lo posible, transparente al diseñador, se aporta un método de decisión basado en *model checking* que, dado un prototipo y una nueva propiedad a incorporar, sintetiza el nuevo prototipo, si ello es posible. Para tal fin se ha restringido la sintaxis completa de la lógica admitiendo, para síntesis, todas aquellas fórmulas SCTL que puedan ser sintetizadas sin una toma de decisiones del diseñador. Adicionalmente, la sintaxis se ha ampliado para permitir el manejo imperativo de relojes en una especificación basada en lógica, característica típicamente asociada a los formalismos orientados a modelo. Tal incorporación permite la síntesis automática, incluso para reglas de síntesis incluyendo puntualidad. Sin embargo, propiedades comúnmente utilizadas en la especificación de propiedades basada en lógica, caen fuera de las reglas de síntesis consideradas. Tal limitación del conjunto de reglas de síntesis admitidas, fuerza una síntesis basada en escenarios identificados en el sistema a sintetizar. Con el ánimo de proporcionar una metodología de síntesis completa, en el apartado de las líneas de trabajo futuro, se propone una extensión basada en el cálculo de sugerencias.

En base a la técnica de síntesis presentada, se ha definido la relación de simulación especificada que permite establecer un orden en el conjunto de prototipos generados por síntesis incremental en la metodología SCTL/MUS-T. Informalmente, un modelo sintetizado es considerado de evolución de otro modelo base, cuando contiene, al menos, todos los comportamientos "especificados" en el modelo base; pudiendo contener, a mayores, otros comportamientos "especificados" adicionales.

Además de aportar una relación de "evolución" en el dominio de los modelos MUS-T, hemos probado la corrección del algoritmo de síntesis a partir del orden parcial en nivel de conocimiento en SCTL-T. Dado que el procedimiento de síntesis, bajo un criterio RCT, respeta el nivel de conocimiento tanto de arcos discretos como de arcos temporales en el modelo, se preserva el nivel de conocimiento de cualquier fórmula SCTL. Es decir, la síntesis recoge la intuición básica de un

proceso incremental: el conocimiento sobre el prototipo en desarrollo crece fase a fase en el proceso *software*.

En la aplicación de la síntesis incremental en MUS-T es posible aplicar distintos criterios de compatibilidad entre regiones. Sin embargo, en base al caso de estudio desarrollado, podemos decir que la obtención de modelos manejables y visualmente acordes con una especificación operacional se obtiene de la aplicación de criterios RCM y RCMReset. Como se ha demostrado, dichos criterios, no respetan necesariamente el nivel de conocimiento de fórmulas genéricas, obligando a una reevaluación tanto de reglas de síntesis como de requisitos en el sistema.

Eficiencia de la metodología

Si bien el objetivo central de esta tesis no radicaba en la optimización de los procedimientos de decisión en la metodología propuesta, su desarrollo ha puesto de relieve que los métodos de decisión, basados en la construcción explícita del grafo de regiones, estaban muy lejos de garantizar de forma mínima una transferencia tecnológica real. El problema de explosión del espacio de estados asociado a los métodos de decisión, que se basan en la exploración exhaustiva de tal espacio, no es un problema exclusivo de los sistemas de tiempo real. Sin embargo, para tales sistemas, se ve fuertemente agravado por la explosión heredada del espacio de estados de control más la que tiene su origen en el manejo del tiempo en el modelo.

Con el ánimo de aliviar este problema, en el estado del arte actual se han estudiado distintas técnicas que o bien tratan de evitar la construcción explícita de la abstracción (simbólicas), o bien tratan de reducir el espacio de estados. De entre las técnicas de reducción, la minimización, centrada en encontrar una abstracción del espacio de estados a explorar que respete un conjunto de propiedades, se ajusta perfectamente a los procedimientos de decisión diseñados en la metodología. Mediante la adaptación de los procedimientos de minimización a la semántica multivalorada de un modelo MUS-T, se obtiene una abstracción que respeta o bien una propiedad SCTL-T genérica (para verificación), o bien una regla de síntesis. Con el grafo abstracto construido, los algoritmos de verificación y síntesis desarrollados son totalmente trasladables a la nueva abstracción, mediante el cambio del elemento básico sobre el que trabajan: de una región a un conjunto de estados que forman una clase de equivalencia para la relación de bisimulación de abstracción de tiempo definida.

Implementación

La metodología SCTL/MUS introducida en (García Duque, 2000) nace acompañada del diseño de una herramienta que habilite su aplicación bajo las premisas de integración y facilidad de uso.

La arquitectura propuesta para dicha herramienta supone la distribución del trabajo computacional, centrado en los métodos de decisión, las facilidades de visualización, conferidas a clientes Java, y el almacenamiento del conocimiento adquirido en la aplicación de la metodología, traducido en una base de datos orientada al mantenimiento y reutilización de prototipos MUS y requisitos SCTL.

En base a tal distribución se detallan un *Módulo de algoritmos*, un *Módulo de datos* y un *Módulo de Visualización*; cada uno de ellos, de forma aislada, siguiendo una arquitectura cliente-servidor.

Paralelamente a toda la labor teórica descrita en esta memoria, se ha realizado una labor de implementación con el objetivo de comprobar la validez de la extensión propuesta para tiempo real. Dicha labor de implementación se ha centrado en el *Módulo de algoritmos* de la herramienta.

Dado que el servidor del *Módulo de algoritmos* en la arquitectura se debe implementar en un lenguaje compilado (no interpretado), y conscientes de las ventajas de un paradigma orientado a objetos se ha seleccionado el lenguaje de programación C++. Dicha selección vino también condicionada por la posibilidad de incorporar la biblioteca de algoritmos y estructuras de datos LEDA (*Library of Efficient Data Types and Algorithms*) (Mehlhorn y Maher, 1999), beneficiándonos de la innumerables ventajas a la hora de trabajar con grafos y estructuras de datos complejas de forma eficiente.

El *Módulo de algoritmos* se ha divido en tres paquetes principales:

- Paquete bases: Consta, principalmente de la clase MUST, la clase SCTLT y la clase IPM, reflejando los elementos básicos de las bases formales en la metodología SCTL/MUS-T. La clase MUST tiene una dependencia composicional con la clase DBML del paquete lógico tiempo. Se ha dotado a las clases en este paquete lógico de una interfaz orientada a los métodos de decisión desarrollados (Paquete algoritmos).

- Paquete tiempo: Este paquete recoge los aspectos referentes al modelado y operación del tiempo en la metodología. Se han implementado dos representaciones alternativas: la clase LRelojes (expuesta brevemente en el capítulo 8) y la clase DBML (con las operaciones detalladas en el apéndice A), siendo la segunda la más general. La clase DBML se ha definido por agregación de objetos de la clase DBM, reflejando la construcción de poliedros genéricos a partir de la unión de poliedros convexos. Dado que se trata de dos representaciones alternativas del tiempo se ha definido la correspondencia, en forma de constructores, entre objetos de la clase LRelojes y la clase DBML (siempre posible, dada la naturaleza general de la segunda) y, de forma inversa y siempre que sea posible, la traducción de objetos de la clase DBML a objetos de la clase LRelojes.

- Paquete algoritmos: El paquete de algoritmos se ha estructurado en una clase por algoritmo. Cada clase, representando un algoritmo, define una interfaz pública de acceso destinada a dar cabida a la interacción con el cliente de algoritmos. La utilización de un paradigma orientado a objetos ha permitido la implementación de la clase SI (síntesis en el capítulo 9) como una subclase de la clase MCGR (*model checking* en el capítulo 8), siendo la clase SI una clase derivada que sobrecarga los métodos de verificación para la síntesis imperativa. Además y dado el algoritmo de minimización desarrollado, se ha optado por una implementación de los algoritmos de *model checking* y de síntesis operando sobre grafos genéricos en clases abstractas. Bajo la jerarquía de dicha clase abstracta, se ha implementando la subclase que opera sobre el grafo de regiones. En base a esta jerarquía, y dado el grafo

generado por el algoritmo de minimización, se posibilita la implementación directa de los algoritmos de síntesis y de *model checking* operando sobre dicho grafo.

El *Módulo de datos*, encargado de mantener la información relativa a las especificaciones SCTL/MUS, propone un almacenamiento del proceso de diseño destinado al mantenimiento y la reutilización. Con tal objetivo se identifican los elementos básicos de almacenamiento en grafos MUS y requisitos SCTL. En base a la estructura del modelo de ciclo de vida se definen incrementos de grafo e incrementos de requisitos, destinados al almacenamiento eficiente del proceso de diseño visto desde el punto de vista de la síntesis incremental en la metodología. Dada la correspondencia directa entre las extensiones con tiempo realizadas y sus versiones sin tiempo, además de las referentes a los métodos de decisión, el trabajo realizado se adapta adecuadamente a la filosofía subyacente en el *Módulo de datos*.

Por último el *Módulo de Visualización*, destinado a proporcionar una vista adecuada del marco de trabajo, no ha sido abordado en los trabajos de implementación desarrollados. Dadas las facilidades gráficas de la biblioteca LEDA, centradas en la representación y animación de grafos, se ha implementado una interfaz prototípica que permite la interacción simple con el *Módulo de algoritmos*. Sin embargo, y como ya hemos mencionado, la herramienta propone el uso de clientes Java, siendo necesario, por tanto, la actualización de los clientes de visualización para la extensión realizada.

13.2 Líneas de trabajo futuro

Durante el desarrollo de esta tesis han surgido varios aspectos no incluidos en los objetivos iniciales, algunos ya incorporados en el trabajo y otros, que directamente relacionados con los métodos de decisión de la metodología propuesta, se exponen a continuación. Además existen aspectos más alejados de los objetivos de esta tesis que merecen un trabajo de investigación más profundo.

13.2.1 Directas

Bajo este término englobamos todas aquellas líneas de trabajo futuro que son consecuencia de los trabajos expuestos en esta tesis. Estas líneas futuras se centran en la adecuación de la lógica para una mayor facilidad de uso y en la mejora de los distintos métodos de decisión propuestos.

Requisitos

Como hemos adelantado en las conclusiones, la expresión de los requisitos en lógica SCTL-T exige un cierto conocimiento y experiencia en el uso de lógicas temporales. Si bien es de esperar que este tipo de formación alcance a los ingenieros *software* en un futuro, la práctica actual de la ingeniería del *software* se encuentra bastante alejada de una satisfacción media de este objetivo. Los ingenieros *software* trabajan durante la captura de requisitos con documentos, en su mayoría, en

lenguaje natural, y con notaciones, en su mayoría visuales. Acorde a esta realidad, se plantean dos líneas de actuación principales:

- Definición de un diccionario de patrones en base a fórmulas SCTL-T parametrizables de una forma similar a la que se propone en (Gotzhein et al., 1998). Los patrones debe incluir la fórmula SCTL-T parametrizable, la explicación en lenguaje natural del requisito expresado en lógica, y distintos ejemplos de uso que lo conviertan en un almacén de la experiencia recogida de la aplicación del formalismo en un ámbito de aplicación concreto.

- Definición de una notación visual para la especificación de fórmulas SCTL-T. En este sentido encontramos notables ejemplos en (Lee y Sokolsky, 1997) con la definición de una notación gráfica para T_μ, y en (Moser et al., 1997) para una lógica de intervalo de tiempo real. El objetivo será la obtención de una notación visual que por un lado registre la relación causa-efecto implícita en un requisito SCTL y, por otro lado, permita la anotación de tales relaciones causa-efecto con la restricción de tiempo requerida. La estructura de una relación causa-efecto puede obedecer a distintos patrones de ordenamiento temporal, dicho patrón será representado bajo distintos iconos correspondiéndose con los distintos operadores básicos y operadores macros definidos. Adicionalmente, debe ser posible la definición de nuevos operadores macros en la notación visual, y la interrelación de la herramienta de notación visual con el diccionario de patrones.

Procedimientos de decisión

Como ya hemos mencionado, la validación de los algoritmos de síntesis y de verificación se ha realizado mediante la utilización del grafo de regiones. Tal práctica es impensable en un sistema de tamaño medio, o incluso en modelos reducidos cuando la composición o el número de relojes hace aumentar el espacio de estados a explorar. Para aliviar este problema hemos definido la bisimulación de abstracción de tiempo en los modelos MUS-T, permitiendo así la obtención de una abstracción finita manejable. En este sentido es necesario adaptar los algoritmos de síntesis y de verificación para que operen sobre el grafo obtenido por minimización.

Por otro lado, si bien se propone la simulación simbólica del sistema a partir de un modelo MUS-T, tal mecanismo de validación ha de ser completado para permitir al diseñador conocer la razón del no cumplimiento de una propiedad SCTL-T genérica, y aportar información (si es posible) que ayude a la toma de decisiones necesaria para la realización de esta propiedad. Para tal fin, se proponen dos soluciones con filosofías y aplicación diferente en el modelo de ciclo de vida propuesto.

En lo que se refiere a la síntesis incremental de modelos MUS-T, es nuestro objetivo aportar un algoritmo de generación de sugerencias similar al desarrollado en nuestro grupo de investigación para el tratamiento de datos en esta metodología (Gil Solla, 1999). En relación con la incorporación de datos, es posible alcanzar la integración con una perturbación mínima. Los relojes considerados en el modelo MUS-T no son más que datos que incrementan su valor al mismo ritmo, paralelamente, los datos no serán mas que relojes con una regla de variación cero.

En lo que se refiere a la identificación del no cumplimiento, se propone una generación de contraejemplos en trazas de ejecución realizada durante el cómputo del *model checking*. La información de no cumplimiento devolverá la traza (trazas) con grado de satisfacción máximo en el caso de la aplicación de una subfórmula con cuantificación universal, e inversamente, la traza (trazas) con grado de satisfacción mínimo en un contexto con cuantificación existencial. Si bien la estrategia local en el algoritmo desarrollado facilita la incorporación de contraejemplos, es necesario investigar la aplicación de algún lenguaje de simplificación de trazas, que permita la generación de trazas de un tamaño legible (Havelund et al., 1997).

El lenguaje de trazas y de sugerencias definido debe ser interpretado por el simulador simbólico, proporcionando, de esta forma, capacidades de ejecución a partir de una traza, y de modificación a partir de una sugerencia.

Sin reflejo en los objetivos de esta tesis, ni en los procedimientos de decisión aportados, apuntamos a una de las conclusiones inferidas del caso de estudio. Nos hemos encontrado con la necesidad de un mecanismo que compute distancias mínimas y máximas entre los cumplimientos de requisitos en un modelo. Si bien el algoritmo de *model checking* puede comprobar si dicha distancia obedece a cierta restricción de tiempo, no permite el cálculo del caso peor y del caso mejor en ese contexto. Dicho procedimiento de decisión puede ser ocultado, bajo *model checking*, mediante la incorporación de un reloj de especificación asociado al origen de la distancia temporal.

Conexión flexible de SCTL/MUS con SCTL/MUS-T

Para una correspondencia más suave entre ambas visiones, sería necesario una anotación más flexible en los modelos MUS-T, permitiendo que los requisitos con tiempo se conviertan en una especificación a menor nivel de abstracción que los requisitos sin tiempo. En este sentido, por ejemplo, la especificación posible de una acción en un requisito sin tiempo debería convertirse en un acción posible en un modelo MUS-T con subespecificación en tiempo. La acción es posible, pero a este nivel de abstracción no se ha llegado a identificar las restricciones de tiempo a las que dicha acción se encuentra sujeta. En el estado de desarrollo actual, tal objetivo sólo puede ser alcanzado mediante la definición de una acción parcialmente subespecificada. Sin embargo, una acción parcialmente subespecificada no se ve forzada a convertirse en posible en una evolución posterior.

13.2.2 Indirectas

Como de líneas de trabajo futuro indirectas consideramos los trabajos de investigación necesarios en la fase de refinamientos para su adaptación al ámbito del diseño de sistemas de tiempo real críticos, así como la natural extensión de la metodología para el tratamiento de sistemas híbridos y requisitos de tiempo *soft*. Adicionalmente, consideramos interesantes otros aspectos, alejados de la metodología, que podrían ser incorporados tanto en su versión sin tiempo como en su versión temporizada.

Fase de refinamientos

Esta tesis se centra en la fase del proceso *software* denominada *objetivos iniciales*. El resultado de esta macro-fase será un primer prototipo sobre el que aplicar una fase de refinamientos sucesivos, mediante la utilización de un álgebra de procesos, en la que la estructura se introduce, a mayor nivel de detalle, en el proceso. Para tal fin, consideramos del todo adecuado la utilización de un álgebra de procesos mas específica para tiempo real. Cuando hablamos de un álgebra de procesos específica para tiempo real, no nos referimos, únicamente, a un álgebra de procesos temporizada, característica mínima que se debe pedir a un álgebra de procesos para este tipo de sistemas. Nos referimos a la introducción explícita de conceptos como recursos o prioridades. En esta línea apuntamos a realizar una adecuación de E-LOTOS, álgebra de procesos en la metodología propuesta, considerando aproximaciones como TAM o ACSR.

Si bien la incorporación de prioridades puede ser llevada a la fase de refinamientos, como ya mencionamos durante la exposición del caso de estudio, resultaría del todo conveniente la introducción de prioridades en las transiciones del modelo MUS-T, que nos permitiese un modelado directo, en las primeras fases, de fenómenos como interrupciones, o una noción de urgencia más flexible.

A la hora de incorporar un álgebra de procesos en la fase de refinamientos, sería conveniente no "obligar" a la pérdida de toda la subespecificación en el modelo. En este sentido, apuntamos a una solución como la adoptada en TMS –*Timed Modal Specifications* (Cerans et al., 1993)–. TMS propone la incorporación de dos modalidades en el prefijo de acción: *may* y *must*, traducidas en la metodología propuesta, a prefijos de acción posibles o subespecificados. Dichas modalidades determinan, al igual que en un modelo operacional MUS-T, qué evoluciones son "necesarias" y qué evoluciones son "admisibles". Tal definición permitiría establecer una noción de refinamiento más relajada.

OO y composicionalidad

Como se ha inferido durante la revisión de los modelos de proceso *software* para su aplicación al ámbito de los sistemas de tiempo real, la práctica profesional adopta, mayoritariamente, una perspectiva orientada a objetos. La adecuación de un modelo de estados al paradigma orientado a objetos es, de largo, mucho más común en el estado del arte, que en el caso de los formalismos orientados a propiedad. Una máquina de estados, identificando el comportamiento interno de un objeto, puede ser introducida en un mundo pensado en objetos, simplemente mediante la definición de las interfaces de comunicación entre los distintos objetos (máquinas de estados). Sin embargo la encapsulación de datos y control no es la única base del paradigma orientado a objetos. Conceptos como herencia o polimorfismo requieren una investigación más profunda para su aplicación a un formalismo orientado a modelo. Al respecto del lenguaje de especificación de requisitos, resaltamos la lógica temporal orientada a objetos TRIO+ (Morzenti y San Prietro, 1994), avalada por su aplicación a casos de estudio reales, como un buen punto de partida en un trabajo que imprima una semántica orientada a objetos a la lógica SCTL-T.

En lo que se refiere a las características composicionales de la metodología

propuesta, es necesario investigar mejoras en los procedimientos de decisión que permitan inferir resultados a partir de los resultados obtenidos en los componentes considerados en la composición. A este respecto, y como hemos mencionado durante el caso de estudio, la información obtenida en el análisis del modelo compuesto debe ser mejorada para permitir su traducción a los distintos componentes, cuando más de uno de ellos incluye subespecificación.

Del proceso software

Relacionadas con la estructura del proceso *software* propuesta en la metodología SCTL/MUS, es natural su extensión a un modelo de ciclo de vida distribuido. En los sistemas grandes y, especialmente, en las primeras fases del ciclo de vida, es habitual la incorporación de varios agentes especificadores que trabajan, durante ciertos intervalos en el proceso, de forma paralela. El objetivo de un proceso *software* distribuido será la obtención de una especificación inicial del sistema a partir de las vistas proporcionadas por cada uno de los agentes involucrados.

En este entorno, los modelo MUS se convierten en vistas MUS proporcionadas por los distintos agentes involucrados en el proceso *software*. Y las inconsistencias, relacionando, ahora, modelos MUS en distintos ciclos del proceso, relacionarían además agentes en ciclos de vida independientes durante un intervalo temporal. Se deben delimitar hitos comunes, que en un cierto punto del proceso *software* fuercen un acuerdo entre las distintas vistas en los procesos *software* paralelos. Es decir, se debe establecer formalmente la obtención de un prototipo MUS a partir de cada una de las vistas MUS en el proceso. Tal obtención debe o bien eliminar inconsistencias entre vistas, o bien permitir temporalmente dichas inconsistencias, convirtiéndolas en inconsistencias identificadas.

Sistemas con requisitos *soft* y sistemas híbridos

Si bien el uso de métodos formales en el diseño de sistemas de tiempo real críticos se hace urgente debido a las catastróficas consecuencias de su funcionamiento incorrecto, nos encontramos habitualmente, y cada día con mayor frecuencia, con sistemas con requisitos de tiempo *soft*. Tales sistemas pueden beneficiarse, igualmente, del uso de la metodología propuesta. Por otro lado, y dado que la mayoría de los sistemas de tiempo real son embebidos, es necesaria la extensión de la metodología para que, además del tiempo, permita el modelado y análisis de magnitudes, en general magnitudes físicas, que cambian de forma continua con el tiempo.

Se plantea, de forma natural, una extensión de la metodología en ambos sentidos. Como ya hemos mencionado, el paso del tratamiento de sistemas con requisitos de tiempo a sistemas híbridos se ha realizado, en la literatura, bajo los mismos paradigmas y técnicas de decisión utilizados con éxito en el tratamiento de los primeros. La extensión, en este sentido, sería directa mediante la inclusión de reglas de variación en el modelo MUS-T y la inclusión de la parte de datos anteriormente propuesta. En lo que se refiere a los sistemas de tiempo real *soft*, la extensión buscada requiere una revisión de la semántica subyacente en el modelo MUS-T y la lógica SCTL-T. Asimismo, los métodos de decisión deberán diseñarse

teniendo en cuenta que para este tipo de sistemas es más importante una medida de los tiempos medios de respuesta que una medida de satisfacción absoluta.

PARTE VI

Apéndices y bibliografía

CAPÍTULO A

DBMs

En este capítulo se presenta la adaptación realizada en la teoría de representación de zonas de tiempo mediante DBMs, para su uso en los métodos de decisión propuestos. La diferencia fundamental reside en la introducción de valores negativos en el dominio temporal, lo que permite un tratamiento simétrico de los relojes de especificación de futuro y de pasado.

A.1 Poliedros

Dado el conjunto de restricciones $\Psi(\mathcal{C})$, se define $\Psi_A(\mathcal{C})$, conjunto de restricciones atómicas, como el subconjunto $\Psi(\mathcal{C})$ de hiperplanos en el espacio de interpretaciones del conjunto de relojes $\mathcal{C}$:

$$\Psi_A(\mathcal{C}) = \{x \prec c \mid x - x' \prec c\} \tag{A.1}$$

donde x, $x' \in \mathcal{C}$, $\prec \in \{<, \leq, >, \geq\}$ y $c \in \mathcal{Z}$, siendo $\mathcal{Z}$ el conjunto de los enteros.

Se define el conjunto de poliedros $\mathcal{P}^{\mathcal{C}}$ en el espacio de interpretaciones de $\mathcal{C}$ como todos aquellos poliedros obtenidos de $\Psi_A(\mathcal{C})$ y las operaciones de intersección y complementación. Denotaremos por $\mathcal{P}$ a cualquier elemento del conjunto $\mathcal{P}^{\mathcal{C}}$.

Se define el conjunto de poliedros convexos $\mathcal{P}^{\mathcal{C},\cap}$ como todos aquellos poliedros obtenidos de $\Psi_A(\mathcal{C})$ y la operación de intersección. Denotaremos por $\mathcal{P}_c$ a cualquier elemento del conjunto $\mathcal{P}^{\mathcal{C},\cap}$.

De la definición de convexidad se sigue que un poliedro $\mathcal{P}_c$ es convexo si y sólo si se puede definir como la intersección de un número finito de hiperplanos ($\in \mathcal{P}^{\mathcal{C},\cap}$). En caso contrario, $\mathcal{P}$ se puede describir como la unión de un número finito de poliedros convexos ($\in \mathcal{P}^{\mathcal{C}}$).

En este capítulo se presenta el almacenamiento de poliedros utilizando DBMs, así como las distintas operaciones sobre poliedros utilizadas en los algoritmos propuestos.

A.2 DBM y representación de poliedros convexos

Un DBM (*Difference Bound Matrix*) (Dill, 1989) de dimensión n es una matriz cuadrada $(n+1) \times (n+1)$, cuyos elementos pertenecen al dominio de las cotas. Una cota, para la representación matricial de un poliedro convexo, es un par $(c, \prec)$ donde $c \in \mathcal{Z} \cup \{\infty\}$ y $\prec \in \{<, \leq\}$. Para $0 \leq i, j \leq n$, denotaremos por $[D]_{i,j}$ al elemento de la fila i y columna j de D. Para un elemento $[D]_{i,j}$ denotaremos $fst([D]_{i,j}))$ al primer el elemento en el par $(c, \prec)$, y $snd([D]_{i,j})$ al segundo elemento.

La base para la representación de poliedros convexos mediante un DBM D es que cada elemento $[D]_{i,j}$ almacena el límite superior de la diferencia entre dos variables (en adelante relojes) $x_i - x_j$. Por ejemplo, la restricción atómica $x_i - x_j < 5$ se almacenará como $[D]_{i,j} = (5, <)$ y la restricción $x_i - x_j \geq 3$ como $[D]_{j,i} = (-3, \leq)$. Las cotas sobre un solo reloj se almacenan en la fila y columna 0 de la matriz; así, por ejemplo, una restricción $x_i < 5$ se representa como $[D]_{i,0} = (5, <)$ y $x_i \geq 3$ como $[D]_{0,i} = (-3, \leq)$.

En general, un poliedro convexo $\mathcal{P}_c$ de dimensión $\#\mathcal{C} = n$ definido mediante conjunción de m restricciones atómicas (es decir, $\mathcal{P}_c = \psi_1 \wedge \psi_2 \wedge \cdots \wedge \psi_m$) se almacenará en un DBM D de dimensión n siguiendo las siguientes reglas:

- Si ψ_k es una restricción atómica de tipo $x_i - x_j < c$ (resp. $\leq c$), entonces $[D]_{i,j} = (c, <)$ (resp. $[D]_{i,j} = (c, \leq)$).
- Si ψ_k es una restricción atómica de tipo $x_i - x_j > c$ (resp. $\geq c$), entonces $[D]_{j,i} = (-c, <)$ (resp. $[D]_{j,i} = (-c, \leq)$).
- Si ψ_k es una restricción atómica de tipo $x_i < c$ (resp. $\leq c$), entonces $[D]_{i,0} = (c, <)$ (resp. $[D]_{i,0} = (c, \leq)$).
- Si ψ_k es una restricción atómica de tipo $x_i > c$ (resp. $\geq c$), entonces $[D]_{0,i} = (-c, <)$ (resp. $[D]_{0,i} = (-c, \leq)$).
- $[D]_{i,i} = (0, \leq)$, para todo i representando $(x_i - x_i \leq 0) \wedge (x_i - x_i \geq 0)$ y, en consecuencia, $x_i = x_i$.
- Todos los demás elementos de D (los que no aparezcan en $\mathcal{P}_c$) serán el elemento trivial en las cotas $(\infty, <)$.

Por ejemplo, un poliedro de dimensión 2 definido como $(x_1 \leq 2) \wedge (x_2 > 3)$ se representará mediante el DBM:

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(\infty, <)$	$(-3, <)$
x_1	$(2, \leq)$	$(0, \leq)$	$(\infty, <)$
x_2	$(\infty, <)$	$(\infty, <)$	$(0, \leq)$

En sentido inverso, cualquier DBM de dimensión n define un poliedro de dimensión n sobre los relojes $\{x_1, \cdots, x_n\}$, tal que:

$$\mathcal{P}_c = (\bigwedge_{1 \leq i \neq j \leq n} x_i - x_j \prec_{snd([D]_{i,j})} fst([D]_{i,j})) \wedge$$
$$(\bigwedge_{1 \leq i \leq n} (x_i \prec_{snd([D]_{i,0})} fst([D]_{i,0}) \wedge \ x_i \prec^{-1}_{snd([D]_{0,i})} -fst([D]_{0,i})))$$

donde, para $\prec \in \{<, \leq\}$, $\prec^{-1}$ es $>$ (resp. $\geq$) sii $\prec$ es $<$ (resp. $\leq$).

Por tanto,

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(\infty, <)$	$(-3, <)$
x_1	$(2, \leq)$	$(0, \leq)$	$(-1, <)$
x_2	$(\infty, <)$	$(\infty, <)$	$(0, \leq)$

será la representación matricial del poliedro $(x_1 \leq 2) \wedge (x_2 > 3)$, ya que:

$$(x_1 \leq 2) \wedge (x_2 > 3) \equiv (x_1 \leq 2) \wedge (x_2 > 3) \wedge (x_2 - x_1 > 1)$$

Por otro lado, el DBM

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(\infty, <)$	$(-3, <)$
x_1	$(2, \leq)$	$(0, \leq)$	$(\infty, <)$
x_2	$(\infty, <)$	$(0, \leq)$	$(0, \leq)$

representa un poliedro vacío, ya que $(x_1 \leq 2) \wedge (x_2 > 3) \wedge (x_2 \leq x_1)$ es una fórmula que no puede ser satisfecha por ninguna interpretación real ($\in \Re$) del conjunto de relojes $\{x_1, x_2\}$.

A.3 Relojes de especificación

En general los algoritmos utilizados manejan, además de los relojes del modelo $\mathcal{C}$, un conjunto de relojes de especificación en los conjuntos $\mathcal{E}^f$ y $\mathcal{E}^p$. Si bien los relojes de especificación de futuro pueden ser asumidos como relojes de modelo, los relojes de especificación de pasado tienen una evolución complementaria a los primeros en los pasos temporales. Para ello, se ha optado por manejar los relojes de especificación de pasado en el dominio $\Re_-$, lo que introduce algunos cambios en la teoría de DBMs comúnmente utilizada. A lo largo de las siguientes secciones nos referiremos como $\mathcal{P}^{\mathcal{C},\cap}$ al dominio de los DBMs sobre los relojes del modelo, y $\mathcal{P}^{\mathcal{C}\cup\mathcal{E},\cap}$ al dominio de los DBMs que además incorpora un conjunto de relojes de especificación $\mathcal{E} = \mathcal{E}^f \cup \mathcal{E}^p$.

A.4 DBM canónico

Como hemos visto en los ejemplos de la sección anterior, dos o más DBMs diferentes pueden representar el mismo poliedro convexo. Esto se debe a que las cotas del DBM no están lo suficientemente ajustadas.

Sin embargo, para cualquier conjunto de restricciones sobre los relojes, existe una representación canónica única. La representación canónica de un conjunto de restricciones en un DBM se obtiene imponiendo un orden total en los elementos de la matriz. Si consideramos $<$ estrictamente menor que $\leq$, y el orden en $\mathcal{Z}$, para cualesquiera dos elementos en $\{\mathcal{Z} \times \{<, \leq\} \cup (\infty, <)\}$, se define el orden en el dominio de las cotas $\sqsubseteq$:

$$(c, \prec) \sqsubseteq (c', \prec') \text{ sii } \begin{cases} c < c' \text{ o} \\ c = c' \text{ y } \prec \text{ es menor o igual a } \prec' \end{cases}$$

La cota $(\infty, <)$ satisface $(c, \prec) \sqsubseteq (\infty, <)$, para todo $(c, \prec)$ en el dominio de las cotas $\{\mathcal{Z} \times \{<, \leq\} \cup (\infty, <)\}$.

Con esta relación de orden entre los elementos de los DBMs, se define un orden parcial $\sqsubseteq_n$ entre DBMs de dimensión n, tal que:

$$D \sqsubseteq_n D' \text{ si } \forall \ 0 \leq i, j \leq n, [D]_{ij} \sqsubseteq [D']_{ij}$$

Para cualquier DBM D (correspondiente a un poliedro no vacío $\mathcal{P}_c \in \mathcal{P}^{\mathcal{C},\cap}$, con $n = \#\mathcal{C}$), existe un DBM D' que representa el mismo poliedro $\mathcal{P}_c$, tal que para todo D'' representando $\mathcal{P}_c$, $D' \sqsubseteq_n D''$. D' se denomina forma canónica o forma mínima de D.

Una vez obtenida la forma canónica para un DBM no vacío, es necesaria una forma canónica para representar los poliedros vacíos (DBMs vacíos). Dado que existen muchos DBMs vacíos, se elige uno por convención. El DBM vacío de dimensión n, denotado como D^0, es el definido como $[D]_{i,j} = (0, <)$ para todo $i, j = 0, \cdots, n$. Identificaremos los DBMs vacíos (sección A.4.2) y los reemplazaremos por D^0. La detección de un poliedro vacío puede realizarse, como veremos en la sección A.4.3, durante el cómputo de la forma canónica del DBM.

Una representación canónica nos permite comprobar de forma sencilla la igualdad entre DBMs y los DBMs vacíos (poliedros convexos vacíos).

A.4.1 Obtención de la forma canónica

Para formalizar la noción de matriz canónica, se define la suma de dos elementos de un DBM.

Definimos la suma sobre el dominio de las cotas $\{\mathcal{Z} \times \{<, \leq\} \cup \{(\infty, <)\}$:

$$(x, r) +_D (x', r') = \begin{cases} (\infty, <) \text{ si } (x, r) \text{ o } (x', r') \text{ es } (\infty, <) \\ (x + x', \leq) \text{ si } x, x' \text{ son finitos y } r \text{ y } r' \text{ son } \leq \\ (x + x', <) \text{ si } x, x' \text{ son finitos y } r \text{ o } r' \text{ es } < \end{cases}$$

Con el orden $\sqsubseteq$ en el dominio de las cotas, el ajuste de las restricciones se obtiene observando que la suma de los límites superiores de las diferencias $x_i - x_j$ y $x_j - x_l$ es un límite superior para la diferencia $x_i - x_l$.

Un DBM de dimensión n es canónico sii para todo $0 \leq i, l \leq n$, $[D]_{il} \leq min_{0 \leq j \leq n}([D]_{ij} + [D]_{jl})$. La matriz canónica de un DBM D se puede calcular aplicando un algoritmo *all-pairs shortest path*.

A.4.2 Detección de poliedros vacíos

Un DBM de dimensión n se dice satisfactible si representa un poliedro no vacío. Un DBM D es no satisfactible si existe una secuencia $0 \leq i_1, i_2, ..., i_j \leq n$ de índices tales que $[D]_{i_1 i_2} + [D]_{i_2 i_3} + \cdots + [D]_{i_j i_1} < (0, \leq)$. Como veremos, la aplicación de un algoritmo *all-pairs shortest path* para obtener la forma canónica de un DBM permite comprobar si un DBM es o no vacío.

A.4.3 Aplicación de un algoritmo *all-pairs shortest path* para obtener la forma canónica

La aplicación de un algoritmo *all-pairs shortest path* sobre un DBM D permite comprobar si se trata de un DBM vacío y, en caso contrario, obtener su forma canónica.

La idea, para poder aplicar un algoritmo de cómputo de los caminos más cortos entre cada par de nodos de un grafo al cálculo de la forma canónica, es ver el DBM D (de dimensión n) como un grafo de $n+1$ nodos numerados de 0 a n y $(n+1) \cdot (n+1)$ arcos:

- Al arco del nodo i al nodo j le asignamos un peso $[D]_{i,j}$ [1].
- El coste de un camino $i_1, i_2, \cdots, i_m$ será la cota:

$$[D]_{i_1,i_2} + [D]_{i_2,i_3} + \cdots + [D]_{i_{m-1},i_m}$$

A partir de esta definición, un DBM será vacío si existe un ciclo $i = i_1, \cdots, i_m = i$ tal que $\mathsf{coste}(i_1, \cdots, i_m) < (0, \leq)$. Si el DBM no es vacío, estará en forma canónica sii para todo $0 \leq i, j \leq n$ no existe ningún camino $i = i_1, i_2, \cdots, i_m = j$ tal que $\mathsf{coste}(i_1, i_2, \cdots, i_m) < \mathsf{coste}(i, j)$.

Para el cálculo de la forma canónica de un DBM D, aplicamos el algoritmo *all-pairs shortest path* de *Floyd-Warshall* (Kershenbaum, 1993), cuyo pseudocódigo se muestra en el algoritmo A.1.

A.4.4 DBM universal

Por último, definimos la matriz universal D^u, en el dominio $\mathcal{P}^{\mathcal{C} \cup \mathcal{E}, \cap}$ como la matriz cuyos elementos son:

- $[D^u]_{i,j} = (0, \leq)$ si $i = j$
- Para $i = 1 \cdots \#(\mathcal{C} \cup \mathcal{E})$:

[1] Si el peso de todos los arcos es 1 encontraremos el camino con menor número de arcos entre cada par de nodos

Algoritmo A.1 Obtención de forma canónica

```
Forma_canonica(D)
{
   foreach k in 0 ··· n do
      foreach i in 0 ··· n do
         foreach j in 0 ··· n do
            D(i,j):=min(D(i,j),D(i,k)+D(k,j));
            if (D(i,i) < (0,≤)) then return D^0
            /* DBM vacio */
         end foreach
      end foreach
   end foreach
   return D
}
```

– Si $i \in \mathcal{C} \cup \mathcal{E}^f$

$$[D^u]_{i,0} = (\infty, <), [D^u]_{0,i} = (0, \leq)$$

– Si $i \in \mathcal{E}^p$

$$[D^u]_{i,0} = (0, \leq), [D^u]_{0,i} = (\infty, <)$$

- Y $[D^u]_{i,j} = (\infty, <)$ para el resto de elementos.

Para $n = 2$ y $\mathcal{E}^p$ vacío:

$D^u =$

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(\infty, <)$	$(\infty, <)$
x_1	$(0, \leq)$	$(0, \leq)$	$(\infty, <)$
x_2	$(0, \leq)$	$(\infty, <)$	$(0, \leq)$

siendo D^u un poliedro en forma canónica. Sin embargo si $\mathcal{E}^p$ es no vacío el DBM universal obtenido no es necesariamente canónico, siendo necesaria la obtención de la forma canónica. Así por ejemplo sea el DBM universal en el dominio $\mathcal{C} = x$ y $\mathcal{E}^p = y$, correspondiendo x al índice 1 del DBM e y al índice 2 del DBM:

$D^u =$

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(0, \leq)$	$(\infty, <)$
x_1	$(\infty, <)$	$(0, \leq)$	$(\infty, <)$
x_2	$(0, \leq)$	$(\infty, <)$	$(0, \leq)$

cuya forma canónica es:

$D^u =$

	x_0	x_1	x_2
x_0	$(0, \leq)$	$(0, \leq)$	$(\infty, <)$
x_1	$(\infty, <)$	$(0, \leq)$	$(\infty, <)$
x_2	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$

La matriz universal D^u en el dominio $\mathcal{C} \cup \mathcal{E}$ será el conjunto de todas las posibles interpretaciones de dichos relojes y, por tanto, equivalente a la constante *true* en $\Psi(\mathcal{C} \cup \mathcal{E})$.

A.5 Operaciones sobre DBMs

Los algoritmos de *model checking* (capítulo 8), síntesis (capítulo 9), minimización (capítulo 10), cómputo del grafo de simulación y detección de bloqueos (capítulo 11) requieren las operaciones sobre DBMs que se detallan en los siguientes apartados.

En las figuras de la tabla A.1 se muestran algunos ejemplos. En dicha tabla podemos observar que todas las operaciones sobre poliedros convexos son cerradas, excepto la unión y la diferencia (en realidad el complemento). Tanto la unión como la diferencia de poliedros convexos pueden dar como resultado poliedros no convexos. En el caso de poliedros no convexos, la representación no es canónica y, en general, las operaciones sobre dichos poliedros requieren una elevada carga computacional. Aunque el resultado de la unión o diferencia de dos poliedros convexos fuese convexa y, por tanto, pudiese ser reemplazada por un único poliedro convexo, el realizar esta comprobación en las iteraciones de algoritmos con dichas operaciones resultaría demasiado costoso. Para el tratamiento de poliedros no convexos, algunas herramientas recurren a representar dichas zonas como una lista de DBMs. Cuanto más "no convexo" sea el poliedro, más matrices serán necesarias para representarlo, lo que hará que la prueba de equivalencia entre poliedros y la eliminación de redundancia en listas de DBMs sea una tarea más compleja. Una de las líneas más exploradas como solución a la no convexidad de la unión de poliedros convexos es la de utilizar aproximaciones convexas a la unión, por ejemplo mediante la operación *convex hull*. Mediante la operación *convex hull*, se aproxima la unión entre poliedros por el menor poliedro convexo conteniendo a los dos poliedros que intervienen en la unión.

A.5.1 Intersección o conjunción

$$\mathcal{P}_c \cap \mathcal{P}_c{}' \qquad \text{o} \qquad \psi \wedge \psi'$$

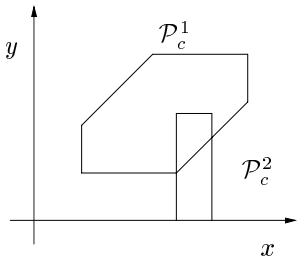
La intersección de dos DBMs es un DBM (la intersección de dos poliedros convexos es un poliedro convexo). Sean D y D' los DBMs de dos poliedros convexos $\mathcal{P}_c$ y $\mathcal{P}_c{}'$. El DBM $D \cap D'$, que representa el poliedro convexo $\mathcal{P}_c \cap \mathcal{P}_c{}'$, será:

$$\forall\ i, j\ [D \cap D']_{i,j} = \min\{[D]_{ij}, [D']_{ij}\}$$

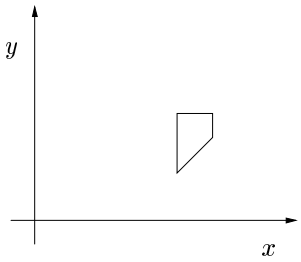
La intersección de dos DBMs canónicos no es necesariamente un DBM canónico, e incluso puede ser un DBM vacío; por tanto, una vez computada la intersección de dos DBMs debe realizarse el cálculo de la forma canónica.

A.5.2 Inclusión o implicación

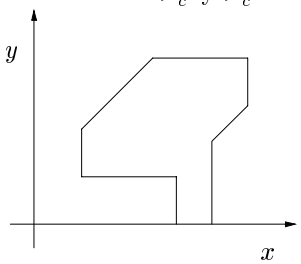
$$\mathcal{P}_c \subseteq \mathcal{P}_c{}' \qquad \text{o} \qquad \psi \rightarrow \psi'$$



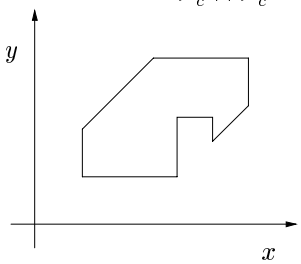
Poliedros $\mathcal{P}_c^1$ y $\mathcal{P}_c^2$



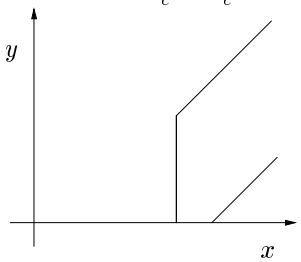
Intersección $\mathcal{P}_c^1 \cap \mathcal{P}_c^2$



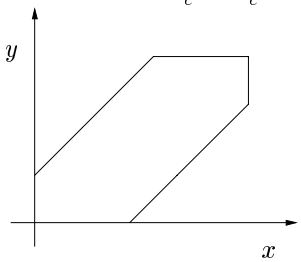
Unión $\mathcal{P}_c^1 \cup \mathcal{P}_c^2$



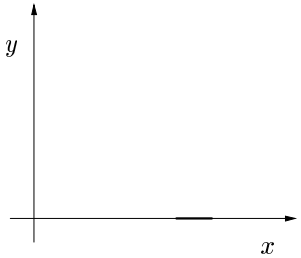
Diferencia $\mathcal{P}_c^1 - \mathcal{P}_c^2$



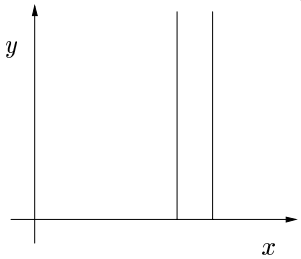
Sucesores en tiempo de $\uparrow(\mathcal{P}_c^2)$



Predecesores en tiempo de $\downarrow(\mathcal{P}_c^1)$



Reset sobre y de $\mathcal{P}_c^2 : R_y(\mathcal{P}_c^2)$



PreReset sobre y de $\mathcal{P}_c^2 : R_y^{-1}(\mathcal{P}_c^2)$

Tabla A.1. Operaciones sobre poliedros convexos

El poliedro $\mathcal{P}_c$ (representado por el DBM D) está incluido en el poliedro $\mathcal{P}_c{}'$ (representado por el DBM D') sii:

$$\forall\ 0 \leq i, j \leq n\ \ [D]_{i,j}\ \sqsubseteq [D']_{i,j}$$

A.5.3 Sucesores en tiempo

$$\uparrow(\mathcal{P}_c) \qquad \text{o} \qquad \psi \xrightarrow{\tau} \psi'$$

Cuando el tiempo se incrementa ($\xrightarrow{\tau}$), las diferencias entre pares de relojes permanecen inalterables, ya que todos los relojes se incrementan al mismo ritmo. Los límites inferiores tampoco cambian, ya que el valor de los relojes no se decrementa. Por contra, los límites superiores en los relojes deben ser llevados a ∞, ya que el tiempo que puede transcurrir es arbitrariamente grande.

Para un DBM D representando el poliedro convexo $\mathcal{P}_c$, el conjunto de sucesores en tiempo $\uparrow(D)$ se obtiene a partir de D eliminando todas las restricciones del tipo $x \leq k$ o $x < k$ que aparecen en la definición del poliedro $\mathcal{P}_c$. El resultado de esta operación en un DBM canónico es el DBM canónico del conjunto de sucesores en tiempo. El DBM resultante que representa el conjunto $\uparrow(D)$ se obtiene haciendo $D_{i,0} = (\infty, <)\ \forall\ i$.

Cuando el DBM corresponde a un dominio con $\mathcal{E}^p$ no vacío, el cálculo de los sucesores en tiempo descarta el valor obtenido para dichos relojes, dado que son relojes de sólo pasado nunca serán evaluados después de un paso hacia delante.

A.5.4 Predecesores en tiempo

$$\downarrow(\mathcal{P}_c) \qquad \text{o} \qquad \psi' \underset{\tau}{\rightarrow} \psi$$

Para un poliedro convexo $\mathcal{P}_c$, representado por el DBM D, la operación predecesores en tiempo, $\downarrow D$, obtiene el poliedro que contiene el conjunto de interpretaciones que pueden alcanzar alguna interpretación $\in \mathcal{P}_c$ mediante un incremento temporal arbitrario.

En el DBM resultante, como en el caso de los sucesores en tiempo, las diferencias entre relojes permanecen inalterables, pero desaparecen los límites inferiores. La operación $\downarrow$ sobre DBMs no respeta necesariamente la calidad canónica de un DBM.

Por tanto, el DBM $\downarrow D$ (no necesariamente canónico) sobre un dominio $\mathcal{C} \cup \mathcal{E}^f \cup \mathcal{E}^p$, se calcula:

$$\forall\ j\ \begin{cases} j \in \mathcal{C} \cup \mathcal{E}^f\ \ [\downarrow D]_{0,j} = (0, \leq) \\ j \in \mathcal{E}^p\ [\downarrow D]_{0,j} = (\infty, <) \end{cases}$$
$$\forall\ j, i \neq 0\ [\downarrow D]_{i,j} = [D]_{i,j}$$

A.5.5 *until*

El operador *until* permite el cómputo de predecesores temporales bajo una condición de especificación posible o subespecificada. El operador $until(D, D')$ calcula el poliedro que contiene el conjunto de interpretaciones que pueden alcanzar alguna interpretación en D, mediante interpretaciones intermedias contenidas en D'.

$$until(D, D') = \{\gamma \in D' \mid \exists\, \gamma' \in D \text{ tal que } \gamma + \tau = \gamma' \text{ y } \forall \tau' \leq \tau, \gamma + \tau' \in D'\}$$

y por tanto $until(D, D') = \downarrow (D \cap D') \cap D'$.

A.5.6 *since*

El operador *since* permite el cómputo de sucesores temporales bajo una condición de especificación posible o subespecificada. El operador $since(D, D')$ calcula el poliedro que contiene el conjunto de interpretaciones que pueden ser alcanzadas desde alguna interpretación en D, mediante interpretaciones intermedias contenidas en D'.

$$since(Z, Z') = \{\gamma \in Z' \mid \exists\, \gamma' \in Z \text{ tal que } \gamma' + \tau = \gamma \text{ y } \forall \tau' \leq \tau, \gamma + \tau' \in Z'\}$$

y por tanto $since(D, D') = \uparrow (D \cap D') \cap D'$.

A.5.7 Reset de un conjunto de relojes

Para poder encontrar, por ejemplo, el conjunto de sucesores tras una transición discreta, necesitamos computar la imagen sobre un conjunto de interpretaciones de una acción que lleve asociado un conjunto de relojes del sistema a reiniciar. Sea λ un subconjunto de relojes $\lambda \subseteq \mathcal{C}$ a reiniciar en un poliedro convexo $\mathcal{P}_c$ representado por un DBM D. Entonces, $R_\lambda(D)$, será la proyección del poliedro $\mathcal{P}_c$ sobre los ejes definidos por el subconjunto de relojes λ.

Para obtener el DBM $R_\lambda(D)$ se descartan todas las restricciones que involucren los relojes en el subconjunto λ y se añaden las restricciones $x = 0$ para todo reloj $x \in \lambda$. Por tanto, sea $\lambda = \{x_i\}$ un subconjunto de relojes x_i (que suponemos el i elemento de la matriz), el DBM $R_\lambda(D)$ será tal que:

$$\begin{array}{ll} \forall\ j = 1 \cdots n\ ,\ i \mid x_i \in \lambda, & [R_\lambda(D)]_{i,j} = (\infty, <) \text{ y } [R_\lambda(D)]_{ji} = (\infty, <) \\ \forall\ i \mid x_i \in \lambda, & [R_\lambda(D)]_{i,0} = [R_\lambda(D)]_{0,i} = (0, \leq) \end{array}$$

Una vez computado $R_\lambda(D)$, debemos aplicar el algoritmo de obtención de la forma canónica.

A.5.8 PreReset

La operación R_λ^{-1} es la operación dual de R_λ. Dado un poliedro convexo $\mathcal{P}_c$ representado por un DBM D, la operación R_λ^{-1} obtiene la representación del

poliedro convexo $\mathcal{P}_c{}'$ formado por todas aquellas interpretaciones γ que poniendo a cero los relojes en λ (γ_λ), están incluidas en $\mathcal{P}_c$. Es decir:

$$\gamma \in \mathcal{P}_c{}' \text{ sii } \gamma_\lambda \in \mathcal{P}_c$$

Para obtener el DBM $R_\lambda^{-1}(D)$ se descartan todas las restricciones que involucren los relojes en el subconjunto λ y se añaden las restricciones $x < \infty$ para todo reloj $x \in \lambda$. Por tanto, sea $\lambda = \{x_i\}$ un subconjunto de relojes x_i (que suponemos el i elemento de la matriz), el DBM $R_\lambda^{-1}(D)$ será tal que:

$$\forall\ j = 1 \cdots n\ ,\ i \mid x_i \in \lambda,\ [R_\lambda^{-1}(D)]_{i,j} = (\infty, <) \text{ y } [R_\lambda^{-1}(D)]_{j,i} = (\infty, <)$$

Una vez computado $R_\lambda^{-1}(D)$, debemos aplicar el algoritmo de obtención de la forma canónica.

A.5.9 *cierre*

El cierre de un poliedro, $cierre(D, k)$, representado por su DBM D con respecto a $k \in \Re_+$, se obtiene considerando $x_i - x_j \leq c$ con $c > k$ equivalente a $x_i - x_j < \infty$, y $x_i - x_j \geq c$ con $c > k$ equivalente a $x_i - x_j > k$. Dicha operación representa la equivalencia de interpretaciones en un modelo temporizado en el que la mayor constante involucrada en una restricción de tiempo es k. Por tanto, definimos $cierre(D, k)$, como:

$$\forall i, \forall j = 0 \cdots i-1$$

$$[D]_{i,j} = (c, \prec) \begin{cases} c > k, [cierre(D,k)]_{i,j} = (\infty, <) \\ c < -k, [cierre(D,k)]_{i,j} = (-k, <) \\ \text{en otro caso } [cierre(D,k)]_{i,j} = [D]_{i,j} \end{cases}$$

$$[D]_{j,i} = (c, \prec) \begin{cases} -c > k, [cierre(D,k)]_{i,j} = (-k, <) \\ -c < -k, [cierre(D,k)]_{i,j} = (\infty, <) \\ \text{en otro caso } [cierre(D,k)]_{j,i} = [D]_{j,i} \end{cases}$$

A.5.10 Unión o disyunción

$$\mathcal{P}_c \cup \mathcal{P}_c{}' \qquad \textbf{o} \qquad \psi \vee \psi'$$

Evidentemente, la unión de dos DBMs no es necesariamente un DBM; es decir, el conjunto de DBMs no es cerrado bajo disyunción, o en otras palabras, la disyunción de dos restricciones no define necesariamente un poliedro convexo. Se representa la disyunción de D y D' como el conjunto $\{D, D'\}$. El trabajo computacional para determinar si dos conjuntos de DBMs representan la misma restricción es elevado, dado que los poliedros no convexos no admiten una representación canónica obvia. Veamos, como ejemplo, el poliedro no convexo de la figura A.1. Dicho poliedro no convexo puede ser descompuesto en uniones de poliedros convexos de varias formas. En (Tripakis, 1998) se conjetura que, si bien se podría definir un orden, la carga computacional de dicha forma canónica sería,

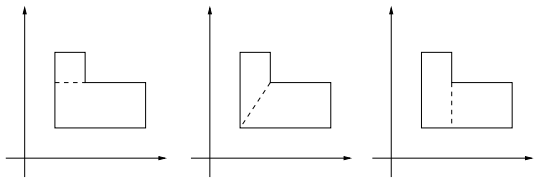


Figura A.1. Poliedro no convexo

por lo menos, tan fuerte como la carga computacional de los algoritmos que no consideran representación canónica.

A.6 Representación de poliedros no convexos: listas de DBMs

Las únicas operaciones que no conservan la convexidad de un poliedro convexo serán la unión, la complementación y la diferencia. Representaremos un poliedro no convexo mediante un conjunto de poliedros convexos. Dado que tanto diferencia como la unión de poliedros convexos se pueden reducir a la complementación, consideramos la complementación como el origen de la no convexidad en el mundo de los poliedros.

- $\mathcal{P} \cup \mathcal{P}' = \overline{\overline{\mathcal{P}} \cap \overline{\mathcal{P}'}}$
- $\mathcal{P} - \mathcal{P}' = \mathcal{P} \cap \overline{\mathcal{P}'}$

A.6.1 Complemento

De la sección anterior se deduce que es suficiente con considerar la complementación de poliedros convexos. El poliedro (posiblemente no convexo) complementario de un poliedro convexo $\mathcal{P}_c$ será la unión de los complementos de los hiperplanos que delimitan $\mathcal{P}_c$.

Para calcular la lista de DBMs $D_1, \cdots, D_k$, representando el poliedro $\overline{\mathcal{P}_c}$, a partir del DBM D representando al poliedro convexo $\mathcal{P}_c$, tenemos que complementar todas las cotas no triviales de D. Sean $\{(c_1, \prec_1), \cdots, (c_k, \prec_k)\}$ el conjunto de cotas no triviales en $[D]_{i,j} \neq (\infty, <)$, con $i \neq j$ en el DBM D. Complementamos los k hiperplanos representados por dichas cotas y, dado que, la complementación de un hiperplano es convexa, representamos cada complemento por un DBM. O sea, para todo $(c_i, r_i) \in \{(c_1, \prec_1), \cdots, (c_k, \prec_k)\}$:

$$\begin{array}{ll} \text{si } [D]_{i,j} = (c, <), & \overline{x_i - x_j < c} = x_i - x_j \geq c \\ \text{si } [D]_{i,j} = (c, \leq), & \overline{x_i - x_j \leq c} = x_i - x_j > c \end{array}$$

Con los k poliedros convexos obtenidos de la complementación de los elementos en $\{(c_1, \prec_1), \cdots, (c_k, \prec_k)\}$, aplicamos un algoritmo de inclusión entre todos los pares para eliminar la posible redundancia; la lista resultante será la representación del poliedro no convexo.

A.6.2 Intersección

La intersección de dos listas de poliedros convexos,

$$\mathcal{P} = \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_k \text{ y } \mathcal{P}' = \mathcal{P}_1{}' \cup \cdots \cup \mathcal{P}_l{}'$$

representados por los DBMs:

$$D_1, \cdots, D_k \text{ y } D_1{}', \cdots, D_l{}'$$

se obtiene:

$$\mathcal{P} \cap \mathcal{P}' = \bigcup_{1 \leq i \leq k, 1 \leq j \leq l} \mathcal{P}_i \cap \mathcal{P}_j$$

A.6.3 Inclusión

La inclusión se reduce a intersección y complementación. Comprobar $\mathcal{P} \subseteq \mathcal{P}'$ es equivalente a comprobar $\mathcal{P} \cap \overline{\mathcal{P}'} = \{\}$. Para ello, se calcula la lista de DBMs representando $\mathcal{P} \cap \overline{\mathcal{P}'}$ y se comprueba que la forma canónica de cada uno de ellos es el DBM vacío.

O sea, dados dos poliedros no convexos $\mathcal{P} = \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_k$ y $\mathcal{P}' = \mathcal{P}_1{}' \cup \cdots \cup \mathcal{P}_l{}'$,

$$\begin{aligned} \mathcal{P} \subseteq \mathcal{P}' \qquad & \text{sii } \{\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_k\} \cap \{\bar{\mathcal{P}_1}{}' \cap \cdots \cap \bar{\mathcal{P}_l}{}'\} = \{\} \\ & \text{y, por tanto } \forall\, \mathcal{P} \in \bigcup_{1 \leq i \leq k, 1 \leq j \leq l} \mathcal{P}_i \cap \mathcal{P}_j, \mathcal{P} = \{\} \end{aligned}$$

Dado que la expresión anterior se puede representar como una lista de poliedros convexos, se comprueba que cada uno de los poliedros convexos en la unión es un poliedro vacío.

A.6.4 Reset, Sucesores, Predecesores, PreReset y cierre

Todas estas operaciones (op), definidas sobre poliedros convexos, son distributivas respecto a la unión, cumpliendo:

$$op(\mathcal{P}_1 \cup \mathcal{P}_2 \cdots \cup \mathcal{P}_n) = op(\mathcal{P}_1) \cup op(\mathcal{P}_2) \cdots \cup op(\mathcal{P}_n)$$

A.6.5 *until* y *since*

Para el caso de invariantes convexas, que ha sido el implementado para la minimización, se cumple:

- $until(\mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{P}) = until(\mathcal{P}_1, \mathcal{P}) \cup until(\mathcal{P}_2, \mathcal{P})$.
- $since(\mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{P}) = since(\mathcal{P}_1, \mathcal{P}) \cup until(\mathcal{P}_2, \mathcal{P})$.

En el caso invariantes no convexas podemos encontrar el cálculo en (Olivero, 1994).

A.6.6 Reducción

Como se sigue de las operaciones definidas sobre poliedros no convexos, el número de DBMs obtenidos puede llegar a ser elevado. Por tanto, será conveniente aplicar una reducción mínima basada en inclusión a cada poliedro no convexo representado por un conjunto de DBMs. Otra reducción más costosa es la presentada en el capítulo 9 para la obtención de las guardas de tiempo en el modelo sintetizado.

A.7 Expresión de guardas de tiempo de MUS-T en DBMs

El modelo MUS-T asocia a sus estados y arcos restricciones de tiempo definidas por la sintaxis:

⟨*restricción*⟩ ::= ⟨*restricción*⟩ AND ⟨*restricción*⟩
| ⟨*restricción*⟩ OR ⟨*restricción*⟩
| NOT ⟨*restricción*⟩
| c ⟨*relación*⟩ reloj
| reloj ⟨*relación*⟩ c
| reloj - reloj ⟨*relación*⟩ c
| c ⟨*relación*⟩ reloj - reloj
| TRUE
| FALSE

⟨*relacion*⟩ ::= > | < | <= | >= | =

Definición A.1. **Operador $\mathcal{P}$.** Para $\mathcal{M}$ un modelo MUS-T con un conjunto de relojes $\mathcal{C}$, el operador $\mathcal{P} : \Psi \to \mathcal{P}^{\mathcal{C}}$, asocia a cualquier predicado de tiempo ψ del modelo, un conjunto de DBMs en el espacio $\mathcal{P}^{\mathcal{C},\cap}$. Dicho conjunto es el resultado de la aplicación de una forma normal disyuntiva a ψ.

Cualquier guarda del tipo ⟨*restricción*⟩ debe ser expresada como

$$\bigvee_i (\bigwedge_j (x - x' \prec c)_j)_i$$

donde $\prec \in \{<, \leq\}$ y tanto x como x' forman parte de los relojes del sistema más el reloj adicional x_0 y c es una constante en $(\mathcal{Z} \cup \infty)$. Cada una de las j conjunciones lógicas $(\bigwedge_j (x - x' \prec c)_j)$ que intervienen en cada uno de los i elementos de $\bigvee$ puede almacenarse en un DBM, de forma que cualquier guarda de tiempo será resultado de aplicar la función $\bigvee$ sobre un conjunto de DBMs (como ya hemos visto, no necesariamente un DBM).

Para poder llegar a la expresión de una guarda de tipo ⟨*restricción*⟩ como una disyunción de conjunciones de restricciones simples debemos reducir la expresión

original que aparece en la sintaxis del modelo a su forma normal disyuntiva. Siendo ψ^s las restricciones simples en la siguiente gramática:

$\langle\psi^s\rangle$ ::= c $\langle$*relación*$\rangle$ reloj
| reloj $\langle$*relación*$\rangle$ c
| reloj - reloj $\langle$*relación*$\rangle$ c
| c $\langle$*relación*$\rangle$ reloj - reloj
| NOT $\langle\psi^s\rangle$
| TRUE
| FALSE

Para ψ^s una restricción simple se aplican las siguientes reglas:

$$
\begin{array}{ll}
(x < c) & \rightarrow (x - x_0 < c) \\
(x \leq c) & \rightarrow (x - x_0 \leq c) \\
(x = c) & \rightarrow (x - x_0 = c) \rightarrow (x - x_0 \leq c) \wedge (x - x_0 \geq c) \\
(x > c) & \rightarrow (x - x_0 > c) \rightarrow (x_0 - x < -c) \\
(x \geq c) & \rightarrow (x - x_0 \geq c) \rightarrow (x_0 - x \leq -c) \\
(x - x' < c) & \rightarrow (x - x' < c) \\
(x - x' \leq c) & \rightarrow (x - x' \leq c) \\
(x - x' = c) & \rightarrow (x - x' \leq c) \wedge (x - x' \geq c) \\
(x - x' > c) & \rightarrow (x' - x < -c) \\
(x - x' \geq c) & \rightarrow (x' - x \leq -c)
\end{array}
$$

Además, definimos la negación de restricciones simples, NOT (ϕ^s), como:

$$
\begin{array}{ll}
\text{NOT } (x < c) & \rightarrow (x \geq c) \\
\text{NOT } (x \leq c) & \rightarrow (x > c) \\
\text{NOT } (x = c) & \rightarrow (x < c) \vee (x > c) \\
\text{NOT } (x > c) & \rightarrow (x \leq c) \\
\text{NOT } (x \geq c) & \rightarrow (x < c) \\
\text{NOT } (x - x' < c) & \rightarrow (x - x' \geq c) \\
\text{NOT } (x - x' \leq c) & \rightarrow (x - x' > c) \\
\text{NOT } (x - x' = c) & \rightarrow (x - x' < c) \vee (x - x' > c) \\
\text{NOT } (x - x' > c) & \rightarrow (x - x' \leq c) \\
\text{NOT } (x - x' \geq c) & \rightarrow (x - x' < c)
\end{array}
$$

Una $\langle$*restricción*$\rangle$ estará en forma normal disyuntiva (FND) si sigue el patrón:

$$\psi_1 \vee \cdots \vee \psi_n$$

con $\psi_i = \psi_1^s \wedge \cdots \wedge \psi_m^s$, siendo ψ_i^s una restricción simple o la negación de una restricción simple. Por tanto, para ψ_i^s restricciones simples y ψ_i restricciones:

- FND(ψ^s) = ψ^s
- FND(NOT ψ^s) = NOT ψ^s
- FND(ψ_1^s AND ψ_2^s) = ψ_1^s AND ψ_2^s
- FND(ψ_1^s OR ψ_2^s) = ψ_1^s OR ψ_2^s
- FND(NOT (ψ_1 OR ψ_2)) = FND(NOT ψ_1) AND FND (NOT ψ_2)
- FND(NOT (ψ_1 AND ψ_2)) = FND(NOT ψ_1) OR FND (NOT ψ_2)
- FND(ψ_1 AND (ψ_2 OR ψ_3)) = FND(ψ_1 AND ψ_2) OR FND(ψ_1 AND ψ_3)
- FND((ψ_1 OR ψ_2) AND ψ_3) = FND(ψ_1 AND ψ_3) OR FND(ψ_2 AND ψ_3)

Una vez obtenida una disyunción de conjunciones de restricciones simples, almacenamos la $\langle$*restricción*$\rangle$ como una lista de DBM $\{D_1, \cdots D_i\}$ donde D_k será la representación en DBM de $(\bigwedge_j (x - x' \prec c)_j)_k$.

Bibliografía

Abrial, J. R., Börger, E. y Langmaack, H. (1996). *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volumen 1165 de *Lecture Notes in Computer Science*. Springer Verlag.

Ajami, K., Haddad, S. y Ilie, J. M. (1998). Exploiting Symmetry in Linear Temporal Logic Model Checking: One Step Beyond. En Steffen, B., editor, *Tools and Algorithms for the Construction and Analysis of Systems, 4th International Conference (TACAS'98)*, volumen 1384 de *Lecture Notes in Computer Science*, páginas 52–67. Springer Verlag.

Altisen, K., Gobler, G. y Sifakis, J. (2000). A Methodology for the Construction of Scheduled Systems. En *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)*.

Altisen, K., Goessler, G., Pnueli, A., Sifakis, J., Tripakis, S. y Yovine, S. (1999). A Framework for Scheduler Synthesis. En *Real-Time Systems, 20th IEEE Symposium (RTSS'99)*, páginas 154–163. IEEE Computer Society Press.

Alur, R. (1991). *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Department of Computer Science, Stanford University.

Alur, R. (1999). Timed Automata. En Halbwachs, N. y Peled, D., editores, *Computer-Aided Verification, 11th International Conference (CAV '99)*, volumen 1633 de *Lecture Notes in Computer Science*, páginas 8–22. Springer Verlag.

Alur, R., Courcoubetis, C. y Dill, D. (1993a). Model Checking in Dense Real-time. *Information and Computation*, 104(1):2–34. Versión preliminar en Proceedings of the 5th IEEE International Symposium on Logic in Computer Science (LICS'90).

Alur, R., Courcoubetis, C., Halbwachs, N., Dill, D. L. y Wong Toi, H. (1992). Minimization of Timed Transition Sytems. En Cleaveland, W. R., editor, *Concurrency Theory, 3rd International Conference (CONCUR'92)*, volumen 630 de *Lecture Notes in Computer Science*, páginas 340–354. Spinger Verlag.

Alur, R., Courcoubetis, C., Henzinger, T. A. y Ho, P. (1993b). Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. En Grossman, R., Nerode, A., Ravn, A. y Rischel, H., editores, *Hybrid Systems*, volumen 736 de *Lecture Notes in Computer Science*, páginas 209–229. Springer Verlag.

Alur, R. y Dill, D. (1990a). A Theory of Timed Automata. En *Automata, Languages and Programming, 17th International Colloquium (ICALP'90)*.

Alur, R. y Dill, D. (1990b). Automata for Modelling Real Time Sytems. En Paterson, M. S., editor, *Automata, Languages and Programming, 17th International Colloquium (ICALP'90)*, volumen 443 de *Lecture Notes in Computer Science*, páginas 322–335. Springer Verlag.

Alur, R., Feder, T. y Henzinger, T. A. (1996). The Benefits of Relaxing Punctuality. *Journal of the ACM*, 43(1):116–146. Por primera vez en *Proc. of the 10th Annual Symposium on Principles of Distributed Computing*, 1991.

Alur, R., Fix, L. y Henzinger, T. (1994). A Determinizable Class of Timed Automata. En Dill, D. L., editor, *Computer-Aided Verification, 6th International Conference (CAV'94)*, volumen 818 de *Lecture Notes in Computer Science*, páginas 1–13. Springer Verlag.

Alur, R., Fix, L. y Henzinger, T. A. (1999). Event-clock Automata: A Determinizable Class of Timed Automata. *Theoretical Computer Science*, 211(1-2):253–273.

Alur, R. y Henzinger., T. A. (1989). A Really Temporal Logic. En *Foundations of Computer Science, 30th Annual Symposium (FOCS'89)*, páginas 164–169. IEEE Computer Society Press.

Alur, R. y Henzinger, T. A. (1992a). Back to the Future: Towards a Theory of Timed Regular Languages. En *Foundations of Computer Science, 33rd Annual IEEE Symposium (FOCS'92)*, páginas 177–186.

Alur, R. y Henzinger, T. A. (1992b). Logics and Models of Real Time: A Survey. En de Bakker, J., Huizing, K., de Roever, W. P. y Rozenerg, G., editores, *Real Time: Theory and Practice, REX Workshop*, volumen 600 de *Lecture Notes in Computer Science*, páginas 74–106. Springer Verlag.

Alur, R. y Henzinger, T. A. (1993). Real-time Logics: Complexity and Expressiveness. *Information and Computation*, 104(1):35–77.

Alur, R., Henzinger, T. A. y Ho, P. H. (1993c). Automatic Symbolic Verification of Embedded Systems. En *Real-time Systems, 14th Annual IEEE Symposium*, páginas 2–11. IEEE Computer Society Press.

Alur, R., Itai, A., Kurshan, R. P. y Yannakakis, M. (1995). Timing Verification by Successive Approximation. *Information and Computation*, 118(1):142–157.

Alur, R. y Kurshan, R. P. (1996). Timing Analysis in COSPAN. En Alur, R., Henzinger, T. y Sontag, E., editores, *Hybrid Systems III: Control and Verification, DIMACS/SYCON Workshop*, volumen 1066 de *Lecture Notes in Computer Science*, páginas 220–231. Springer Verlag.

Andersen, J. H., Kristoffersen, K. J., Larsen, K. G. y Niedermann, J. (1995). Automatic Synthesis of Real Time Systems. En Fülöp, Z. y Gécseg, F., editores, *Automata, Languages and Programming, 22nd International Colloquium (ICALP'95)*, volumen 944 de *Lecture Notes in Computer Science*, páginas 535–546. Springer Verlag.

Angilotti, P., Buttazzo, G., Natale, M. D. y Spuri, M. (1998). Design and Programming Tools for Time Critical Applications. *Real Time Systems*, (14):251–267.

Archer, M. y Heitmeyer, C. (1996). TAME: A Specialized Specification and Verification System for Timed Automata. En Bestavros, A., editor, *Real-Time Systems, 17th IEEE Symposium (RTSS'96)*, volumen 670, páginas 3–6. Work in progress.

Asarin, E., Bozga, M., Kerbrat, A., Maler, O., Pnueli, A. y Rasse, A. (1997). Data-Structures for the Verification of Timed Automata. En Maler, O., editor, *Hybrid and Real-Time Systems, International Workshop (HART'97)*, volumen 1201 de *Lecture Notes in Computer Science*, páginas 346–360. Springer Verlag.

Asarin, E., Maler, O. y Pnueli, A. (1995). Symbolic Controller Synthesis for Discrete and Timed Systems. En Antsaklis, P., Kohn, W., Nerode, A. y Sastry, S., editores, *Hybrid Systems II*, volumen 999 de *Lecture Notes in Computer Science*, páginas 1–20. Springer Verlag.

Asarin, E., Maler, O. y Pnueli, A. (1998a). On Discretization of Delays in Timed Automata and Digital Circuits. En de Simone, R. y Sangiorgi, D., editores, *Concurrency Theory, 9th International Conference (CONCUR'98)*, volumen 1466 de *Lecture Notes in Computer Science*, páginas 470–484. Springer Verlag.

Asarin, E., Maler, O., Pnueli, A. y Sifakis, J. (1998b). Controller Synthesis for Timed Automata . En *System Structure and Control, IFAC Symposium*, páginas 469–474. Elsevier.

Barringer, H., Fisher, M., Gabbay, D., Gough, G. y Owens, R. (1995). METATEM: An Introduction. *Formal Aspects of Computing*, 7(5):533–549.

Beckert, B., Hähnle, R., Oel, P. y Sulzmann, M. (1996). The Tableu-based Theorem Prover 3 TAP, version 4.0. En McRobbie, M. y Slaney, J., editores, *Automated Deduction, 13th International Conference*, volumen 1104 de *Lecture Notes in Computer Science*, páginas 303–307. Springer Verlag.

Belnap, N. D. (1977). A Useful Four-valued Logic. En Epstein, G., editor, *Modern Uses of Multiple-valued Logic*, páginas 8–37. Reidel.

Bengtsson, J., Griffioen, W. O. D., Kristoffersen, K. J., Larsen, K. G., Larsson, F., Pettersson, P. y Yi, W. (1999). Automated Analysis of an Audio Control Protocol Using UPPAAL. Technical report, Uppsala University.

Bengtsson, J., Jonsson, B., Lilius, J. y Yi, W. (1998). Partial Order Reductions for Timed Systems. En *9th International Conference on Concurrency Theory*, Nice, France.

Bengtsson, J., Larsen, K. G., Larsson, F., Pettersson, P. y Yi, W. (1996). UPPAAL: a Tool Suite for Automatic Verification of Real-time Systems. En Alur, R., Henzinger, T. A. y Sontag, E. D., editores, *Hybrid Systems III: Verification and Control, DIMACS/SYCON Workshop*, volumen 1066 de *Lecture Notes in Computer Science*, páginas 232–243. Springer Verlag.

Berthomieu, B. y Diaz, M. (1991). Modelling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273.

Bjøner, N., Browne, A., Chnag, E., Colón, M., Kapur, A., Manna, Z., Sipma, H. B. y Uribe, T. E. (1996). STeP: Deductive-Algorithmic Verification of Reactive and Real-time Systems. En Alur, R. y Henzinger, T. A., editores, *Computer Aided Verification, 8th International Conference (CAV'96)*, volumen 1102 de *Lecture Notes in Computer Science*, páginas 415–418. Springer Verlag.

Bolc, L. y Borowik, P. (2000). *Many-Valued Logics 2: Automated Reasoning and Practical Applications*. Springer Verlag.

Bolognesi, T. y Brinksma, E. (1989). Introduction to the ISO Specification Language LOTOS. En van Eijk, P. H. J., Vissers, C. A. y Diaz, M., editores, *The Formal Description Technique LOTOS*, páginas 23–73. Elsevier Science Publishers North-Holland.

Bornot, S. y Sifakis, J. (1997). Relating Time Progress and Deadlines in Hybrid Systems. En Maler, O., editor, *Hybrid and Real Time Systems, International Workshop (HART'97)*, volumen 1201 de *Lecture Notes in Computer Science*, páginas 286–300. Springer Verlag.

Bornot, S., Sifakis, J. y Tripakis, S. (1997). Modelling Urgency in Timed Systems. En de Roever, W. P., Langmaack, H. y Pnueli, A., editores, *Compositionality: The Significant Difference, International Symposium (COMPOS'97)*, volumen 1536 de *Lecture Notes in Computer Science*, páginas 103–129. Springer Verlag.

Bouajjani, A., Echached, R. y Sifakis, J. (1993). On Model Checking for Real Time Properties with Durations. En *Logic in Computer Science, 8th IEEE International Symposium (LICS'93)*, páginas 147–159. IEEE Computer Society Press.

Bouajjani, A., Fernandez, J. C., Halbwachs, N., Raymond, P. y Ratel, C. (1992). Minimal State Graph Generation. *Science of Computer Programming*, 18(3):247–269.

Bouajjani, A., Lakhnech, Y. y Yovine, S. (1996). Model Checking for Extended Timed Temporal Logics. En *Formal Techniques in Real-Time and Fault-Tolerant Systems, 4th International Symposium (FTRTFT'96)*, volumen 1135 de *Lecture Notes in Computer Science*, páginas 306–326. Springer Verlag.

Bouajjani, A., Tripakis, S. y Yovine, S. (1997). On-the-Fly Symbolic Model Checking for Real-Time Systems. En *Real-Time Systems, 18th IEEE Symposium (RTSS'97)*. IEEE Computer Society Press.

Boussinot, F. y de Simone, R. (1991). The ESTEREL Language. En *Proceedings of the IEEE*, volumen 79, páginas 1293–1304.

Bozga, M., Maler, O., Pnueli, A. y Yovine, S. (1997). Some Progress in the Symbolic Verification of Timed Automata. En Grumberg, O., editor, *Computer Aided Verification, 9th International Conference (CAV'97)*, volumen 1254 de *Lecture Notes in Computer Science*, páginas 179–190. Springer Verlag.

Bozga, M., Maler, O. y Tripakis, S. (1999). Efficient Verification of Timed Automata Using Dense and Discrete Time Semantics. En Pierre, L. y Kropf, T., editores, *Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference (CHARME'99)*, volumen 1703 de *Lecture Notes in Computer Science*, páginas 125–141. Springer Verlag.

Brémond Grégoire, P. y Lee, I. (1997). A Process Algebra of Communicating Shared Resources with Dense Time and Priorities. *Theoretical Computer Science*, 189.

Broy, M., Dederichs, F., Dendorfer, C., Fuchs, M., Gritzner, T. y Weber, R. (1992). The Design of Distributed Systems — an Introduction to FOCUS. Technical Report TUM-I9202, Technische Universitat Munchen.

Bruno, G., Castela, A., Pavesio, I. y Pescarmona, M. P. (1993). A New Petri Net Based Formalism for Specification, Design and Analysis of Real-Time Systems. En *Real-Time Systems, 14th IEEE Symposium (RTSS'93)*, páginas 262–273. IEEE Computer Society Press.

Bryant, R. (1986). Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

Burns, A. y Wellings, A. J. (1994). HRT-HOOD:A Design Method for Hard Real-Time. *Real-Time Systems*, 6(1):73–114.

Cerans, K., Godskesen, J. C. y Larsen, K. G. (1993). Timed Modal Specifications – Theory and Tools. En Courcoubetis, C., editor, *Computer Aided Verification, 5th International Conference (CAV'93)*, volumen 697 de *Lecture Notes in Computer Science*, páginas 253–267. Springer Verlag.

Chaochen, Z., Hansen, M. y Sestoft, P. (1993). Decidability and Undecidability Results for Duration Calculus. En Enjalbert, P., Finkel, A. y Wagner, K. W., editores, *Theoretical Aspects of Computer Science, 10th Annual Symposium (STACS'93)*, volumen 665 de *Lecture Notes in Computer Science*, páginas 58–68. Springer Verlag.

Chaochen, Z., Hoare, C. y Ravn, A. P. (1991). A Calculus of Durations. *Information Processing Letters*, 40(5):269–276.

Chechik, M., Devereux, B. y Easterbrook, S. (2001). Implementing a Multi-Valued Symbolic Model Checker. En *Tools and Algorithms for the Construction and Analysis of Systems, International Conference (TACAS'01)*.

Choi, J. Y., Lee, I. y Xie, H. L. (1995). The Specification and Schedulability Analysis of Real-Time Systems Using ACSR. En *Real-Time Systems, 16th IEEE Symposium (RTSS'95)*, páginas 266–275. IEEE Computer Society Press.

Clarke, D. y Lee, I. (1995). Testing Real-Time Constraints in a Process Algebraic Setting. En *Software Engineering, 17th International Conference (ICSE'95)*.

Coen Porisini, A., Ghezzi, C. y Kemmerer, R. A. (1997). Specification of Real Time Systems Using ASTRAL. *IEEE Transactions on Software Engineering*, 23(9).

Cousot, P. y Cousot, R. (1977). Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. En *Principles of Programming Languages, 4th Annual Symposium (POPL'77)*, páginas 238–252. ACM Press, New York.

Dams, D., Gerth, R. y Grumberg, O. (1997). Abstract Interpretation of Reactive Systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291.

Dasarathy, B. (1985). Timing Constraints of Real-Time Systems : Constructs for Expressing them, Methods of Validating them. *IEEE Transactions on Software Engineering*, 11(1):80–86.

Davies, J. y Schneider, S. (1995). A Brief History of Timed CSP. *Theoretical Computer Science*, 138(2):243–271.

Daws, C. y Tripakis, S. (1998). Model Checking of Real-time Reachability Properties Using Abstractions. En Steffen, B., editor, *Tools and Algorithms for the Construction and Analysis of Systems, 4th International Conference (TACAS'98)*, volumen 1384 de *Lecture Notes in Computer Science*, páginas 313–329. Springer Verlag.

Daws, C. y Yovine, S. (1996). Reducing the Number of Clock Variables of Timed Automata. En *Real-Time Systems, 17th IEEE Symposium (RTSS'96)*, páginas 73–81. IEEE Computer Society, IEEE Computer Society Press.

Dickhofer, M. y Wilke, T. (1999). Timed Alternating Tree Automata: The Automata-theoretic Solution to the TCTL Model Checking Problem. En Wiedermann, J., van Emde Boas, P. y Nielsen, M., editores, *Automata, Languages and Programming, 26th International Colloquium (ICALP'99)*, volumen 1644 de *Lecture Notes in Computer Science*, páginas 281–290. Springer Verlag.

Dierks, H., Fehnker, A., Mader, A. y Vaandrager, F. (1998). Operational and Logical Semantics for Polling Real-time Systems. En Ravn y Rischel, editores, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 5th International Symposium (FTRTFT'98)*, volumen 1486 de *Lecture Notes in Computer Science*, páginas 29–40. Springer Verlag.

Dill, D. L. (1989). Timing Assumptions and Verification of Finite-state Concurrent Systems. En Sifakis, J., editor, *Automatic Verification Methods for Finite State Systems, International Workshop*, volumen 407 de *Lecture Notes in Computer Science*, páginas 197–212. Springer Verlag.

Doerfel, M., Slomka, F. y Hofmann, R. (1999). A Scalable Hardware Library for the Rapid Prototyping of SDL Specifications. En *Rapid System Prototyping, 10th International Workshop (RSP'99)*. IEEE Computer Society.

Dwyer, M. B., Avrunin, G. S. y Corbett, J. C. (1999). Patterns in Property Specifications for Finite-State Verification. En *Software Engineering, Twenty-First International Conference*, páginas 411–420, Los Angeles.

Easterbrook, S. M. y Chechik, M. (2001). A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints. En *Software Engineering, 23rd International Conference (ICSE'01)*. IEEE Computer Society Press.

Emerson, E. A. (1996). Automated Temporal Reasoning about Reactive Systems (8th Banff Higher Order Workshop). En Moller, F. y Birtwistle, G., editores, *Logics for Concurrency: Structure versus Automata*, volumen 1043 de *Lecture Notes in Computer Science*, páginas 41–101. Springer Verlag.

Emerson, E. A. y Clarke, E. (1982). Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Science of Computer programming*, 2:241–266.

Emerson, E. A. y Clarke, E. M. (1981). Characterizing Correctness Properties of Parallel Programs as Fixpoints. En de Bakker, J. W. y van Leeuwen, J., editores, *Automata, Languages and Programming, 7th International Colloquium (ICALP'81)*, volumen 85 de *Lecture Notes in Computer Science*, páginas 169–181. Spinger Verlag.

Emerson, E. A., Mok, A. K., Sistla, A. P. y Srinivasan, J. (1990). Quantitative Temporal Reasoning. En Clarke, E. M. y Kurshan, R. P., editores, *Computer-aided Verification, 2nd International Workshop (CAV'90)*, volumen 531 de *Lecture Notes in Computer Science*, páginas 136–145. Springer Verlag.

Emerson, E. A. y Trefler, R. J. (1998). Model Checking Real-Time Properties of Symmetric Systems. En Brim, L., Gruska, J. y Zlatuska, J., editores, *Mathematical Foundations of Computer Science, 23rd International Symposium (MFCS'98)*, volumen 1450 de *Lecture Notes in Computer Science*, páginas 427–436. Springer Verlag.

Felder, M. y Morzenti, A. (1994). Validating Real-Time Systems by History Checking TRIO Specifications. *ACM Transactions On Software Engineering and Methodologies*, 3(4):308–339.

Fernández Vilas, A. y Pazos Arias, J. J. (2000). An Incremental Formal Approach to Real-time Systems. En *ASE'00 Doctoral Symposium*, Grenoble, France. IRISA technical report, number PI-1353.

Fernández Vilas, A., Pazos Arias, J. J., Barragáns Martínez, B., Díaz Redondo, R. P., García Duque, J. y Gil Solla, A. (2000). Verificación de Requisitos de Tiempo Críticos sobre Sistemas Incompletos. En *VIII Jornadas de Concurrencia*, Cuenca, España.

Fernández Vilas, A., Pazos Arias, J. J., Díaz Redondo, R., García Duque, J. y Gil Solla, A. (1999). Extensión del Modelo MES para el Diseño de Sistemas de Tiempo Real Críticos. En *VII Jornadas de Concurrencia*, Gandía, España.

Fitting, M. (1992). Many-valued modal logics II. *Fundamenta Informaticae*, 17:55–73.

Franzle, M. (1996). Synthesizing Controllers from Duration Calculus. En Jonsson, B. y Parrow, J., editores, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 4th International Symposium (FTRTFT'96)*, volumen 1135 de *Lecture Notes in Computer Science*, páginas 168–187. Spinger Verlag.

Fredette, A. y Cleaveland, R. (1993). RTSL: A Language for Real-Time Schedulability Analysis. En *Real-Time Systems, 14th IEEE Symposium (RTSS'93)*, páginas 274–283. IEEE Computer Society Press.

García Duque, J. (2000). *Especificación, Verificación y Mantenimiento de Requisitos Funcionales con Técnicas de Descipción Formal*. PhD thesis, Departamento de Tecnología de las Comunicaciones - Universidad de Vigo.

Ghezzi, C., Mandrioli, D. y Morzenti, A. (1990). TRIO, a Logic Language for Executable Specifications of Real-Time systems. *Journal of Systems and Software*, 12(2):107–123.

Gil Solla, A. (1999). *Diseño y Verificación de Sistemas Distribuidos mediante la Aplicación Combinada de Métodos Formales*. PhD thesis, Departamento de Tecnología de las Comunicaciones - Universidad de Vigo.

Godskesen, J. C., Larsen, K. G., Niedermann, J. y Kristoffersen, K. J. (1993). *EPSILON – User´s Manual*. Institute of Electronic Systems. Aalborg University.

Gollu, A., Puri, A. y Varaiya, P. (1994). Discretization of Timed Automata. En *Decision and Control, 33rd IEEE conference*, páginas 957–958.

Gotzhein, R., Kronenburg, M. y Peper, C. (1998). Reuse in Requirements Engineering: Discovery and Application of a Real-Time Requirement Pattern. En *Formal Techniques in Real-Time and Fault-Tolerant Systems, 5th International Symposium (FTRTFT 98)*, volumen 1468 de *Lecture Notes in Computer Science*, páginas 65–74, Lyngby, Denmark. Springer Verlag.

Halang., W. (1990). Real-Time Systems: Another Perspective. En *Real-Time Systems: Abstractions, Languages, and Design Methodologies*. IEEE Computer Society Press.

Halbwachs, N., Caspi, P. y Raymond, P. (1991). The Synchronous Data Flow Programming Language LUSTRE. En *Proceedings of the IEEE*, volumen 79, páginas 1305–1320.

Hansson, H. y Jonsson, B. (1994). A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535.

Harel, D. (1987). Statecharts: A Visual Approach to Complex Systems. *Science of Computer Programming*, 8(3):231–274.

Harel, E., Lichtenstein, O. y Pnueli, A. (1990). Explicit-clock Temporal Logic. En *Logic in Computer Science, 5th IEEE International Symposium (LICS'90)*, páginas 402–413. IEEE Computer Society Press.

Havelund, K., Larsen, K. G. y Skou, A. (1999). Formal Verification of a Power Controller Using the Real-Time Model Checker UPPAAL. En Katoen, J. P., editor, *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop (ARTS'99)*, volumen 1601 de *Lecture Notes in Computer Science*, páginas 277–298. Springer Verlag.

Havelund, K., Skou, A., Larsen, K. G. y Lund, K. (1997). Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL. En *Real Time Systems, 18th IEEE Symposium*, páginas 2–13, San Francisco, California, USA.

Heitmeyer, C. (1998). On the Need for Practical Formal Methods. En Ravn, A. P. y Rischel, H., editores, *Formal Techniques in Real-Time and Fault Tolerant Systems, 5th International Symposium (FTRTFT'98)*, volumen 1486 de *Lecture Notes in Computer Science*, páginas 18–26. Springer Verlag.

Heitmeyer, C., Kirby, J., Labaw, B., Archer, M. y Bharadway, R. (1998). Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications. *IEEE Transactions on Software Engineering*, 24(11):927–948.

Henessy, M. y Regan, T. (1991). A Process Algebra for Timed Systems. Technical Report 5, University of Sussex.

Henessy, M. y Regan, T. (1995). A Process Algebra for Timed Systems. *Information and Computation*, 117(2):221–239.

Henzinger, T., Ho, P. H. y Wong Toi, H. (1997). HyTech: A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1:110–122.

Henzinger, T., Kupferman, O. y Vardi, M. (1996). A Space-efficient On-the-fly Algorithm for Real-Time Model-checking. En Montanari, U. y Sassone, V., editores, *Concurrency Theory, 7th International Conference (CONCUR'96)*, volumen 1119 de *Lecture Notes in Computer Science*, páginas 514–529.

Henzinger, T. A. y Ho, P. H. (1994). Model Checking Strategies for Linear Hybrid Systems. En *Formalisms for Representing and Reasoning about Time, Workshop as part of the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*.

Henzinger, T. A., Ho, P. H. y Wong Toi, H. (1995). HyTech: the Next Generation. En *Real-Time System, 16th Annual IEEE Symposium (RTSS'95)*, páginas 56–65. IEEE Computer Society Press.

Henzinger, T. A., Manna, Z. y Pnueli, A. (1990). An Interleaving Model for Real Time. En *Information Technology, 5th Jerusalen Conference*, páginas 717–730. IEEE Computer Society Press.

Henzinger, T. A., Manna, Z. y Pnueli, A. (1991). Temporal Proof Methodologies for Real-time Systems. En *Principles of Programming Languages, 18th ACM Symposium (POPL'91)*, páginas 353–366. ACM Press.

Henzinger, T. A., Manna, Z. y Pnueli, A. (1992a). What Good Are Digital Clocks? En Kuich, W., editor, *Automata, Languages and Programming, 19th International Colloquium (ICALP'92)*, volumen 623 de *Lecture Notes in Computer Science*, páginas 545–558. Springer Verlag.

Henzinger, T. A., Nicollin, X., Sifakis, J. y Yovine, S. (1992b). Symbolic Model-checking for Real Time Systems. En *Logics in Computer Science, 7th Symposium*, páginas 394–406. IEEE Computer Society Press.

Henzinger, T. A., Raskin, J. F., y Schobbens, P. Y. (1998). The Regular Real-time Languages. En Larsen, K. G., Skyum, S. y Winskel, G., editores, *Automata, Languages and Programming, 25th International Colloquium (ICALP'98)*, volumen 1443 de *Lecture Notes in Computer Science*, páginas 580–591. Springer Verlag.

Henzinger, T. A. y Toi, H. W. (1996). Using HyTech to Synthesize Control Parameters for a Steam Boiler. En (Abrial et al., 1996), páginas 265–282.

Hähnle, R. y Escalada-Imaz, G. (1997). Deduction in Many-valued Logics: a Survey. *Mathware & Soft Computing*, IV(2):69–97.

Ho, P. H. y Wong-Toi, H. (1995). Automated Analysis of an Audio Control Protocol. En Wolper, P., editor, *Computer-aided Verification, 7th International Conference (CAV'95)*, volumen 939 de *Lecture Notes in Computer Science*, páginas 381–394. Springer Verlag.

Hoare, C. (1985). *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall.

Hunter, A. y Nuseibeh, B. (1998). Managing Inconsistent Specifications: Reasoning, Analysis and Action. *ACM Transactions on Software Engineering and Methodology*, 7(4):335–367.

ISO/IECJTC1/SC21/WG7 (1998). Revised Working Draft on Enhancements to LOTOS (V4). Technical report.

Jahanian, F. y Mok, A. K. (1986). Safety Analysis of Timing Properties in Real-time Systems. *IEEE Transactions on Software Engineering*, 12(9):890–904.

Jahanian, F. y Mok, A. K. (1994). Modechart: A Specification Language for Realtime Systems. *IEEE Transactions on Software Engineering*, 20(12):933–947.

Jalote, P. (1997). *An Integrated Approach to Software Engineering*, capítulo 2. Springer Verlag.

Jonsson, B., Larsen, K. G. y Yi, W. (1999). Probabilistic Extensions in Process Algebras. En *Handbook of Process Algebras*. Elsevier.

Joseph, M. (1996). *Time and Real Time*, capítulo 1, páginas 1–14. Prentice Hall.

Kapur, A., Henzinger, T. A., Manna, Z. y Pnueli, A. (1994). Proving Safety Properties of Hybrid Systems. En Langmaack, H., de Roever, W. P. y Vytopil, J., editores, *Formal Techniques in Real-time and Fault-tolerant Systems, 3rd International Symposium (FTRTFT'94)*, volumen 863 de *Lecture Notes in Computer Science*, páginas 431–454. Springer Verlag.

Kershenbaum, A. (1993). *Telecommunications Network Design Algorithms*, capítulo 4, páginas 162–165. Mc Graw Hill.

Koymans, R. (1990). Specifying Real Time Properties with Metric Temporal Logic. En *Real-Time Systems: Abstractions, Languages, and Design Methodologies*, volumen 2, páginas 255–299. IEEE Computer Society Press.

Koymans, R., Shyamasundar, R. K., de Roever, W. P., Gerth, R. y Arun-Kumar, S. (1988). Compositional Semantics for Real-Time Distributed Computing. *Information and Computation*, 79(3):210–256.

Krupnova, H., Vu, D. D. A. y Boubal, M. (1998). Real Time Prototyping Method and a Case Study. En *Rapid System Prototyping, 9th International Workshop (RSP'98)*. IEEE Computer Society.

Kupferman, O. y Pnueli, A. (1995). Once and For All. En *Logic in Computer Science, 10th IEEE International Symposium (LICS'95)*, páginas 25–35.

La Torre, S. y Napoli, M. (1997). Timed Tree Automata with and Application to Temporal Logic. Technical report, Dipartimento di Informatica ed Applicazioni, Universita degli Studi di Salerno, Italy.

La Torre, S. y Napoli, M. (2000). A Decidable Dense Branching-time Temporal Logic. En Kapoor, S. y Prasad, S., editores, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference (FSTTCS'00)*, volumen 1974 de *Lecture Notes in Computer Science*, páginas 139–150. Springer Verlag.

Laroussinie, F., Larsen, K. G. y Weise, C. (1995). From Timed Automata to Logic - And Back. En Wiedermann, J. y Hájek, P., editores, *Mathematical Foundations of Computer Science, 20th International Symposium (MFCS'95)*, volumen 969 de *Lecture Notes in Computer Science*, páginas 529–539. Springer Verlag.

Larsen, K. G., Larsson, F., Pettersson, P. y Yi, W. (1997a). Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction. En *Real-Time Systems, 18th IEEE Symposium (RTSS'97)*, páginas 14–24.

Larsen, K. G., Petterson, P. y Yi, W. (1995a). Compositional and Symbolic Model-Checking of Real-Time Systems. En *Real-Time Systems, 16th IEEE Symposium (RTSS'95)*. IEEE Computer Society Press.

Larsen, K. G., Pettersson, P. y Yi, W. (1995b). Diagnostic Model-Checking for Real-Time Systems. En *Proc. of Workshop on Verification and Control of Hybrid Systems III*, número 1066 en Lecture Notes in Computer Science, páginas 575–586. Springer Verlag.

Larsen, K. G., Pettersson, P. y Yi, W. (1997b). UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1–2:134–152.

Larsen, K. G., Weise, C., Yi, W. y Pearson, J. (1999). Clock Difference Diagrams. *Nordic Journal of Computing*, 6(3):271–298.

Lee, D. y Yannakakis, M. (1992). Online Minimization of Transition Systems. En *Theory of computing, 24th annual ACM symposium*, páginas 264–274. ACM Press.

Lee, I., Ben-Abdellah, H., Kannan, S., Kim, M., Sokolsky, O. y Viswanathan, M. (1998). A Monitoring and Checking Framework for Run-time Correctness Assurance. En *1998 Korea-U.S. Technical Conference on Strategic Technologies*, Vienna.

Lee, I. y Sokolsky, O. (1997). A Graphical Property Specification Language. En *2nd IEEE Workshop on High-Assurance Systems Engineering*.

Leeb, G. y Lynch, N. (1996). Proving Safety Properties of the Steam Boiler Controller. En (Abrial et al., 1996).

Leveson, N. G. y Stolzy, J. L. (1987). Safety Analysis Using Petri Nets. *IEEE Transactions on Software Engineering*, 13(3):386–397.

Lichtenstein, O., Pnueli, A., y Zuck, L. (1985). The Glory of the Past. En *Logics of Programs Conference*, volumen 193 de *Lecture Notes in Computer Science*, páginas 196–218. Springer Verlag.

Lin, H. y Yi, W. (2000). A Proof System for Timed Automata. En Tiuryn, J., editor, *Foundations of Software Science and Computation Structures, 3rd International Conference (FOSSACS'00)*, volumen 1784 de *Lecture Notes in Computer Science*, páginas 208–222. Springer Verlag.

Linger, R. C. (1994). Cleanroom Process Model. *IEEE Software*, 11(2):50–58.

Liu, X., Ramakrishnan, C. y Smolka, S. (1998). Fully Local and Efficient Evaluation of Alternating Fixed Points. En Steffen, B., editor, *Tools and Algorithms for the Construction and Analysis of Systems, 4th International Conference (TACAS'98)*, volumen 1384 de *Lecture Notes in Computer Science*, páginas 5–19. Springer Verlag.

Lowe, G. (1995). Scheduling-Oriented Models for Real-Time Systems. *Computer Journal*, 38(6).

Lynch, N., Segala, R., Vaandrager, F. y Weinberg, H. (1996). Hybrid I/O Automata. En Alur, R., Henzinger, T. y Sontag, E., editores, *Hybrid Systems III: Verification and Control (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems*, volumen 1066 de *Lecture Notes in Computer Science*, páginas 496–510. Springer Verlag.

Lynch, N. A. y Vaandrager, F. W. (1992). Forward and Backward Simulations for Timing-based Systems. En de Bakker, J., Huizing, C., de Roever, W. P. y Rozenberg, G., editores, *Real-Time: Theory in Practice, REX Workshop*, volumen 600 de *Lecture Notes in Computer Science*, páginas 397–446. Springer Verlag.

Maler, O., Manna, Z. y Pnueli, A. (1992). From Timed to Hybrid Systems. En de Bakker, J. W., Huizing, K., de Roever, W. P. y Rozenberg, G., editores, *Real Time: Theory in Practice, REX Workshop*, volumen 600 de *Lecture Notes in Computer Science*, páginas 447–484. Springer Verlag.

Maler, O., Pnueli, A. y Sifakis, J. (1995). On the Synthesis of Discrete Controllers for Timed Sytems. En Mayr, E. W. y Puech, C., editores, *Theoretical Aspects of Computer Science, 12th Annual Symposium (STACS'95)*, volumen 900 de *Lecture Notes in Computer Science*, páginas 229–242. Springer Verlag.

Manna, Z. y Pnueli, A. (1995). Clocked Transition Systems. En *Verification and Control of Hybrid Systems, International Workshop*.

Mehlhorn, K. y Maher, S. (1999). *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press.

Meritt, M., Modugno, F. y Tuttle, M. R. (1991). Time-Constrained Automata. En Baeten, J. C. M. y Groote, J. F., editores, *Concurrency Theory, 2nd International Conference (CONCUR'91)*, volumen 527 de *Lecture Notes in Computer Science*, páginas 408–423. Springer Verlag.

Merlin, P. (1976). Recoverability of Communication Protocols. *IEEE Transactions on Communications*, COM-24(9):1036–1043.

Merz, S. (1995). Efficiently Executable Temporal Logic Programs. En Fisher, M. y Owens, R., editores, *Executable Modal and Temporal Logics*, volumen 897 de *Lecture Notes in Computer Science*, páginas 69–85. Springer Verlag.

Messing, B. (1997). Combining Knowledge with Many-valued Logics. *Data and Knowledge Engineering*, 23(3):297–316.

Milner, R. (1980). *A Calculus of Communicating Systems*, volumen 94 de *Lecture Notes in Computer Science*. Springer Verlag.

Moffett, J. D., Hall, J. G., Coombes, A. C. y McDermid, J. A. (1996). A Model for a Causal Logic for Requirements Engineering. *Journal of Requirements Engineering*, 1(1):27–46.

Mok, A. K. (1991). Towards Mechanization of Real-time System Design. En van Tilborg, A. M. y Koob, G. M., editores, *Foundations of Real-Time Computing: Formal specifications and Methods*. Kluwer Academic Publishers.

Mok, A. K. y Liu, G. (1997). Efficient Run-time Monitoring of Timing Constraints. En *Real-Time Technology and Applications Symposium*.

Mok, A. K., Tsou, D. C. y de Rooij, R. C. M. (1996). The MSP.RTL Real-Time Scheduler Synthesis Tool. En *Real-Time Systems, 17th IEEE Symposium (RTSS'96)*, páginas 118–128. IEEE Computer Society Press.

Moller, F. y Tofts, C. (1990). A Temporal Calculus of Communicating Systems. En Baeten, J. y Klop, J., editores, *Theories of Concurrency: Unification and Extension (CONCUR'90)*, volumen 458 de *Lecture Notes in Computer Science*, páginas 401–415. Springer Verlag.

Morzenti, A. y San Prietro, P. (1994). Object-Oriented Logic Specifications of Time Critical Systems. *ACM TOSEM – Transactions on Software Engineering and Methodologies*, 3(1).

Moser, L. E., Ramakrishna, Y. S., Kutty, G., Melliar Smith, P. M. y Dillon, L. K. (1997). A Graphical Environment for Design of Concurrent Real-Time Systems. *ACM Transactions on Software Engineering and Methodology*, 6(1):31–79.

Nicollin, X. (1992). *ATP: une Algèbre pour la Spécification et Lánalyse des Systèmes Temps Réel*. PhD thesis, Institut National Polytechnique de Grenoble.

Nicollin, X., Richier, J. L., Sifakis, J. y Voiron, J. (1990). ATP: an Algebra for Timed Processes. En Broy, M. y Jones, C., editores, *Programming Concepts and Methods, IFIP TC 2 Working Conference*, páginas 415–442. Elsevier Science.

Nicollin, X. y Sifakis, J. (1991). An Overview and Synthesis on Timed Process Algebras. En de Bakker, J., Huizing, C., de Roever, W. P. y Rozenberg, G., editores, *Real-Time: Theory in Practice, REX Workshop*, volumen 600 de *Lecture Notes in Computer Science*, páginas 526–548. Springer Verlag.

Nicollin, X. y Sifakis, J. (1994). The Algebra of Timed Processes ATP: Theory and Application. *Information and Computation*, 114:131–178.

Nitsche, U. (1994). A Verification Method Based on Homomorphic Model Abstractions. En *Principles of Distributed Computing, 13th Annual ACM Symposium (PODC'94)*, página 393. ACM Press.

Olivero, A. (1994). *Modélisation et Analyse de Systèmes Temporisès et Hybrides.* PhD thesis, Institut National Polytechnique de Grenoble.

Ostroff, J. S. (1989). Automated Verification of Timed Transition Models. En Sifakis, J., editor, *Automated Verification Methods for Finite State Systems*, volumen 407 de *Lecture Notes in Computer Science*, páginas 247–256. Springer Berlin.

Ostroff, J. S. (1990a). A Logic for Real-Time Discrete Event Processes. *IEEE Control Systems Magazine*, 10(4):95–102.

Ostroff, J. S. (1990b). Deciding Properties of Timed Transition Models. *Transactions on Parallel and Distributed Systems*, 1(2):170–183.

Ostroff, J. S. (1992). Formal Methods for the Specification and Design of Real-Time Safety Critical Systems. *Journal of Systems and Software*, 18(1):33–60.

Pagani, F. (1996). Partial Orders and Verification of Real Time Systems. En Jonsson, B. y Parrow, J., editores, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 4th International Symposium (FTRTFT'96)*, volumen 1135 de *Lecture Notes in Computer Science*, páginas 327–346. Spinger Verlag.

Pazos Arias, J. J. (1995). *Transformación y Verificación con LOTOS.* PhD thesis, Departamento de Ingeniería de Sistemas Telemáticos - Universidad Politécnica de Madrid.

Pazos Arias, J. J. y García Duque, J. (2001). SCTL-MUS: A Formal Methodology for Software Development of Distributed Systems. A Case Study. *Formal Aspects of Computing*, 13:50–91.

Pfleeger, S. L. (1998). *Software Engineering: Theory and Practice*, capítulo 2. Prentice Hall.

Puchol, C. y Mok, A. K. (1998). Integrated Design Tools for Hard Real Time Systems. En *Real-Time Systems, 19th IEEE Symposium (RTSS'98).*

Ramchandani, C. (1976). Analysis of Asynchronous Concurrent Systems Using Petri Nets. Technical Report MAC-TR 120, Massachusetts Institute of Technology. Project MAC.

Raskin, J. F. y Schobbens, P. Y. (1997). State Clock Logic: A Decidable Real Time Logic. En Maler, O., editor, *Hybrid and Real Time Systems, International Workshop (HART'97)*, volumen 1201 de *Lecture Notes in Computer Science*, páginas 33–47. Springer Verlag.

Raskin, J. F. y Schobbens, P. Y. (1999). The Logic of Event Clocks. *Journal of Algebra, Languages and Combinatorics.*

Raymond, P., Nicollin, X., Halbwachs, N. y Weber, D. (1998). Automatic Testing of Reactive Systems. En *19th IEEE Real-Time Systems Symposium*, Madrid, Spain.

Reed, G. M. y Roscoe, A. W. (1988). A Timed Model for Communicating Sequential Processes. *Theoretical Computer Science*, 58:249–261.

Reed, G. M., Roscoe, A. W., Schneider, S., Davies, J., Jackson, D. M. y Reed, J. (1991). Timed CSP: Theory and Practice. En de Bakker, J. W., Huizing, C., de Roever, W. P. y Rozenberg, G., editores, *Real-Time: Theory in Practice, REX Workshop*, volumen 600 de *Lecture Notes in Computer Science*, páginas 640–675. Springer Verlag.

Rushby, J. (1995). Formal Methods and Their Role in the Certification of Critical Systems. Technical Report CSL-95-01, Computer Science Laboratory, SRI International, Menlo Park, CA.

Selic, B., Gullekson, G. y Ward, P. T. (1994). *ROOM – Real-Time Object-Oriented Modelling*. Wiley & Sons.

Sifakis, J. y Yovine, S. (1996). Compositional Specification of Timed Systems. En Puech, C. y Reischuk, R., editores, *Theoretical Aspects of Computer Science, 13th Annual Symposium (STACS'96)*, volumen 1046 de *Lecture Notes in Computer Science*, páginas 347–359. Springer Verlag.

Sokolsky, O. V. y Smolka, S. A. (1995). Local Model Checking for Real-Time Systems. En Wolper, P., editor, *Computer Aided Verification, 7th International Conference (CAV'95)*, volumen 939 de *Lecture Notes in Computer Science*, páginas 211–224. Springer Verlag.

Somé, S. y Dssouli, R. (1998). An Enhancement of Timed Automata generation from Timed Scenarios using Grouped States. *Electronic Journal on Networks and Distributed Processing*, 6(2):67–84.

Sommerville, I. (1997). *Software Engineering*. Addison-Wesley Publishing Company Inc.

Springintveld, J. G., Vaandrager, F. W. y D ' Argenio, P. R. (1997). Testing Timed Automata. Technical Report CSI-R9712, Computing Science Institute, University of Nijmegen. Full version to appear in Theoretical Computer Science.

Stankovic, J. A. (1988). Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems. *IEEE Computer*, 21(10):10–19.

Stankovic, J. A. (1996). Strategic Directions in Real-Time and Embedded Systems. *ACM Computing Surveys*, 28(4):751–763.

Stirling, C. y Bradfield, J. (2001). Modal Logics and μ-calculi: An Introduction. En Bergstra, J., Ponse, A. y Smolka, S., editores, *Handbook of Process Algebra*, páginas 293–332. Elsevier.

Stirling, C. y Walker, D. J. (1991). Local Model Checking in the Modal μ-calculus. *Theoretical Computer Science*, 89(1):161–177.

Tarski, A. (1955). A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5:285–309.

Tripakis, S. (1998). *L'Analyse Formelle des Systèmes Temporisès en Pratique*. PhD thesis, Université Joseph Fourier.

Tripakis, S. (1999). Verifying Progress in Timed Systems. En Katoen, J. P., editor, *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop (ARTS'99)*, volumen 1601 de *Lecture Notes in Computer Science*, páginas 299–314. Springer Verlag.

Tripakis, S. y Altisen, K. (1999). On-The-Fly Controller Synthesis for Discrete and Dense-Time Systems. En Wing, J. M., Woodcock, J. y Davies, J., editores, *World Congress on Formal Methods in the Development of Computing Systems (FM'99)*, volumen 1708 de *Lecture Notes in Computer Science*, páginas 233–252. Springer Verlag.

Tripakis, S. y Courcoubetis, C. (1996). Extending Promela and Spin for Real Time. En Margaria, T. y Steffen, B., editores, *Tools and Algorithms for the Construction and Analysis of Systems, 2nd International Workshop (TACAS'96)*, volumen 1055 de *Lecture Notes in Computer Science*, páginas 329–348. Springer Verlag.

Tripakis, S. y Yovine, S. (1996). Analysis of Timed Systems Based on Time-abstracting Bisimulations. En Alur, R. y Henzinger, T. A., editores, *Computer-Aided Verification, 8th International Conference (CAV'96)*, volumen 1102 de *Lecture Notes in Computer Science*, páginas 232–243. Springer Verlag.

Tripakis, S. y Yovine, S. (1998). Verification of the Fast Reservation Protocol with Delayed Transmission Using the Tool Kronos. En *Real-Time Technology and Applications, 4th IEEE Symposium (RTAS'98)*. IEEE Computer Society Press.

Veloudis, S. y Nissanke, N. (1998). Duration Calculus in the Specification of Safety Requirements. En Ravn, A. P. y Rischel, H., editores, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 5th International Symposium (FTRTFT'98)*, volumen 1486 de *Lecture Notes in Computer Science*, páginas 103–112. Spinger Verlag.

Wang, F., Mok, A. K. y Emerson, E. A. (1993). Distributed Real-Time System Specification and Verification in APTL. *ACM Transactions on Software Engineering and Methodology*, 2(4):346–378.

Wong Toi, H. (1995). *Symbolic Approximation for Verifying Real Time Systems*. PhD thesis, Department of Computer Science. Stanford University.

Wong Toi, H. y Hoffmann, G. (1991). The Control of Dense Real Time Discrete Event Sytems. En *Decision and Control, 30th IEEE Conference*, páginas 1527–1528.

XiaoShan, L. y JuAn, W. (1996). Specifying Optimal Design of a Steam-Boiler System. En (Abrial et al., 1996), páginas 339–358.

Yi, W. (1990). Real-Time Behaviour of Asynchronous Agents. En Baeten, J. y Klop, J., editores, *Theories of Concurrency: Unification and Extension (CONCUR'90)*, volumen 458 de *Lecture Notes in Computer Science*, páginas 502–520. Springer Verlag.

Yi, W. (1994). Algebraic Reasoning for Real-Time Probabilistic Processes with Uncertain Information. En Langmaack, H., de Roever, W. P. y Vytopil, J., editores, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 3rd International Symposium (FTRTFT'94)*, volumen 863 de *Lecture Notes in Computer Science*, páginas 680–693. Springer Verlag.

Yi, W., Peterson, P. y Daniels, M. (1994). Automatic Verification of Real-Time Communicating Systems by Constraint Solving. En Hogrefe, D. y Leue, S., editores, *Formal Description Techniques for Distributed Systems and Communication Protocolos, 7th International Conference (FORTE'94)*, páginas 243–258.

Yovine, S. (1993). *Methodes et Outils pour la Verification Symbolique de Systemes Temporises*. PhD thesis, Institut National Polytechnique de Grenoble.

Yovine, S. (1997). KRONOS: A Verification Tool for Real-time Systems. *Springer International Journal of Software Tools for Technology Transfer*, 1(1/2).