DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Tesis de licenciatura

Una nueva estructura de datos basada en *BDDs* para el *model checking* temporizado

12 de abril de 2006, Buenos Aires, Argentina.

Resumen

Los sistemas de tiempo real son por naturaleza críticos. Sus fallas pueden resultar en serias pérdidas, tanto materiales como también de vidas humanas. Además, en general están descriptos por la interacción de varios componentes y resulta muy difícil asegurar que determinadas propiedades (que representan de alguna manera requisitos o condiciones deseables del sistema) se cumplan.

Hoy en día, existen herramientas denominadas model checkers (por ejemplo, UPPAAL, HyTech, KRONOS) utilizadas para expresar y verificar propiedades sobre este tipo de sistemas. Una de las propiedades más requerida es la de establecer si cierto conjunto de estados del sistema es o no alcanzable. Lamentablemente, esta verificación es costosa, y a veces prohibitiva, tanto en términos de tiempo como de memoria, debido al problema de la explosión combinatoria de estados. Por otra parte, recientemente se ha estudiado el uso de estructuras de decisión como posible alternativa a las representaciones clásicas utilizadas en la verificación, de forma de lograr reducir, en cierto grado, los efectos de tal explosión.

En este trabajo se presenta una nueva estructura de datos, basada en árboles de decisión, que apunta a reducir el tiempo y espacio requeridos para estas verificaciones. Además, se realiza una implementación de la misma, integrándola al *model checker* Zeus, y con ella se verifican algunos ejemplos de la literatura, obteniéndose resultados promisorios.

Alumno:
Esteban José Pavese
Libreta universitaria: 471/98
Correo electrónico: epavese@dc.uba.ar

Directores:
Lic. Fernando Pablo Schapachnik
Dr. Alfredo Olivero

Agradecimientos

Desde hace ya un tiempo (difícil de precisar) vengo acumulando algo así como un “déficit” con quienes me han venido acompañando, y que me han ayudado a llegar a este momento. Creo que este es un buen momento para reconocer, al menos en parte, a todos ellos. Espero no olvidarme de nadie en esta pequeña rendición de cuentas; si lo hago, me disculpo de antemano, a sabiendas de que cada quién sabrá reconocerse entre líneas.

Debo agradecer a todos los docentes que fueron orientándome en el camino, tanto a los profesores como a los ayudantes que pusieron en todo momento lo mejor de sí. Tanto es así, que hasta me contagiaron el entusiasmo por la docencia. Dentro de este grupo, agradezco muy especialmente a Fer y Alfredo; sin las prontas acotaciones de Alfredo y las pequeñas “batallas obsesivas” con Fer, este trabajo no sería lo que es. A los también profes, pero del secundario, Tony, Susana, Claudia y Matías les debo esto del gustito por los números.

Del otro lado, el apoyo de los afectos ha sido, como siempre, fundamental. A mis viejos y hermanos les agradezco la paciencia; haber aguantado esos momentos donde la tensión se escapa hasta por los poros. Y está también esa extraña mezcla de “afectos–colegas” con quienes peleamos la carrera juntos: Alejo, Martín, Javier, Pablo B., el otro Pablo B. (el pelado), Analía, Lautaro, Daniel B., Diego, Matías, Leo y también Marito y Daniel (el cabezón), que merecen un aparte (junto con el “cuco” V.d.S.); la gente de Algo2; los chicos (y chica) de Dependex (Diego G., Nico K., Fer, Guido, Lucía, Andran, Juan y Marcelo); y hasta algunos que primero me sufrieron como docente y ahora como amigo: Emi, SW, Nico B., Lu, Eze, Majo, Josi, Tomás, Nico S. y Ana.

Y dejo para el final (aunque de ninguna manera es la última) a Caro, que con mucho amor (e infinita paciencia) me acompaña día a día. Sin ella no estaría acá (¡o al menos no tan feliz!).

Índice general

Índice general I

Índice de figuras III

Índice de tablas IV

1. Introducción y definiciones previas 1

1.1. Motivación . . . 1

1.2. Autómata temporizado . . . 2

1.2.1. Relojes . . . 2

1.2.2. Restricciones sobre relojes . . . 3

1.2.3. Sintaxis de los autómatas temporizados . . . 3

1.2.4. Semántica de los autómatas temporizados . . . 4

1.2.5. Ejecuciones . . . 4

1.3. Análisis utilizando regiones temporales . . . 6

1.3.1. Discretizando las valuaciones de relojes . . . 6

1.3.2. Discretizando el sistema de transiciones . . . 7

1.3.3. Predicados sobre los sistemas de transiciones . . . 7

1.3.4. Los operadores suc_t, suc_e y $suc_{\triangleright}$ . . . 8

1.3.5. El problema de la *alcanzabilidad* . . . 9

1.3.6. Estrategia de evaluación . . . 9

2. Antecedentes bibliográficos 11

2.1. DBMs . . . 11

2.1.1. RDBMs . . . 13

2.2. Diagramas de decisión y sus aplicaciones al model checking . . . 14

2.2.1. Diagramas Binarios de Decisión . . . 14

2.2.2. Diagramas de Diferencia de Relojes . . . 16

2.2.3. Diagramas de Restricción de Relojes . . . 17

2.2.4. Diagramas de Codificación de Regiones . . . 18

2.3. Resumiendo . . . 18

3. Extendiendo la estructura—*e*CDDs 20

3.1. Estructura, invariantes y objetivos . . . 20

3.1.1. Descripción general . . . 20

3.2. Operaciones básicas . . . 24
3.2.1. Unión de regiones temporales . . . 24
3.2.2. Intersección de regiones temporales . . . 25
3.2.3. Chequeo de inclusión de una zona en una región . . . 25
3.2.4. Cálculo de sucesores temporales de una zona . . . 25
3.2.5. Reseteo de un conjunto de relojes sobre una región . . . 26
3.3. Semántica de los *e*CDDs . . . 27
3.3.1. Relación entre los *e*CDDs y las restricciones temporales . . . 27
3.3.2. Satisfacibilidad de un *e*CDD. . . . 28
3.4. Algoritmos iterativos sobre *e*CDDs . . . 29
3.4.1. Enfoque . . . 29
3.4.2. Descripción del algoritmo iterativo de intersección de *e*CDDs . . . 31
3.4.3. Descripción del algoritmo de unión de *e*CDDs . . . 38
3.4.4. Descripción del algoritmo de determinación de inclusión . . . 44
3.4.5. Descripción del algoritmo de reseteo de relojes . . . 47
3.4.6. Descripción del algoritmo de obtención de sucesores temporales de una zona . . . 51

4. Implementación **55**
4.1. Implementación de la estructura . . . 55
4.2. Optimizaciones . . . 57
4.2.1. Reducción de relojes . . . 57
4.2.2. Ajuste de cotas . . . 57
4.2.3. Reducción de la cola de estados por visitar . . . 58

5. Resultados obtenidos **59**
5.1. *Train-Gate-Controller* . . . 59
5.2. Pipeline . . . 60
5.3. Pruebas y resultados . . . 61
5.3.1. *Train-Gate-Controller* . . . 61
5.3.2. Pipeline . . . 64
5.3.3. Utilizando *e*CDDs como regiones . . . 64

6. Trabajo e investigación futura **66**

7. Conclusiones **68**

Bibliografía **69**

Índice de figuras

2.1. Una DBM y la zona representada. 12
2.2. Poliedro no convexo y posibles representaciones mediante convexos 13
2.3. Un BDD 14
2.4. RBDDs con distinto orden 15
2.5. Un CDD y la región representada 16
2.6. Un CRD. 17
2.7. Un nodo de RED 18

3.1. Ejemplo de intersección de dos *e*CDDs con idéntica raíz. 30
3.2. Ejemplo de unión de dos *e*CDDs con raíces distintas. 30

4.1. Una diferencia de relojes, representada con la técnica de *bitstuffing*. 56
4.2. Un *e*CDD representando una región, notando la aplicación de la técnica de *bitstuffing*. . . 57

5.1. Observador del ejemplo *Train-Gate-Controller*. 60

Índice de tablas

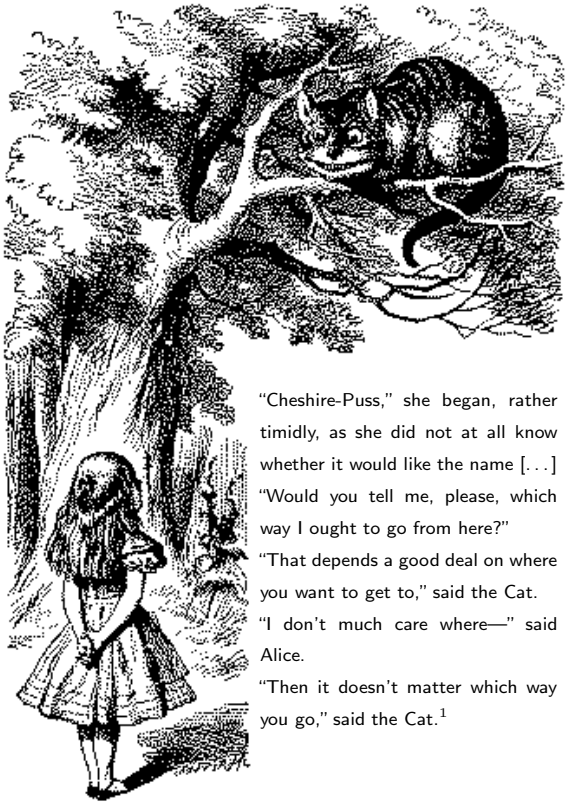
5.1. Dimensiones del caso de estudio de *Train-Gate-Controller*. 61
5.2. Resultados para el caso TGC utilizando RDBMs y *e*CDDs, sin reducir relojes. 62
5.3. Detalle del caso TGC utilizando RDBMs y *e*CDDs, sin reducir relojes. 62
5.4. Detalle de canonizaciones efectuadas en los casos de estudio de TGC, sin minimizar relojes. 62
5.5. Resultados para el caso TGC utilizando RDBMs y *e*CDDs, reduciendo relojes inactivos. . 63
5.6. Detalle del caso TGC utilizando RDBMs, reduciendo relojes inactivos. 63
5.7. Detalle de canonizaciones efectuadas en los casos de estudio de TGC, minimizando relojes. 63
5.8. Resultados para el caso pipeline utilizando RDBMs y *e*CDDs. 64
5.9. Detalle del caso pipeline utilizando RDBMs y *e*CDDs. 64
5.10. Detalle de canonizaciones efectuadas en los casos de estudio de Pipe. 64
5.11. Detalle de los casos medidos utilizando *e*CDDs como repositorio de regiones. 65
5.12. Comparación de los recursos insumidos en el uso de *e*CDDs para verificar el TGC de 4 trenes. 65

Índice de algoritmos

1.3.1.Algoritmo de alcanzabilidad con exploración *forward*. 10
3.2.1.Algoritmo recursivo de unión de *e*CDDs 24
3.2.2.Algoritmo recursivo de intersección de *e*CDDs 25
3.2.3.Algoritmo recursivo de chequeo de inclusión de *e*CDDs 26
3.2.4.Algoritmo recursivo para el cálculo de sucesores temporales de un *e*CDD 26
3.2.5.Algoritmo recursivo para el reseteo de relojes sobre un *e*CDD 27
3.4.1.Algoritmo iterativo de intersección de *e*CDDs—Parte 1. 32
3.4.2.Algoritmo iterativo de intersección de *e*CDDs—Parte 2. 33
3.4.3.Algoritmo iterativo de unión de *e*CDDs—Parte 1. 39
3.4.4.Algoritmo iterativo de unión de *e*CDDs—Parte 2. 40
3.4.5.Algoritmo iterativo de chequeo de inclusión de *e*CDDs. 45
3.4.6.Algoritmo iterativo de reseteo de relojes en un *e*CDD—Parte 1. 49
3.4.7.Algoritmo iterativo de reseteo de relojes en un *e*CDD—Parte 2. 50
3.4.8.Algoritmo iterativo de cálculo de sucesores temporales de una zona utilizando *e*CDDs. . . 53

Capítulo 1

Introducción y definiciones previas



"Cheshire-Puss," she began, rather timidly, as she did not at all know whether it would like the name [...]
"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the Cat.
"I don't much care where—" said Alice.
"Then it doesn't matter which way you go," said the Cat.[1]

1.1. Motivación

Los sistemas de tiempo real son por naturaleza críticos. Sus fallas pueden poner en peligro la vida humana o representar cuantiosas pérdidas materiales. Además, en general están dados por la interacción de varios componentes y resulta muy difícil asegurar que determinadas propiedades (que representan de alguna manera requisitos o condiciones deseables del sistema) se cumplan.

En las últimas décadas se han desarrollado diversos formalismos tanto para representar los sistemas como para expresar propiedades sobre ellos. En particular, surgieron los autómatas temporizados [Alu92] para describir comportamiento y la lógica TCTL [ACD93] para predicar propiedades sobre éstos. Al poco tiempo aparecieron los primeros *model checkers*, como UPPAAL [BLL+96], HyTech [HHWT95]

[1] Lewis Carroll, *Alice's Adventures in Wonderland*, 1865. La ilustración es de John Tenniel, y corresponde a la edición original.

y KRONOS [BDM+98, DOTY96] para este formalismo, programas que toman la especificación de un sistema y una propiedad sobre él y determinan si ésta se cumple o no. Lamentablemente, el problema de verificar una propiedad sobre un sistema suele ser muy costoso en tiempo y en espacio; a veces prohibitivo.

En los últimos tiempos, se han buscado nuevas formas de representar los estados posibles del sistema, generados durante el proceso de verificación del mismo. Dicha representación resulta crucial en el problema de la explosión combinatoria de estados, ya que se busca abarcar problemas más grandes en menos tiempo. En uno de los tantos esfuerzos para obtener esta reducción, se han planteado interrogantes acerca de la posibilidad de utilizar estructuras de datos diferentes a las que se han utilizado hasta ahora. En particular, se ha comenzado a analizar la posibilidad de utilizar estructuras de decisión para representar los distintos valores que pueden ir tomando los distintos relojes, que modelan el paso del tiempo, en el sistema. Existen algunos resultados teóricos con este tipo de estructuras [Wan03, Wan00, BLP+99], pero aún no hay suficientes resultados prácticos que avalen (o refuten) estas ideas. El desafío que plantean estas estructuras no es trivial: se necesita proveer no sólo la estructura (lo cual presenta problemas de eficiencia tanto en tiempo de ejecución de las operaciones como en espacio utilizado), sino también nuevos algoritmos que resuelvan el problema tal como se lo venía resolviendo, probando además que el método utilizado es correcto.

Para comenzar la presente introducción será necesario primero recorrer una larga lista de definiciones previas que servirán para fijar la nomenclatura. En este capítulo, entonces, presentamos la terminología a utilizar a lo largo del trabajo. Entonces, en las secciones que siguen nos dedicaremos exclusivamente a comentar las definiciones correspondientes al model checking.

Para ello, se tomará como marco la pregunta fundacional de esta disciplina:

> Dado el autómata temporizado G, el conjunto de requerimientos R y la relación $\models$ que expresa *satisfacción*, vale $G \models R$?

En las secciones subsiguientes definiremos formalmente cada uno de los componentes que permitirán responder esta pregunta. Esta introducción tendrá como base aquella presente en [Yov96].

1.2. Autómata temporizado

Para lograr el objetivo de definir un autómata temporizado y su semántica, resulta necesario presentar antes algunos conceptos preliminares estrechamente relacionados.

1.2.1. Relojes

Durante el transcurso de toda esta introducción, $\mathbb{N}$ denotará al conjunto de números naturales, $\mathbb{Z}$ al conjunto de números enteros, $\mathbb{R}$ al conjunto de números reales y $\mathbb{R}^+$ a los reales no negativos.

Definición 1.2.1 (Relojes, valuaciones sobre relojes) *Sea X un conjunto finito de variables, a las que se denominarán* relojes. *Una* valuación *es una función que asigna un número real no negativo a cada reloj de X. Intuitivamente, una valuación de un reloj es un valor particular de éste.*

El conjunto de valuaciones sobre X se denota como $\mathcal{V}_X$, y está compuesto por las funciones totales que van de X a $\mathbb{R}^+$: $[X \to \mathbb{R}^+]$.

Sea $v \in \mathcal{V}_X$ y $\delta \in \mathbb{R}^+$. Se denota $v + \delta$ la valuación definida de la siguiente forma:

$$(\forall x \in X)(v + \delta)(x) = v(x) + \delta$$

Definición 1.2.2 (Asignaciones sobre relojes) *Sea X^* el conjunto $X \cup \{0\}$. Una* asignación *es una función que asigna a cada reloj el valor de otro reloj, o bien el valor 0.*

El conjunto de asignaciones sobre los relojes de X se denota como Γ_X, y está compuesto por las funciones totales que van de X en X^*: $[X \to X^*]$.

Si $v \in \mathcal{V}_X$ es una valuación y $\gamma \in \Gamma_X$ es una asignación, se denota con $v[\gamma]$ a la valuación definida como:

$$v[\gamma](x) = \begin{cases} v(\gamma(x)) & \text{si } \gamma(x) \in X \\ 0 & \text{si } \gamma(x) = 0 \end{cases}$$

1.2.2. Restricciones sobre relojes

Con el objetivo de poder predicar sobre los valores de los relojes, introducimos a continuación una sintaxis y una semántica para expresar restricciones sobre ellos. Todo ello permitirá comparar los relojes entre ellos y contra valores constantes.

Definición 1.2.3 (Sintaxis de las restricciones sobre relojes) *Dado un conjunto finito de relojes X, se define inductivamente la gramática del lenguaje Ψ_X que permite expresar restricciones sobre los relojes de X de la siguiente manera:*

$$\begin{array}{l} \forall x, x' \in X, c \in \mathbb{N}, d \in \mathbb{Z}: \\ \quad \psi \;\rightarrow\; c \prec x \;\mid\; x \prec c \;\mid\; x - x' \prec d \;\mid\; \psi \wedge \psi \;\mid\; \neg\psi \\ \quad \prec \;\rightarrow\; < \;\mid\; = \end{array}$$

Sin pérdida de generalidad, a partir de este momento sólo haremos referencia a restricciones sobre relojes de longitud finita, que son las que serán de interés en el trabajo.

Definición 1.2.4 (Semántica de las restricciones sobre relojes) *El lenguaje Ψ_X se interpreta sobre las valuaciones. La satisfacibilidad de una restricción $\psi \in \Psi_X$ por una valuación $v \in \mathcal{V}_X$, denotada como $v \models \psi$, se define inductivamente de la siguiente forma:*

$$\begin{array}{l} \forall x, x' \in X, c \in \mathbb{N}, d \in \mathbb{Z}: \\ \begin{array}{lcl} v \models x < c & \Leftrightarrow & v(x) < c \\ v \models x = c & \Leftrightarrow & v(x) = c \\ v \models c < x & \Leftrightarrow & c < v(x) \\ v \models c = x & \Leftrightarrow & c = v(x) \\ v \models x - x' < d & \Leftrightarrow & v(x) - v(x') < d \\ v \models x - x' = d & \Leftrightarrow & v(x) - v(x') = d \\ v \models \psi \wedge \psi' & \Leftrightarrow & v \models \psi' \wedge v \models \psi'' \\ v \models \neg\psi' & \Leftrightarrow & v \not\models \psi' \end{array} \end{array}$$

Definición 1.2.5 (Conjunto característico) *Se llama* conjunto característico *de ψ—y se nota $[\![\psi]\!]$—al conjunto de valuaciones que satisfacen ψ. Formalmente:*

$$[\![\psi]\!] = \{v : v \in \mathcal{V}_X \wedge v \models \psi\}$$

En lo siguiente, será usual referirnos a *zonas temporales* para entender el conjunto de valuaciones que satisface cierta restricción temporal, y *regiones temporales* para hablar de conjuntos de zonas temporales.

1.2.3. Sintaxis de los autómatas temporizados

Una vez presentados los conceptos anteriores, estamos en condiciones de definir los autómatas:

Definición 1.2.6 (Autómata temporizado) *Un* autómata temporizado *G es una tupla $\langle S, X, E, A, I\rangle$ donde:*

- *$S = \{s_0, \ldots, s_m\}$ es un conjunto finito de* estados discretos *o* locaciones.
- *$X = \{x_0, \ldots, x_n\}$ es un conjunto finito de* relojes.
- *E es un conjunto finito de* etiquetas.
- *A es un conjunto finito de arcos de la forma $\langle s_i, e, cond, asig, s_j\rangle$ donde:*
  - *$s_i \in S$ es el* origen,

- $e \in E$ *es la* etiqueta,
- $cond \in \Psi_X$ *es la* condición de activación,
- $asig \in \Gamma_X$ *es la* asignación *y*
- $s_j \in S$ *es el* destino.

- $I \in [S \to \Psi_X]$ *(si* $s \in S$ *decimos que* $I(s)$ *es el* invariante *del estado discreto* s*).*

1.2.4. Semántica de los autómatas temporizados

Los autómatas temporizados introducidos anteriormente se interpretan de la siguiente forma:

Definición 1.2.7 (Sistema de transiciones etiquetado) *Dado un autómata temporizado* G *tal que* $G = \langle S, X, E, A, I\rangle$*, el significado de* G *es un* sistema de transiciones etiquetado *infinito* $(Q_X, \to)$ *donde* Q_X *es el* conjunto de estados *y* $\to$ *es la* relación de transición *que se define formalmente como sigue:*

- $Q_X = \{(s, v) : s \in S \wedge v \in \mathcal{V}_X \wedge v \models I(s)\}$ *(Es decir,* Q_X *está compuesto por tuplas de estados discretos y valuaciones de relojes que satisfacen el invariante del estado discreto.)*
- *La relación de transición* $\to \subseteq Q_X \times (E \cup \mathbb{R}^+) \times Q_X$ *está definida por las siguientes reglas:*
  - *Transiciones discretas:*

$$\frac{\langle s_i, e, cond, asig, s_j\rangle \in A \quad v \models cond \quad v[asig] \models I(s_j)}{(s_i, v) \to^e (s_j, v[asig])}$$

    El estado $(s_j, v[asig])$ *se denomina* sucesor discreto *de* (s_i, v) *y este último* predecesor discreto *del primero. Intuitivamente, se entiende que a partir de un estado el sistema puede evolucionar a través de un arco que lo conduce a otro estado donde las valuaciones de los relojes son las mismas (i.e., no ha transcurrido tiempo) excepto por aquellos relojes que se ha vuelto a cero o se les ha asignado otro valor (asig). Para que este tipo de transiciones pueda ser tomada las valuaciones del estado original deben satisfacer la condición de activación del arco y las resultantes de la asignación del arco, el invariante del estado destino.*
  - *Transiciones temporales:*

$$\frac{\delta \in \mathbb{R}^+ \quad (\forall \delta')(\delta' \in \mathbb{R}^+ \wedge \delta' \leq \delta \implies v + \delta' \models I(s))}{(s, v) \to^\delta (s, v + \delta)}$$

    El estado $(s, v + \delta)$ *se denomina* sucesor temporal *de* (s, v) *y este último* predecesor temporal *del primero. Intuitivamente, en ellas sólo transcurre tiempo. No hay asignaciones de valores a relojes, ni hay un cambio de estado discreto. El único requisito para poder tomarla es que el invariante del estado discreto se siga cumpliendo durante la duración (transcurso del tiempo) de la transición.*

Para simplificar la notación, y cuando el contexto lo permita se notará Q para referirse a Q_X.

1.2.5. Ejecuciones

El concepto de ejecución sobre un autómata temporizado es la abstracción de una sucesión de eventos. Es el primer paso para describir comportamientos deseados y no deseados. En ésta y las siguientes secciones se siguen definiciones tomadas de [Yov96].

Definición 1.2.8 (Ejecución) *Sea* $G = \langle S, X, E, A, I\rangle$ *un autómata temporizado y* $(Q, \to)$ *el sistema de transición etiquetado asociado a dicho autómata. Una* ejecución r *de* G *es una secuencia infinita de aplicaciones de la relación* $\to$ *de la forma*

$$\begin{array}{l}(\forall i)(i \in \mathbb{N} \implies q_i \in Q \wedge l_i \in E \cup \mathbb{R}^+) \\ r \quad = \quad q_0 \to^{l_0} q_1 \to^{l_1} q_2 \to^{l_2} \ldots \to^{l_{i-1}} q_i \to^{l_i} \ldots\end{array}$$

Luego, si $q \in Q$, se denotará $R_G(q)$ el conjunto de ejecuciones de G tal que $q_0 = q$ y $R_G = \bigcup_{q \in Q} R_G(q)$ al conjunto de ejecuciones de G.

A partir de esta noción de ejecución se define una posición *p de una ejecución r como un par $(i, \delta) \in \mathbb{N} \times \mathbb{R}^+$ donde $\delta = 0$ si $l_i \in E$ y $0 < \delta \leq l_i$ en otro caso. Se llamará Pos_r al conjunto de posiciones de r. Para todo $i \geq 0$ el conjunto de posiciones de la forma (i, δ) caracteriza el conjunto de estados de Q por los que la ejecución r pasa mientras el tiempo fluye del estado q_i al q_{i+1}. Adicionalmente, si $q_i = (s_i, v_i)$ se notará $\Xi(i, \delta)$ al estado $(s_i, v_i + \delta)$.*

Si $r \in R_G$, se define a $\tau_r : \mathbb{N} \to \mathbb{R}^+$ como el tiempo transcurrido *en la ejecución r hasta alcanzar un estado dado de la ejecución partiendo de q_0:*

$$\begin{array}{lcl} \tau_r(0) & = & 0 \\ \tau_r(i+1) & = & \begin{cases} \tau_r(i) & si\ l_i \in E \\ \tau_r(i) + l_i & si\ l_i \in \mathbb{R}^+ \end{cases} \end{array}$$

A su vez se define $\tau_r(i, \delta) = \tau_r(i) + \delta$.

Definición 1.2.9 (Paso de duración δ) *Sea $G = \langle S, X, E, A, I \rangle$ un autómata temporizado, $(Q, \to)$ el sistema de transición etiquetado asociado a dicho autómata, $\delta \in \mathbb{R}^+$ y q, $q' \in Q$. El* paso de duración δ *entre los estados q y q' se define como:*

$$\begin{array}{rcl} q \rhd^\delta q' & \Leftrightarrow & q \to^\delta q' \vee q \to^\delta q'' \to^e q' \\ & & \text{donde } q'' \in Q \text{ y } e \in E \end{array}$$

Dados q, $q' \in Q$ y $\delta \in \mathbb{R}^+$ tales que $q \rhd^\delta q'$ se llamará estado intermedio *del paso $q \rhd^\delta q'$ a todo $\hat{q}$ que satisfaga $(\forall \delta')(\delta' \in \mathbb{R}^+ \wedge \delta' \leq \delta \implies q \to^{\delta'} \hat{q})$.*

Como consecuencia de la definición anterior se obtiene una forma alternativa de representar una ejecución $r \in R_G$:

$$\begin{array}{l} (\forall i)(i \in \mathbb{N} \implies q_i \in Q \wedge \delta_i \in \mathbb{R}^+) \\ r = q_0 \rhd^{\delta_0} q_1 \rhd^{\delta_1} q_2 \rhd^{\delta_2} \ldots \rhd^{\delta_{i-1}} q_i \rhd^{\delta_i} \ldots \end{array}$$

Definición 1.2.10 (Ejecución divergente) *Se dice que una ejecución $r \in R_G$ es* divergente *si y sólo si $\lim_{i \to \infty} \tau_r(i) = \infty$.*

Por extensión, dado $q \in Q$ se llama $R_G^\infty(q)$ al conjunto de ejecuciones divergentes que comienzan en q y $R_G^\infty = \bigcup_{q \in Q} R_G^\infty(q)$ al conjunto de ejecuciones divergentes de G.

Intuitivamente, una ejecución es divergente si el tiempo no se "estanca" dentro de ella.

La importancia de la definición 1.2.10 podrá ser apreciada cuando se presente la definición de autómata bien temporizado (definición 1.2.12).

Los elementos presentados hasta ahora permiten definir un orden total entre los componentes de una ejecución:

Definición 1.2.11 (Orden total de las posiciones de una ejecución) *Sea $r \in R_G$. Se define como $\ll_r \subseteq Pos_r \times Pos_r$ a la siguiente relación de orden total:*

$$(i, \delta) \ll_r (j, \delta') \quad \Leftrightarrow \quad (i < j \vee (i = j \wedge \delta \leq \delta'))$$

Definición 1.2.12 (Autómata bien temporizado o *non-zeno*) *Sea $G = \langle S, X, E, A, I \rangle$ un autómata temporizado y $(Q, \to)$ el sistema de transición etiquetado asociado a dicho autómata. Se dice que G es* non-zeno *o* bien temporizado *si y sólo si $(\forall q \in Q)(R_G^\infty(q) \neq \emptyset)$.*

Es decir, un autómata está bien temporizado si en ninguna de sus ejecuciones el tiempo necesariamente "se traba".

1.3. Análisis utilizando regiones temporales

Cualquier análisis sobre el formalismo recién presentado debe resolver varias dificultades para que sea factible. En primer lugar, dado que las valuaciones de relojes son *densas*, el sistema de transiciones $(Q, \rightarrow)$ resulta ser infinito. Es necesario encontrar alguna manera de hacer manejable esta cantidad infinita de estados.

1.3.1. Discretizando las valuaciones de relojes

Como primer paso a tomar para resolver este problema, introducimos una relación que permitirá "partir" a las valuaciones de los relojes en una cantidad finita de clases de equivalencia.

Definición 1.3.1 (Relación de equivalencia entre valuaciones) *Sea $\widehat{\Psi} \subseteq \Psi_X$ un conjunto no vacío y finito de restricciones sobre relojes del conjunto X. Sea $C \in \mathbb{N}$ la constante más chica que es mayor o igual a c para toda constante $c \in \mathbb{Z}$ que aparece en alguna restricción del conjunto $\widehat{\Psi}$.*

Se define $\simeq_{\widehat{\Psi}} \subseteq \mathcal{V}_X \times \mathcal{V}_X$ como la relación reflexiva y simétrica más grande *tal que, dados v, $v' \in \mathcal{V}_X$ se satisface $v \simeq_{\widehat{\Psi}} v'$ si y sólo si para todo x, $x' \in X$ valen simultáneamente:*

- *si $v(x) > C$ entonces $v'(x) > C$;*
- *si $v(x) \leq C$ entonces:*
  - $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$,
  - $v(x) - \lfloor v(x) \rfloor = 0 \implies v'(x) - \lfloor v'(x) \rfloor = 0$; *y*
- *para toda restricción $r \in \widehat{\Psi}$, si $v \models r$, entonces $v' \models r$.*

La relación $\simeq_{\widehat{\Psi}}$ se llama entonces relación de equivalencia de regiones temporales sobre el conjunto de restricciones sobre relojes $\widehat{\Psi}$. *Por comodidad, se nota $[v]$ para referirse a la clase de equivalencia (o la* zona*) de v.*

A continuación presentamos varias propiedades respecto de esta relación de equivalencia. Omitimos sus demostraciones; el lector interesado puede encontrarlas en [Oli94].

Propiedad 1.3.1

$\simeq_{\widehat{\Psi}}$ es una relación de equivalencia con una cantidad finita de clases. Si se nota $n = \#X$, la cantidad de clases es $O(n!2^n C^n)$ [TXJS92].

Es válido notar que la propiedad 1.3.1 nos da un marco finito para comenzar a trabajar, amén de ser significativamente grande.

Propiedad 1.3.2

Si v, $v' \in \mathcal{V}_X \wedge [v] = [v']$, $(\forall \psi \in \widehat{\Psi})(v \models \psi \Leftrightarrow v' \models \psi)$.

La propiedad 1.3.2 establece que las valuaciones de relojes dentro de una misma clase de equivalencia son indistinguibles para el conjunto $\widehat{\Psi}$ de restricciones sobre relojes.

Propiedad 1.3.3

Dado un autómata temporizado G, sea $\widehat{\Psi}_G$ el conjunto de restricciones sobre relojes que aparecen en G. Dadas dos valuaciones v, $v' \in \mathcal{V}_X$ tales que $v \simeq_{\widehat{\Psi}_G} v'$ se cumple que:

- *para toda $asig \in \Gamma_X$, $v[asig] \simeq_{\widehat{\Psi}_G} v'[asig]$; y*
- *para todo $\delta \in \mathbb{R}^+$, $(\exists \delta' \in \mathbb{R}^+)(v + \delta \simeq_{\widehat{\Psi}_G} v' + \delta')$.*

La propiedad 1.3.3 establece que, tomando como origen un estado del autómata en una región determinada, cualquier transición (sea ésta discreta o temporal), resulta en la misma región que se habría obtenido de haber tomado como origen cualquier otro estado dentro de la misma clase de equivalencia, y haber tomado la misma transición. Dicho de otro modo, $\simeq_{\widehat{\Psi}_G}$ es una congruencia respecto de las operaciones que transforman valuaciones.

1.3.2. Discretizando el sistema de transiciones

A continuación se extiende la relación de equivalencia $\simeq_{\widehat{\Psi}_G}$ a los sistemas de transiciones, para poder obtener una cantidad finita de regiones en éstos.

Definición 1.3.2 (Relación de equivalencia entre estados) *Dado G un autómata temporizado, el sistema de transiciones asociado a G $(Q, \rightarrow)$, y $\widehat{\Psi}_G$ el conjunto de restricciones sobre los relojes que aparecen en G, se define $\simeq_{\widehat{\Psi}_G} \subseteq Q \times Q$ tal que para todo $q = (s, v)$, $q' = (s', v') \in Q$ se satisface $q \simeq_{\widehat{\Psi}_G} q' \Leftrightarrow (s = s' \wedge v \simeq_{\widehat{\Psi}_G} v')$.*

Se nota $[q]$ para referirse a la clase de equivalencia *o* zona *de q.*

La relación $\simeq_{\widehat{\Psi}_G}$ satisface algunas propiedades que resultan interesantes.

Propiedad 1.3.4

$\simeq_{\widehat{\Psi}_G}$ es una relación de equivalencia con una cantidad finita de clases. Si k es la cantidad de clases de equivalencia en las que se particiona $\mathcal{V}_X$ y n es la cantidad de estados discretos del autómata, $\simeq_{\widehat{\Psi}_G}$ genera $O(kn)$ clases.

Propiedad 1.3.5

Dado $G =< S, X, E, A, I >$ un autómata temporizado, $\widehat{\Psi}_G$ el conjunto de restricciones sobre los relojes que aparecen en G y $q_1, q_2 \in Q$ tales que $q_1 \simeq_{\widehat{\Psi}_G} q_2$, se satisfacen las siguientes dos propiedades:

- *para toda $e \in E$ tal que existe $q_1{}' \in Q$ y $q_1 \rightarrow^e q_1{}'$, existe $q_2{}' \in Q$ tal que:*
  - $q_2 \rightarrow^e q_2{}'$;
  - $q_1{}' \simeq_{\widehat{\Psi}_G} q_2{}'$; *y*
- *para todo $\delta \in \mathbb{R}^+$ tal que existe $q_1{}' \in Q$ y $q_1 \rightarrow^\delta q_1{}'$, existe $q_2{}' \in Q$ y $\delta' \in \mathbb{R}^+$ tal que:*
  - $q_2 \rightarrow^{\delta'} q_2{}'$;
  - $q_1{}' \simeq_{\widehat{\Psi}_G} q_2{}'$

Esta propiedad tiene el mismo sentido que la propiedad 1.3.3, pero en referencia al sistema de transiciones.

1.3.3. Predicados sobre los sistemas de transiciones

Como se mencionó anteriormente, se cuenta con un formalismo lógico (la lógica temporal TCTL) para expresar propiedades sobre los autómatas temporizados. Sin embargo, gran parte de las propiedades que nos interesará verificar puede verse, más simplemente, como el problema de la *alcanzabilidad*. Informalmente, un estado será alcanzable desde un estado q (generalmente el inicial) si existe una ejecución perteneciente a $R_G(q)$ tal que dicho estado se encuentre en ella. Además, como parte del método de análisis, será necesario expresar propiedades sobre los estados alcanzables. Para ello, se define el conjunto $Pred$ de predicados sobre los estados del sistema de transiciones.

Definición 1.3.3 (Sintaxis de los predicados sobre Q) *Dado un autómata temporizado G tal que $G =< S, X, E, A, I >$ y $(Q, \rightarrow)$ su sistema de transiciones asociado, se define el conjunto $Pred(Q)$ de predicados sobre Q de manera inductiva sobre la siguiente gramática:*

$$\phi \rightarrow @ = s \quad | \quad \psi \quad | \quad \phi \wedge \phi \quad | \quad \neg\phi$$
$$\text{donde } s \in S \text{ y } \psi \in \Psi_X$$

Definición 1.3.4 (Semántica de los predicados sobre Q) *Dado un autómata temporizado G tal que $G =< S, X, E, A, I >$ y $(Q, \rightarrow)$ su sistema de transiciones asociado, la satisfacibilidad de un predicado $\phi \in Pred(Q)$ por un estado $(s_0, v_0) \in Q$, denotado como $(s_0, v_0) \models \phi$, se define inductivamente de la siguiente forma:*

$$\begin{array}{lcl}(s_0, v_0) \models @ = s & \Leftrightarrow & s_0 = s \\ (s_0, v_0) \models \psi & \Leftrightarrow & v_0 \models \psi \\ (s_0, v_0) \models \phi' \wedge \phi'' & \Leftrightarrow & (s_0, v_0) \models \phi' \wedge (s_0, v_0) \models \phi'' \\ (s_0, v_0) \models \neg\phi' & \Leftrightarrow & (s_0, v_0) \not\models \phi'\end{array}$$

donde $s \in S$ *y* $\psi \in \Psi_X$.

Se llama conjunto característico*—y se denota* $[\![\phi]\!]$*—al conjunto de estados que satisfacen* ϕ*:*

$$[\![\phi]\!] = \{(s, v) : s \in S \wedge v \in \mathcal{V}_X \wedge (s, v) \models \phi\}$$

Los predicados de $Pred(Q)$ satisfacen también algunas propiedades.

Definición 1.3.5 (Especialización en un estado) *Sea* $G =< S, X, E, A, I >$ *un autómata temporizado,* $s \in S$ *un estado discreto de dicho autómata,* $(Q, \rightarrow)$ *su sistema de transiciones y* $\phi \in Pred(Q)$*. Se define:*

$$\phi_s \equiv \phi|_s^{@}$$

donde se entiende a $\phi|_s^{@}$ *como el resultado de reemplazar a* $@$ *en* ϕ *por* s*.*

Propiedad 1.3.6

Sea $G =< S, X, E, A, I >$ *un autómata temporizado,* $s \in S$ *un estado discreto de dicho autómata,* $(Q, \rightarrow)$ *su sistema de transiciones y* $\phi \in Pred(Q)$*. Existe* $\psi \in \Psi_X$ *tal que* $\psi \equiv \phi_s$

La propiedad 1.3.6 establece que dado cualquier predicado de $Pred(Q)$ especializado en una locación, existe una restricción sobre relojes que lo caracteriza.

1.3.4. Los operadores suc_t, suc_e y $suc_\triangleright$

A continuación se describe una familia de operadores que serán utilizados directamente en el cálculo de satisfacibilidad de un conjunto de requisitos por parte de un autómata temporizado. En primer lugar, se definirán operadores sobre valuaciones de relojes. Estos operadores se definen tomando como referencia a los que se hallan en [Sch02], donde se presentan operadores análogos, pero para el cálculo de predecesores.

Definición 1.3.6 (El operador suc_t**)** *Sea* $G =< S, X, E, A, I >$ *un autómata temporizado,* $s \in S$*,* $\psi \in \mathcal{P}(\mathcal{V}_X)$*,* $v \in \mathcal{V}_X$ *y* $\delta \in R^+$*. Se define el operador* $suc_t^s : \mathcal{P}(\mathcal{V}_X) \rightarrow \mathcal{P}(\mathcal{V}_X)$ *de la siguiente forma:*

$$(v + \delta) \in suc_t^s(\psi) \Leftrightarrow v \in \psi \wedge (s, v) \rightarrow^{\delta} (s, v + \delta)$$

Definición 1.3.7 (El operador suc_e**)** *Sea* $G =< S, X, E, A, I >$ *un autómata temporizado,* $s \in S$*,* $\psi \in \mathcal{P}(\mathcal{V}_X)$ *y* $v \in \mathcal{V}_X$*. Se define* $suc_e^s : \mathcal{P}(\mathcal{V}_X) \rightarrow \mathcal{P}(\mathcal{V}_X)$ *de la siguiente forma:*

$$\begin{array}{rcl}v \in suc_e^s(\psi) & \Leftrightarrow & (\exists < s, e, cond, asig, s' >, v')(\langle s, e, cond, asig, s' \rangle \in A \wedge v' \in \mathcal{V}_X \implies \\ & & v' \in \psi \wedge v'[asig] = v) \wedge (s, v') \rightarrow^e (s', v)\end{array}$$

Propiedad 1.3.7

Los operadores suc_t *y* suc_e *son monótonos.*

A continuación se introducen definiciones análogas a las recién presentadas, pero para operar sobre elementos de $\mathcal{P}(Q)$. Estas definiciones se derivan también de [Sch02].

Definición 1.3.8 (El operador suc_t **sobre** Q**)** *Sea* $G =< S, X, E, A, I >$ *un autómata temporizado,* $(Q, \rightarrow)$ *su sistema de transiciones asociado,* $(s, v) \in Q$ *y* $\phi \in \mathcal{P}(Q)$*. Se define* $suc_t : \mathcal{P}(Q) \rightarrow \mathcal{P}(Q)$ *de la siguiente forma:*

$$(s, v) \in suc_t(\phi) \Leftrightarrow v \in suc_t^s(\phi_s)$$

Definición 1.3.9 (El operador suc_e sobre Q) *Sea $G =< S, X, E, A, I >$ un autómata temporizado, $(Q, \rightarrow)$ su sistema de transiciones asociado, $(s, v) \in Q$ y $\phi \in \mathcal{P}(Q)$. Se define $suc_e : \mathcal{P}(Q) \rightarrow \mathcal{P}(Q)$ de la siguiente forma:*

$$(s, v) \in suc_e(\phi) \Leftrightarrow v \in \bigcup_{\langle s', e, cond, asig, s\rangle \in A} suc_e^{s'}(\phi_{s'})$$

Definición 1.3.10 (El operador $suc_\triangleright$ sobre Q) *Sea $G =< S, X, E, A, I >$ un autómata temporizado, $(Q, \rightarrow)$ su sistema de transiciones asociado, $(s, v) \in Q$ y $\phi \in \mathcal{P}(Q)$. Se define $suc_\triangleright : \mathcal{P}(Q) \rightarrow \mathcal{P}(Q)$ de la siguiente manera:*

$$(s, v) \in suc_\triangleright(\phi) \Leftrightarrow (s, v) \in suc_t(\phi \cup suc_e(\phi))$$

1.3.5. El problema de la *alcanzabilidad*

El problema de la *alcanzabilidad* puede expresarse como:

> Dado un autómata temporizado G, un conjunto de requisitos R, cuáles son aquellos estados desde los que existe una ejecución válida del autómata, de tal forma que dicha ejecución lo hace evolucionar a estados donde los requisitos expuestos en R valen?

Dentro del model checking, resolver el problema de la alcanzabilidad es crucial. Esto se debe no sólo a que muchas propiedades pueden ser expresadas como propiedades de alcanzabilidad, sino también porque la pregunta "$G \models R$?" puede resumirse, teniendo en cuenta la respuesta a la cuestión anterior, como:

> Se encuentra el estado inicial del autómata G entre aquellos desde los cuales $[\![R]\!]$ es alcanzable?

Dicho en otras palabras, la alcanzabilidad se plantea si, partiendo del estado inicial, es posible *alcanzar* estados que satisfagan R.

A continuación describimos el método a seguir para resolver esta cuestión mediante los formalismos previamente presentados.

1.3.6. Estrategia de evaluación

Existen dentro de la bibliografía dos estrategias para abordar la cuestión de la alcanzabilidad. Son conceptualmente opuestas, en el sentido de que el recorrido que realizan sobre el sistema de transiciones es inverso.

En esta presentación, se aborda una estrategia de evaluación *forward*. Como su nombre lo indica, esta estrategia consiste en recorrer el sistema de transiciones comenzando en los estados iniciales, utilizando el operador $suc_\triangleright$ para intentar llegar a aquellos estados cuya alcanzabilidad se quiere verificar.

Existe también otra estrategia, llamada *backwards*, la cual no se presentará en este trabajo. A modo introductorio, alcanza con decir que su enfoque es el inverso: intenta verificar la alcanzabilidad de un conjunto de estados comenzando el recorrido en éstos mismos, utilizando un operador llamado $pred_\triangleright$ (análogo a $suc_\triangleright$, pero tomando como base los predecesores en lugar de los sucesores). Utilizando este operador, se intenta llegar a los estados iniciales del sistema. El método para la verificación se basa, a su vez, en un cálculo de punto fijo. Los detalles de este enfoque pueden encontrarse en [Yov96].

A continuación presentamos el procedimiento básico para la exploración *forward* del sistema, junto a algunas notas para su mejor comprensión.

La idea del algoritmo 1.3.1 consiste en explorar el espacio completo de posibilidades, chequeando en cada paso si el estado que se ha encontrado (mediante el operador $suc_\triangleright$) es realmente nuevo o si, en algn otro momento de la ejecución, ya se ha pasado por dicho estado. La función del contenedor *por_explorar* es almacenar los estados que aún no han sido visitados, mientras que la del contenedor *visitados* es, análogamente al anterior, almacenar aquellos que sí se han recorrido. El mayor problema de este algoritmo (y que lo hace extremadamente costoso) es chequear si cierto estado pertenece o no al conjunto de estados ya visitado; aquellas operaciones que se llevan a cabo en las líneas (11) y (12). Reducir la complejidad de este paso, que se realiza para cada nuevo estado generado del sistema, será el punto central que atacaremos en esta tesis.

```
1: function AlcanzabilidadForward(Requisito φ)
   ▷ Devuelve los estados alcanzables que satisfacen φ.
2:     alcanzables ← ∅
3:     visitados ← ∅
4:     por_explorar ← {(l_0, R_0)}      ▷ l_0 es la locación inicial, y R_0 la zona que representa la valuación
   inicial.
5:     while por_explorar ≠ ∅ do
6:         {(l, R)} ← Obtener(por_explorar)
7:         Quitar({(l, R)}, por_explorar)
8:         if {(l, R)} ⊨ φ then
9:             Agregar({(l, R)}, alcanzables)
10:        end if
11:        Rs ← ⋃_{{(l,R')}∈visitados} R'
12:        if R ⊈ Rs then
13:            Agregar({(l, R)}, visitados)
14:            siguientes ← { {(l'R')}/{(l', R')} ∈ suc_▷({(l, R)})}
15:            por_explorar ← por_explorar ∪ siguientes
16:        end if
17:    end while
18:    devolver alcanzables
19: end function
```

Algoritmo 1.3.1: Algoritmo de alcanzabilidad con exploración *forward*.

Capítulo 2

Antecedentes bibliográficos

Dada la complejidad a veces privativa de verificar propiedades en modelos temporizados, los trabajos realizados en el campo del model checking han tenido en gran parte un denominador común: sea mediante la distribución del trabajo en varias unidades de cómputo, optimización de algoritmos [BPO03] y estructuras, o reducción del sistema a verificar [BGO04, Daw98], siempre se busca atacar el problema de la explosión combinatoria de estados. Esta explosión es producto de la composición de los distintos autómatas que representan el sistema. Por otra parte, ya vimos que la discretización del sistema genera una cantidad inmensa de regiones de equivalencia.

En el transcurso de esta tesis nos concentraremos en desarrollar una estructura de datos que logre el objetivo de reducir la complejidad de resolver el problema de la alcanzabilidad. Por ello, en este capítulo presentaremos los esfuerzos realizados hasta el momento en esta área.

En los model checkers temporizados existentes que han alcanzado cierto grado de éxito, se han utilizado casi con exclusividad las DBMs (del inglés *Difference Bound Matrices*, matrices de cotas de diferencias) para representar las clases de equivalencia de los valores de los relojes. Estas estructuras están presentadas en detalle en [Dil90], por lo que nos limitaremos a dar una introducción que nos permita referirnos a ellas con comodidad en el resto del trabajo.

2.1. DBMs

Definición 2.1.1 (Cotas) *Una cota es un par $(c, \prec)$, donde $c \in \mathbb{Z} \cup \{\infty\}$ (entendiendo a ∞ como una constante más) y $\prec \in \{<, \leq\}$. Se define también el orden en $\mathbb{Z} \cup \{\infty\}$ extendiendo el orden en $\mathbb{Z}$ de forma que para todo $c \in \mathbb{Z}$, valga que $c < \infty$. Asimismo, se define el orden en $\{<, \leq\}$ de forma que $<$ es menor que $\leq$. Además, dada una cota $C = (c, \prec)$, notaremos $-C$ a la cota $(-c, \prec)$.*

Definición 2.1.2 (Orden total entre cotas) *Se dice que una cota $(c, \prec)$ es más estricta que otra $(c', \prec')$—notado $(c, \prec) < (c', \prec')$—si $c < c'$ o bien $c = c'$ y $\prec < \prec'$.*

Definición 2.1.3 (DBM) *Una DBM de dimensión n es una matriz cuadrada de $(n+1) \times (n+1)$, cuyos elementos son cotas.*

La idea detrás de esta estructura es, dada una DBM M, que M_{ij} representa la cota superior de la diferencia de relojes $x_i - x_j$. Por ejemplo, $x_i - x_j \leq 8$ se representará guardando en M_{ij} la cota $(8, \leq)$, mientras que $x_i - x_j > 2$ se codificará con la cota $(-2, <)$ en M_{ji}. Mediante esta codificación, todas las posibles DBMs representan el subconjunto de los poliedros convexos[1] que pueden denotar las soluciones de una restricción temporal. Por ejemplo, la restricción $x_1 \leq 2 \wedge x_2 > 3$ se ve representada, con una DBM y geométricamente, en la figura 2.1.

Las DBMs no sólo tienen utilidad para representar las zonas temporales, sino que además las operaciones sobre estas zonas se traducen directamente como operaciones simbólicas sobre las DBMs. Además,

[1] Informalmente, se llama *convexos* a los poliedros (abiertos o cerrados) tales que dados dos puntos cualesquiera del mismo, el segmento que los une se encuentra íntegramente dentro del poliedro.

$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} (0,\leq) & (\infty,<) & (-3,<) \\ (2,\leq) & (0,\leq) & (\infty,<) \\ (\infty,<) & (\infty,<) & (0,\leq) \end{pmatrix} \end{array}$$

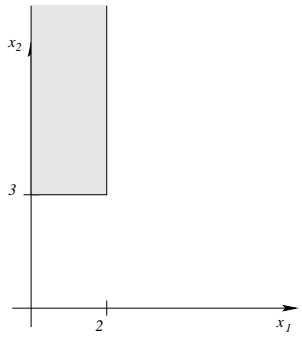


Figura 2.1: Una DBM y la zona representada.

cumplen con ciertas propiedades que las hacen aún más útiles para la verificación temporizada. A continuación mencionamos las operaciones y propiedades más importantes, dejando al lector la posibilidad de ahondar en ellas en [Dil90].

Observación 2.1.1 *Si bien existe más de una manera de representar una zona sobre una DBM, puede definirse una* forma canónica *tal que dos DBMs M y M' representan la misma zona si y sólo si son idénticas. Esto reduce la verificación de igualdad entre DBMs a un chequeo prácticamente sintáctico.*

La observación 2.1.1 ofrece una forma fácil de verificar vacuidad (o universalidad) de la zona representada por la DBM; de otra manera, sería necesario resolver las correspondientes restricciones temporales y verificar su equivalencia. Por otra parte, es también útil para la mayoría de las operaciones, ya que requerirán que sus parámetros se hallen en forma canónica. A continuación detallamos cómo puede transformarse una DBM cualquiera a su equivalente en forma canónica.

Definición 2.1.4 (Obtención de forma canónica de una DBM) *Dada cualquier DBM M, es posible obtener la forma canónica de la misma. Informalmente, esta forma canónica es tal que ninguna de las cotas puede ajustarse*[2] *más sin alterar el poliedro original.*

En la práctica, se puede obtener la forma canónica de una DBM representándola como un grafo con ejes pesados, y aplicando algoritmos de obtención de caminos mínimos sobre la misma; por ejemplo, el algoritmo de Floyd-Warshall. Si n es la cantidad de relojes presentes en el sistema, obtener la forma canónica de una DBM es $O(n^3)$, por lo que se intenta mantener la forma canónica durante la ejecución de las demás operaciones, ya que canonizar resulta una operación costosa.

Operaciones sobre DBMs

A continuación definiremos las operaciones más salientes a realizar con las DBMs; aquellas que serán necesarias para llevar a cabo la verificación temporizada.

Definición 2.1.5 (Intersección) *Dadas dos DBMs M y M' de dimensión n, se define la DBM intersección de ambas, I, también de dimensión n, de la siguiente forma, para todo $0 \leq i, j \leq n, i \neq j$:*

$$I_{ij} = \text{mín}(M_{ij}, M'_{ij})$$

donde mín *denota el mínimo según el orden presentado en la definición 2.1.2.*

Definición 2.1.6 (Chequeo de inclusión) *Dadas dos DBMs M y M' de dimensión n, y tales que ambas se encuentran en forma canónica, puede determinarse si M representa una zona incluida en M' mediante la siguiente definición:*

$$M \subseteq M' \Leftrightarrow \forall 0 \leq i, j \leq n, M_{ij} \leq M'_{ij}$$

[2]Se entiende por ajustar una cota al hecho de tornarla más estricta.

Observación 2.1.2 *En realidad, para asegurar la correcta operación de chequeo de inclusión de la DBM M en M', sólo es necesario que M' se encuentre en forma canónica, y no M [BY]. Esta observación cobrará mayor importancia más adelante cuando presentemos las RDBMs.*

Las próximas dos operaciones que introduciremos conforman la abstracción del uso de relojes como medición de tiempo sobre los poliedros que utilizamos para representarlos. Si bien son operaciones geómetricas, las notaremos con el nombre que se les suele dar en el algoritmo simbólico.

Definición 2.1.7 (Reseteo de relojes) *Dada una DBM M se define la DBM R, que se obtiene como representación del reseteo de los relojes de un conjunto X, de la siguiente manera, para todo $0 \leq i, j \leq n$:*

$$R_{ij} = \begin{cases} (0, \leq) & si\ (x_i \in X \wedge x_j = 0) \vee (x_j \in X \wedge x_i = 0) \vee (x_i \in X \wedge x_j \in X) \\ M_{0j} & si\ x_i \in X \wedge x_j \notin X \\ -M_{i0} & si\ x_j \in X \wedge x_i \notin X \\ M_{ij} & en\ otro\ caso \end{cases}$$

Puede verse fácilmente que la operación presentada en la definición 2.1.7 es en realidad una proyección del poliedro sobre los ejes que representan los relojes que se resetean.

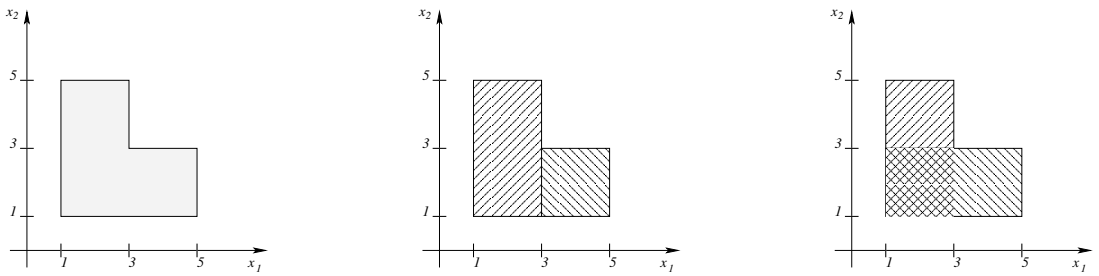
Definición 2.1.8 (Avance del tiempo) *Dada una DBM M, se define la DBM A, que representará las valuaciones de los relojes al avanzar el tiempo, de la siguiente manera, para todo $0 \leq i, j \leq n$:*

$$A_{ij} = \begin{cases} (\infty, <) & si\ j = 0 \\ (-\infty, <) & si\ i = 0 \\ M_{ij} & en\ otro\ caso \end{cases}$$

2.1.1. RDBMs

Las RDBMs (por *Reduced* DBMs, DBMs reducidas), descriptas entre otros en [BY], son una manera alternativa de representar las zonas. La idea es idéntica a la de las DBMs; sin embargo, el espacio que consumen es menor. La motivación detrás de la reducción es que, en general, resulta innecesario almacenar *todas* las diferencias entre los relojes, ya que varias de ellas se deducen a partir de las demás. El procedimiento de reducción consiste, como en el caso de la obtención de la forma canónica, en encontrar las cotas redundantes mediante algoritmos de determinación de caminos mínimos.

La representación obtenida mediante este proceso efectivamente anula las cotas innecesarias. Sin embargo, dado que podrían existir varios subconjuntos minimales de cotas que describan la misma DBM, la representación de las RDBMs no es canónica. Esto, en general, no es un problema ya que la mayoría de las operaciones que describimos anteriormente para DBMs pueden trasladarse a RDBMs sin cambios, ya que no exigen que las DBMs se hallen en forma canónica. La excepción a esta apreciación es la operación de inclusión, pero la observación 2.1.2 ofrece un resultado que permite, de todas maneras, obtener una reducción del tiempo requerido también para esta operación.



(a) Poliedro no convexo (b) Posible representación con uniones de convexos (c) Otra posible representación

Figura 2.2: Poliedro no convexo y posibles representaciones mediante convexos

Más allá de la reducción del tamaño, los espacios que pueden representarse tanto con DBMs como con RDBMs son los mismos. Por otra parte, si bien ambas estructuras pueden representar sólo cierto

subconjunto de poliedros convexos, suelen utilizarse conjuntos de DBMs (o RDBMs) para representar espacios no convexos. Evidentemente, tampoco existe una forma canónica de representar este espacio: múltiples conjuntos de poliedros convexos pueden representar, unidos, el mismo poliedro no convexo. Por ejemplo, el poliedro de la figura 2.2a puede representarse mediante esta noción conjuntista de varias formas: las figuras 2.2b y 2.2c son posibles representaciones. Puede notarse en la figura 2.2c que los conjuntos no son necesariamente disjuntos. Este tipo de problemas, como veremos luego, es el punto que se ataca utilizando el resto de las estructuras.

2.2. Diagramas de decisión y sus aplicaciones al model checking

Las estructuras de decisión no son ajenas al model checking, al menos en el caso de los sistemas no temporizados, ya que han tenido considerable éxito en varios de ellos, como SPIN [Hol97] y Rabbit [BLN03]. Por otra parte, ya desde sus comienzos se han utilizado para representar simbólicamente los estados discretos del sistema de transiciones en varios model checkers temporizados„ lo cual se muestra en [DWT95]. A continuación presentamos entonces los BDDs, que conforman la base de todos los trabajos posteriores en esta subárea del problema.

2.2.1. Diagramas Binarios de Decisión

Los Diagramas Binarios de Decisión (BDDs, por el inglés *Binary Decision Diagrams*), son las estructuras en las cuales se funda nuestro trabajo. Estas estructuras no fueron desarrolladas con el model checking temporizado en mente, sino que fueron presentadas en [Bry86] como una forma de representar funciones booleanas (es decir, aquellas donde sus variables y su resultado son booleanas).

Definición 2.2.1 (BDD) *Un BDD es un grafo dirigido, con conjunto de vértices V, con un nodo distinguido llamado* raíz*, Cada vértice $v \in V$ puede ser* terminal *o* no terminal*. Los vértices no terminales poseen un índice—$indice(v) \in \{1..n\}$—y dos hijos $cero(v)$ y $uno(v)$, tal que ambos pertenecen a V. Los vértices terminales tienen un valor—$valor(v) \in \{0, 1\}$.*

Notaremos, cuando el contexto lo permita, tan sólo v para denotar el BDD cuya raíz es v.

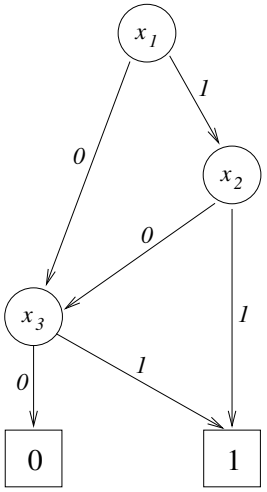


Figura 2.3: Un BDD

Intuitivamente, los vértices no terminales de un BDD representan las variables de la función booleana, mientras que sus hijos *cero* y *uno* representan las funciones que se obtienen fijando sus variables, justamente, en 0 y 1 respectivamente. Realizando este proceso de manera sucesiva, se alcanza algún vértice terminal, que representará o bien la función 0, o bien 1. Vale notar que dada cierta asignación a las variables, el camino a recorrer es único. Además, no existen nodos del BDD que sean inalcanzables: siempre existe una asignación inicial que resulta en un camino que llega a un nodo dado. Por ejemplo, el BDD que se exhibe en la figura 2.3 es la representación de la función booleana $(x_1 \wedge x_2) \vee x_3$.

Sin embargo, los BDDs alcanzan mayor potencial al *reducirlos*. Avanzaremos con unas definiciones y propiedades que nos permitirán ver esta particularidad. No incluiremos las demostraciones, las cuales pueden ser consultadas en la fuente citada anteriormente.

Definición 2.2.2 (Isomorfismo de BDDs) *Dos BDDs B y B', con conjuntos de vertices V y V' respectivamente, se dicen* isomorfos *si existe una función biyectiva $\sigma : V \to V'$ tal que para todo $v \in V$, si $\sigma(v) = v'$, se cumple que:*

- *o bien tanto v como v' son terminales y $valor(v) = valor(v')$;*
- *o bien ambos son no terminales, y tales que $indice(v) = indice(v')$, $\sigma(cero(v)) = \sigma(cero(v'))$ y $\sigma(uno(v)) = \sigma(uno(v'))$.*

Definición 2.2.3 (RBDD (BDD reducido)) *Un BDD se dice* reducido*—y se nota* RBDD*, por el inglés* reduced *BDD—si no tiene ningún vértice v tal que $cero(v) = uno(v)$, ni tampoco posee dos vértices v, v' tales que $v \neq v'$ y además sean isomorfos.*

De esta manera, puede verse que dada una función booleana f y un orden inicial entre las variables, existe un único RBDD que la representa (exceptuando isomorfismos). En la literatura, suele llamarse *ROBDD*—BDD reducido y ordenado, por el inglés *Reduced Ordered BDD*—a los RBDDs que respetan un orden entre sus variables.

Este orden de variables no es un factor menor; el tamaño que puede llegar a desarrollar un RBDD depende fuertemente del ordenamiento de sus variables (en el sentido en que se encuentran tomando el recorrido desde la raíz). Como puede verse en la figura 2.4, ambos RBDDs representan la misma función booleana, $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$, pero el ordenamiento resulta crítico en el tamaño del grafo. En general, deben estudiarse la naturaleza y estructura del problema para determinar el ordenamiento de las variables que producirán el RBDD más compacto.

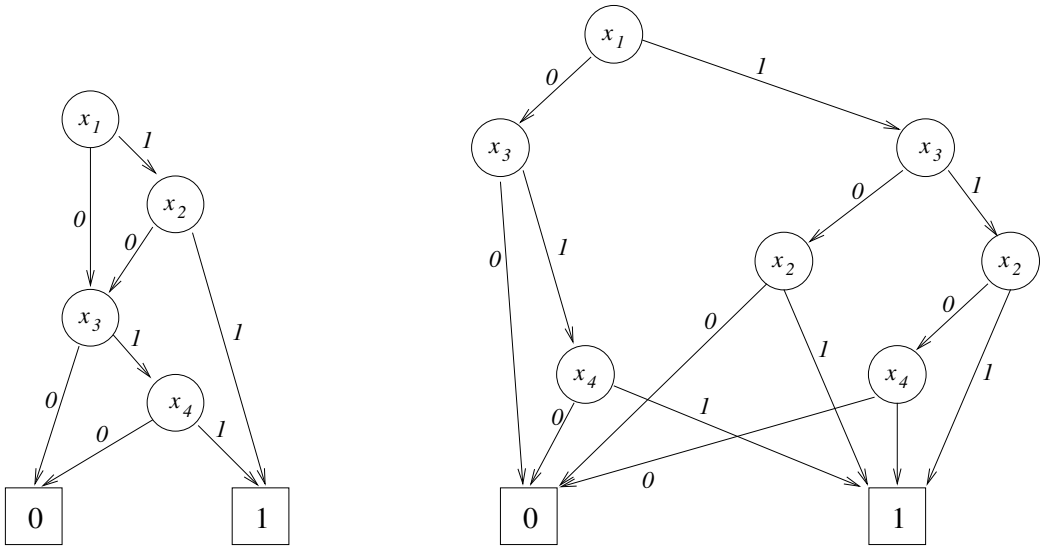


Figura 2.4: RBDDs con distinto orden

En la última década, varios trabajos han avanzado con el objetivo de generalizar los resultados obtenidos mediante los RBDDs en otras áreas, intentando integrarlos en las herramientas de model checking como una forma de representar las regiones temporales. La idea detrás de este enfoque es intentar representarlas de manera compacta. Esto contrasta con la representación mediante DBMs, donde las regiones simplemente se representan mediante un conjunto de zonas. Esta representación no es óptima, ya que este conjunto podría estar demasiado atomizado, o también ser muy redundante, con muchas zonas tales que sus intersecciones son no vacías. El objetivo buscado es obtener una representación tal que estos problemas sean minimizados.

De esta manera, se intenta reducir tanto el tamaño de las estructuras como el tiempo de ejecución necesario para arribar a un resultado. En esta dirección, en [Bal96] se realizó un primer intento de utilizar

BDDs para codificar directamente la unión de las DBMs. Sin embargo, al utilizar esta representación el problema de calcular la unión de clases de equivalencia no se soluciona de manera exacta, sino de manera aproximada, mediante el uso de la cápsula convexa de la solución real. Esta aproximación es conservadora, dado que resulta en casos de falsos positivos para la alcanzabilidad, pero no cae en falsos negativos.

Como consecuencia de estos trabajos, el esfuerzo fue enfocándose cada vez más en el desarrollo de soluciones que permitiesen la representación discreta pero *no necesariamente de bifurcación binaria* de las distintas posibilidades de decisión. En este sentido es que surgen los Diagramas de Diferencias de Relojes (CDDs) [BLP+99], los Diagramas de Restricción de Relojes (CRDs) [Wan03] y los Diagramas de Codificación de Regiones (REDs) [Wan00], que presentaremos más adelante. En pasos intermedios, existieron otros trabajos de menor impacto, como los Diagramas de Decisión Algebraicos (ADDs, por el inglés *Algebraic Decision Diagrams*) [REC+93], y los Diagramas de Decisión Numéricos (NDDs, del inglés *Numerical Decision Diagrams*) [EMA+97]. Estas estructuras, si bien se plantearon con la posibilidad de su uso en el ámbito del model checking, sufren de diversas debilidades, principalmente debido a la naturaleza exclusivamente binaria de las decisiones en cada nodo.

El paso siguiente, entonces, fue desarrollar estructuras con el objetivo de representar el espacio denso de los valores de los relojes, ya no codificándolos mediante sus representaciones binarias, sino utilizando fuertemente el contexto de los problemas surgidos durante el recorrido de los sistemas temporizados. Como resultado de esta línea de trabajo aparecen los Diagramas de Decisión de Intervalos (IDDs, proveniente del inglés *Interval Decision Diagrams*) [KL98], y otras variantes, donde se comienza a abandonar las decisiones binarias, transformándolas en n-arias.

Como mencionamos anteriormente, otros han tomado la posta y, heredando las ideas de estas estructuras, hoy en día cuentan con mayor impulso otras alternativas que, sin embargo, carecen de validación independiente que pueda sostener (o dar por tierra) sus virtudes. A continuación presentaremos estas tres estructuras.

2.2.2. Diagramas de Diferencia de Relojes

Los diagramas de diferencias de relojes (CDDs, del inglés *Clock Difference Diagrams*) [BLP+99] surgen como variante a los IDDs. En un CDD, los nodos del diagrama representan diferencias de relojes (a diferencia de los IDDs, que sólo representan a cada reloj), mientras que cada eje saliente de un nodo está etiquetado con un intervalo. De esta manera, el eje representa el camino a seguir en el diagrama si la diferencia de las valuaciones se encuentra comprendida en dicho intervalo, dado que los ejes salientes de un mismo nodo contienen intervalos disjuntos entre sí. Por ejemplo, la figura 2.5 ilustra un CDD junto con la región que éste representa; para no complicar la lectura del gráfico, se han omitido los intervalos cuyos ejes guían directamente al nodo $False$ (y se ha omitido también el mismo nodo $False$), aunque es válido aclarar que dicha omisión no existe en la definición de la estructura.

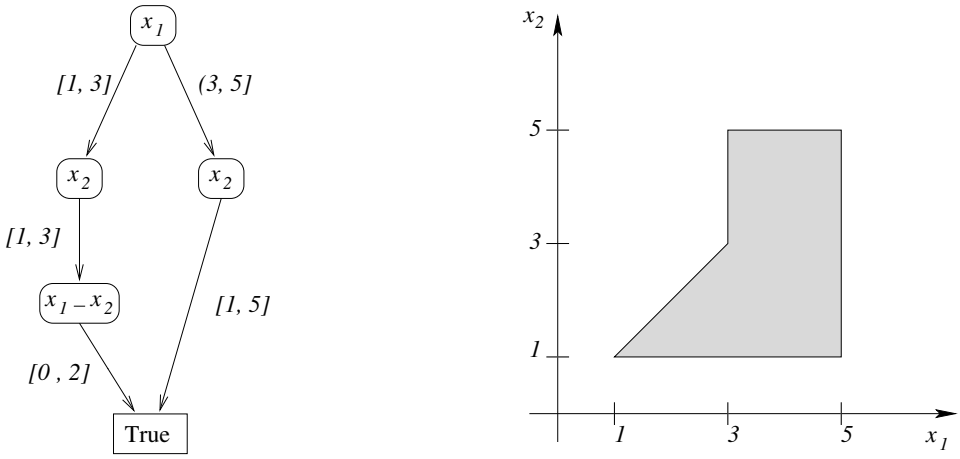


Figura 2.5: Un CDD y la región representada

En su presentación, los autores analizan también las posibilidades de compartir los subdiagramas isomorfos dentro del mismo diagrama, pero no exhiben detalles acerca de cómo hacerlo. Esto es impor-

tante, ya que es el principal argumento para sostener su bajo consumo de memoria. Además, presentan algunos algoritmos sobre esta estructura, todos ellos recursivos. Sin embargo, aparte de las cuestiones de eficiencia que surgen por el uso de recursión, estas operaciones sólo apuntan a utilizar la estructura como repositorio de regiones ya visitadas, es decir, como implementación del contenedor *visitados* del algoritmo 1.3.1, sin utilizarlas para el ciclo completo del model checking. Por otra parte, si bien exhiben algunos resultados positivos, no nos fue posible contrastarlos ya que las versiones disponibles de la herramienta que implementa esta estructura no presentan opciones para el uso de las mismas. De todas maneras, esta estructura parece ser la más promisoria de las tres que presentamos, razón por la cual nos sostendremos en ella para desarrollar esta tesis.

2.2.3. Diagramas de Restricción de Relojes

Los diagramas de restricción de relojes (CRDs, por su nombre en inglés, *Clock Restriction Diagrams*) [Wan03] son conceptualmente muy similares a los CDDs. La mayor diferencia radica en que, a diferencia de los anteriores, tan sólo codifican la cota superior de cada diferencia de relojes. De esta manera, efectivamente el tamaño de la estructura es considerablemente menor. Esta reducción de tamaño no sólo se debe al ahorro de espacio al no codificar la cota inferior (que queda implícitamente representada por $-\infty$) sino que también se reduce la expansión a lo ancho de cada nodo. Esto último se debe a que múltiples caminos pueden representarse mediante la misma cota superior. Sin embargo, esta ventaja en el uso del espacio es contrastada con un presumible aumento significativo en el tiempo de ejecución, ya que dada una valuación de relojes, no siempre podrá verificarse si se satisface siguiendo un sólo camino. El mismo solapamiento obliga a recorrer múltiples ramas del diagrama. Además, dado que sólo es posible codificar la cota superior de una diferencia, se vuelve potencialmente necesario representar ambos pares de diferencias: tanto $x_i - x_j$ como $x_j - x_i$ para cada par de relojes x_i, x_j. Esto trae como consecuencia directa un crecimiento vertical del diagrama. Por otra parte, si bien en [Wan03] se presentan distintas ideas para reducir el tamaño de un CRD, ninguna de ellas resulta ser una forma canónica; y existen casos en que los algoritmos presentados no encuentran una representación satisfactoria de ciertas regiones.

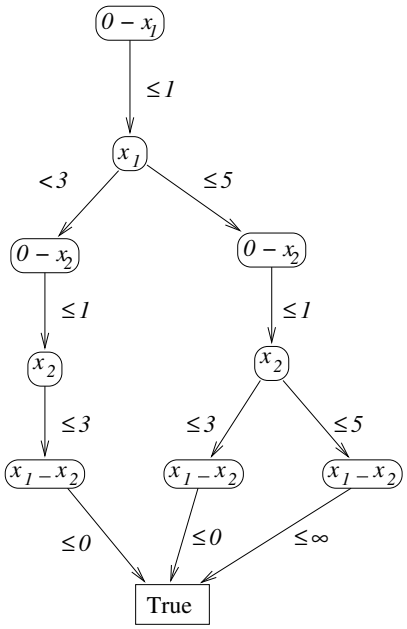


Figura 2.6: Un CRD.

La figura 2.6 muestra un ejemplo de CRD. El problema principal, como se dijo anteriormente, se refleja en la superposición de intervalos. Las soluciones a este problema residen en mayor atomización de los hijos de cada nodo, lo cual se aproxima rápidamente a la definición de los mismos CDDs. Por otra parte, los algoritmos presentados en las publicaciones son de naturaleza recursiva, y fundados en otras operaciones donde no se expresa de qué manera pueden lograrse de manera eficiente; por ejemplo, hacen uso intensivo de técnicas de *aliasing* para reducir espacio, pero no se explica cómo es posible detectarlo.

2.2.4. Diagramas de Codificación de Regiones

Los diagramas de codificacion de regiones (REDs, por su nombre en inglés *Region Encoding Diagrams*) [Wan00] surgen casi paralelamente a los CRDs (de hecho, sus autores son los mismos en ambos casos). Su naturaleza es un poco más compleja, dado que los nodos del diagrama de decisión no representan diferencias de relojes, sino tan sólo relojes. Esto los hace muy similares a los IDDs; sin embargo, el *rationale* de esta estructura es dar un ordenamiento dinámico entre los relojes: este orden se desprende de la parte fraccionaria de la valuación de los relojes, de forma tal que un reloj a se encuentra antes dentro del orden que otro b si su parte fraccionaria es menor a la de este último, sin que importe la parte entera de la valuación. Esto permite que los límites de las ramas sean *disjuntos*, con un rango de disyunción no mayor a 1. Es necesario notar que esta situación conlleva dos problemas de solución no trivial:

- En primer lugar, el hecho de no conocer la diferencia explícita de los relojes, y poseer un orden dinámico de los mismos, obliga a recorrer el árbol en ambos sentidos (no sólo de "arriba hacia abajo"). Este recorrido se vuelve cada vez más complejo a medida que se agregan ramas en un mismo nodo. El orden dinámico obliga también a que la estructura deba ser reordenada muy frecuentemente.
- Además, y en estrecha relación con lo anterior, el hecho de tener intervalos disjuntos a *distancia uno* causa que, cuanto mayor sea la máxima constante en el sistema, mayor sea la atomización de las ramas. La combinación entre no codificar la diferencia completa y no tener en cuenta las partes enteras de las valuaciones resulta en que la subdivisión de las ramas sea mayor que en el caso de las otras estructuras.

Por otra parte, la bibliografía es poco detallada respecto de esta estructura. Debido a estas dificultades, no se muestra un RED, sino que ilustramos sólo la forma de sus nodos. Este diagrama, presentado en la figura 2.7 ha sido extraído fielmente de [Wan00].

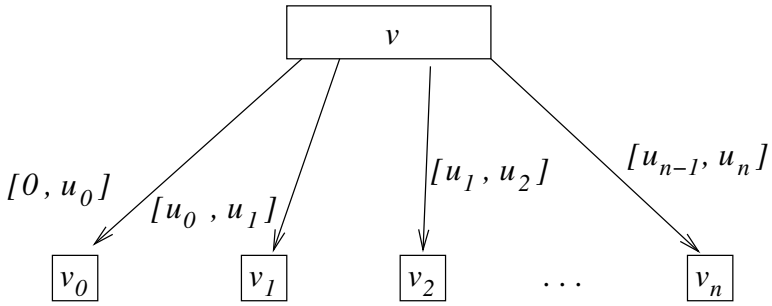


Figura 2.7: Un nodo de RED

Además de este problema, los autores hacen notar que la estructura sólo se utiliza en un subconjunto muy estricto de los problemas a verificar: se usa solamente en aquellos casos donde el sistema es una composición de procesos *iguales*, y donde además cada proceso cuenta con un sólo reloj local, sin existencia de reloj global. Este factor hace que la estructura no sea muy útil para el caso general.

2.3. Resumiendo

Las estructuras clásicas (las DBMs) presentan propiedades que las hacen eficientes para el cálculo de zonas, pero sin embargo son extremadamente ineficientes para representar regiones, ya que no permiten una codificación homogénea de espacios no convexos. Por otra parte, los intentos por adaptar la estructura de búsqueda de los BDDs a fin de representar regiones han sido varios, con diferentes grados de éxito. De las estructuras mencionadas en este capítulo, sólo aquellas cuyas ramificaciones no son binarias parecen ser las indicadas. Esto surge del hecho de que estas estructuras no han sido directamente adaptadas de los BDDs, sino que han intentado tomar un camino propio, aplicando directamente las características del problema a resolver. Sin embargo, ninguna de ellas ha completado el ciclo necesario para realmente resolver el problema del model checking de manera completa. Más aún, no nos ha sido posible contrastar las afirmaciones manifestadas por los autores: en el caso de los REDs y CRDs, no

sólo sirven exclusivamente para cierto subconjunto de problemas, sino que además nos ha sido imposible ejecutar la herramienta que los implementa; mientras que en el caso de las herramientas que operan con CDDs no se encuentran opciones para utilizarla como medio de verificación; aún así, no permiten realizar la verificación completa, ya que no se independizan de las DBMs (las siguen utilizando en todos los cálculos intermedios).

Capítulo 3

Extendiendo la estructura—*e*CDDs

Or il y avait des graines terribles sur la planète du petit prince... c'étaient les graines de baobabs. Le sol de la planète en était infesté. Or un baobab, si l'on s'y prend trop tard, on ne peut jamais plus s'en débarrasser. Il encombre toute la planète. Il la perfore de ses racines. Et si la planète est trop petite, et si les baobabs sont trop nombreux, ils la font éclater.[1]

3.1. Estructura, invariantes y objetivos

3.1.1. Descripción general

En capítulos anteriores presentamos el problema del model checking temporizado, y las distintas estructuras sobre las cuales se ha estado trabajando en los últimos tiempos para mejorar los rendimientos de las herramientas que realizan este tipo de verificación. También hemos destacado los aspectos más salientes de cada una de ellas, mostrando sus debilidades y fortalezas. Luego de dicho análisis, concluimos que no todas las estructuras resultaban eficientes o útiles para el objetivo que se persigue, aunque destacamos algunas que se encontraban por arriba de la media en función de sus características positivas. Sin embargo, encontramos el problema de que no se hallan completamente definidas ni hemos podido realizar experimentos que nos permitiesen argumentar sobre resultados empíricos.

[1] Antoine de Saint-Exupéry, *Le Petit Prince*, 1943. La ilustración es del autor.

Como conclusión de los capítulos anteriores, notamos que los CDDs se posicionaban como las estructuras más promisorias en ese sentido. En el transcurso de este capítulo, extenderemos las nociones de los mismos, de forma de suplir las falencias que posee dicha estructura, además de perseguir el objetivo de realizar la *totalidad* del procedimiento de verificación utilizando sólo este tipo de estructuras. En este desarrollo, intentamos en todo momento obtener funciones tales que fuesen más simples (desde el punto de vista computacional, no necesariamente desde el punto de vista conceptual), especialmente en lo que se refiere a uniones de zonas (que resultan en regiones no convexas) y el chequeo de inclusiones entre ellas que, como notamos anteriormente, son críticas para el chequeo de alcanzabilidad que nos proponemos realizar.

Este capítulo recorrerá el siguiente camino:

- En primer lugar, formalizaremos las nociones de los CDDs, extendiéndolas de manera tal de obtener la estructura con la cual trabajaremos, los *e*CDDs.
- A continuación, introduciremos las operaciones a implementar sobre estas estructuras. Lo haremos conceptualmente, utilizando para su definición funciones recursivas.
- Una vez alcanzados estos objetivos, definiremos formalmente las nociones de satisfacibilidad de restricciones temporales por medio de los *e*CDDs.
- Utilizando los resultados y definiciones anteriores, exhibiremos implementaciones eficientes para las operaciones, demostrándolas correctas en cada uno de los casos.

Intuitivamente, cada nodo de un *e*CDD constará de:

(I) En algunos casos, sólo un valor de verdad (*True* o *False*) indicando si la valuación determinada por las distintas decisiones tomadas hasta llegar a dicho nodo pertenece o no a la región representada. Este tipo de nodos será terminal, de manera que el recorrido del diagrama culminará en ellos.

(II) En los casos que no se encuentren comprendidos por las características anteriores, los nodos incluirán la diferencia de relojes que representa; por ejemplo, $x_4 - x_2$, o $x_5 - x_0$. Recordemos que el reloj x_0 representará al reloj ficticio, e indicará el tiempo consumido desde el inicio del sistema.

(III) Los diferentes intervalos que componen el espectro total de valores que dicha diferencia puede comprender. Los valores de estos intervalos estarán sujetos a ciertas restricciones, que formarán parte del invariante de la estructura.

(IV) Para cada uno de estos intervalos, un nodo asociado que, o bien se trata de una nueva decisión, o bien del valor de verdad final; alguno de los dos tipos de nodos descriptos anteriormente.

Con estas nociones en mente, a continuación los definiremos de manera formal, lo cual nos será útil tanto para describir los algoritmos, como para probar la corrección de los mismos.

Definición 3.1.1 (*e*CDDs) *Dado un conjunto C de relojes, $C' = C \cup \{x_0\}$, se define un* e*CDD por medio de su raíz k, para la cual se definen los siguientes atributos:*

- *diff*$(k) \in (C \times C') \cup \{TRUE, FALSE\}$*, la* diferencia *de relojes representada por el nodo. Se extiende con los valores TRUE y FALSE, en cuyo caso k se dirá* terminal.
- *Para cada nodo $k \neq TRUE, FALSE$; se define la función $Ints(k)$, que obtiene los intervalos asociados a la diferencia de relojes representada por dicho nodo. $Ints(k)_i$ representará el i-ésimo intervalo asociado al nodo*[2]*. Notamos $\#_{Ints}(k)$ a la cantidad de intervalos obtenidos mediante $Ints(k)$.*
- *Análogamente a lo anterior, para cada nodo k, se define también la función $S(k)$, que obtiene los sub-*e*CDDs asociados a cada uno de los intervalos obtenidos mediante $Ints(k)$*[3]*. Definimos $S(k)_i$ y $\#_S(k)$ de manera análoga a $Ints(k)_i$ y $\#_{Ints}(k)$. Claramente, para cualquier nodo k tal que $k \neq TRUE, FALSE$; $\#_{Ints}(k) = \#_S(k)$, y $S(k)_i$ es tal que devolverá el sub-*e*CDD asociado al i-ésimo intervalo.*

[2] La noción de "i-ésimo intervalo" cobrará mayor sentido una vez que se haya visto el invariante de la estructura.

[3] La definición formal de un sub-*e*CDD se dará más adelante, una vez establecido el invariante. Por el momento, puede suponerse que un sub-*e*CDD es equivalente a un *e*CDD.

Así, un nodo k de un eCDD puede expresarse como una tupla $[d, Is, Ss]$, donde d es la diferencia de relojes representada (o $TRUE$ o $FALSE$ en los casos que correspondan)—$diff(k) = d$—, Is es la lista de intervalos posibles para dicha diferencia—$Ints(k) = Is$—, y Ss la lista de sub-eCDDs correspondiente a cada uno de los intervalos anteriores—$S(k) = Ss$. En algunos casos, y sólo al efecto de simplificar las notaciones diremos, además, que un nodo k de por sí representa un eCDD, y que un eCDD k es $TRUE$ si $diff(k) = TRUE$, y que es $FALSE$ si $diff(k) = FALSE$. Estas abreviaturas serán claras según el contexto en que se expresen. Además, ocasionalmente extenderemos la definición de S, de forma de poder obtener el sub-eCDD correspondiente a un intervalo dado I. Formalmente, definiremos que $S(k, I) = k_s \Leftrightarrow \exists\, i\,, 1 \leq i \leq \#_{Ints}(k), Ints(k)_i = I \wedge S(k)_i = s$.

Los eCDDs, de manera muy similar a los CDDs, revisten la posibilidad de mantener en una misma estructura, de manera homogénea, el conjunto completo de zonas que conforman una región. Sin embargo, para lograr esa homogeneidad, se hace necesario que se cumplan ciertas restricciones, que conformarán el *invariante* de la estructura, extendiendo las nociones introducidas en la presentación de los CDDs. Sin embargo, antes de proceder a dicha definición, necesitaremos establecer la noción de *orden* entre las diferencias de relojes presentes en un sistema.

Definición 3.1.2 (Orden entre diferencias de relojes) *Dado el conjunto $C \cup \{x_0\}$ de relojes de un sistema, y tomando como base un* orden total *$\prec \subseteq (C \cup \{x_0\}) \times (C \cup \{x_0\})$ entre ellos, de manera tal que para cada $x_i \in C$, $x_0 \prec x_i$, se define la relación de* orden total entre diferencias de relojes *de $C \cup \{x_0\}$ de la siguiente manera: (abusaremos del uso de $\prec$ para definir también este orden)*

$\forall x_a, x_c \in C; x_b, x_d \in C \cup \{x_0\}, x_a - x_b \prec x_c - x_d$ *si y sólo si se cumple alguna de las siguientes:*

(I) $x_b = x_0$ *y* $x_d \neq x_0$;

(II) $x_b = x_d = x_0$ *y* $x_a \prec x_c$;

(III) $x_b \neq x_0, x_d \neq x_0$ *y* $x_a \prec x_c$;

(IV) $x_b \neq x_0, x_d \neq x_0, x_a = x_c$ *y* $x_b \prec x_d$.

Observación 3.1.1 *Es importante notar que las definiciones* I *y* II *aseguran que las diferencias que determinan cotas individuales para cada reloj sean, efectivamente, menores que las diferencias que aparean relojes distintos de x_0.*

De aquí en adelante abusaremos de las notaciones $<$ y $\leq$ para referirnos tanto al orden usual entre relojes como también al orden entre diferencias de relojes inducido por el mismo.

Por otra parte, también nos será útil la noción de sub-eCDD para expresar el invariante.

Definición 3.1.3 (sub-*e*CDD) *Dado un* e*CDD $k = [d, Is, Ss]$, se llama sub-*e*CDD de k a cada uno de los elementos de Ss, junto con el mismo nodo k. Además, los sub-*e*CDDs de un sub-*e*CDD de k son a su vez sub-*e*CDDs de k.*

La noción intuitiva de la definición 3.1.3 es que, dado un eCDD, cualquier diagrama saliente de un nodo interno del mismo es un sub-eCDD (además del mismo eCDD).

Observación 3.1.2 *Se desprende de la definición 3.1.3 que todo* e*CDD es sub-*e*CDD de sí mismo. Por lo tanto, en lo siguiente utilizaremos los términos equivalentemente, siempre que el contexto lo permita.*

El orden definido en la definición 3.1.2, que deberá ser mantenido globalmente para todos los eCDDs pertenecientes al sistema, nos permite (junto con la definición de sub-eCDD y otros conceptos) establecer el invariante de la estructura.

Definición 3.1.4 (Invariante de un *e*CDD) *La estructura descripta anteriormente, para conformarse como* e*CDD, debe cumplir las siguientes restricciones:*

- *Si k es un nodo de la forma descripta anteriormente, tal que $diff(k)$ es $TRUE$ o $FALSE$, entonces tanto $Is(k)$ como $Ss(k)$ son listas vacías.*

- *Si k es tal que diff(k) no es TRUE ni FALSE, entonces ningún elemento de $Ss(k)$ es FALSE. Además, esta condición vale también recursivamente para cada elemento de $Ss(k)$.*
- *Dado un orden entre diferencias determinado $\{<, \leq\}$, el invariante de un e*CDD *exige que, si un sub-e*CDD *k_1 del e*CDD *k tiene a otro nodo k_2 como resultado de aplicar la función S para cierto valor i; entonces sea cierto que diff$(k_1) <$ diff(k_2), o bien que diff(k_2) sea TRUE.*
- *Debido a que las decisiones son basadas en intervalos, la existencia simultánea de diferencias $x_i - x_j$ y $x_j - x_i$ es redundante. Por lo tanto, no debe existir ningún nodo k en un e*CDD *donde la diferencia de relojes representada sea $x_j - x_i$ y, según el orden de relojes usado para establecer el orden entre diferencias, sea cierto que $x_j \geq x_i$*[4]*.*
- *Los intervalos que representan los posibles valores que puede tomar cada diferencia de relojes en cada nodo deben ser* disjuntos *y estar* ordenados *de menor a mayor. Se entenderá que un intervalo I será menor que otro J si para todo par de elementos i, j tales que $i \in I, j \in J$, vale que $i < j$.*

Estas restricciones que se imponen a la estructura tienen diversos objetivos:

- El hecho de que los nodos *TRUE* y *FALSE* no posean intervalos ni subdiagramas es al sólo fin de que sean realmente terminales.
- La restricción de que no existan subdiagramas *FALSE* conlleva un alto grado de reducción de la estructura, que deberá ser mantenido en los algoritmos. La única forma de representar un sub-eCDD vacío será por medio del mismo nodo *FALSE* (cuando se trate de la raíz), o bien omitiendo los intervalos cuyos sub-eCDDs sean vacíos.
- El orden establecido entre las diferencias de relojes establece un orden de recorrido de un eCDD. Como este orden se establece de manera global para *todas* las instancias en el sistema podremos, como veremos más adelante, asegurar ciertas propiedades acerca de los resultados de las operaciones entre ellos.
- La aparición de sólo una forma de diferencias entre dos relojes cualesquiera provee, por una parte, un factor de reducción de la estructura, y por otra, cierto nivel de "canonicidad", aunque no una canonicidad total.
- La disyunción de los intervalos provee *unicidad de opciones* (y por lo tanto determinismo) a la hora de recorrer un eCDD para verificar si una valuación lo satisface o no.

Como conclusión, llamaremos eCDDs exclusivamente a aquellos CDDs que cumplen con las condiciones estipuladas en los puntos anteriores. Estas consideraciones recién presentadas se distinguen de la definición original de CDDs presentada en [BLP+99], en el sentido de que en aquella introducción, las definiciones se focalizan tan sólo en operaciones de naturaleza recursiva. De esta forma, siendo de por sí poco eficiente, no se toman en consideración varias características de la estructura que resultan esenciales para lograr algoritmos eficientes para la verificación. En particular, distinguimos fuertemente la necesidad de asegurar orden entre los intervalos correspondientes a cada nodo, lo cual determinará un orden de recorrido único para ellos. Por otra parte, como veremos más adelante, también nos concentraremos en hallar algoritmos eficientes para todas las operaciones utilizadas durante el proceso de model checking, a diferencia del caso de la herramienta que utiliza CDDs [BLL+96] (que recurre a DBMs en varias de sus operaciones).

Por otra parte, los CRDs presentados en[Wan03] tienden a crecer mucho más hacia abajo, ya que en el caso general deben incluir, para cualquier par de relojes x_i y x_j, las cotas correspondientes a ambas diferencias $x_i - x_j$ y $x_j - x_i$. Si bien en su presentación se argumenta que este incremento "vertical" se ve contrarrestado por un menor crecimiento "horizontal" (es decir que la cantidad de cotas por diferencia es menor), esto resulta en algoritmos que deben recorrer mayor cantidad de subdiagramas de manera innecesaria, o incluso casos particulares que no nos fue posible representar mediante esta estructura.

[4] Esta restricción es arbitraria; podría exigirse en su lugar que *sólo* apareciesen differencias de relojes $x_j - x_i$ tales que $x_j \geq x_i$, pero sólo una de ellas.

3.2. Operaciones básicas

Las operaciones principales a implementar son las mismas que se implementan en el caso de las DBMs para la transformación de las zonas temporales. Es decir, necesitaremos diseñar e implementar algoritmos para la *unión* e *intersección* de regiones, *reseteo* de relojes dentro de alguna de dichas regiones, y cálculo de *sucesores temporales* de las zonas temporales. Adicionalmente, es necesaria alguna forma de discernir, dadas dos regiones (o más específicamente, una zona y una región) cualesquiera, si la primera se encuentra incluida dentro de la segunda.

Como primera aproximación a este objetivo, a continuación presentamos una descripción conceptual y recursiva de las funciones en cuestión.

3.2.1. Unión de regiones temporales

El algoritmo 3.2.1 presenta el enfoque recursivo para la unión de eCDDs. Es preciso notar que en cada nodo, de encontrarse la diferencia de relojes en cuestión sólo en uno de los dos diagramas, deben generarse los intervalos de manera de cubrir el espacio total de $\mathbb{R}$. Esto se debe a que, en caso de no existir un nodo con cierta diferencia de reloj, se entiende que dicha diferencia de reloj se encuentra irrestricta. Las diferencias de relojes de la forma $x - x_0$ sólo necesitarían abarcar tan sólo $\mathbb{R}^+$; sin embargo, por simplicidad, omitiremos esta sutileza. Por otra parte, los intervalos representados son tanto aquellos por los cuales puede avanzarse hasta llegar a las hojas que determinan que la valuación satisface las restricciones; como también los que indican que la misma no es satisfecha. De esta manera, cada intervalo correspondiente a una diferencia de relojes se verá completamente intersecado por los intervalos correspondientes a la misma diferencia en el otro diagrama.

```
1:  function UNION (eCDD_A, eCDD_B)                          ▷ Devuelve un nuevo eCDD.
2:      eCDD retValue
3:      if eCDD_A = FALSE then
4:          return eCDD_B
5:      else if eCDD_A = TRUE then
6:          return TRUE
7:      else if diff(eCDD_A) < diff(eCDD_B) then        ▷ eCDD_A ≠ TRUE ∧ eCDD_A ≠ FALSE
8:          for i = 1; i ≤ #_Ints(eCDD_A); i++ do
9:              I_i ← Ints(eCDD_A)_i
10:             AgregarSub − eCDD(retValue, I_i, UNION(S(eCDD_A)_i, eCDD_B))
11:         end for
12:     else if diff(eCDD_A) > diff(eCDD_B) then          ▷ análogo al caso anterior...
13:         ⋮
14:     else                                              ▷ diff(eCDD_A) = diff(eCDD_B)
15:         for i = 1; i ≤ #_Ints(eCDD_A); i++ do
16:             for j = 1; j ≤ #_Ints(eCDD_B); j++ do
17:                 I_i ← Ints(eCDD_A)_i
18:                 J_j ← Ints(eCDD_B)_j
19:                 if ¬Vacio(I_i ∩ J_j) then
20:                     AgregarSub − eCDD(retValue, I_i ∩ J_j, UNION(S(eCDD_A)_i, S(eCDD_B)_j))
21:                 end if
22:             end for
23:         end for
24:     end if
25:     return retValue
26: end function
```

Algoritmo 3.2.1: Algoritmo recursivo de unión de eCDDs

3.2.2. Intersección de regiones temporales

La intersección de eCDDs, presentada en el algoritmo 3.2.2, es en esencia similar a la unión. Notemos que sólo cambian los casos base de la recursión, y el hecho de que no se generan los intervalos faltantes, además del evidente cambio en las llamadas recursivas.

```
1:  function INTERSECCION (eCDD_A, eCDD_B)                    ▷ Devuelve un nuevo eCDD.
2:      eCDD retValue
3:      if eCDD_A = FALSE then
4:          return FALSE
5:      else if eCDD_A = TRUE then
6:          return eCDD_B
7:      else if diff(eCDD_A) < diff(eCDD_B) then        ▷ eCDD_A ≠ TRUE ∧ eCDD_A ≠ FALSE
8:          for i = 1; i ≤ #_Ints(eCDD_A); i++ do
9:              I_i ← Ints(eCDD_A)_i
10:             AgregarSub − eCDD(retValue, I_i, INTERSECCION(S(eCDD_A)_i, eCDD_B))
11:         end for
12:     else if diff(eCDD_A) > diff(eCDD_B) then          ▷ análogo al caso anterior...
13:         ⋮
14:     else                                              ▷ diff(eCDD_A) = diff(eCDD_B)
15:         for i = 1; i ≤ #_Ints(eCDD_A); i++ do
16:             for j = 1; j ≤ #_Ints(eCDD_B); j++ do
17:                 I_i ← Ints(eCDD_A)_i
18:                 J_j ← Ints(eCDD_B)_j
19:                 if ¬Vacio(I_i ∩ J_j) then
20:                     AgregarSub − eCDD(retValue, I_i ∩ J_j, INTERSECCION(S(eCDD_A)_i, S(eCDD_B)_j)
21:                 end if
22:             end for
23:         end for
24:     end if
25:     return retValue
26: end function
```

Algoritmo 3.2.2: Algoritmo recursivo de intersección de eCDDs

3.2.3. Chequeo de inclusión de una zona en una región

Para verificar si una zona está o no incluida dentro de una región (representada por un eCDD), debemos notar, en primera instancia, que:

- Una zona puede representarse con un eCDD que consiste de un sólo camino. En particular, este camino siempre puede ser representado de manera equivalente por una DBM en estado ajustado y completo (canónica), por lo que también existe una forma ajustada y completa de representación como eCDD. Intuitivamente, esta representación se logra simplemente traduciendo cada elemento de la DBM a un nodo del eCDD, sujeto a las restricciones enunciadas en el invariante de la estructura.
- Dicho camino podría extenderse a lo largo de varios de los caminos pertenecientes al eCDD que representa la región, producto de la subdivisión en intervalos de los nodos del eCDD.

Teniendo en cuenta las observaciones anteriores, el algoritmo 3.2.3 verifica la inclusión de una zona representada por medio de un eCDD en una región representada por la misma estructura.

3.2.4. Cálculo de sucesores temporales de una zona

Esta operación representa el paso uniforme del tiempo sobre las variables de reloj del sistema. Al igual que en el caso anterior, aplicaremos esta operación sobre zonas temporales que se encuentren ajustadas

```
1: function INCLUIDO? (eCDD_zona, eCDD_A)        ▷ En este caso sólo se devuelve un valor de verdad.
2:     if eCDD_zona = FALSE then                  ▷ FALSE ⊆ ψ para cualquier región ψ.
3:         return True
4:     else if eCDD_zona = TRUE then
5:         return eCDD_A = TRUE
6:     else if eCDD_A = FALSE then
7:         return FALSE
8:     else if eCDD_A = TRUE then
9:         return TRUE
10:    else
11:        I ← intervalo saliente de eCDD_zona                    ▷ El único posible.
12:        candidatos ← {J ∈ Ints(eCDD_A) / I ∩ J ≠ ∅}
13:        return ⋀_{J∈candidatos}(INCLUIDO?(S(eCDD_zona, I), S(eCDD_A, J)))   ▷ Debe estar incluido en
   cada posible camino.
14:    end if
15: end function
```

Algoritmo 3.2.3: Algoritmo recursivo de chequeo de inclusión de eCDDs

y acotadas, de manera similar a como se logra en DBMs[5]. Como resultado de este proceso se obtiene el algoritmo 3.2.4.

```
1: function SUCESORTEMPORALZONA (eCDD_zona)   ▷ Se devuelve un eCDD que representa la zona
   resultante.
2:     eCDD retValue
3:     if eCDD_zona = FALSE ∨ eCDD_zona = TRUE then
4:         return eCDD_zona                                   ▷ No se registran cambios.
5:     else if diff (eCDD_zona) es de la forma x − x_0 para algún reloj x then
6:         I ← intervalo saliente de eCDD_zona                ▷ El único posible.
7:         J ← ⟨I.min, +∞⟩
8:         retValue.diff ← diff(eCDD_zona)
9:         AgregarIntervalo (retValue, J)
10:        S(retValue, J) ← SUCESORTEMPORALZONA(S(eCDD_zona, I))
11:    else                          ▷ Ya no hay diferencias individuales más abajo (por orden).
12:        return eCDD_zona
13:    end if
14:    return retValue
15: end function
```

Algoritmo 3.2.4: Algoritmo recursivo para el cálculo de sucesores temporales de un eCDD

3.2.5. Reseteo de un conjunto de relojes sobre una región

Esta última operación representa el pasaje a cero de un conjunto de relojes del sistema dentro de la región sobre la que se opera. La idea de esta operación es obtener una región tal que sea satisfecha sólo por las valuaciones tales que valen cero en los relojes reseteados y que, en los demás relojes, valgan lo mismo que las valuaciones que satisfacían la región original. Se utilizará, por claridad, una función INTERVALODELRELOJ, que obtiene el intervalo en que se halla comprendido el reloj parámetro, siempre dentro de la rama que se evalúa. Por simplicidad, no especificaremos esta función. El algoritmo que presentamos en 3.2.5 representa la operación obtenida para este reseteo, con la cual completamos el conjunto de operaciones que se utilizan en el análisis *forward* presentado en el algoritmo 1.3.1.

Esta introducción a la estructura nos permite definir a continuación cómo entenderemos las restricciones temporales sobre la misma.

[5]Ver [Tri98] para detalles del método sobre DBMs.

```
1: function RESET (C, eCDD_A)                                   ▷ Devuelve un nuevo eCDD.
2:     eCDD retValue
3:     if eCDD_A = FALSE ∨ eCDD_A = TRUE then                   ▷ No hay cambios.
4:         return eCDD_A
5:     else if diff(eCDD_A) es de la forma x − x_0 y x ∈ C then  ▷ El único intervalo admitido será [0,0].
6:         CDDacum ← FALSE
7:         for i ≤ #_Ints(eCDD_A) do
8:             RESET (C, S(eCDD_A)_i)
9:             CDDacum ← UNION(CDDacum, S(eCDD_A)_i)
10:        end for
11:        diff (retValue) ← diff (eCDD_A)
12:        AgregarIntervalo (retValue, [0,0], CDDacum)
13:    else if diff(eCDD_A) es de la forma x − y y (x ∈ C ∨ y ∈ C) then  ▷ Se reemplaza por la cota del que no se resetea. Notar que si se resetean ambos, queda el intervalo [0,0].
14:        if y ∈ C then
15:            CDDacum ← FALSE
16:            for i ≤ #_Ints(eCDD_A) do
17:                RESET (C, S(eCDD_A)_i)
18:                CDDacum ← UNION(CDDacum, S(eCDD_A)_i)
19:            end for
20:            diff (retValue) ← diff (eCDD_A)
21:            AgregarIntervalo (retValue, INTERVALODELRELOJ (x), CDDAcum)
22:        else                          ▷ Se invierte la diferencia, para mantener el intervalo del reloj y
23:            CDDacum ← FALSE
24:            for i ≤ #_Ints(eCDD_A) do
25:                RESET (C, S(eCDD_A)_i)
26:                CDDacum ← UNION(CDDacum, S(eCDD_A)_i)
27:            end for
28:            diff (retValue) ← −1×diff (eCDD_A)
29:            AgregarIntervalo (retValue, INTERVALODELRELOJ (y), CDDAcum)
30:        end if
31:    end if
32:    return retValue
33: end function
```

Algoritmo 3.2.5: Algoritmo recursivo para el reseteo de relojes sobre un eCDD

3.3. Semántica de los *e*CDDs

3.3.1. Relación entre los *e*CDDs y las restricciones temporales

Siguiendo con las ideas presentadas anteriormente, los eCDDs serán la abstracción para representar simbólicamente las regiones temporales durante el análisis del sistema. Como tal, esa representación debe mantener una relación correcta respecto a las restricciones temporales que apunta a modelar. Definimos, entonces, la función de traducción de restricciones temporales a eCDDs, inductivamente, de la siguiente manera:

Definición 3.3.1 (Función de representación de restricciones temporales en *e*CDDs, $rToCDD$)

$\forall \psi_1, \psi_2 \in \Psi_X, x, y \in X, x < y, c \in \mathbb{N}, d \in \mathbb{Z}$, *se define la función rToCDD de la siguiente manera:*

$$
\begin{aligned}
rToCDD(True) &= [TRUE, \lambda, \lambda] && (3.1)\\
rToCDD(False) &= [FALSE, \lambda, \lambda] && (3.2)\\
rToCDD(c < x) &= [x - x_0, «(c, \infty)», «rToCDD(True)»] && (3.3)\\
rToCDD(c \leq x) &= [x - x_0, «[c, \infty)», «rToCDD(True)»] && (3.4)\\
rToCDD(x < c) &= [x - x_0, «[0, c)», «rToCDD(True)»] && (3.5)\\
rToCDD(x \leq c) &= [x - x_0, «[0, c]», «rToCDD(True)»] && (3.6)\\
rToCDD(x - y < c) &= [x - y, «(-\infty, c)», «rToCDD(True)»] && (3.7)\\
rToCDD(x - y \leq c) &= [x - y, «(-\infty, c]», «rToCDD(True)»] && (3.8)\\
rToCDD(\psi_1 \wedge \psi_2) &= Interseccion(rToCDD(\psi_1), rToCDD(\psi_2)) && (3.9)
\end{aligned}
$$

3.3.2. Satisfacibilidad de un *e*CDD.

La función presentada anteriormente, *rToCDD*, cumple con el objetivo de describir la relación que se establece entre una restricción temporal y un *e*CDD. Ahora, al igual que en su momento se definió la semántica de las restricciones sobre relojes, será necesario definir la semántica de los *e*CDDs; es decir, establecer formalmente qué significa que un *e*CDD sea satisfecho por una valuación temporal. Serán necesarias algunas definiciones preliminares.

Definición 3.3.2 (Camino en un sub-*e*CDD) *Se define un* camino *en un sub-*e*CDD A como una secuencia* $C[1] \dots C[n]$ *de tuplas de la forma* $\langle relojes, intervalo \rangle$*, tal que:*

- *relojes es una diferencia de relojes o alguno de los valores TRUE y FALSE.*
- *intervalo es un intervalo* $\subseteq \mathbb{R}$
- *vale alguna de las siguientes condiciones:*
  - *o bien* $diff(A) = relojes(C[1]) = TRUE$ *y* $n = 1$.
  - *o bien* $diff(A) = relojes(C[1]) = FALSE$ *y* $n = 1$.
  - *o en su defecto, valen las siguientes condiciones:*
    1. $diff(A) = relojes(C[1]) = x - y$.
    2. $intervalo(C[1])$ *es un intervalo saliente de A.*
    3. $C[2] \dots C[n]$ *es un camino en* $S(A, intervalo(C[1]))$.
  - *Finalmente,* $relojes(C[n])$ *deberá ser igual a TRUE o FALSE, efectivamente indicando el final del camino.*

Definición 3.3.3 (Satisfacibilidad de un camino por una valuación) *Dados un camino* C[6] *y una valuación* $v \in \mathcal{V}_X$*, se dice que v* satisface *C—y lo notamos* $v \models C$*—si para cada* $1 \leq i \leq longitud(C)$*,* $relojes(C[i]) = TRUE \vee (relojes(C[i]) \notin \{TRUE, FALSE\} \wedge v(relojes(C[i])) \in intervalo(C[i]))$.

Definición 3.3.4 (Satisfacibilidad de un sub-*e*CDD) *La satisfacibilidad de un sub-*e*CDD A por una valuación* $v \in \mathcal{V}_X$*, denotada como* $v \models A$*, se define inductivamente de la siguiente forma:*$\forall x, x' \in X, c \in \mathbb{N}, d \in \mathbb{Z}$:

$$
\begin{aligned}
v &\models [TRUE, \lambda, \lambda] && (3.10)\\
v &\not\models [FALSE, \lambda, \lambda] && (3.11)\\
v &\models [x - y, Is, Ss] \equiv \bigvee_{i=1}^{\#(Is)} \{ (v(x) - v(y) \in Is_i) \wedge v \models Ss_i \} && (3.12)
\end{aligned}
$$

Observación 3.3.1 *Alternativamente, puede definirse la satisfacibilidad de* $rToCDD(\psi)$ *notando que la definición 3.3.4 es equivalente a decir que un* e*CDD A es satisfacible por una valuación v si y sólo si existe un* camino *en A tal que es satisfecho por v.*

[6] Nótese que por el hecho de ser un camino cumple con las condiciones especificadas en la definición 3.3.2

Observación 3.3.2 *Vale notar también que, por la disyunción de los intervalos que asegura el invariante de la estructura, de existir ese camino, es* único.

A continuación, demostraremos además que para cualquier valuación $v \in \mathcal{V}_X$ y toda restricción temporal $\psi \in \Psi_X$, $v \models \psi \Leftrightarrow v \models rToCDD(\psi)$.

Teorema 3.3.1 (Corrección de la función *rToCDD*) *La función rToCDD es correcta respecto de la noción de satisfacibilidad, es decir que*

$$\forall \psi \in \Psi_X, v \in \mathcal{V}_X, v \models \psi \Leftrightarrow v \models rToCDD(\psi)$$

La demostración de este teorema quedará pendiente hasta tanto tengamos una demostración de la corrección de los algoritmos planteados, lo cual se encontrará más adelante. Continuaremos entonces con la descripción de los algoritmos iterativos para la estructura.

3.4. Algoritmos iterativos sobre *e*CDDs

3.4.1. Enfoque

Las operaciones presentadas en la sección 3.2 nos han servido para sentar las ideas a desarrollar en torno a nuestra estructura. Sin embargo estas operaciones, al igual que las planteadas en la presentación de los CDDs, son en extremo simplistas, de forma tal que su utilización en la realidad sería impracticable. Por lo tanto, es necesario llevar a cabo la tarea de crear versiones iterativas, logrando operaciones eficientes que puedan ser utilizadas en la práctica en un model checker.

En la versión iterativa de estas funciones mantendremos el concepto expuesto en las versiones recursivas. Por otra parte, para lograr la eficiencia deseada, las transformaciones de *e*CDDs y sub-*e*CDDs se realizarán a través de una *pila* conceptual (que mantendrá la información sobre el estado de los diagramas que se están operando), para evitar la sobrecarga que conlleva la llamada recursiva. En general, los elementos de estas pilas guardarán la siguiente información:

- Para cada uno de los diagramas que se están operando, se mantendrá cada nodo que se está operando, y los últimos intervalos "analizados". Estos intervalos pueden no ser los originales, como se verá más adelante.
- Adicionalmente, información acerca del estado actual del *e*CDD resultante, lo cual permitirá realizar recorridos hacia abajo (y arriba) en dicho diagrama en cada paso del algoritmo.

La idea general de la operación con la pila es la comparación de las raíces de los diagramas a operar, de la misma manera en que se comparan en la presentación conceptual. La diferencia se observa en el caso en que ambos poseen la misma raíz: en ese caso, los intervalos de ambos se iteran y se "funden" de acuerdo a ciertas reglas. A modo de ilustración, en la figura 3.1 puede verse como se maneja el caso en que se intersecan dos *e*CDDs que presentan una misma raíz (pero varios subdiagramas). Por otra parte, en la figura 3.2 se presenta la situación inicial de la unión de dos *e*CDDs con las características opuestas, es decir, donde las raíces de los mismos son distintas. En cada caso, la operación termina cuando la pila donde se acumulan los resultados y subdiagramas sin explorar se vacía. Debido a los invariantes de la estructura, una rama de un diagrama cualquiera no puede exceder la longitud de $((\#C)^2 + \#C)/2$, donde $\#C$ es la cantidad de relojes del sistema, por lo que la altura de la pila está *acotada*. Esta cota es siempre válida, aún en el peor caso en que todas las diferencias de relojes estén representadas. El número de intervalos también es finito por cada nodo (dado que puede tomarse la mayor constante M que aparezca en el sistema y, cada vez que un intervalo se extienda más allá de M, extenderlo hasta ∞. Algo análogo puede hacerse con $-M$ y $-\infty$. Veremos más acerca de este tema en el capítulo 4.

En general, notaremos los nodos de la pila de la siguiente manera:

$$[eCDD_1, eCDD_2, eCDD_{res}]$$

donde $eCDD_1$ y $eCDD_2$ son sub-*e*CDDs a operar dentro de la pila[7], es decir, a medida que avanza el algoritmo; y $eCDD_{res}$ es el sub-*e*CDD que mantiene el resultado de la operación hasta el momento.

[7] En algunos casos, y dependiendo de la operación, sólo se almacenará uno de ellos.

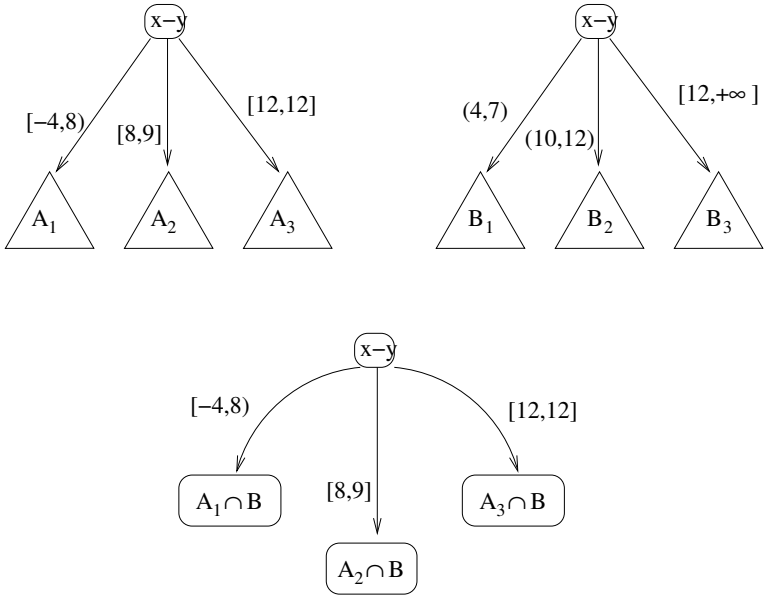


Figura 3.1: Ejemplo de intersección de dos *e*CDDs con idéntica raíz.

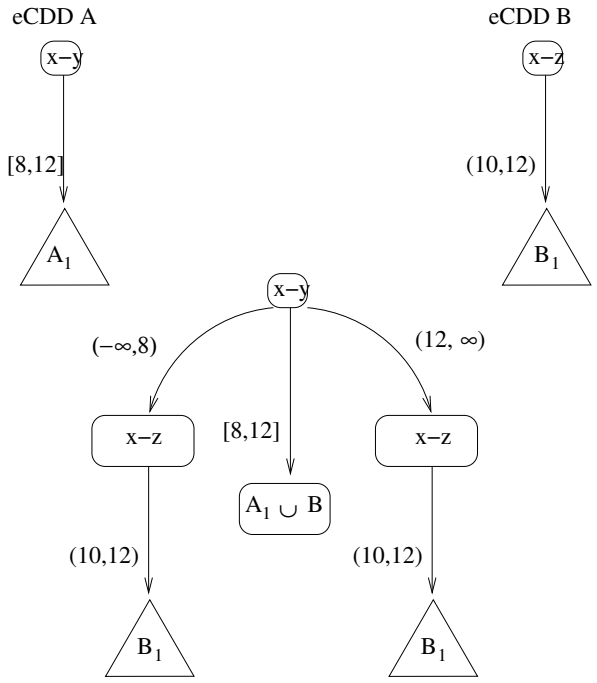


Figura 3.2: Ejemplo de unión de dos *e*CDDs con raíces distintas.

Haremos uso también de una estrategia de marcado de intervalos que, para cada sub-eCDD siendo operado, servirá para indicar cuáles de estos intervalos deben ser tenidos en cuenta en cada paso de la operación.

3.4.2. Descripción del algoritmo iterativo de intersección de *e*CDDs

A continuación se muestra el algoritmo de intersección de eCDDs, en su versión iterativa, según la técnica descripta en la sección 3.4.1.

En el caso del algoritmo de intersección, la estructura de los elementos de esta pila es como sigue:

$$[eCDD_A, eCDD_B, eCDD_{res}]$$

donde:

- $eCDD_A$ y $eCDD_B$ representan los diagramas a operar (intersecar, en este caso).
- Los intervalos correspondientes a $eCDD_A$ y $eCDD_B$ son "marcados" durante el algoritmo, para indicar si se han procesado o no.
- $eCDD_{res}$ es el eCDD resultante de la operación.

Además, en el desarrollo del algoritmo, presentado en los listados 3.4.1 y 3.4.2, se usa una función denominada PreprocesarIntervalos. Esta función, que recibe dos nodos de eCDDs, tiene como objetivo realizar una subdivisión de los intervalos originales correspondientes a cada nodo, de forma tal que,si se la invoca con dos eCDDs; $eCDD_A$ y $eCDD_B$, para cada par de intervalos $I \in eCDD_A$ y $J \in eCDD_B$ siempre valdrá que o bien $I \cap J = \emptyset$ o $I = J$. El hecho de utilizar esta función nos permitirá abstraernos de los distintos casos en los cuales dos intervalos pueden intersecarse. Esta operación de preprocesamiento de los intervalos no corrompe el invariante de la estructura; el impacto en el tamaño de la estructura, por otra parte, no es menor[8]. Esta función PreprocesarIntervalos es muy simple y, a fin de concentrarnos en los aspectos importantes de los algoritmos, no será presentada aquí. Cabe aclarar que esta función mantiene las marcas que se hayan asociado previamente a los intervalos (extendiendo dichas marcas a las potenciales particiones que realice).

Tanto en el caso de este algoritmo como en los que siguen, presentaremos una noción informal de sus complejidades temporales; sus complejidades espaciales estarán limitadas al tamaño inicial de los operandos, más el tamaño del resultado donde corresponda. En todos estos casos será simple presentar la complejidad para las ocasiones en que los operandos posean una única rama, ya que en las mismas se comportarán de manera similar a los algoritmos clásicos sobre RDBMs. En los casos en los que los operandos presenten ramificaciones, sin embargo, dicho análisis no será tan simple. Presentaremos entonces una noción aproximada, fundamentándonos en la dependencia de las ramificaciones particulares de cada caso: en general, cuanto más ramificada se halle una estructura, más compleja será la operación.

Observación 3.4.1 *Si llamamos n a la cantidad de relojes distintos con los que se define la zona, el algoritmo es $O(n^2)$ en el caso de hallarse una sola rama. Si las estructuras son tales que poseen ramificaciones, entonces si k_1 y k_2 son las cantidades de nodos presentes en cada uno de los operandos respectivamente, la complejidad estará* acotada *por $k_1 + k_2$. Esta cota es, sin embargo, extremadamente conservadora: en el caso general la complejidad será mucho menor ya que, en primer lugar, la estructura se encontrará debidamente reducida (sin representar las n^2 diferencias de relojes. Además, no todas las intersecciones resultarán en subdiagramas en común. Vale aclarar también que tanto k_1 como k_2 son $O(n^2)$, multiplicados por un factor directamente ligado a cuán ramificadas se encuentren las estructuras respectivas.*

Teorema 3.4.1 (Corrección del algoritmo de intersección) *El algoritmo de intersección de regiones representadas por medio de* e*CDDs presentado es correcto.*

A continuación ofrecemos una demostración de lo aseverado en el teorema 3.4.1.

Queremos ver que el algoritmo de intersección de eCDDs computa, como se espera, la conjunción de las dos restricciones temporales introducidas por cada uno de los eCDDs originales. Para ello, debemos verificar que:

[8] En el capítulo 4 volveremos sobre este tema en particular.

```
1: function INTERSECCION (eCDD_A, eCDD_B)                    ▷ Devuelve un nuevo eCDD.
2:     Pila(sub − eCDD, sub − eCDD, sub − eCDD) pila
3:     eCDD retValue
4:     Apilar (pila, [eCDD_A, eCDD_B, retValue])
5:     while ¬ Vacia(pila) do
6:         tope ← Desapilar (pila)
7:         if diff(tope.eCDD_A) = FALSE then                 ▷ análogo para tope.eCDD_B.
8:             tope.eCDD_res ← FALSE
9:         else if diff(tope.eCDD_A) = TRUE then             ▷ análogo para tope.eCDD_B.
10:            tope.eCDD_res ← copia(tope.eCDD_B)

    ▷ Para el resto, diff(eCDD_A) ≠ TRUE ∧ diff(eCDD_A) ≠ FALSE.

11:        else if diff(tope.eCDD_A) < diff(tope.eCDD_B) then
12:            tope.eCDD_res.diff ← diff(tope.eCDD_A)
13:            if no hay intervalos en tope.eCDD_A sin marcar then
    ▷ Eliminación de FALSE del resultado.

14:                for cada intervalo I saliente de tope.eCDD_res do
15:                    if S(tope.eCDD_res, I) = FALSE then
16:                        EliminarIntervalo(tope.eCDD_res, I)
17:                    end if
18:                end for
19:                if ∀ sub-eCDD de tope.eCDD_res S, , S = FALSE then
20:                    tope.eCDD_res ← FALSE
21:                    continue
22:                end if
23:                continue
24:            end if
25:            I ← primer intervalo de tope.eCDD_A sin marcar
26:            Marcar (tope.eCDD_A, I)
27:            AgregarIntervalo (tope.eCDD_res, I)

    ▷ Apilar, para más tarde considerar los intervalos restantes.

28:            Apilar (pila, [tope.eCDD_A, tope.eCDD_B, tope.eCDD_res])
29:            nuevo_eCDD_B ← DesmarcarIntervalos(tope.eCDD_B)

    ▷ Apilar el siguiente nivel, hasta llegar a las hojas.

30:            Apilar (pila, [S(tope.eCDD_A, I), nuevo_eCDD_B, S(tope.eCDD_res, I)])
```

Algoritmo 3.4.1: Algoritmo iterativo de intersección de eCDDs—Parte 1.

```
31:         else if diff(tope.eCDD_A) > diff(tope.eCDD_B) then

    ▷ Análogo al caso anterior...
                ⋮
32:
33:         else                                          ▷ diff(tope.eCDD_A) = diff(tope.eCDD_B)
34:             tope.eCDD_res.diff ← diff(tope.eCDD_A)
35:             PreprocesarIntervalos (tope.eCDD_A, tope.eCDD_B)
36:             for cada intervalo I en tope.eCDD_A do

    ▷ Se ignoran los intervalos que no intersecan.

37:                 if ∀ intervalo J en tope.eCDD_B, I ≠ J then
38:                     Marcar (tope.eCDD_A, I)
39:                 end if
40:             end for
41:             for cada intervalo J en tope.eCDD_B do
42:                 if ∀ intervalo I en tope.eCDD_A, J ≠ I then
43:                     Marcar (tope.eCDD_B, J)
44:                 end if
45:             end for
46:             if no hay intervalos sin marcar en tope.eCDD_A then
47:                 if ∀ sub-eCDD de tope.eCDD_resS, , S = FALSE then

    ▷ Eliminación de FALSE del resultado.

48:                     tope.eCDD_res.diff ← FALSE
49:                     continue
50:                 else
51:                     continue
52:                 end if
53:             end if
54:             I ← primer intervalo de tope.eCDD_A sin marcar
55:             Marcar (tope.eCDD_A, I)
56:             Marcar (tope.eCDD_B, I)
57:             AgregarIntervalo (tope.eCDD_res, I)

    ▷ Procesar siguiente nivel y apilar para seguir procesando el actual.

58:             Apilar (pila, [tope.eCDD_A, tope.eCDD_B, tope.eCDD_res])
59:             Apilar (pila, [S(tope.eCDD_A, I), S(tope.eCDD_B, I), S(tope.eCDD_res, I)])
60:         end if
61:     end while
62:     return retValue
63: end function
```

Algoritmo 3.4.2: Algoritmo iterativo de intersección de *e*CDDs—Parte 2.

1. el algoritmo termina;
2. el invariante de la estructura se verifica en el diagrama resultante; y
3. para cualquier par de restricciones $\psi_1, \psi_2 \in \Psi_X$, y $\forall v \in \mathcal{V}_X$:

$$v \models rToCDD(\psi_1) \wedge v \models rToCDD(\psi_2) \Leftrightarrow v \models \textsc{Interseccion}(rToCDD(\psi_1), rToCDD(\psi_2))$$

Como comprobación del primer punto, es suficiente notar que ambos parámetros son finitos, que en cada inserción de un estado dentro de la pila se introduce al menos un diagrama más pequeño (o con menos intervalos a verificar), y que durante la ejecución del algoritmo cada camino se explora a lo sumo una vez (esto sucede porque al revisitar estados de la pila, no se analiza más de una vez ningún intervalo). Por otra parte, dada la estructura de árbol de los *e*CDDs y sub-*e*CDDs, y que el recorrido es siempre hacia las hojas, no es posible que se presenten ciclos en un recorrido. Como conclusión de esto, se desprende que eventualmente el algoritmo termina; y que la estructura resultante es también finita.

Respecto del segundo punto, debemos verificar que, en la estructura resultante, se verifiquen las siguientes propiedades:

1. Cuando existan nodos *TRUE* o *FALSE*, que entonces no tengan intervalos salientes ni sub-*e*CDDs.
2. Efectivamente se eliminen los caminos que resulten en la existencia de nodos *FALSE* dentro del diagrama.
3. En todo camino perteneciente al *e*CDD resultante, los nodos se encuentren ordenados según la relación ya presentada.
4. Los intervalos salientes de cada nodo sean disjuntos.

En realidad, los puntos más interesantes son los últimos tres: el primer punto es trivial, ya que es simple verificar que en cada momento en que se crea un nodo *TRUE* o *FALSE* se lo hace sin intervalos ni sub-*e*CDDs; el segundo es un poco menos evidente, sin embargo puede verse que si bien al computar la intersección existe la posibilidad de que se generen caminos insatisfacibles, éstos se eliminan a medida que aparecen, por lo que no existen nodos *FALSE* dentro del diagrama (el código presentado incluye comentarios en los puntos exactos donde se eliminan los nodos *FALSE*). Obviamente en caso de que todos los caminos resulten falsos, el diagrama que se devolverá constará de un único nodo *FALSE*.

Nos concentraremos entonces en los dos puntos restantes. De éstos, el primero es fácil de verificar ya que, para cada estado que se encuentra en la pila, siempre se utiliza como nueva raíz a la menor de las diferencias de relojes encontradas. Como, a su vez, siempre se insertan en la pila subdiagramas que tienen sus nodos ordenados, este orden se mantiene en la estructura resultante. Otro enfoque, tal vez más intuitivo, es entender a esta operación como una especie de *merge* de los intervalos ya ordenados.

Por otra parte, la disyunción de los intervalos ingresados también está asegurada debido a que, o bien se ingresan intervalos correspondientes a uno sólo de los diagramas parámetro (que ya eran disjuntos), o bien se ingresan aquellos resultantes de las operaciones PreprocesarIntervalos, que también genera intervalos disjuntos entre sí. Esto asegura el cumplimiento del segundo punto requerido.

Finalmente, a continuación presentamos una demostración para el último punto a comprobar, es decir, la corrección de la operación de intersección respecto de la semántica de las restricciones:
$\forall \psi_1, \psi_2 \in \Psi_X, \forall v \in \mathcal{V}_X$,

$$v \models rToCDD(\psi_1) \wedge v \models rToCDD(\psi_2) \Leftrightarrow v \models \textsc{Interseccion}(rToCDD(\psi_1), rToCDD(\psi_2))$$

Demostración

Demostraremos cada implicación por separado.
$\Longrightarrow$ *) Sea* $v \in \mathcal{V}_X$ *tal que* $v \models rToCDD(\psi_1)$ *y* $v \models rToCDD(\psi_2)$*. Esto es equivalente a decir (por definición de satisfacibilidad de los* e*CDDs) que existe un camino* C_1 *en* $rToCDD(\psi_1)$ *y otro,* C_2*, en* $rToCDD(\psi_2)$ *tal que* $v \models C_1$ *y* $v \models C_2$*. Queremos ver que existe un camino* $C_{1\cap 2}$ *en el resultado ofrecido por* Interseccion$(rToCDD(\psi_1), rToCDD(\psi_2))$ *tal que* $v \models C_{1\cap 2}$ *(es decir, queremos probar que* v *satisface también al diagrama resultante de la intersección). Denotemos, sin pérdida de generalidad,*

$$\begin{aligned} C_1 &= \langle x_1 - y_1, I_1\rangle, \langle x_2 - y_2, I_2\rangle, \ldots, \langle x_n - y_n, I_n\rangle \\ C_2 &= \langle z_1 - w_1, J_1\rangle, \langle z_2 - w_2, J_2\rangle, \ldots, \langle z_m - w_m, J_m\rangle \end{aligned}$$

donde $I_1, \ldots, I_n, J_1, \ldots, J_m$ *son intervalos* $\subseteq \mathbb{R}$ *tales que,* $x_i, y_i, z_j, w_j \in X \cup \{x_0\}$ *para* i *y* j *tales que* $1 < i < n$ *y* $1 < j < m$*, y de tal forma que se mantiene el orden en las diferencias de relojes exigido por el invariante (es decir que para cada* i *tal que* $1 < i < n-1, (x_i - y_i) \prec (x_{i+1} - y_{i+1})$*; y para cada* j *tal que* $1 < j < m-1$*, vale análogamente que* $(z_j - w_j) \prec (z_{j+1} - w_{j+1})$*). Mostraremos entonces que el camino* $C_{1\cap 2}$ *existe, a partir de la forma en que el algoritmo se ejecuta y construye este camino. La idea general es mostrar que, para cada elemento que encontremos dentro de la pila y cumpla la hipótesis enunciada a continuación, el algoritmo siempre continúa creando un camino satisfacible. La hipótesis en cuestión, que debe cumplir un elemento de la pila para ser considerado dentro de la construcción, es que los dos sub-*e*CDDs a operar mantengan la propiedad de ser satisfacibles por* v*, es decir, que contengan un camino que* v *satisface. Una vez hallado el caso base (la hoja del diagrama), el* e*CDD resultante ya contendrá un camino satisfacible por* v*.*

*Del análisis de los elementos de la pila, se desprende que los sub-*e*CDDs (denotados tope.*eCDD_A *y tope.*eCDD_B*) pueden tomar alguna de las siguientes formas:*

- *Si alguno de ellos es FALSE (supongamos, sin pérdida de generalidad, que se trata de tope.*eCDD_A*), debe ser que rToCDD* (ψ_1) *= FALSE; es decir que* $\nexists v \in \mathcal{V}_X$ *tal que* $v \models rToCDD(\psi_1)$*. Dado que suponíamos lo contrario, no tiene sentido continuar con este caso.*

- *Si, en cambio, uno de ellos es TRUE (nuevamente, sin pérdida de generalidad, podemos suponer que es tope.*eCDD_A*), cualquier* $v \in \mathcal{V}_X$ *lo satisface. Nos concentramos entonces en aquellas* $v \in \mathcal{V}_X$ *que satisfacen tope.*eCDD_B*. Sin embargo, por hipótesis, sabemos que* $v \models$ *tope.*eCDD_B*. Dado que es el mismo tope.*eCDD_B *el que se apila en el resultado (ver líneas (9) y subsiguientes), el caso queda demostrado.*

- *Puede darse el caso de que ninguno de ellos sea TRUE ni FALSE. En ese caso, los nodos superiores de ambos sub-*e*CDDs representan diferencias de relojes. Supongamos inicialmente que la situación es tal que una de ellas sea menor a la otra, según el orden definido anteriormente; sin pérdida de generalidad, que diff (tope.*eCDD_A*) < diff (tope.*eCDD_B*). Dado que ambos sub-*e*CDDs eran satisfechos por* v*, mediante caminos* C_1 *y* C_2 *respectivamente, podemos afirmar, equivalentemente, que relojes*$(C_1[1]) <$ *relojes*$(C_2[1])$*. Dado este caso, el algoritmo coloca a diff (tope.*eCDD_A*) en diff (tope.*eCDD_{res}*). Además, agrega todos los intervalos salientes de tope.*eCDD_A *en tope.*eCDD_{res}*, entre ellos, intervalo*$(C_1[1])$ *(ver líneas (25) a (30)). Comenzamos entonces a construir el camino de forma tal que* $C_{1\cap 2}[1] = \langle$*relojes*$(C_1[1])$*, intervalo*$(C_1[1])\rangle$*. Como* $v \models C_1$*, debe ser también que* $v \models C_{1\cap 2}[1]$*. Notemos que, para el caso en que se agrega este intervalo al diagrama resultante, el siguiente estado de la pila que se agrega viene dado por (ver línea (30)):*

$$[S(tope.\text{e}CDD_A, intervalo(C_1[1])), tope.\text{e}CDD_B, S(tope.\text{e}CDD_{res}, intervalo(C_1[1]))]$$

Como el camino C_1 *que era satisfecho por* v *en* eCDD_A *tiene su continuación en* S*(tope.*eCDD_A*, intervalo*$(C_1[1])$*), sabemos también que* $v \models S$*(tope.*eCDD_A*, intervalo* $(C_1[1])$*). Sabiendo esto y recurriendo a la hipótesis, resulta que:*

$$\begin{aligned} v &\models tope.\text{e}CDD_B \\ v &\models S(tope.\text{e}CDD_A, intervalo(C_1[1])) \end{aligned}$$

Con esto estamos en condiciones de asegurar que el estado ingresado en la pila cumple las hipótesis necesarias para continuar construyendo un camino satisfacible por v*. Dado que eventualmente se considera este elemento de la pila, la continuidad de la construcción de* $C_{1\cap 2}$ *está asegurada, con lo que este caso queda demostrado. Por otra parte, el resto del sub-*e*CDD sigue siendo considerado ya que se apila el elemento actual, pero con los intervalos analizados ya marcados (ver línea (28)).*

- *De darse el caso de que relojes*$(C_1[1]) =$ *relojes*$(C_2[1])$*, se construye el resultado estableciendo como primer nodo uno cuya diferencia de relojes sea ésta misma (ver línea (34)). De esta manera, el algoritmo asigna a diff*(*tope.*eCDD_{res}) *la diferencia representada en relojes*$(C_1[1])$*. Es preciso notar que luego de esta asignación vale que*

$$diff(tope.\text{e}CDD_{res}) = relojes(C_1[1]) = diff(tope.\text{e}CDD_A) = diff(tope.eCDD_B) = relojes(C_2[1])$$

Sabemos además, por hipótesis, que $v \models C_1$ *y* $v \models C_2$*. Por la noción de satisfacibilidad de los caminos,* $v($*relojes*$(C_1[1]) \in$ *intervalo*$(C_1[1]))$ *y* $v($*relojes*$(C_2[1])) \in$ *intervalo*$(C_2[1])$*. Por lo tanto,*

$intervalo(C_1[1]) \cap intervalo(C_2[1]) \neq \emptyset$: al menos tendrán un valor en común, aquel que la valuación v asigna a la diferencia. Por ello, el algoritmo incluirá en la pila, en este paso, el siguiente nodo (ver líneas (54) a (59)):

$$\begin{bmatrix} S(tope.\text{e}CDD_A, intervalo(C_1[1])), \\ S(tope.\text{e}CDD_B, intervalo(C_2[1])), \\ S(tope.\text{e}CDD_{res}, intervalo(C_1[1]) \cap intervalo(C_2[1])) \end{bmatrix}$$

Es preciso notar que los intervalos que se agregan en el algoritmo representan la intersección *de los intervalos que originalmente pertenecían a cada sub-*e*CDD. Estos nuevos intervalos se obtienen a partir de la función* PreprocesarIntervalos *(ver línea (35)). Volviendo a la demostración, es necesario remarcar también que:*

$$\begin{aligned} v &\models S(tope.\text{e}CDD_A, intervalo(C_1[1])) \\ v &\models S(tope.\text{e}CDD_B, intervalo(C_2[1])) \end{aligned}$$

por las mismas razones expuestas en el caso anterior. Por lo tanto, no sólo $v \models C_{1\cap 2}[1]$ sino que además este nuevo estado de la pila cumple las hipótesis requeridas para la demostración. Esto permite asegurar que de aquí en más se logrará también una intersección satisfacible por v.

Habiendo considerado todos los casos, cerramos esta parte de la demostración.

$\Longleftarrow$) Demostraremos esta segunda implicación valiéndonos de la contrarrecíproca. Es decir, para demostrar que:
$\forall v \in \mathcal{V}_X, \forall \psi_1, \psi_2 \in \Psi_X$,

$$v \models \text{Interseccion}(rToCDD(\psi_1), rToCDD(\psi_2)) \implies v \models rToCDD(\psi_1) \wedge v \models rToCDD(\psi_2)$$

en realidad demostraremos, de manera equivalente, que:
$\forall v \in \mathcal{V}_X, \forall \psi_1, \psi_2 \in \Psi_X$,

$$\neg(v \models rToCDD(\psi_1) \wedge v \models rToCDD(\psi_2)) \implies v \not\models \text{Interseccion}(rToCDD(\psi_1), rToCDD(\psi_2))$$

Sea entonces $v \in \mathcal{V}_X$ tal que $\neg(v \models rToCDD(\psi_1) \wedge v \models rToCDD(\psi_2))$. Sin pérdida de generalidad, supongamos que $v \not\models rToCDD(\psi_1)$. Esto significa que no existe ningún camino en $rToCDD(\psi_1)$ que sea satisfecho por v. La demostración que presentaremos consiste en mostrar que, partiendo de dicha hipótesis, no existirán tampoco en el diagrama resultante de Interseccion$(rToCDD(\psi_1), rToCDD(\psi_2))$ *caminos satisfacibles por v.*

Notemos $rToCDD(\psi_1) =$ eCDD_A y $rToCDD(\psi_2) =$ eCDD_B, por comodidad. La idea, al igual que en la demostración anterior, será demostrar que, para cada estado que se encuentra en la pila al avanzar el algoritmo, las intersecciones que se generan a partir de ellos cumplen con la tesis de que v no los satisface. Para lograr ello, plantearemos la hipótesis de que todos *los elementos encontrados en la pila serán tales que v no satisfacerá a alguno de los sub-*e*CDDs que se están operando. Partiendo de esta idea, sea $[tope.\text{e}CDD_A, tope.\text{e}CDD_B, tope.\text{e}CDD_{res}]$ el estado de la pila encontrado.*

- *Si alguno de ellos es FALSE, entonces el algoritmo apila un diagrama FALSE en el resultado, que ninguna valuación $v \in \mathcal{V}_X$ satisface, por lo que es correcto (ver línea (7)).*
- *Si alguno de ellos, por el contrario, es TRUE, entonces toda valuación lo satisface. Como sabemos, por hipótesis, que la valuación v no satisface* alguno *de los dos sub-*e*CDDs, entonces claramente debe ser que v no satisface al otro sub-*e*CDD siendo operado. Como el algoritmo de intersección apila este último como resultado (ver línea (9)), v no satisface esta intersección resultante, lo que deriva en la corrección del algoritmo en este caso.*
- *Si ninguno de ellos es TRUE ni FALSE, y además $diff(tope.\text{e}CDD_A) \neq diff(tope.\text{e}CDD_B)$, debe ser que una de las diferencias es menor que la otra. Analizando cada caso se desprende que:*

  1. *Si $diff(tope.\text{e}CDD_A) < diff(tope.\text{e}CDD_B)$:*
     recordemos también que habíamos supuesto que $v \not\models tope.\text{e}CDD_A$. Bajo esta hipótesis, puede suceder que:

a) *no exista un intervalo saliente de tope.eCDD_A tal que $v($diff$($tope.e$CDD_A))$ pertenezca a dicho intervalo (ver líneas (13) a (24)). En ese caso, habremos encontrado el motivo por el cual* eCDD_A *no puede ser satisfecho por* v*. Además, como en este paso del algoritmo se genera un diagrama con raíz* diff$($tope.e$CDD_A)$ *y cuyos intervalos salientes son* todos *y* sólo *los salientes del mismo, es imposible que el resultado sea satisfecho por* v*. Esto se traduce en que todos los sub-*eCDD*s que salen del resultado terminan siendo FALSE, lo cual redunda en que el resultado mismo es FALSE.*

b) *exista un intervalo* I *saliente de tope.eCDD_A tal que $v($diff$($tope.e$CDD_A))) \in I$. En ese caso, de haber un camino en la intersección tal que sea satisfecho por* v*, el mismo debe pasar por este intervalo* I*. En este paso del algoritmo se introduce a la pila el siguiente nodo (ver línea (30)):*

$$[S(\text{tope.e}CDD_A, I), \text{tope.e}CDD_B, S(\text{tope.e}CDD_{res}, I)]$$

Este nodo cumple con la hipótesis de que $v \not\models S($tope.e$CDD_A, I)$, por lo que podemos argumentar también que eventualmente se alcanzará uno de los estados descriptos en esta demostración.

2. *Si se diera el caso en que* diff$($tope.e$CDD_A) >$ diff$($tope.e$CDD_B)$*, todos los estados que se agregan a la pila son de la forma:*

$$[\text{tope.e}CDD_A, S(\text{tope.e}CDD_B, J), S(\text{tope.e}CDD_{res}, J)]$$

para cada intervalo J *que sale de tope.eCDD_B (ver líneas (25) a (30)). Sin embargo, como $v \not\models$ tope.eCDD_A, y sin importar cómo se comporte respecto de tope.eCDD_B, cada uno de estos estados cumple con la hipótesis de la demostración. Esto último nos permite argumentar el eventual hallazgo del caso conflictivo.*

- *Finalmente, si* diff$($tope.e$CDD_A) =$ diff$($tope.e$CDD_B)$*, la raíz del diagrama resultante vendrá dada por* diff$($tope.e$CDD_A)$ *(línea (34)), de forma que los estados que se ingresen en la pila en este paso del algoritmo serán de la forma*

$$[S(\text{tope.e}CDD_A, K), S(\text{tope.e}CDD_B, K), S(\text{tope.e}CDD_{res}, K)]$$

donde los intervalos K *surgen del preprocesamiento, y son tales que* K *es intervalo saliente tanto de tope.eCDD_A como de tope.eCDD_B. Entonces,*

1. *si* no *existe un* K *de la forma mencionada anteriormente tal que $v($diff$($tope.e$CDD_A))) \in K$, se habrá encontrado el caso conflictivo, ya que todas las intersecciones generadas a partir de ese estado no serán satisfacibles por* v *(este caso se encuentra desarrollado en las líneas (46) a (53)).*
2. *Si, en cambio, dicho* K sí *existiera, sabremos sin embargo que $v \not\models S($tope.e$CDD_A, K)$, por lo que el estado satisface la hipótesis de la demostración y nos permite asegurar que eventualmente se encontrará el estado conflictivo.*

De esta forma, analizando cada caso por separado, demostramos que en ninguna corrida del algoritmo bajo las hipótesis planteadas se puede construir un resultado satisfacible por v*.*

□

Además de haber probado la corrección del algoritmo INTERSECCION, demostraremos también el siguiente resultado (su utilidad se verá más adelante).

Lema 3.4.1 (Intersección de convexos) *Dadas ψ_1 y ψ_2 dos zonas temporales, y por ende convexas, con* eCDD_1 *y* eCDD_2 *sus representaciones tales que poseen un único camino cada una, entonces la aplicación de* INTERSECCION(eCDD_1, eCDD_2) *devuelve también la representación de una zona de forma tal que posee un sólo camino.*

Demostración

En primer lugar, dada la corrección de INTERSECCION*, es seguro que el algoritmo computará correctamente la intersección de las zonas. Vale aclarar que, también, la intersección de dos zonas es una zona,*

por lo que una representación que posea un sólo camino existe. *Sólo falta ver que* INTERSECCION *genera, en este caso, una representación tal.*

*La demostración en cuestión es simple: es fácil ver que, cualquiera sea el caso que se encuentre al analizar las raíces de cada sub-*e*CDD encontrado en la pila, cada intervalo o se agrega, o se ignora, o bien se agrega la intersección de ambos, que sólo puede ser un intervalo. Por otra parte, cada uno de los* e*CDDs parámetro, al poseer un sólo camino, posee un único intervalo por cada nodo. De esta manera, no hay forma de que el resultado posea más de un intervalo por nodo, y por ende poseerá solamente un camino.*

□

Finalmente, agregaremos una observación más al resultado del algoritmo.

Observación 3.4.2 *En los casos en que la intersección de dos zonas o regiones se detecta como FALSE en el transcurso del algoritmo (ver líneas (19) y (47)), el* e*CDD resultante es siempre el nodo FALSE.*

Es necesario notar dos cosas de la observación 3.4.2:

I En primer lugar, el hecho de que esto suceda indica que efectivamente la zona resultado de la intersección está efectivamente en su forma canónica, simplificando todas las operaciones posteriores que la involucren.

II Sin embargo, *no* es cierto que siempre que la intersección de dos zonas o regiones sea vacía, el algoritmo lo detecte mientras se ejecuta. En algunos casos será necesario canonizar la estructura de todas maneras.

Este comportamiento se deriva exclusivamente del hecho de que los *e*CDDs mantienen, para cada diferencia de relojes, ambas cotas (la superior y la inferior) simultáneamente. Esta característica les permite detectar la ausencia de intersección al mismo momento de computarla. Como veremos en capítulos posteriores, esta reducción de canonizaciones resulta muy satisfactoria a la hora de reducir los tiempos de ejecución de la verificación, con mínimo impacto en el uso de memoria.

En este momento, nos encontramos en condiciones de presentar la demostración del teorema 3.3.1.

Demostración

Plantearemos la demostración probando, para cada parte de la definición de rToCDD, la corrección de dicha definición. Demostraremos cada uno de los casos por separado.

- *Las proposiciones 3.1 a 3.8 se satisfacen por la definición de satisfacibilidad de un* e*CDD.*
- *La proposición 3.9 es consecuencia directa del teorema 3.4.1, que acabamos de demostrar.*

□

A continuación, seguimos con la descripción de los algoritmos restantes.

3.4.3. Descripción del algoritmo de unión de *e*CDDs

El algoritmo de unión de *e*CDDs presenta varias similitudes respecto del de intersección. En particular, nos interesa destacar que se comparte la noción de preprocesamiento de intervalos; y que también se hace uso de una pila de estados. Además, utilizaremos una función COMPLETARINTERVALOS que, dados los intervalos I_i salientes de un nodo de un *e*CDD, devuelve los mismos intervalos más los existentes en $\mathbb{R} - \bigcup_i I_i$. Es decir, la función COMPLETARINTERVALOS no hace más que devolver también los intervalos intermedios que no pertenezcan al conjunto de intervalos pasado como parámetro. Más allá de estas cuestiones, las similitudes y diferencias guardan una fuerte analogía con las encontradas en la versión recursiva ya presentada. El algoritmo es presentado en los listados 3.4.3 y 3.4.4.

```
 1: function UNION (eCDD_A, eCDD_B)                          ▷ Devuelve un nuevo eCDD.
 2:     Pila(sub − eCDD, sub − eCDD, sub − eCDD) pila
 3:     eCDD retValue
 4:     Apilar (pila, [eCDD_A, eCDD_B, retValue])
 5:     while ¬ Vacia(pila) do
 6:         tope = Desapilar (pila)
 7:         if tope.eCDD_A = FALSE then                      ▷ Análogo para tope.eCDD_B.
 8:             tope.eCDD_res ← copia(tope.eCDD_B)
 9:         else if tope.eCDD_A = TRUE then                  ▷ Análogo para tope.eCDD_B.
10:             tope.eCDD_res ← TRUE
11:         else if diff(tope.eCDD_A) < diff(tope.eCDD_B) then

    ▷ Además, eCDD_A ≠ TRUE ∧ eCDD_A ≠ FALSE.

12:             tope.eCDD_res.diff ← diff(tope.eCDD_A)
13:             PREPROCESARINTERVALOS (tope.eCDD_A, tope.eCDD_B)
14:             intervalos ← COMPLETARINTERVALOS (tope.eCDD_A)
15:             if no hay intervalos en intervalos sin marcar then
16:                 continue
17:             end if
18:             I ← primer intervalo de intervalos sin marcar
19:             Marcar (intervalos, I)
20:             AgregarIntervalo (tope.eCDD_res, I)

    ▷ Notar que la reducción del caso para la recursión viene dada por el marcado de intervalos.

21:             Apilar (pila, [tope.eCDD_A, tope.eCDD_B, tope.eCDD_res])
22:             nuevo_eCDD_B ← DesmarcarIntervalos(tope.eCDD_B)

    ▷ Analizar qué sub-eCDDs se ingresan a la pila.

23:             if I ∈ intervalos salientes de tope.eCDD_A then
24:                 Apilar (pila, [S(tope.eCDD_A, I), nuevo_eCDD_B, S(tope.eCDD_res, I)])
25:             else
26:                 S(tope.eCDD_res, I) ← copia(nuevo_eCDD_B)
27:             end if
28:         else if diff(tope.eCDD_A) > diff(tope.eCDD_B) then      ▷ Análogo al caso anterior...
29:             ⋮
30:         else                                  ▷ diff(tope.eCDD_A) = diff(tope.eCDD_B)
31:             tope.eCDD_res.diff ← diff(tope.eCDD_A)
32:             PREPROCESARINTERVALOS (tope.eCDD_A, tope.eCDD_B)
33:             recorriendo ← 'A'
```

Algoritmo 3.4.3: Algoritmo iterativo de unión de eCDDs—Parte 1.

```
34:             if no hay intervalos sin marcar en tope.eCDD_A then
35:                 recorriendo ← 'B'
36:             end if
37:             if recorriendo = 'B' ∧ no hay intervalos sin marcar en tope.eCDD_B then
38:                 continue
39:             end if

    ▷ Con lo anterior, se decide el orden de recorrido de los intervalos.
    ▷ En ambos casos se realiza un merge según los intervalos

40:             if recorriendo = 'A' then
41:                 I ← primer intervalo de tope.eCDD_A sin marcar
42:                 Marcar (tope.eCDD_A, I)
43:             else
44:                 I ← primer intervalo de tope.eCDD_B sin marcar
45:                 Marcar (tope.eCDD_B, I)
46:             end if
47:             AgregarIntervalo (tope.eCDD_res, I)
48:             if recorriendo = 'A' then
49:                 nuevo_eCDD_A ← S(tope.eCDD_A, I)
50:                 if I ∈ tope.eCDD_B then
51:                     Marcar (tope.eCDD_B, I)
52:                     nuevo_eCDD_B ← S(tope.eCDD_B, I)
53:                 else
54:                     nuevo_eCDD_B ← FALSE
55:                 end if
56:             else                    ▷ recorriendo = 'B' implica que no hay más intervalos en eCDD_A.
57:                 nuevo_eCDD_B ← S(tope.eCDD_B, I)
58:                 nuevo_eCDD_A ← FALSE
59:             end if
60:             Apilar (pila, [tope.eCDD_A, tope.eCDD_B, tope.eCDD_res])
61:             Apilar (pila, [nuevo_eCDD_A, nuevo_eCDD_B, S(tope.eCDD_res, I)])
62:         end if
63:     end while
64:     return retValue
65: end function
```

Algoritmo 3.4.4: Algoritmo iterativo de unión de *e*CDDs—Parte 2.

Observación 3.4.3 *A diferencia del caso de la intersección, en general la unión resultará en estructuras ramificadas. Esto resulta en que, si los operandos están comprendidos por un sólo camino, la complejidad sea un poco superior, en particular de* $r \times O(n^2)$*. El factor* r *mencionado es derivado de cuán disjuntas sean ambas estructuras: su valor oscilará entre* 1*, si no hay tal disyunción, y* 3*, en el caso de que además de existir disyunción existan partes en común. En el caso de que las estructuras a ser operadas se encuentren ya ramificadas, el cálculo se extiende sobre* k_1 *y* k_2 *de manera similar a lo argumentado en la observación 3.4.1.*

Teorema 3.4.2 (Corrección del algoritmo de unión) *El algoritmo presentado para la unión de regiones representadas por* e*CDDs es correcto.*

Demostración

Demostraremos, al igual que en el caso de la intersección, ambas implicaciones por separado. De la misma manera en que se ha demostrado la corrección del algoritmo de intersección, demostraremos que el algoritmo de Union *también lo es. Nuevamente, tendremos que probar primero la terminación del mismo, la conservación del invariante, y luego que*
$\forall \psi_1, \psi_2 \in \Psi_X; \forall v \in \mathcal{V}_X$,

$$v \models rToCDD(\psi_1) \vee v \models rToCDD(\psi_2) \Leftrightarrow v \models \textsc{Union}(rToCDD(\psi_1), rToCDD(\psi_2))$$

Por otra parte, las garantías de terminación del algoritmo y de conservación de invariantes vienen dadas por las mismas razones que se exhibieron en la demostración de la intersección. Es también preciso notar que, a diferencia del algoritmo de intersección, no es posible generar caminos insatisfacibles (recordemos que en un e*CDD no existen caminos insatisfacibles, ya que los nodos FALSE no se explicitan). Esto es consecuencia de que durante el algoritmo no se reducen ni eliminan intervalos; obviamente la excepción es el caso de que se estén uniendo dos diagramas FALSE. A continuación presentamos la demostración de la corrección del resultado del algoritmo desde el punto de vista semántico.*

$\Longrightarrow$*) Queremos ver que, para todo par de restricciones* $\psi_1, \psi_2 \in \Psi_X$*, y para toda valuación* $v \in \mathcal{V}_X$*,*

$$v \models rToCDD(\psi_1) \vee v \models rToCDD(\psi_2) \implies v \models \textsc{Union}(rToCDD(\psi_1), rToCDD(\psi_2))$$

Sea entonces v *una valuación en* $\mathcal{V}_X$*. De la misma manera en que demostramos esta implicación para la intersección, usaremos fuertemente la hipótesis de que* $v \models rToCDD(\psi_1) \vee v \models rToCDD(\psi_2)$*; es decir que o bien existe un camino* C *en* $rToCDD(\psi_1)$ *tal que* $v \models C$*, o bien ese camino* C *existe en* $rToCDD(\psi_2)$*. Sin pérdida de generalidad, supondremos que ese camino* C *existe en* $rToCDD(\psi_1)$ *al cual, por comodidad, notaremos* eCDD_A*, notando también* eCDD_B *a* $rToCDD(\psi_2)$*.*

Queremos ver entonces que el algoritmo construye un camino $C_{1\cup 2}$ *en la unión tal que* v *lo satisface. La demostración será, en espíritu, análoga a la demostración de la intersección.*

Partiremos entonces del hecho de que existe un camino $C \in$ eCDD_A *satisfecho por* v*. Notamos*

$$C = \langle x_1 - y_1, I_1\rangle, \langle x_2 - y_2, I_2\rangle, \ldots, \langle x_n - y_n, I_n\rangle$$

y veremos entonces que, para cada estado que se puede encontrar en la pila, es posible seguir construyendo un camino en la unión que sea satisfacible por v*, siempre y cuando ese estado tope sea tal que cumpla con la hipótesis de que existe algún camino satisfacible en tope.*eCDD_A*. De esta manera, para cada estado de la pila*

$$[tope.\text{e}CDD_A, tope.\text{e}CDD_B, tope.\text{e}CDD_{res}]$$

tal que $v \models tope.\text{e}CDD_A$ *se suscita alguno de los siguientes casos:*

- *Si alguno de los sub-*e*CDDs encontrados es TRUE (sea tope.*eCDD_A *o tope.*eCDD_B*), entonces la unión de ambos vendrá dada por otro diagrama TRUE a apilar (ver línea (9)). Dado que cualquier valuación satisface un* e*CDD TRUE,* v *es sólo un caso particular.*
- *Por el contrario, puede darse el caso de que alguno de ellos sea FALSE. De ser éste el caso, el sub-*e*CDD FALSE* debe *ser tope.*eCDD_B*, ya que por hipótesis tope.*eCDD_A *era satisfecho por* v *(y ninguna valuación satisface FALSE). En ese caso, la unión resultante es el mismo diagrama tope.*eCDD_A *(ver línea (7)). Como sabemos, por hipótesis, que* $v \models tope.\text{e}CDD_A$*, la unión es también satisfacible por* v*.*

- *Si ninguno de ellos es TRUE ni FALSE, cabe esperar dos posibilidades:*

  1. *Si diff(tope.eCDD_A) ≠ diff(tope.eCDD_B), puede darse que*

     a) *diff(tope.eCDD_A) < diff(tope.eCDD_B): en este caso, la raíz del diagrama resultado se compondrá de la misma raíz que tope.eCDD_A (ver línea (12)). Además, como se agregan todos los intervalos salientes de tope.eCDD_A (aunque potencialmente con distintos subdiagramas), dentro de la pila se ingresará:*

     $$[S(tope.\text{e}CDD_A, intervalo(C[1])), tope.\text{e}CDD_B, S(tope.\text{e}CDD_{res}, intervalo(C[1]))]$$

     Este proceso se realiza en las líneas (21) y (23). Dado que esto es equivalente a "recorrer" el primer paso dentro del camino C que v satisface, podemos asegurar que la situación es tal que $v \models S(tope.\text{e}CDD_A, intervalo(C[1]))$. Como se satisface el primer paso del camino, y lo que se ingresa en la pila cumple con la hipótesis de la demostración, este caso queda demostrado.

     b) *diff(tope.eCDD_A) > diff(tope.eCDD_B): en este caso, el algoritmo coloca como raíz del diagrama resultado el valor contenido en diff(tope.eCDD_B) (ver línea (12)). Además, como los sub-eCDDs están ordenados (y se analizan también en orden), podemos afirmar que el valor diff(tope.eCDD_B)* no *es tenido en cuenta en tope.eCDD_A, es decir que no se impone ninguna restricción sobre él. Sin embargo, es fácil ver que en este paso del algoritmo se introducen estados en la pila que abarcan* todo *el espectro de* $\mathbb{R}$ *(ver líneas (14) a (20)). Por lo tanto, existe un intervalo I ingresado en alguno de los estados de la pila tal que $v(diff(tope.\text{e}CDD_B)) \in I$, lo cual permite obtener el primer paso del camino $C_{1\cup 2}$. Dependiendo de si v satisface o no diff(tope.eCDD_B), se darán los siguientes casos respecto del intervalo I:*

        1) *Si $I \in$ intervalos salientes de tope.eCDD_B, se ingresa en la pila el estado*

        $$[tope.\text{e}CDD_A, S(tope.\text{e}CDD_B, I), S(tope.\text{e}CDD_{res}, I)]$$

        2) *Si no, al resultado se copia tope.eCDD_A.*

        En ambos casos, o bien se introduce dentro de la pila a tope.eCDD_A para su consideración, o bien directamente se lo constituye como resultado. En el caso de que se lo inserte como estado de la pila, y sabiendo que $v \models tope.\text{e}CDD_A$, dicho estado cumple la hipótesis necesaria para continuar con la demostración. Si se lo devuelve como resultado, el resultado es satisfecho por v por lo mencionado anteriormente.

  2. *Finalmente, de no darse ninguno de los casos expuestos anteriormente, debe ser cierto que diff(tope.eCDD_A) = diff(tope.eCDD_B). En ese caso, sólo se tienen en cuenta los intervalos que salen o bien de tope.eCDD_A o de tope.eCDD_B. Además, se tiene que:*

     a) *para los intervalos que* sólo *salen de tope.eCDD_A, se copia al mismo como resultado.*

     b) *para los que* sólo *salen de tope.eCDD_B, se copia al resultado, análogamente al caso anterior, dicho subdiagrama.*

     c) *para los que salen de* ambos, *se ingresa a la pila el estado* $[S(tope.\text{e}CDD_A, I), S(tope.\text{e}CDD_B, I), S(tope.\text{e}CDD_{res}, I)]$

     Todas estas situaciones son analizadas en las líneas (48) a (59). Como v satisface al nodo contenido en tope.eCDD_A, necesariamente debe existir un intervalo saliente J de tope.eCDD_A, tal que se verifique que $v(diff(tope.\text{e}CDD_A)) \in J$, tal que, además, $v \models S(tope.\text{e}CDD_A, J)$. Claramente, entonces, el segundo caso de los presentados anteriormente es irrelevante, ya que el I planteado no *pertenece a los intervalos salientes de tope.eCDD_A. Por otra parte es fácil verificar que, sea el primero o el tercero el caso relevante según el valor de $v(diff(tope.\text{e}CDD_A))$, en ambos casos $S(tope.\text{e}CDD_A, I)$ se ingresa al estado, de forma que las hipótesis necesarias se verifican.*

$\Longleftarrow$) *Como se había anticipado antes, esta parte de la demostración se llevará a cabo mediante su recíproca. Es decir, en vez de demostrar de manera directa que*
$\forall \psi_1, \psi_2 \in \Psi_X; \forall v \in \mathcal{V}_X$,

$$v \models \textsc{Union}(rToCDD(\psi_1), rToCDD(\psi_2)) \implies v \models rToCDD(\psi_1) \vee v \models rToCDD(\psi_2)$$

demostraremos que
$\forall \psi_1, \psi_2 \in \Psi_X; \forall v \in \mathcal{V}_X$,

$$v \not\models rToCDD(\psi_1) \wedge v \not\models rToCDD(\psi_2) \implies v \not\models \textsc{Union}(rToCDD(\psi_1), rToCDD(\psi_2))$$

En esta parte de la demostración también notaremos eCDD_A *y* eCDD_B*, en vez de rToCDD* (ψ_1) *y rToCDD* (ψ_2) *respectivamente. Sea entonces una valuación v tal que no satisface a* eCDD_A *ni a* eCDD_B*, lo cual es equivalente a la hipótesis planteada. Sabiendo entonces que no hay caminos satisfacibles por v en* eCDD_A *ni en* eCDD_B*, verificaremos que tampoco los habrá en* Union (eCDD_A, eCDD_B)*. Siguiendo un esquema similar a la demostración anterior, mostraremos que para cada estado ingresado en la pila,* ninguno *de los que se agregan en ese paso del algoritmo podrán ser satisfacibles. Esto se reducirá a mostrar, para cada estado, que si v no satisface a ninguno de los dos subdiagramas ingresados, entonces también será imposible que satisfaga el resultado obtenido por la unión de ellos.*

Sea entonces

$$[tope.\text{e}CDD_A, tope.\text{e}CDD_B, tope.\text{e}CDD_{res}]$$

el estado que se encuentra en el tope de la pila, y que cumple la hipótesis antedicha. Así, nos encontramos en alguna de las siguientes situaciones:

- *Si alguno de ellos es FALSE (supongamos, sin pérdida de generalidad, que se trata de tope.*eCDD_A*), el nuevo diagrama que se devuelve es el* otro*, tope.*eCDD_B*. Dado que v no satisfacía a ninguno de los dos sub-*e*CDDs, claramente no satisface a éste que devolvemos (ver línea (7)).*
- *Si tope.*eCDD_A *o tope.*eCDD_B *son TRUE, se niega la hipótesis, ya que partíamos de suponer que v no satisfacía a ninguno de los dos. Por lo tanto, este caso se verifica inmediatamente al no cumplir la hipótesis.*
- *Si, además de ser ambos distintos de TRUE y FALSE, son también distintos entre sí, sucederá que o bien diff* $(tope.\text{e}CDD_A) < diff\,(tope.\text{e}CDD_B)$ *o diff* $(tope.\text{e}CDD_A) > diff\,(tope.\text{e}CDD_B)$*. Ambos casos son análogos, por lo que analizaremos el caso en que diff* $(tope.\text{e}CDD_A) < diff\,(tope.\text{e}CDD_B)$ *sin que esto resulte en perjuicio para la generalidad de la demostración. Siendo este el caso, el algoritmo coloca a diff* $(tope.\text{e}CDD_A)$ *como raíz, y genera intervalos que cubren completamente el espectro de* $\mathbb{R}$ *(ver línea (13)). Sin embargo, como sabemos que* $v \not\models tope.\text{e}CDD_A$*, debe ser que* $v(diff\,(tope.\text{e}CDD_A))$ *pertenezca a un intervalo de los que originalmente* no *estaban presentes en tope.*eCDD_A*. Por lo tanto, para que se dé el caso de que v satisfaga la unión, debe pasar por uno de estos nuevos intervalos. Sin embargo, resulta que para todos estos intervalos I se introducen estados de la forma:*

  $$[FALSE, tope.\text{e}CDD_B, S(tope.\text{e}CDD_{res}, I)]$$

  Esto se ve claramente en las líneas (23) y subsiguientes. Dado que, por definición, $v \not\models FALSE$ *y además, por la hipótesis, sabíamos que* $v \not\models tope.\text{e}CDD_B$*, todos estos estados cumplen la hipótesis que nos permite asegurar que ninguno de los caminos subsiguientes serán satisfacibles por v.*
- *Finalmente, si diff* $(tope.\text{e}CDD_A) = diff\,(tope.\text{e}CDD_B)$*, la raíz del diagrama resultante vendrá dada por esa misma diferencia de relojes. Sin embargo, los intervalos que se agregan son sólo aquellos que o bien pertenecen a tope.*eCDD_A *o a tope.*eCDD_B*, o a ambos (ver líneas (48) a (59)). Sabemos que v no satisface a ninguno de los dos sub-*e*CDDs; esto puede ser consecuencia de varias cosas:*

  1. *que el valor de* $v(diff\,(tope.\text{e}CDD_A))$ *(que es el mismo que* $v(diff\,(tope.\text{e}CDD_B))$*), no pertenece a ninguno de los intervalos salientes de tope.*eCDD_A *ni tope.*eCDD_B*. En ese caso, sin importar qué estados se ingresen a la pila, v ya no puede satisfacer la unión.*
  2. *que v satisfaga a alguno de los dos topes (sin pérdida de generalidad, supongamos que se trata de tope.*eCDD_A*), o a ambos. En ese caso, indefectiblemente tendrá que encontrarse más adelante una situación conflictiva, ya que sabemos por hipótesis que v no satisface a ninguno de los dos, y no se agregan intervalos que no existieran previamente.*

Con estas consideraciones, queda culminada la segunda parte de la demostración.

□

3.4.4. Descripción del algoritmo de determinación de inclusión

Una de las operaciones que cumplen un rol fundamental dentro del algoritmo de alcanzabilidad para la verificación formal de requerimientos temporales, como anticipamos anteriormente, es la de verificar la *inclusión* de una zona dentro de una región. Si bien las regiones temporales son representadas por medio de *e*CDDs, que potencialmente poseen varios caminos, las zonas cumplen con la particularidad de ser conjuntos *convexos*. Esto hace que sea posible representarlas por medio de un *e*CDD conteniendo un sólo camino. Por lo tanto, nuestro algoritmo de chequeo de inclusiones sólo se ocupará del caso en que se desee verificar si un *e*CDD que representa una *zona* se encuentra incluido dentro de otro, que representa una *región*.

Sin embargo, dado que no proveemos una *forma canónica* para las representaciones de las regiones utilizando *e*CDDs, este algoritmo podría devolver resultados erróneos si no se cumplen ciertas condiciones sobre los parámetros de entrada. En particular, como única condición para asegurar el resultado correcto, es preciso que el *e*CDD que representa la zona a chequear se encuentre *ajustado* (en el sentido de que sus cotas deben ser lo más estrictas posibles); y *completo* (queriendo decir que todas las diferencias de relojes posibles deben estar representadas, restringidas a lo explicitado por el invariante). Esto sí es posible de asegurar en un *e*CDD que represente sólo una zona, de la siguiente forma:

1. construyendo una matriz equivalente al *e*CDD que representa la zona[9];
2. encontrando los caminos mínimos sobre la matriz recién construida mediante, por ejemplo, el algoritmo de Floyd-Warshall. Esto efectivamente resulta en la expresión canónica de la DBM; y luego
3. reconstruyendo un *e*CDD equivalente al original, pero a partir de la matriz obtenida por el paso anterior (lo cual resulta en una representación canónica de la misma pero en la forma de un *e*CDD).

De no cumplirse esta condición, el algoritmo podría devolver *falsos negativos*, forzando a la revisión de zonas ya visitadas. Esto podría derivar en que la verificación no terminase nunca, por no encontrar el punto fijo.

Este algoritmo, a diferencia de los ya presentados, no genera un nuevo *e*CDD, sino que sólo realiza un chequeo entre ellos. De todas maneras, también hace uso de una pila, ya que utiliza la técnica de *backtracking* y, en los casos negativos, realiza cortes prematuros de ser posible. Por otra parte, al no ser un *e*CDD el resultado de la función, los elementos de la pila no son de la misma forma que en los algoritmos de Union e Interseccion. En cambio, contienen la siguiente información:

$$[eCDD_{zona}, eCDD_A]$$

donde querremos determinar si $eCDD_{zona} \subseteq eCDD_A$. También usaremos la técnica de marcado de intervalos para no repetir búsquedas ya realizadas. Además, se hará uso de una función CubreIntervalos? que, dados un intervalo y un conjunto de intervalos, devuelve si la "diferencia" entre ambos es no vacía; es decir, si existen (sub)intervalos comprendidos en el intervalo pasado como segundo parámetro que *no* se encuentran abarcados en ninguno de los intervalos del conjunto que se ha pasado como segundo parámetro, o que no se encuentran abarcados siquiera por la unión de dichos intervalos.

Observación 3.4.4 *En el caso del algoritmo de determinación de inclusión, presentado en el listado 3.4.5, el tamaño de la estructura que representa la región no es tan determinante para calcular su complejidad. Esto se debe a que todas las decisiones efectuadas en el mismo son en realidad determinadas por la estructura de la zona (que posee un único camino) y no por la de la región. Sin embargo, es cierto que cuanto más ramificada se encuentre la región, mayor será la dificultad en encontrar el subdiagrama correspondiente. En general, podemos dar una cota para la operación asegurando que su complejidad es* $O(n^2 \times Mr)$*, donde* Mr *representa el máximo factor de ramificación en la región. Nuevamente, esta cota es en extremo conservadora, ya que para su cálculo se asume el peor escenario (zonas y regiones poco reducidas, fragmentación extrema de la región).*

Teorema 3.4.3 (Corrección del algoritmo de determinación de inclusión) *El algoritmo presentado para la determinación de inclusión de una zona en una región representadas sobre* e*CDDs es correcto.*

[9] Esta matriz es reminiscente de una DBM en su concepto.

```
1: function INCLUIDO? (eCDD_zona, eCDD_A)
2:     Boolean retValue ← True
3:     Pila(sub-eCDD, sub-eCDD) pila
4:     Apilar (pila, [eCDD_zona, eCDD_A])
5:     while ¬ Vacía(pila) ∧ retValue do
6:         tope ← Desapilar (pila)
7:         if tope.eCDD_A = FALSE then
8:             retValue ← (tope.eCDD_zona = FALSE)
9:         else if tope.eCDD_A = TRUE then
10:            retValue ← True
11:        else
12:            if diff(tope.eCDD_A) < diff(tope.eCDD_zona) then      ▷ tope.eCDD_A es más restrictiva.
13:                retValue ← False
14:            else if diff(tope.eCDD_zona) < diff(tope.eCDD_A) then
    ▷ tope.eCDD_zona restringe más, al menos explícitamente.
    ▷ Notar que tope.eCDD_zona consta de un único camino.

15:                I ← intervalo saliente de tope.eCDD_zona                ▷ El único posible.
16:                Apilar (pila, [S(tope.eCDD_zona, I), tope.eCDD_A])
17:            else                                  ▷ diff(tope.eCDD_A) = diff(tope.eCDD_zona)
18:                I ← intervalo saliente de tope.eCDD_zona
19:                if ¬ CUBREINTERVALOS? (I, tope.eCDD_A) then
20:                    retValue ← False    ▷ Porque hay (sub)intervalos de tope.eCDD_zona no satisfechos
21:                    continue
22:                end if
23:                J ← primer intervalo sin marcar en tope.eCDD_A
24:                Marcar (tope.eCDD_A, J)
25:                if no existe tal J ∨ I ∩ J = ∅ then                    ▷ Ya fue recorrido.
26:                    continue
27:                else if I ∩ J ≠ ∅ then          ▷ Continuamos descendiendo por el camino.
28:                    Apilar (pila, [tope.eCDD_zona, tope.eCDD_A])
29:                    Apilar (pila, [S(tope.eCDD_zona, I), S(tope.eCDD_A, J)])
30:                else                                        ▷ I ∩ J = ∅, no nos sirve.
    ▷ Lo que se apila está reducido debido al marcado de intervalos.
31:                    Apilar (pila, [tope.eCDD_zona, tope.eCDD_A])
32:                end if
33:            end if
34:        end if
35:    end while
36:    Vaciar(pila)                                  ▷ Puede existir corte prematuro.
37:    return retValue
38: end function
```

Algoritmo 3.4.5: Algoritmo iterativo de chequeo de inclusión de *e*CDDs.

Demostración

En este caso, sólo nos ocuparemos de demostrar la terminación del algoritmo y la corrección de su resultado. No hay necesidad de demostrar tal cosa como la conservación del invariante, ya que las estructuras tan sólo se examinan y no se generan nuevas como resultado de la operación. El argumento que asegura la terminación del algoritmo es el mismo que en casos anteriores, por lo que no ahondaremos en el mismo. Queda por demostrar entonces que:

$$\forall \psi_1, \psi_2 \in \Psi_X Incluido?(Canonizada(rToCDD(\psi_1)), rToCDD(\psi_2)) \Leftrightarrow \psi_1 \subseteq \psi_2$$

donde Canonizada *es una función que, dada una zona representada por un* e*CDD de un único camino, computa su equivalente* completo *y con las* cotas ajustadas*. La definición de* completitud *de un* e*CDD de un único camino es análoga a la de aquella para DBMs.*

Como ya es usual, demostraremos ambas implicaciones por separado.
$\Longrightarrow$ *) En primer lugar, demostraremos que si el algoritmo devuelve True, entonces efectivamente la zona pasada como parámetro en el* e*CDD que posee una única rama está incluida en la región representada por el otro diagrama. Para lograr este objetivo, tendremos en cuenta el resultado correspondiente a los chequeos de inclusión utilizando RDBMs ya planteado en la observación 2.1.2. Es fácil ver que el procedimiento realiza exactamente la tarea descripta en dicha observación. La única salvedad es que al no tener una única cota por diferencia de reloj, sino dos (el intervalo completo abarcado por las posibles valuaciones de la diferencia de relojes), en cada paso existe la posibilidad de tener que* dividir *la zona en* subzonas*; de esta forma se hace necesaria la verificación de que las* subzonas *estén incluidas dentro de las* subregiones *resultantes.*

Supongamos entonces que el algoritmo de determinación de inclusiones devuelve True al ser ejecutado con los parámetros eCDD_A *y* eCDD_{zona}*. Esto resulta en la inclusión, en primer lugar, del elemento*

$$[\text{e}CDD_{zona}, \text{e}CDD_A]$$

dentro de la pila. Mostramos a continuación, en detalle, un análisis de qué sucede cada vez que se encuentra un estado de la pila, cuando el algoritmo devuelve True.

- *si* eCDD_A *es FALSE, equivalente a tener relojes con circuitos negativos en una DBM clásica, la única posibilidad de que* eCDD_{zona} *esté incluido es que el mismo sea también FALSE (ver línea (7)). En ese caso el valor de retorno será True, en caso contrario, False.*

- *si* eCDD_A *es TRUE, entonces toda región (y por ende toda zona) estará incluida dentro de la región que representa. Por ello el algoritmo devuelve True, sin importar de qué zona se trate (ver línea (9)).*

- *si diff* (eCDD_A) < *diff* (eCDD_{zona})*, lo que sucede es que existe una diferencia de relojes (la representada en diff* (*tope*.eCDD_A)*) que está restringida dentro de* eCDD_A *(porque nunca se incluyen las diferencias que sólo tienen como intervalo permitido a* $\langle -\infty, +\infty \rangle$*); pero que no está restringida dentro de* eCDD_{zona}*. Esta situación es análoga a que existan* $1 \leq i, j \leq \#(C)$ *en un par de DBMs* D, D' *tales que* $D_{ij} > D'_{ij}$*. Esto indica que la zona* no *está incluida dentro de la región, por lo que se devolvería False. Sin embargo, estamos contemplando los casos en que el algoritmo devuelve True, por lo que ignoraremos esta posibilidad hasta la segunda sección de la demostración.*

- *si, en cambio, diff* (eCDD_A) > *diff* (eCDD_{zona})*, lo que sucede es que existe una diferencia de relojes que,* a priori*, no está restringida dentro de* eCDD_A*, pero sí en* eCDD_{zona}*. Decimos* a priori *porque esta diferencia podría estar* implícita *por las otras diferencias de* eCDD_A*. Sin embargo, gracias a la observación 2.1.2, podemos "ignorar" por ahora esta diferencia ya que, si la zona realmente no está incluida dentro de la región, entonces existirá más adelante una diferencia de relojes* explícita *que lo demostrará. Esto se realiza en las líneas (12) y subsiguientes.*

- *finalmente, si diff* (eCDD_A) = *diff* (eCDD_{zona})*, sólo resta verificar que todos los intervalos salientes de diff* (eCDD_{zona}) *(en realidad uno sólo) sean abarcados por los de diff* (eCDD_A)*. Si hay porciones que no están abarcadas, quiere decir que existen valuaciones que satisfacen* eCDD_{zona} *pero no hacen tal cosa con* eCDD_A*, de lo cual se deduce que la zona* no *está incluida en la región (ver línea (18) a (22) para el análisis detallado). Si todos los intervalos están abarcados, debemos continuar la verificación para poder responder con seguridad la pregunta de si la zona está o no*

incluida en la región, recorriendo el resto del sub-eCDD que representa la zona, y contrastándolo contra los sub-eCDDs indicados por cada uno de los intervalos en tope.eCDD_A que abarque el intervalo de tope.eCDD_{zona}. Esto se lleva a cabo en las líneas (25) a (33).

Estas consideraciones cierran esta primera parte de la demostración.

$\Longleftarrow$) *Como en los casos anteriores, demostraremos esta implicación utilizando su recíproca. Es decir, demostraremos que* $\forall \psi_1, \psi_2 \in \Psi_X$

$$\neg Incluido?(Completo(rToCDD(\psi_1)), rToCDD(\psi_2)) \implies \psi_1 \nsubseteq \psi_2$$

es decir que $\exists v \in \mathcal{V}_X, v \models \psi_1 \wedge v \not\models \psi_2$.

Se desprende inmediatamente del algoritmo que, para que el resultado devuelto sea False debe pasarse, en algún momento, el valor de retValue a False (ya que el mismo comienza siendo True). Evaluemos entonces las situaciones en que este cambio de valuación acontece.

- *El primer caso en que el valor de retValue puede ser cambiado a FALSE es que suceda que tope.eCDD_A = FALSE y tope.e$CDD_{zona} \neq$ FALSE. Si esto ocurre, entonces toda valuación que satisfaga al nodo tope.eCDD_{zona} es candidata a ser dicha valuación v, ya que no hay valuación que satisfaga a FALSE (ver línea (7)).*
- *Si se da el caso en que se cumple que diff(tope.eCDD_A) < diff(tope.eCDD_{zona}), entonces el nodo diff(tope.eCDD_A)* no *puede encontrarse dentro de los nodos de tope.eCDD_{zona}, ya que el mismo era, por hipótesis, completo y ordenado. Esto quiere decir que diff(tope.eCDD_A) no está restringido dentro de diff(tope.eCDD_{zona}). Por lo tanto, puede hallarse una valuación v, tal que $v \models$ tope.eCDD_{zona} y v(diff(tope.eCDD_A)) no pertenezca a ninguno de los intervalos salientes de tope.eCDD_A. Como consecuencia, $v \not\models$ tope.eCDD_A.*
- *Finalmente, la última posibilidad es que diff(tope.eCDD_A) = diff(tope.eCDD_{zona}) y exista un intervalo I saliente de tope.eCDD_{zona}*[10] *tal que alguno de sus subintervalos* no *esté comprendido dentro de los intervalos salientes de tope.eCDD_A. Dado que tope.eCDD_{zona} estaba completo y, además, ajustado, existe una valuación v tal que $v \models$ tope.eCDD_{zona} y que además toma, en diff(tope.eCDD_{zona}) algún valor comprendido en dicho subintervalo. Se desprende inmediatamente que $v \not\models$ tope.eCDD_A.*

Esta última argumentación concluye la demostración de la corrección de este algoritmo.

□

3.4.5. Descripción del algoritmo de reseteo de relojes

Sumándose a las demás operaciones ya presentadas, es necesario también implementar una operación que abstraiga el *reseteo de relojes* de regiones temporales; es decir, una función que dada una región y un conjunto de relojes, obtenga la región resultante de modificar la original pasando los valores de los relojes del conjunto a 0.

Es necesario notar que el reseteo de un reloj x cualquiera conlleva las siguientes consecuencias directas:

- el único valor posible para ese reloj es ahora 0. Por lo tanto, en una estructura que represente esta región, cada vez que aparezca el nodo $x - x_0$, deberá existir uno y sólo un intervalo saliente, *cerrado*, y con tanto su mínimo como su máximo iguales a 0.
- las relaciones que estaban establecidas con anterioridad entre x y algún otro reloj del sistema (excepto el ficticio), dejan de verificarse y cambian radicalmente; sólo existirán relaciones entre ellos si éstas vienen implícitas por otras relaciones ya contempladas en el sistema. En particular, las restricciones de la forma $y - x$, donde x es el reloj a resetear, tomarán la misma forma que las restricciones sobre y (ya que $y - x$ pasa a ser $y - 0 = y$).
- todos los demás (y las relaciones entre ellos) no sufren variación alguna respecto de sus posibles valores.

[10] Nuevamente, es válido recordar que, como *tope.eCDD_{zona}* tiene un sólo camino, I es único.

Presentaremos entonces un algoritmo que, dada una región representada por un eCDD y un conjunto de relojes, obtiene la región resultante de resetear dichos relojes en la región representada por el eCDD pasado como parámetro. Es decir, dados la región ψ_1, un conjunto de relojes C y una valuación v tal que $v \models rToCDD(\psi_1)$, se obtendrá por medio del algoritmo un eCDD, $eCDD_A$, tal que $v[z := 0, \forall z \in C] \models eCDD_A$. Este algoritmo se encuentra en los listados 3.4.6 y 3.4.7.

Se hace uso, además, de una función llamada IntervaloDelReloj. Esta función devuelve, para el reloj pasado como parámetro, el intervalo en el cual se halla comprendido. Esta información se sintetiza de cada rama del eCDD, que representa una zona. La aplicación de esta función, y la inserción de las diferencias de relojes que se realiza con dicho intervalo, no son de importancia para este algoritmo en particular (es fácil ver que dicha diferencia de relojes no agrega información nueva, ya que se obtiene de las diferencias ya representadas). Sin embargo, será de extrema utilidad para el correcto desempeño de la función SucesorTemporalZona, como se verá en la sección 3.4.6.

Como ya es de esperarse, también en este caso haremos uso de una pila para recorrer la estructura y crear el nuevo eCDD que representará el resultado. En este caso, al ser un sólo eCDD el que se operará, y al ser un nuevo eCDD el resultado, cada uno de los elementos de la pila tendrá la siguiente estructura:

$$[eCDD_A, eCDD_{res}]$$

donde tanto $eCDD_A$ como $eCDD_{res}$ son en realidad sub-eCDDs. Sin embargo, los notamos con esos nombres por comodidad.

Observación 3.4.5 *En el caso de la operación de reseteo de relojes sobre una región, la complejidad dependerá exactamente del tamaño de esta estructura. Si k es el número de nodos dentro de esta estructura, entonces la misma tendrá una complejidad $O(k)$. Este k se hallará determinado, como en los casos anteriores, por la cantidad de ramificaciones presentes en la estructura. Sin embargo, es válido notar que en el transcurso de nuestro algoritmo de alcanzabilidad sólo se realizarán operaciones de reseteo sobre* zonas*. En esos casos, dicho k estará acotado por n^2, siendo en general menor debido a la reducción de la estructura. Por otra parte, será necesario un incremento de $O(n)$ en el espacio utilizado para la ejecución, debido al guardado de cotas individuales; el no hacerlo implicaría un aumento del tiempo de ejecución debido a la necesidad de retrotraerse para determinarlas.*

Teorema 3.4.4 (Corrección del algoritmo de reseteo de relojes) *El algoritmo presentado para el cómputo de la región correspondiente al reseteo de relojes sobre otra región, ambas representadas mediante* e*CDDs, es correcto.*

Demostración

Como en los casos anteriores, nos interesará demostrar que el algoritmo presentado en los listados 3.4.6 y 3.4.7 computa de manera correcta la región correspondiente al reseteo de un conjunto de relojes de una región pasada como parámetro. Tendremos que mostrar no sólo que las valuaciones mantienen la relación esperada, sino que además el algoritmo termina en todos los casos (en un tiempo finito) y que la estructura generada cumple con los invariantes planteados acerca de ella.

La finalización es demostrada de la misma manera que en los algoritmos anteriores, por lo que no profundizaremos en sus argumentos. Al igual que en el caso del algoritmo de obtención de sucesores temporales, también debemos referirnos a la corrección del algoritmo de unión de regiones. Respecto de la conservación del invariante, al igual que en los demás casos, podemos ver claramente que el orden de las diferencias de relojes se mantiene (por recorrerlo y generarlo en el mismo orden). La disyunción de los intervalos, como sucedía también al obtener los sucesores temporales, es destruida durante la operación; de todas maneras, es recuperada mediante la operación de unión. Por otra parte, no se generan nodos FALSE, por lo que tampoco son de preocupar los aspectos del invariante que lidiaban con ese tipo de nodos terminales.

Por último, resta demostrar que la región obtenida es efectivamente aquella que se obtiene por medio del reseteo, es decir que para toda región ψ y cualquier valuación $v \in \mathcal{V}_X$:

$$v \models \psi \implies v[z := 0, \forall z \in C] \models rToCDD(reset(C, \psi)) \Leftrightarrow v[z := 0, \forall z \in C] \models \textsc{Reset}(C, rToCDD(\psi))$$

Al igual que en el caso de la operación de cálculo de sucesores temporales, podemos ver que lo enunciado anteriormente es equivalente a las siguientes dos implicaciones:

```
1: function RESET (C, eCDD_A)                                   ▷ Devuelve un eCDD nuevo
2:     Pila(sub-eCDD, sub-eCDD) pila
3:     eCDD retValue
4:     Apilar (pila, [eCDD_A, retValue])
5:     while ¬ Vacía?(pila) do
6:         tope ← Desapilar (pila)
7:         if diff(tope.eCDD_A) = y − z, con y ∉ C; z ∉ C then
   ▷ Esto incluye al caso (diff(tope.eCDD_A) = y − x_0, con y ∉ C), ya que x_0 nunca se resetea.
   ▷ En este caso, los sub-eCDDs se mantienen iguales.
8:             diff(tope.eCDD_res) ← diff(tope.eCDD_A)
9:             I ← primer intervalo sin marcar en tope.eCDD_A
10:            if existe tal I then
11:                Marcar (tope.eCDD_A, I)
12:                AgregarIntervalo (tope.eCDD_res, I)
13:                Apilar (pila, [tope.eCDD_A, tope.eCDD_res])
14:                Apilar (pila, [S(tope.eCDD_A, I), S(tope.eCDD_res, I)])
15:            end if
16:        else if diff(tope.eCDD_A) = x − x_0, con x ∈ C then
   ▷ "Aplanar" en el intervalo [0, 0]
17:            I ← primer intervalo sin marcar en tope.eCDD_A
18:            if existe talI then
19:                Marcar (tope.eCDD_A, I)
20:                AgregarIntervalo (tope.eCDD_res, I)                ▷ Se eliminará luego.
21:                Apilar (pila, [tope.eCDD_A, tope.eCDD_res])
22:                Apilar (pila, [S(tope.eCDD_A, I), S(tope.eCDD_res, I) ▷ Reseteo de los subdiagramas.
23:            else
24:                diff(tope.eCDD_res) ← diff(tope.eCDD_A)
25:                AgregarIntervalo (tope.eCDD_res, [0, 0])
26:                S(tope.eCDD_res, [0, 0]) ← S(tope.eCDD_A)_1
27:                for indice = 2 a #_I(tope.eCDD_A) do        ▷ Los intervalos se "pliegan" en uno sólo.
28:                    S(tope.eCDD_res, [0, 0]) ← UNION (S(tope.eCDD_res, [0, 0]), S(tope.eCDD_A, indice))
29:                end for
30:                for indice = 2 a #_I(tope.eCDD_A) do
31:                    EliminarIntervalo(tope.eCDD_res, indice)        ▷ Queda sólo el [0, 0].
32:                end for
33:            end if
34:        else                                  ▷ diff(tope.eCDD_A) = y ≠ x_0, e y − x o x − y con x ∈ C
   ▷ Aplanar y eliminar la restricción
35:            I ← primer intervalo sin marcar en tope.eCDD_A
36:            if existe tal I then
37:                Marcar (tope.eCDD_A, I)
38:                Apilar (pila, [tope.eCDD_A, tope.eCDD_res])
39:                Apilar (pila, [S(tope.eCDD_A, I), S(tope.eCDD_res, I)
```

Algoritmo 3.4.6: Algoritmo iterativo de reseteo de relojes en un eCDD—Parte 1.

```
40:             else
41:                 for
42:                     doS(tope.eCDD_A)_1 ← Union (S(tope.eCDD_A)_1, S(tope.eCDD_A)_indice)
43:                     EliminarIntervalo(tope.eCDD_res, indice)          ▷ Queda sólo el primero.
44:                 end for
45:                 tope.eCDD_res ← copia(tope.eCDD_A)
46:                 if diff(tope.eCDD_A) es de la forma y − x then
47:                     I(tope.eCDD_res)_1 ← IntervaloDelReloj (tope.eCDD_res, y)
48:                 else
   ▷ es x − y
   ▷ Se entiende por multiplicar por -1 un intervalo < a, b > al intervalo < −b, −a >.
   ▷ La inclusión o no de los extremos se invierte análogamente.
49:                     I(tope.eCDD_res)_1 ← −1×IntervaloDelReloj (tope.eCDD_res, y)
50:                 end if
51:             end if
52:         end if
53:     end while
54:     return retValue
55: end function
```

Algoritmo 3.4.7: Algoritmo iterativo de reseteo de relojes en un eCDD—Parte 2.

$v \models \psi \implies$

$$v[z := 0, \forall z \in C] \models rToCDD(reset(C, \psi)) \implies v[z := 0, \forall z \in C] \models \textsc{Reset}(C, rToCDD(\psi)) \quad (3.13)$$

$$v[z := 0, \forall z \in C] \models \textsc{Reset}(C, rToCDD(\psi)) \implies v[z := 0, \forall z \in C] \models rToCDD(reset(C, \psi)) \quad (3.14)$$

Demostraremos primero la propiedad 3.13, mostrando que la manera de armar el e*CDD resultante es tal que lo enunciado anteriormente es verdadero. Para esto, hay que notar dos puntos esenciales sobre la valuación $v[z := 0, \forall z \in C]$. En primer lugar es necesario notar que, por definición de la valuación con asignación de relojes,*

$$v[z := 0, \forall z \in C](x) = \begin{cases} v(x) & si\ x \notin C. \\ 0 & si\ x \in C. \end{cases}$$

Esto conlleva como consecuencia que, dados dos relojes x e y que no se encuentren en el conjunto de relojes reseteados C, $v[z := 0, \forall z \in C](x) - v[z := 0, \forall z \in C](y) = v(x) - v(y)$. Veamos entonces cómo estas observaciones influyen durante la construcción del diagrama Reset *$(C, rToCDD(\psi_1))$. Analizando los distintos estados que podemos encontrar dentro de la pila, encontramos los siguientes casos:*

- *si diff*$(tope.eCDD_A)$ no *involucra a relojes presentes en C, dicho nodo (que representa una diferencia de relojes), y los intervalos salientes del mismo no se ven afectados (esto es así debido a que en este caso el diagrama se mantiene íntegro, ver líneas (7) a (15)). En primer lugar, esto demuestra que ya existía un intervalo I tal que $v($diff$(tope.eCDD_A)) \in I$ (pues $v \models tope.eCDD_A$). Por otra parte, gracias a lo que notábamos más arriba acerca de la valuación $v[z := 0, \forall z \in C]$, $v[z := 0, \forall z \in C]($diff$(tope.eCDD_A)) = v($diff$(tope.eCDD_A)) \in I$. En conclusión, resulta que la valuación $v[z := 0, \forall z \in C]$ satisface este nodo del resultado (veremos que satisface también sus subdiagramas al analizar los demás estados posibles de la pila).*
- *si, en cambio, diff*$(tope.eCDD_A)$ *se trata de una diferencia donde alguno (o ambos) de los relojes involucrados* sí *pertenece al conjunto de relojes a resetear; y el reloj x_0* no *asume ningún rol, entonces la diferencia en cuestión puede ser tanto de la forma $y - z$ como $z - y$, siendo ninguno de ellos igual a x_0. En este caso, la valuación claramente se verá modificada. Sin embargo, veremos que este hecho termina siendo irrelevante, ya que cuando encontramos este estado en la pila se realiza lo siguiente (ver líneas (34) a (52)):*
  1. *para cada sub-*e*CDD relacionado a los intervalos salientes de tope.*eCDD_A*, se obtiene el resultado de aplicar el reseteo de relojes del conjunto C, insertándolo en la pila. En particular, el subdiagrama correspondiente al intervalo donde se encontraba el resultado de aplicar la valuación v a diff*$(tope.eCDD_A)$ *será satisfacible por $v[z := 0, \forall z \in C]$ (siguiendo el razonamiento de la demostración).*

2. *una vez obtenidos todos los subdiagramas con el reloj reseteado, se unen de forma de obtener un sólo diagrama. Como cada valuación v que satisfacía a tope.eCDD_A lo hacía con alguno de los intervalos, deberá ser cierto que (como se unen todos), también satisfaga al nuevo diagrama.*
3. *este nuevo diagrama se coloca como subdiagrama de lo que ya se tenía en la pila (y que por hipótesis ya era satisfecho por la misma valuación $v[z := 0, \forall z \in C]$).*

- *finalmente, en el caso de que diff$(tope.\text{e}CDD_A) = z - x_0$, donde $z \in C$, el procedimiento es similar al caso anterior, con la salvedad de que se coloca un nodo $z - x_0$ como raíz, cuyo único intervalo saliente es $[\,0, 0\,]$. Esto lo hace directamente satisfacible por $v[z := 0, \forall z \in C]$.*

Esto concluye la demostración de la primera implicación.

A continuación, demostramos que la propiedad 3.14 también se verifica; es decir, que cada vez que una valuación de la forma $v[z := 0, \forall z \in C]$ satisface el e*CDD resultante, es porque dicha valuación satisface también la región resultante de setear esos relojes a 0. Es decir, si $v' = v[z := 0, \forall z \in C]$ satisface el* e*CDD resultante debe ser que*

$$v'(x) = \begin{cases} 0 & si\ x \in C \\ v(x) & si\ no \end{cases}$$

Citando la demostración anterior, se verifica inmediatamente que:

- *Los nodos correspondientes a las diferencias entre relojes $x \in C$ y x_0 ven sus intervalos modificados a $[\,0, 0\,]$. De esta forma, si v' satisface dicho nodo es porque sin duda se verifica que $v'(x) = 0$.*
- *Los nodos correspondientes a diferencias de relojes y no pertenecientes a C y x_0 no se ven modificados en absoluto. Es decir, $v'(x) = v(x)$.*

Si bien es cierto que el e*CDD resultante no posee algunas de las restricciones que el original sí planteaba (recordemos que el original se encontraba canonizado, y que se eliminaban aquellas restricciones correspondientes a diferencias de relojes donde alguno de ellos perteneciese a C), estas omisiones no influyen en las valuaciones que satisfacen al* e*CDD. Esto es así pues dichas restricciones omitidas pueden ser deducidas de las restricciones impuestas sobre cada reloj individual.*

□

3.4.6. Descripción del algoritmo de obtención de sucesores temporales de una zona

Finalmente, la obtención de sucesores temporales de una zona es absolutamente crítica para nuestro algoritmo de alcanzabilidad por medio de una exploración *forward*. Esta operación es la abstracción del *paso del tiempo*, que incrementa cada uno de los relojes del sistema de manera uniforme. Es decir, dada una zona ψ_1, se busca obtener la zona $suc_{temp}(\psi_1)$, la cual es tal que se verifica que $\forall v \in \mathcal{V}_X, v \models \psi_1 \implies \forall \delta \in \mathbb{R}_0^+\ (v + \delta) \models suc_{temp}(\psi_1)$. Dado que, justamente, los relojes se incrementan uniformemente, las diferencias entre ellos *se mantienen constantes*, sea cual fuere el incremento temporal.

Una vez presentado el algoritmo, veremos que tendrá una utilidad evidente (apuntando a la obtención de sucesores temporales de *regiones*) a partir del siguiente lema:

Lema 3.4.2 (Cálculo por zonas de sucesores temporales) *Dada una región ψ y un conjunto de zonas temporales $Z = \{z_1, \ldots z_n\}$ tales que $\bigcup_{z_i \in Z} = \psi$, vale que*

$$suc_{temp}(\psi) = \bigcup_{z_i \in Z} suc_{temp}(z_i)$$

Demostración

Demostraremos esta igualdad mediante una doble inclusión.
$\subseteq$) En primer lugar, nos preocupará verificar que

$$suc_{temp}(\psi) \subseteq \bigcup_{z_i \in Z} suc_{temp}(z_i)$$

es decir que

$$\forall v \in \mathcal{V}_X, v \models suc_{temp}(\psi) \implies \exists z_i \in Z \text{ tal que } v \models suc_{temp}(z_i)$$

Notemos que, como $\psi = \bigcup_{z_i \in Z} z_i$*, para cualquier* $v' \in \mathcal{V}_X$ *tal que* $v' \models \psi$*, debe ser que existe* $z_i \in Z$ *tal que* $v' \models z_i$ *(de no existir dicha zona* z_i*, la unión no podría ser satisfecha por* v'*). Por otra parte, todos las valuaciones pertenecientes al conjunto característico de los sucesores temporales de* ψ *tienen la forma* $v' + \delta$ *para algún* $\delta \in \mathbb{R}^+$ *(por definición). Pero además, si* $v' \models z_i$*, entonces* $v' + \delta \models suc_{temp}(z_i)$ *(nuevamente, por definición), y por lo tanto satisface a cualquier región que la contenga (en particular, la unión planteada).*

$\supseteq$*) En este caso, nos interesará mostrar que*

$$\bigcup_{z_i \in Z} suc_{temp}(z_i) \subseteq suc_{temp}(\psi)$$

es decir, que

$$\forall v \in \mathcal{V}_X, v \models \bigcup_{z_i \in Z} suc_{temp}(z_i) \implies v \models suc_{temp}(\psi)$$

Análogamente al caso anterior, si v *es tal que* $v \models \bigcup_{z_i \in Z} suc_{temp}(z_i)$*, es porque existe una zona* $z_i \in Z$ *tal que* $v \models suc_{temp}(z_i)$*. Por definición, esto implica que existen un* $\delta \in \mathbb{R}^+$ *y una valuación* $v' \in \mathcal{V}_X$ *tales que* $v = v' + \delta$ *y* $v' \models z_i$*. Dado que* $z_i \subseteq \psi$*, se cumple que* $v' \models \psi$ *y, además vale que* $\forall \delta \in \mathbb{R}^+, v' + \delta \models suc_{temp}(\psi)$*. Esto concluye la demostración del lema.*

□

A partir del resultado planteado por el lema 3.4.2, si se tiene una forma de calcular los sucesores temporales correspondientes a una zona, entonces pueden calcularse los sucesores temporales correspondientes a una región uniendo todos los sucesores temporales de las zonas que la componen (en cualquiera de las descomposiciones posibles). Es válido notar que la descomposición de una región en zonas no necesariamente es única ni tampoco disjunta. Sin embargo, ello no redunda en ninguna diferencia respecto de la corrección del método, aunque seguramente la *performance* será peor cuanto más fragmentada se encuentre la región.

Presentamos entonces un algoritmo, en el listado 3.4.8, para calcular los sucesores temporales correspondientes a una zona, representada por medio de un *e*CDD que, al igual que en el caso de la determinación de inclusión, se encuentra *ajustado* y *completo*. Esto permite calcular los sucesores temporales de una manera muy simple y similar a como se hace con DBMs, como se verá en el algoritmo; y sentará la base para calcular los sucesores temporales de una región. Es importante notar que en el análisis *forward* las zonas sufren un reseteo de relojes para luego calcular sus sucesores temporales. Dado un último requisito de tener la zona en forma completa, cobra importancia el uso de IntervaloDelReloj en la función de reseteo.

Observación 3.4.6 *Al igual que en el caso del cálculo del reseteo de relojes, nos interesa más analizar qué sucede al calcular los sucesores de una* zona*, dado que ése será el uso que se le dará dentro de nuestros algoritmos. Análogamente a lo explicado en aquel caso, la complejidad dependerá sólo de la cantidad de nodos, generalmente acotada por* n^2*, aunque en general menor debido a reducciones. La complejidad del algoritmo es, entonces,* $O(n^2)$*.*

Teorema 3.4.5 (Corrección de la función de cálculo de sucesores temporales) *La función presentada para el cálculo de los sucesores temporales de una zona representada por medio de un* e*CDD es correcta.*

Demostración

Demostraremos entonces el teorema 3.4.5, es decir, que la función presentada calcula correctamente los sucesores temporales de la zona pasada como parámetro.

La terminación en un tiempo finito del algoritmo, y la conservación del invariante en la estructura resultante se fundamentan en los mismos argumentos que hemos utilizado para la demostración realizada

```
1: function SucesorTemporalZona (eCDD_zona)        ▷ Calcula la zona sucesora de eCDD_zona
2:     Pila(sub-eCDD, sub-eCDD) pila
3:     eCDD retValue
4:     Apilar (pila, [eCDD_zona, retValue]
5:     while ¬ Vacía(pila) do
6:         tope ← Desapilar (pila)
7:         if diff(tope.eCDD_zona) no tiene la forma x − x_0 para algún reloj x then
8:             tope.eCDD_res ← copia(tope.eCDD_zona)                ▷ Copiar completa la zona.
9:         else
10:            diff(tope.eCDD_res) ← diff(tope.eCDD_zona)
11:            I ← intervalo saliente de tope.eCDD_zona             ▷ Al ser una zona, I es único.
12:            if ¬ Marcado(I) then
    ▷ Proyectarlo al futuro.

13:                Marcar (I, eCDD_zona)
14:                AgregarIntervalo (tope.eCDD_res, ⟨I.min, +∞⟩)
15:                Apilar (pila, [tope.eCDD_zona, tope.eCDD_res)
16:                Apilar (pila, [S(tope.eCDD_zona, I), S(tope.eCDD_res, ⟨I.min, +∞⟩)])
17:            end if
18:        end if
19:    end while
20:    return retValue
21: end function
```

Algoritmo 3.4.8: Algoritmo iterativo de cálculo de sucesores temporales de una zona utilizando *e*CDDs.

acerca de los algoritmos anteriores. Por este motivo, no ahondaremos en ellos y, en cambio, profundizaremos en la demostración de corrección del resultado. Es decir, verificaremos que para toda zona z, y para toda valuación $v \in \mathcal{V}_X$,

$$v \models z \implies \forall \delta \in \mathbb{R}_0^+ (v+\delta) \models SucesorTemporalZona(rToCDD(z)) \Leftrightarrow (v+\delta) \models suc_{temp}(z)$$

Utilizando la corrección de rToCDD y desdoblando la doble implicación, esto resulta equivalente a demostrar por separado que
$v \models rToCDD(z) \implies$

$$\forall \delta \in \mathbb{R}_0^+ (v+\delta \models rToCDD(suc_{temp}(z)) \implies v+\delta \models SucesorTemporalZona(rToCDD(z))) (3.15)$$
$$\forall \delta \in \mathbb{R}_0^+ (v+\delta \models SucesorTemporalZona(rToCDD(z)) \implies v+\delta \models rToCDD(suc_{temp}(z))) (3.16)$$

Comenzaremos por demostrar la propiedad 3.15. Inmediatamente resulta claro, por definición, que se verifica que $v \models rToCDD(z) \implies \forall \delta \in \mathbb{R}_0^+ v+\delta \models rToCDD(suc_{temp}(z))$. Falta ver entonces que es cierto también que $v+\delta \models SucesorTemporalZona(rToCDD(z))$. Veamos el algoritmo para verificar esto, observando en todo momento que la representación de una zona en un e*CDD posee* un único camino. *Es necesario notar también que para cualquier par de relojes x e y tales que $x \neq x_0$ e $y \neq x_0$, para cualquier δ real positivo, $v(x)-v(y) = (v+\delta)(x)-(v+\delta)(y)$. Pueden notarse (del algoritmo) los siguientes aspectos:*

- *No se agregan nuevas diferencias de relojes. Es decir, la altura del* e*CDD se mantendrá inalterada tras la ejecución del algoritmo.*
- *No se modifican las diferencias de relojes. Añadido a lo anterior, se deduce que el* e*CDD resultante posee los mismos nodos, y en el mismo orden.*
- *Los intervalos correspondientes a los nodos cuya diferencia de relojes* no *incluye al reloj x_0 se mantienen inalterados. Esto hace que las diferencias entre relojes se mantengan constantes, como se observaba anteriormente.*
- *Finalmente, los intervalos $\langle c_1, c_2 \rangle$ salientes de los nodos de la forma $x - x_0$ son modificados y pasan a tomar la forma $\langle c_1, +\infty \rangle$.*

Sea entonces un $\delta \in \mathbb{R}_0^+$ cualquiera, y una valuación v tal que $v \models rToCDD(\psi)$. Para cada reloj x tal que $x - x_0$ pertenece al eCDD, sabemos que el valor de $v(x)$ pertenece a cierto intervalo $\langle \alpha_x, \beta_x \rangle$. Además, $(v+\delta)(x) \geq v(x)$ por lo que $(v+\delta)(x) \in \langle \alpha_x, +\infty \rangle$, que es el intervalo con el cual $x - x_0$ se ingresa al resultado. Además, vimos que en el caso de las otras diferencias de relojes, los intervalos se mantienen inalterados, y también lo hace el valor de $(v+\delta)$ respecto de v. Por lo tanto, $(v+\delta)$ satisface el eCDD resultante, que es lo que queríamos ver.

Resta ver que el eCDD resultante no es satisfacible sino por valuaciones que satisfacen los sucesores temporales de la zona (lo expuesto en la propiedad 3.16). Para lograr este objetivo, nos fundamentaremos en exhibir que el algoritmo aquí presentado es equivalente a aquel que calcula sucesores temporales utilizando DBMs. La corrección del método sobre DBMs ya ha sido mostrada en [Tri98]. Este algoritmo consiste en, dada una DBM completa y ajustada, eliminar las cotas superiores de los relojes, pasándolas a $+\infty$. Es decir, si M es la DBM representando la zona original, se calcula M_{suc}:

$$M_{suc}[x,y] = \begin{cases} M[x,y] & si\ y \neq 0 \\ (<, +\infty) & si\ y = 0 \end{cases}$$

Resulta inmediato de una lectura del algoritmo que esto es exactamente lo que se realiza: cuando el algoritmo encuentra una diferencia de relojes de la forma $x - x_0$, elimina la cota superior correspondiente, proyectando la nueva cota a $+\infty$. Esta misma idea puede expresarse geométricamente, donde los sucesores temporales son las proyecciones uniformes de la zona en relación a la recta $x = y$.

□

Finalmente, cabe aclarar que, si bien es correcto calcular los sucesores temporales de una región mediante el algoritmo anterior y la unión de los resultados, sólo se necesita el cálculo sobre zonas para la verificación *forward* de alcanzabilidad.

Estas últimas consideraciones cierran este capítulo. En los próximos capítulos analizaremos de qué manera se llevó todo esto a la práctica, discutiendo las optimizaciones que se implementaron en pos de la obtención de mejores resultados, y comentaremos algunos experimentos llevados a cabo.

Capítulo 4

Implementación

En capítulos anteriores, describimos la problemática del model checking temporizado, y presentamos las estructuras que se han usado hasta ahora en la mayoría de las herramientas para representar las zonas y regiones temporales. Luego, presentamos nuestra estructura junto con los algoritmos que implementan las operaciones necesarias para el model checking, y las demostraciones de su corrección.

En este capítulo se describirá tanto la implementación de las modificaciones realizadas al model checker Zeus, como también las distintas optimizaciones que se agregaron a fin de mejorar los tiempos de ejecución y uso de memoria del verificador. Algunas de estas optimizaciones son independientes de la estructura utilizada (fueron agregadas también a la versión anterior que trabaja mediante DBMs), mientras que otras son específicas para la estructura que resulta de esta tesis. Además, detallaremos algunas diferencias significativas que exhibe lo realmente implementado respecto de lo presentado en el capítulo 3.

El desarrollo completo se llevó a cabo en ANSI C, trabajando sobre el sistema operativo Unix. Además de la necesidad de eficiencia, que hubiese determinado de todas maneras utilizar el lenguaje C, la razón de mayor peso es que Zeus ya se hallaba implementado en este lenguaje. Esto resultó en una mayor facilidad de integración con el código ya existente.

La implementación de la nueva estructura comprende más de nueve mil líneas de código, además de las modificaciones menores necesarias para la correcta integración. Esto contrasta notoriamente con la implementación clásica sobre RDBMs, cuya implementación tomó algo menos de tres mil líneas de código en el mismo lenguaje.

A continuación describiremos la implementación de la estructura, junto con las optimizaciones que se le realizaron. En la siguiente sección, comentaremos una optimización ajena a la estructura en sí, para finalmente comentar las diferencias más notorias comparando con las versiones presentadas anteriormente.

4.1. Implementación de la estructura

En un primer momento, se optó por implementar la estructura en la manera clásica en que se implementan las estructuras arbóreas. Esto es, se diseñó la estructura en base a nodos unidos por punteros, guardando información adicional para codificar el intervalo correspondiente a cada puntero. Lamentablemente, el *overhead* producido por la proliferación de punteros resultó extremadamente perjudicial en términos de eficiencia ya que, además de la gran pérdida de espacio asociada a la cantidad de punteros, la mayoría de las operaciones hace uso de al menos tres indirecciones a memoria. Esto no hace más que ralentizar la ejecución.

Por esta razón, procedimos a analizar la situación más en detalle. Este análisis arrojó como resultado la siguiente observación: en general, el algoritmo trabaja con *zonas* y no con *regiones* temporales. Estas zonas, como se mostró en el capítulo 3, se representan tan sólo mediante *tiras* de diferencias de relojes, sin bifurcaciones de ningún tipo. Como resultado, en el caso de las zonas, la estructura es completamente lineal. Por otra parte, en la representación de las regiones sólo se generan bifurcaciones a medida que se unen zonas cuyas valuaciones para una misma diferencia de reloj son completamente distintas. En

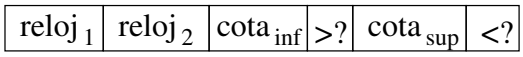


Figura 4.1: Una diferencia de relojes, representada con la técnica de *bitstuffing*.

muchos de los casos de unión esto no sucede para todas las diferencias de relojes, con lo que se preserva la linealidad de la estructura, al menos hasta cierto punto. Buscamos, entonces, alguna manera de explotar al máximo esta linealidad, con el objetivo de demorar la ramificación en punteros hasta el momento en que fuese estrictamente necesario.

La solución que fue finalmente implementada consistió en representar linealmente esta tira de datos. Definimos al nodo de manera alternativa, de forma tal que bajo esta nueva representación el nodo consta tanto de los relojes que representan la diferencia como también del intervalo en el que se halla comprendida. Dado que se trata de una zona, este intervalo es único. Claramente, cada nodo puede representarse con los datos adyacentes en memoria, sin utilizar punteros. Más aún, la sucesión de nodos que representa la tira puede representarse colocando cada nodo directamente a continuación del otro. Además, como las tiras generalmente están *reducidas*, y no contienen la representación de la totalidad de las diferencias de relojes, sólo requerimos de la memoria una fracción del total de diferencias para esta representación. De esta forma, sólo es necesario recurrir a punteros en el caso de que esta estimación sea superada por la realidad, en cuyo caso se recurre a obtener más espacio libre, para luego encadenar las representaciones resultantes.

Como optimización adicional, además, utilizamos una técnica similar a la de *bit stuffing*, de manera que sólo guardamos en estas tiras la cantidad de bits necesarios para representar cada valor. De esta forma minimizamos la cantidad de memoria utilizada para la representación de las zonas. Esta técnica se ilustra en la figura 4.1. En la misma se muestran distintos campos, cada uno de los cuales delimita una secuencia de bits que representará un valor determinado. Los valores $reloj_1$ y $reloj_2$ representan los relojes cuya diferencia se codificará, mientras que los números $cota_{inf}$ y $cota_{sup}$ denotan los valores entre los cuales puede oscilar la diferencia de sus valuaciones. Los campos marcados <? y >? son dos bits individuales que especifican la estrictez de dichas cotas, es decir si dicho valor está o no incluido dentro de los valores permitidos. El hecho de utilizar campos de bits a bajo nivel permite un ahorro importante de espacio de memoria respecto de utilizar los tipos ya alineados; por ejemplo, supongamos un estudio en el cual se tienen cinco relojes, cuyos valores nunca excederán el valor 12, o donde no se los compara contra constantes mayores a dicho valor. Entonces, sólo necesitamos tres bits para codificar los relojes, y otros cuatro para codificar cada cota. En total, codificar uno de estos campos nos insume tan sólo dieciséis bits (dos relojes, dos cotas y dos bits de estrictez), contra treinta y cuatro (suponiendo que usáramos bytes para relojes y cotas y bits para la estrictez). Es esencial notar que para que la aplicación de la técnica sea posible es necesario conocer de antemano los máximos valores que podrán tomar tanto los relojes como sus valuaciones. En el caso de la cantidad de relojes, esta información se obtiene trivialmente del modelo. El caso de las valuaciones es un poco más complejo, y describiremos la idea de cómo se soluciona un poco más adelante en este mismo capítulo.

Esta optimización no es exclusiva de la representación de las zonas; la misma idea puede hacerse extensiva a las regiones. Para lograr este objetivo, se utiliza la misma técnica, con la diferencia de que se hace necesario bifurcar al encontrar una diferencia de relojes cuyas valuaciones se encuentran comprendidas en más de un intervalo. En ese caso, simplemente dividimos la estructura en tantas otras como intervalos se encuentren en el nodo que se bifurca. Cada una de las estructuras resultantes es, nuevamente, una subregión en sí misma, por lo que esta idea se aplica nuevamente, hasta llegar a cada una de las hojas. Esta técnica está ilustrada para mejor comprensión en la figura 4.2.

Finalmente, queda por detallar el mecanismo de recorrido de los intervalos correspondientes a un nodo en las distintas operaciones. En el capítulo 3 definimos la función PreprocesarIntervalos para evitarnos la discusión sobre este aspecto. Sin embargo, como notábamos en ese momento, el uso de esta función genera una estructura mucho más fragmentada que lo que realmente es necesario. Por otra parte, el uso de la función PreprocesarIntervalos agrega también cierta cantidad de tiempo de proceso que sería deseable eliminar. En realidad, en la implementación de la herramienta, no existe tal cosa como la función PreprocesarIntervalos. En cambio, el tratamiento de los intervalos se realiza analizando cada caso de intersección por separado, sin generar intervalos superfluos ni aumentando la cantidad original de intervalos de cada nodo. Este tratamiento se realiza aprovechando fuertemente el hecho de que los intervalos correspondientes a un nodo en particular se encuentran ordenados de menor a mayor. De esta

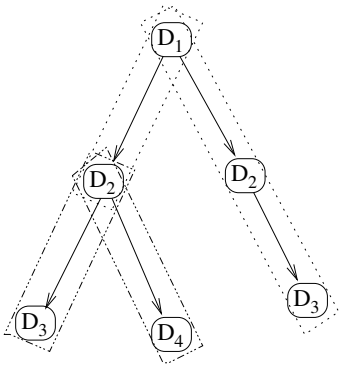


Figura 4.2: Un *e*CDD representando una región, notando la aplicación de la técnica de *bitstuffing*.

manera, operar con los intervalos correspondientes a dos nodos distintos puede hacerse en tiempo lineal en la cantidad total de intervalos, efectuando una especie de *"merge"* entre ambos; las particularidades de este *"merge"* dependerán de cada operación. Esto se diferencia de la versión presentada anteriormente, donde se mostraba que todos los intervalos eran trabajados de a pares; por medio de esta técnica sólo se trabaja con los pares cuya comparación pueda potencialmente arrojar resultados relevantes.

4.2. Optimizaciones

Describiremos en esta sección varias optimizaciones que se introdujeron al model checker, independientemente de la estructura utilizada: la técnica de *reducción de relojes*, el *ajuste de cotas*, y la *reducción de la cola de estados por visitar*. Todas estas optimizaciones se describen en detalle en [DT98], por lo cual nos limitaremos a comentar sus características más salientes.

4.2.1. Reducción de relojes

De manera independiente al desarrollo de la estructura, se implementó la técnica de reducción de relojes. El *rationale* detrás de la técnica es que en cada locación discreta sólo un conjunto de los relojes se encuentra realmente activo. Dicho de otra forma, las valuaciones de algún subconjunto de los relojes no tendrán incidencia en las posibles acciones a realizar en dicha locación. Por un lado, no influirán en el invariante del estado, de forma que el avance del tiempo no se verá restringido por esos relojes; mientras que por el otro, las transiciones habilitadas tampoco estarán limitadas por las valuaciones de esos mismos relojes. Intuitivamente, un reloj sólo se considera *activo* desde el momento en que su valor se resetea, hasta que el mismo es observado.

Gracias a las observaciones recién descriptas, puede obtenerse el conjunto de relojes activos correspondiente a cada locación. De esta manera, alcanza con tener en cuenta sólo las valuaciones de dichos relojes para llevar a cabo todas las operaciones. Además del evidente ahorro en espacio (debido a lo innecesario de representar diferencias que incluyan relojes inactivos), se obtiene también una reducción de los tiempos de ejecución, ya que al ignorar varias diferencias el trabajo de las operaciones es menor.

4.2.2. Ajuste de cotas

Como mencionamos en capítulos anteriores, las particiones que pueden realizarse a los intervalos no son infinitamente grandes. De hecho, sabemos que si M es la constante más grande que aparece en cualquiera de las comparaciones del autómata a analizar (ignorando el signo de estas constantes, considerando sólo su módulo), entonces en el momento en que los relojes (o diferencias de relojes) superen ese valor (o bien se vuelvan inferiores a $-M$), puede suponerse su valor como equivalente a ∞ o $-\infty$ respectivamente. Esto, sumado a la discretización de los intervalos, resulta en que no se encontrarán más de $2M$ intervalos por nodo. Esta propiedad ya nos permitía acotar el tamaño de nuestra estructura, además de permitir establecer una cota para el espacio requerido por las constantes, lo cual a su vez posibilita el uso de la técnica de *bitstuffing* ya descripta.

Sin embargo, también merece algo de atención notar el hecho de que, si bien la mayor constante puede ser M, *no toda diferencia de relojes alcanzará dicha cota*. Por lo tanto, prestar atención a las constantes del autómata de manera individual para cada reloj resulta en unas cotas más ajustadas. Esto, a su vez, redunda en una menor cantidad de intervalos en el caso de que se realice una partición máxima de los mismos.

4.2.3. Reducción de la cola de estados por visitar

Como vimos anteriormente en el algoritmo 1.3.1, durante el procedimiento de verificación de alcanzabilidad se mantienen dos repositorios con estados y zonas temporales: *visitados*, donde se almacenan aquellos que ya se han visitado, y *por_explorar*, que contiene todos los que se han encontrado como sucesores pero aún no se han recorrido. A lo largo de la tesis, nos hemos concentrado en trabajar de forma de minimizar el costo de las operaciones de chequeo de inclusión en el conjunto de *visitados*, lo cual es necesario para la terminación del algoritmo. Sin embargo, es posible analizar también un poco más a fondo lo que sucede con la cola de estados *por_explorar*. Analicemos qué sucede cuando se genera un sucesor $< s, z >$ en el algoritmo (líneas (14) a (15)): los nuevos estados generados se agregan, sin análisis ulteriores, a la cola de estados por explorar. Sin embargo, este agregado *naïf* de los nuevos estados en la cola puede generar duplicados en la misma, ya que podrían agregarse estados ya existentes. Este agregado redundante no amenaza la terminación del algoritmo (ya que de todas maneras se recorren los estados que pueden llegar a ser los buscados por el algoritmo de alcanzabilidad), pero sí resulta en una mayor cantidad de cálculos, que claramente podrían evitarse. Entonces, en principio resulta evidente la necesidad de verificar que los estados que se están ingresando a la cola no pertenezcan ya a la misma.

Por otra parte, puede darse también la posibilidad de que no exista en la cola el mismo estado que se está ingresando, pero sí exista uno que esté *estrictamente incluido en él*. Cuando definimos los operadores suc_t y suc_e notamos, en la propiedad 1.3.7, que dichos operadores son *monótonos*. Como consecuencia de esto, si se tienen dos estados por explorar S y S' tales que $S \subseteq S'$, todos los estados que se obtengan como sucesores de S se obtendrán también al calcular los sucesores de S'. Este último cálculo, además, potencialmente contendrá otros estados adicionales que no se obtendrían por medio de S. Como resultado de este razonamiento, se deduce que en el caso de tener que agregar S' a la cola de estados por explorar, puede procederse a la remoción de S de la misma, sin modificar la corrección del algoritmo, pero efectivamente reduciendo el tiempo requerido para su terminación.

A modo de resumen, vale aclarar que para cada una de las optimizaciones, si bien la optimización no depende de la estructura, fue necesario modificar levemente los algoritmos de la misma para implementarlos y, al mismo tiempo, mantener la validez de sus invariantes.

Capítulo 5

Resultados obtenidos

A lo largo de esta tesis hemos perseguido el objetivo de mejorar el rendimiento de una herramienta de model checking, mediante una nueva estructura y algunas optimizaciones adicionales ajenas a ella. En este sentido, presentamos un resumen de la situación previa, para luego introducir y desarrollar una nueva estructura que nos permitiese, al fin y al cabo, conseguir dicho objetivo.

El presente capítulo, finalmente, presenta los resultados a lo que hemos arribado. Estos resultados se han obtenido aplicando la herramienta modificada según lo descripto en capítulos anteriores a diversos casos de estudio, comunes en la literatura. Por otra parte, compararemos los resultados obtenidos mediante la versión modificada de Zeus contra aquellos obtenidos en su versión original, cuyas restricciones temporales se encuentran implementadas por medio de RDBMs. Además, compararemos los resultados teniendo en cuenta también si se han utilizado o no las optimizaciones presentadas en el capítulo 4.

Para obtener los resultados que exponemos en este capítulo, utilizamos dos casos de estudio: el modelo de *Train-Gate-Controller* [Alu92], y el de *pipeline* de procesos. Antes de presentar los resultados, daremos una introducción a estos dos modelos.

5.1. *Train-Gate-Controller*

El modelo del *Train-Gate-Controller*—TGC, para abreviar—es muy utilizado en la bibliografía. Este modelo especifica el cruce de un ferrocarril por medio de tres elementos: tren, barrera y controlador.

Informalmente, este modelo se entiende como describimos debajo.

- Cuando el tren se acerca a la barrera, envía una señal al controlador anunciando su llegada. Este proceso insume no menos de 6 unidades de tiempo; mientras que se considera que un tren esta "cerca" de la barrera si se encuentra a menos de 10 unidades de tiempo.
- El tren permanece dentro del cruce entre 11 y 15 unidades de tiempo.
- Cuando el controlador es informado acerca del advenimiento de un tren, tiene no más de 1 unidad de tiempo para indicar a la barrera que debe bajar. El controlador recién reaccionará nuevamente una vez que el tren haya pasado por el cruce, momento en el cual vuelve a tener no más de una unidad de tiempo, en este caso para indicar a la barrera que ascienda.
- La barrera, por su parte, sólo entiende órdenes de subir y bajar. Cada una de estas operaciones le insume 3 unidades de tiempo, aunque puede ser interrumpida en el medio de la operación por una orden contraria (es decir, puede ordenársele que descienda mientras esta subiendo, o viceversa).

Por medio de este modelo, intentan detectarse casos de error tales como que el tren pase por el cruce cuando la barrera no está baja, o que la barrera continúe abajo aún cuando no hay trenes en proximidad. Este modelo puede generalizarse, de manera de controlar simultánemente cualquier cantidad de trenes. En los casos de prueba que presentaremos, instanciamos al modelo utilizando dos, cuatro y hasta cinco autómatas representando trenes en cada caso. Además, para cada una de las instancias anteriores se cuenta también con un autómata para la barrera, otro para el controlador, y un último

autómata denominado *observador*. Este observador es un autómata que avanza junto con el resto del sistema, pero que evoluciona a un estado de error en cuanto detecta una situación indeseada. En el caso que nos ocupa, el observador intenta detectar el caso en que el tren 1 intenta cruzar con la barrera aún arriba. Este autómata se presenta en la figura 5.1. Debido a los parámetros particulares del problema se puede hacer que el autómata evolucione al estado de error o no dependiendo de los valores de sus guardas. Esto nos da la flexibilidad de crear un ejemplo donde el error sea alcanzable y otro donde no.

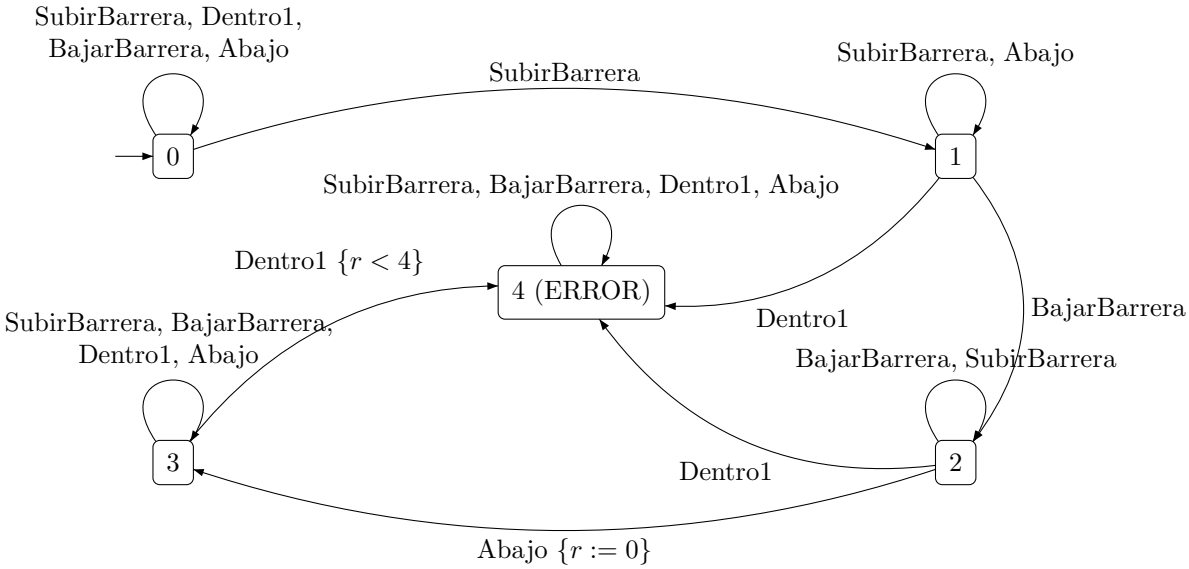


Figura 5.1: Observador del ejemplo *Train-Gate-Controller*.

Por ejemplo, la secuencia `(SubirBarrera, Dentro1)`, donde el tren 1 pasó por el cruce luego de que el controlador indicara levantar la barrera, lleva a este observador directamente al estado de error. Otra secuencia que hace evolucionar al observador hacia un estado de error es `(SubirBarrera, BajarBarrera, Dentro1)`, donde el tren 1 pasó por el cruce luego de que el controlador indicara bajar la barrera, pero antes de que ésta bajara. Finalmente, la secuencia `(SubirBarrera, BajarBarrera, Abajo, SubirBarrera, Dentro1)` representa la situación en la que el tren pasa estando la barrera baja, pero habiendo recibido orden de levantar. Ésa es claramente una situación no deseada ya que el tren 1 estaría pasando por el cruce mientras la barrera se levanta.

Por otra parte, en la tabla 5.1, se muestran las distintas magnitudes a tener en cuenta para cada uno de los casos instanciados. En ella se aprecian tanto los parámetros de cada autómata en particular, como también aquellos correspondientes a su composición. Es esta composición, en definitiva, el autómata que representa el sistema a verificar.

5.2. Pipeline

El modelo del *pipeline* de procesos consiste de varios procesos idénticos, cada uno de ellos consistente de varias etapas o fases. Si bien dos fases iguales no pueden ejecutarse simultáneamente, sí puede hacerse con fases distintas de sendos procesos. Esto permite paralelismo en la ejecución; por ejemplo, el componente encargado de la primera fase puede ejecutar dicha fase del primer proceso, y a continuación ejecutar la misma fase del segundo proceso, mientras el primero ya se encuentra ejecutando su fase siguiente. En los ejemplos hemos manipulado la cantidad de fases y el retraso de la señal que provoca el avance. Por otra parte, el observador en este caso se construyó de tal manera de permitir variar el tiempo requerido

Autómata	Locaciones	Transiciones	Relojes
Cada tren	3	3	1
Barrera	4	10	1
Controlador (2 trenes)	9	48	1
Controlador (4 trenes)	15	140	1
Controlador (5 trenes)	18	205	1
Observador	5	22	1
Composición (2 trenes)	126	392	5
Composición (4 trenes)	1134	5828	7
Composición (5 trenes)	4764	27850	8

Tabla 5.1: Dimensiones del caso de estudio de *Train-Gate-Controller*.

para la finalización de los procesos.

5.3. Pruebas y resultados

A continuación presentaremos las pruebas efectuadas, y realizaremos un análisis de los resultados obtenidos por medio de las mismas. En este análisis nos ocuparemos además de diferenciar aquellos casos en que las corridas se realizaron implementando el repositorio de *visitados* mediante un conjunto de *e*CDDs de aquellos casos en los que se implementó por medio de un único *e*CDD representando la unión.

Esta última diferencia no es nada menor: en el primer caso, la cantidad de estados generados y explorados es exactamente igual a aquella que se obtiene mediante la versión anterior implementada sobre RDBMs. Esto es así debido a que en estos casos los *e*CDDs actúan de manera similar a las mismas, existiendo una transformación conceptual directa que nos permite representar las zonas de una u otra manera. Sin embargo, veremos que aún así se obtiene una importante ganancia en varios de los casos, debido a las características que le permiten ahorrar tiempo en canonizaciones, como notáramos en la observación 3.4.2. Luego, en la última parte del capítulo, presentaremos por separado algunos resultados obtenidos utilizando la versión unificada del repositorio de *visitados*.

Además, como punto de partida para cotejar los resultados, se implementaron en la versión original sobre RDBMs las optimizaciones de ajuste de cotas, reducción de relojes y reducción de cola de estados por explorar, a fin de poder medir todos los resultados con la misma vara.

Finalmente, contrastaremos también los resultados obtenidos al utilizar la optimización de reducción de relojes contra aquellos donde no se ha utilizado. No se ha realizado el mismo experimento con las otras optimizaciones ya que ambas eran más difíciles de aislar dentro de la herramienta.

Todas las pruebas se realizaron en un Pentium 4 HyperThreaded, de 3 GHz, con 2 GB de memoria RAM.

5.3.1. *Train-Gate-Controller*

Como primer paso, exhibiremos los resultados obtenidos por medio de las RDBMs y *e*CDDs en los casos en que no hemos utilizado la optimización de reducción de relojes. Los resultados acerca del tiempo y memoria consumidos se encuentran en la tabla 5.2. Es necesario notar que las indicaciones "(V)" y "(F)" presentes en estos casos denotan cuándo se han utilizado observadores cuya propiedad es verdadera y cuándo falsa respectivamente.

Tanto en el caso de las ejecuciones con RDBMs como con *e*CDDs, omitimos verificar los modelos instanciados con 5 trenes por practicidad, ya que hubiese tomado excesivo tiempo y consumo de memoria.

Como puede deducirse de la comparación de los tiempos obtenidos para ambas versiones, la versión implementada con *e*CDDs resulta alrededor de un 50 % más veloz, sin que esa diferencia se traslade significativamente al uso de memoria. Es decir, si bien el uso de memoria se incrementa lo hace en un factor mucho menor a dicho 50 % (el incremento es de 37 % en el caso donde el mismo es máximo).

[1] Aproximadamente 118 minutos.

	RDBM		*e*CDD			
Caso	Tiempo (s)	Mem. (KB)	Tiempo (s)	Mem. (KB)	Dif. tiempo	Dif. mem.
2 trenes (V)	0,10	≈ 0	0,10	≈ 0	0 %	0 %
2 trenes (F)	0,31	1784	0,20	1824	−35,48 %	+2,24 %
4 trenes (V)	30,20	5628	14,19	6448	−53,01 %	+14,57 %
4 trenes (F)	7081,36[1]	35996	3685,20	49260	−47,96 %	+36,85 %

Tabla 5.2: Resultados para el caso TGC utilizando RDBMs y *e*CDDs, sin reducir relojes.

Por ejemplo, en el caso de mayor tamaño (cuatro trenes con propiedad falsa), la versión implementada con *e*CDDs resulta un 47,96 % más rápida que la implementada con DBMs y, si bien se encuentra un incremento en memoria, el mismo es de tan sólo un 36,85 %. Un poco más dramático aún es el resultado en el caso de que la propiedad fuese verdadera (con la misma cantidad de trenes), donde el incremento en velocidad es de un 53 %, con un incremento de memoria de tan sólo 14,57 %.

Estas diferencias cobran sentido al analizar en detalle los resultados. Estos detalles están presentados en las tablas 5.3, donde se manifiestan las magnitudes comunes a ambos casos. Las últimas dos columnas de estas tablas merecen una explicación más detallada. La primera de ellas denota la cantidad de chequeos de inclusión que se ejecutaron (ver línea 12 del algoritmo 1.3.1), constando entre paréntesis cuántas de ellas arrojaron resultado positivo. La última columna conlleva el mismo sentido, pero estas inclusiones se refieren a aquellas ejecutadas como medio de optimización para reducir la cola de estados por explorar. A su vez, en la tabla 5.4 nos concentramos en detallar la cantidad de operaciones de obtención de formas canónicas efectuadas en cada caso, además del porcentaje de reducción obtenido en el caso de la ejecución realizada mediante *e*CDDs.

Caso	Locaciones	Zonas	Inclusiones (exitosas)	Inversas (exitosas)
2 trenes (V)	36	218	1288 (91)	462 (13)
2 trenes (F)	45	510	7154 (383)	1147 (39)
4 trenes (V)	188	14929	1496705 (6797)	1080155 (2833)
4 trenes (F)	273	8274	804969853 (560208)	132812463 (48636)

Tabla 5.3: Detalle del caso TGC utilizando RDBMs y *e*CDDs, sin reducir relojes.

Caso	Canon. RDBM	Canon. *e*CDD	Variación (%)
2 trenes (V)	1735	1133	−34,70 %
2 trenes (F)	4847	3364	−30,60 %
4 trenes (V)	123284	85915	−30,31 %
4 trenes (F)	3718055	2970139	−20,12 %

Tabla 5.4: Detalle de canonizaciones efectuadas en los casos de estudio de TGC, sin minimizar relojes.

Como habíamos anticipado, las magnitudes correspondientes a la exploración (locaciones y zonas exploradas) son idénticas en ambos casos, al igual que los chequeos de inclusión. Sin embargo, la fuerte mejora se debe a la diferencia de canonizaciones efectuadas. Como se explicó en el capítulo 1, esta operación es costosa; el tiempo de ejecución de este algoritmo está en $O(n^3)$ donde n es la cantidad de relojes del sistema. Las reducciones en este sentido son las que han contribuido a la ganancia en tiempo mencionada anteriormente.

A continuación presentamos los resultados correspondientes a los mismos casos de estudio, pero activando la optimización de reducción de relojes. Las diferencias obtenidas entre la versión implementada con RDBMs y aquella con *e*CDDs son mucho más ajustadas en estos casos. Sin embargo, las ganancias comparadas con lo obtenido sin activar la optimización de los relojes es inmensa; tanto así que en estos casos sí verificamos los modelos correspondientes a la instanciación con 5 trenes. Todos estos resultados se encuentran exhibidos en las tablas 5.5 a 5.7.

Claramente, el uso de la optimización de reducción de relojes es extremadamente ventajoso. Por una

	RDBM		*e*CDD			
Caso	Tiempo (s)	Mem. (KB)	Tiempo (s)	Mem. (KB)	Dif. tiempo	Dif. mem.
2 trenes (V)	0,10	1464	0,10	1328	0,00 %	−9,29 %
2 trenes (F)	0,10	1324	0,10	1328	0,00 %	+0,30 %
4 trenes (V)	2,65	2564	2,45	2728	−7,55 %	+6,40 %
4 trenes (F)	8,26	3204	6,23	3368	−24,58 %	+4,87 %
5 trenes (V)	91,00	9212	91,52	12356	+0,57 %	+34,13 %
5 trenes (F)	1859,23	34580	1608,98	41296	−13,46 %	+19,42 %

Tabla 5.5: Resultados para el caso TGC utilizando RDBMs y *e*CDDs, reduciendo relojes inactivos.

Caso	Locaciones	Zonas	Inclusiones (exitosas)	Inversas (exitosas)
2 trenes (V)	36	70	113 (24)	63 (4)
2 trenes (F)	45	103	248 (50)	109 (9)
4 trenes (V)	188	3875	218814 (3104)	182605 (869)
4 trenes (F)	273	8274	619148 (8225)	457831 (2070)
5 trenes (V)	556	41776	9016053 (34858)	8532948 (11019)
5 trenes (F)	1617	281547	263366258 (383365)	78723108 (83880)

Tabla 5.6: Detalle del caso TGC utilizando RDBMs, reduciendo relojes inactivos.

Caso	Canon. RDBM	Canon. *e*CDD	Variación
2 trenes (V)	661	345	−47,81 %
2 trenes (F)	1108	598	−46,03 %
4 trenes (V)	39742	24742	−50,43 %
4 trenes (F)	101684	64714	−36,36 %
5 trenes (V)	432294	265361	−38,62 %
5 trenes (F)	3784223	2514074	−33,56 %

Tabla 5.7: Detalle de canonizaciones efectuadas en los casos de estudio de TGC, minimizando relojes.

parte, no incide en un incremento notorio del consumo de memoria. Por otra, los tiempos obtenidos al ejecutar los casos de estudio sin la optimización resultan ser mucho mayores. En particular, en el caso del chequeo del sistema consistente de cuatro trenes, con propiedad falsa, al utilizar la optimización de relojes se obtiene, en la versión implementada sobre *e*CDDs, el proceso es casi 600 veces más rápido y, además, resulta 15 veces más eficiente en términos de memoria utilizada. Por otra parte, las comparaciones entre las versiones implementadas con RDBMs y *e*CDD en los casos optimizados son, como anticipamos, más ajustadas: en los casos más pequeños no hay diferencias apreciables, mientras que en el mayor de los casos (cinco trenes con propiedad falsa), el incremento de velocidad en la versión con *e*CDDs respecto de aquella con RDBMs es de sólo $13,46\,\%$, con un incremento de memoria de un $19,42\,\%$, transformando el caso en un *trade-off*. Es muy particular el caso de cinco trenes con propiedad falsa, por ser el único en el cual observamos mejor tiempo de procesamiento para la versión con RDBMs, a pesar de que esta diferencia es ínfima.

5.3.2. Pipeline

A continuación presentamos los resultados obtenidos para los casos de estudio del *pipeline*. Los análisis son equivalentes a los ya presentados en la sección anterior; por lo tanto, nos limitaremos a exhibir los resultados sin ulteriores análisis. Además, dado que la cantidad de casos es menor a la de aquellos de TGC, los presentaremos condensados en menos tablas. En estos casos de estudio, además de mostrar resultados utilizando la optimización de reducción de relojes, se redujo también el modelo utilizando la herramienta ObsSlice, descripta en [BGO04]. Dentro de la tabla, notaremos "OS" cuando se haya utilizado dicha técnica. Anotaremos también con "min" en los casos en que se haya aplicado la técnica de minimización de relojes.

	RDBM		*e*CDD			
Caso	Tiempo (s)	Mem. (KB)	Tiempo (s)	Mem. (KB)	Dif. tiempo	Dif. mem.
Pipe6 OS (F)	58,30	13360	40,61	11044	−30,34 %	−17,34 %
Pipe6 OS (T)	172,05	18588	147,22	26668	−14,43 %	+43,47 %
Pipe6 (F)	250,57	35220	160,59	29420	−35,91 %	−16,47 %

Tabla 5.8: Resultados para el caso pipeline utilizando RDBMs y *e*CDDs.

Caso	Locs.	Zonas	Inclusiones (exitosas)	Inversas (exitosas)
Pipe6 OS (F)	1822	24581	550351 (36878)	438073 (9053)
Pipe6 OS (T)	1724	59940	4831702 (91413)	2764671 (19293)
Pipe6 (F)	4552	80280	2487603 (104627)	2289587 (29117)

Tabla 5.9: Detalle del caso pipeline utilizando RDBMs y *e*CDDs.

Caso	Canon. RDBM	Canon. *e*CDD	Variación
Pipe6 OS (F)	349239	214286	−38,64 %
Pipe6 OS (T)	781167	517246	−33,79 %
Pipe6 (F)	1168801	674672	−42,28 %

Tabla 5.10: Detalle de canonizaciones efectuadas en los casos de estudio de Pipe.

5.3.3. Utilizando *e*CDDs como regiones

Como vimos en las secciones anteriores, el sólo uso de los *e*CDDs como reemplazo de las RDBMs, una a una, ofrece resultados que van de buenos a muy buenos (entre un 10 % y un 50 %). Sin embargo, habíamos explorado inicialmente la estructura con el objetivo de utilizarla como repositorio completo de los estados visitados, representado una región completa en un único *e*CDD. Esta representación,

como adelantamos anteriormente, persigue el fin de detectar inclusiones de nuevas zonas, aún cuando se encuentren abarcadas dentro de varias otras zonas dentro de los estados visitados. Como notamos al presentar los algoritmos, al mantener una región en una estructura homogénea podríamos realizar esta tarea mucho más eficientemente que mediante el uso de un conjunto de RDBMs.

Mediante la versión del verificador implementada con un único *e*CDD como repositorio, obtuvimos unos pocos resultados que sostienen esta idea. Estos resultados, en la tabla 5.11, se corresponden con casos ya presentados en tablas anteriores; se trata de casos particulares del modelo TGC. En estos casos, notaremos con un número la cantidad de trenes en el sistema y, salvo donde se note lo contrario, se ha aplicado la técnica de reducción de relojes.

Caso	Locs.	Zonas	Canonizaciones	Inclusiones (exitosas)	Inversas (exitosas)
TGC, 2, sin min. (V)	36	218	1133	273 (91)	427 (11)
TGC, 2 (V)	36	68	337	56 (24)	57 (4)
TGC, 2 (F)	45	100	566	106 (51)	91 (8)
TGC, 4 (V)	188	3362	22819	6288 (3114)	93881 (648)

Tabla 5.11: Detalle de los casos medidos utilizando *e*CDDs como repositorio de regiones.

Como puede apreciarse analizando estos resultados, el tamaño del espacio explorado (es decir, la cantidad de locaciones y zonas temporales) efectivamente ha decrecido. Este aumento se hace más notorio cuanto mayor es el caso analizado: en el caso más grande estudiado (el TGC de cuatro trenes), la diferencia en la cantidad de zonas temporales exploradas es de un $13,25\,\%$. Estos resultados son promisorios en cuanto a la aplicación de este enfoque para la resolución de problemas de magnitudes mayores.

Lamentablemente, la implementación actual de las estructuras utiliza demasiado espacio en memoria, ya que durante las uniones se generan numerosas copias de sub-*e*CDDs ya disponibles. Esto impacta de manera muy negativa en la performance de la verificación, tanto en el consumo de memoria como en el tiempo de ejecución, ya que la copia requiere vastas cantidades de ambos. Esta diferencia de tiempos es apreciable en el caso de los cuatro trenes: la verificación de este caso con la versión de repositorios insume cerca de una hora, utilizando más de un gigabyte de memoria. Sin embargo, esto puede llegar a entenderse observando justamente el comportamiento respecto de la cantidad de copias generadas (y el tiempo necesario para su liberación). Estos datos pueden apreciarse en la tabla 5.12, donde se presenta una comparación de la cantidad de copias de sub-*e*CDDs realizadas, tanto en la implementación con un único *e*CDD como repositorio como en aquella que los utiliza puramente en forma de zonas. La misma tabla muestra también información adicional, obtenida por medio del *profiling* de la ejecución, acerca del tiempo insumido por las copias dentro de la ejecución del proceso. Resulta de este *profiling* que en la versión sin repositorio se insume un $2,36\,\%$ del tiempo en copias, mientras que en la versión con repositorios se invierte un $83,72\,\%$ del tiempo en el mismo concepto. Por otra parte, el *profiling* es bastante rústico, ya que no nos permite observar los tiempos generados por el *overhead* resultante de manejar una mayor cantidad de estructuras, por lo que conjeturamos que el tiempo aplicado a copias en la versión que utiliza un único *e*CDD sea probablemente mayor. Estos indicios nos hacen no descartar esta técnica, al menos hasta obtener resultados en los que la cantidad de copias no sea un factor determinante.

	Zonas puras		Repositorio	
Funciones	Llamadas	Tiempo (% total)	Llamadas	Tiempo (% total)
Copias	96146	1,48	1282774645	25,13
Liberaciones	112244	0,09	19049555	14,16
Operaciones de pila	0	0,00	11095328040	11,93
Operaciones bit-a-bit	2741403	0,79	7296619949	32,50
Totales		2,36		83,72

Tabla 5.12: Comparación de los recursos insumidos en el uso de *e*CDDs para verificar el TGC de 4 trenes.

En el próximo capítulo discutiremos un poco más acerca de esta dificultad, proponiendo incluso algunas ideas a desarrollar para eliminar este problema, o al menos minimizar su impacto.

Capítulo 6

Trabajo e investigación futura

En el presente capítulo nos ocuparemos de identificar con mayor precisión los puntos a mejorar en la herramienta y la estructura propuesta, y además presentaremos algunas otras cuestiones que han quedado fuera del alcance de este trabajo.

En primer lugar, al final del capítulo 5 notamos que para lograr una verificación realmente eficiente mediante el uso de regiones representadas como eCDDs es necesario poder realizar estas uniones con mayor rapidez y menor uso de memoria. La razón por la cual esto no sucede ya ha sido identificada: durante el proceso de la unión, se duplican muchísimos sub-eCDDs. Esto resulta extremadamente costoso, no sólo en el consumo de memoria necesario para mantener todas estas copias, sino también por el tiempo de ejecución que insume recorrer esta estructura arbórea. De todas maneras, la solución es conceptualmente simple: si en vez de generar copias de los sub-eCDDs se mantuviesen referencias a los mismos, esta "duplicación" no tomaría tiempo adicional. Este punto, que ya había sido identificado durante los capítulos introductorios, resultó crucial.

Sin embargo, la implementación de esta solución no es tan simple. Lamentablemente no alcanza con sólo mantener referencias cuando se necesita una copia—esto sería suficiente si la estructura se mantuviese estática, pero no es el caso. El mayor problema es que el repositorio se encuentra en constante cambio: cada vez que se une una nueva zona a la región, el repositorio es susceptible de ser modificado completamente. En este caso, las referencias deben ser tratadas con sumo cuidado, ya que seguramente durante este proceso habrá nodos cuyos intervalos sean modificados. De ser este el caso, y de estar además este nodo referenciado desde otros n nodos, este nodo deberá ser replicado al menos una vez (una para mantener las viejas referencias; otra para ser efectivamente modificado). Lo mismo sucede con el borrado de los nodos—hay que ser cuidadoso de mantener la integridad del resto de la estructura. Estos factores, sumados a lo intrincado de la estructura al carecer en gran parte de punteros y concebirse sobre mapas de bits, hacen la cuestión bastante compleja. Por estas razones, este trabajo será realizado a futuro.

Como segundo trabajo a realizar, identificamos una nueva optimización que puede realizarse para reducir la cantidad de estados a explorar. Así como intentamos reducirla por medio de los chequeos de inclusiones de estados de la cola por explorar dentro de los nuevos estados generados, podría chequearse también lo inverso. Es decir, si se genera un nuevo estado $< s, z >$ tal que la zona z se encuentra comprendida en la unión de las zonas correspondientes a la locación s en la cola por explorar, es seguro que los sucesores de z serán analizados eventualmente, sin necesidad de incluir a $< s, z >$ dentro de la cola. Claramente, para poder resolver esto sería necesario almacenar la cola de estados por explorar de manera similar a los estados visitados, es decir, con un nuevo repositorio sobre un eCDD. Esto conlleva dos factores a solucionar: en primer lugar, que no es factible implementar esta nueva optimización sin antes solucionar el problema de la duplicación de sub-eCDDs; y en segundo lugar que la extracción de zonas desde una región implementada por un eCDD no es un tema estudiado aún. En principio, las zonas podrían estar muy atomizadas dentro del eCDD, haciendo su extracción poco práctica—convendría en esos casos duplicar la región y el conjunto de zonas a explorar.

Aún a más bajo nivel en la estructura, si bien se notó la necesidad de poseer un orden entre las diferencias de relojes, ésta se define de manera totalmente arbitraria. Sin embargo, al presentar los BDDs notamos que el orden en que se presentan las variables dentro de los mismos impacta muy fuertemente en el tamaño final de la estructura. Resta entonces investigar acerca de las posibilidades de generar distintos órdenes para las diferencias de relojes, en función de alguna predicción que pueda realizarse

acerca del tamaño que finalmente alcanzará la estructura, de manera estática sobre el modelo. Este orden seguramente dependerá de cada caso en particular, por lo que una estrategia general muy probablemente no rendiría frutos.

Finalmente, y en otro orden de cosas, no olvidemos que Zeus surge como una herramienta *distribuida*. Esta es, entonces, otra rama a investigar: la posibilidad (o imposibilidad) de distribuir las zonas y regiones implementadas sobre *e*CDDs en los distintos nodos de cómputo es un tema del cual no hemos llegado ni a rozar la superficie.

Capítulo 7

Conclusiones

El presente capítulo cierra esta tesis. En él, nos dedicaremos a destacar los que creemos son los puntos salientes del trabajo.

Ante todo, pudimos concretar una implementación de una estructura de datos basada en decisiones para representar las zonas y regiones temporales. Logramos suplir la falta de completitud de definiciones anteriores, además de detectar posibles orígenes de conflictos o reducciones de eficiencia. Además, logramos realizar el proceso de verificación completo utilizando exclusivamente esta nueva estructura, sin depender en ningún momento de aquella que queríamos evitar, las DBMs o RDBMs.

Aplicando esta nueva implementación de la misma forma que se venía haciendo con las DBMs, es decir, almacenando conjuntos de zonas para representar los estados visitados, alcanzamos ganancias en tiempo de hasta un 50 % comparado con los resultados obtenidos mediante RDBMs. Esta ganancia, además, no es opacada por un uso mucho mayor de memoria. Adicionalmente implementamos otras optimizaciones ajenas a la estructura, obteniendo también importantes beneficios respecto del tiempo de ejecución. Para comprender mejor estos resultados, realizamos análisis de los mismos, encontrando las causas detrás de ellos.

Paralelamente, desarrollamos una versión del verificador representando el conjunto de estados visitados mediante sus regiones temporales de manera homogénea. Esta posibilidad existe gracias a esta nueva estructura. Los resultados obtenidos muestran también una reducción de los espacios a explorar, lo cual debería redundar como en los demás casos en una reducción del tiempo de ejecución. Lamentablemente, la herramienta en su estado actual sufre de algunas limitaciones técnicas—que hemos identificado y sobre las cuales continuaremos trabajando—que causan que esta reducción no se vea aprovechada aún.

Finalmente, abrimos las puertas para futuros desarrollos mediante esta estructura, en especial aquellos relacionados con obtener una herramienta paralela de verificación.

Bibliografía

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[Alu92] Rajeev Alur. *Techniques for automatic verification of real-time systems*. PhD thesis, Stanford University, Stanford, CA, USA, 1992.

[AMY03] Eugene Asarin, Oded Maler, and Sergio Yovine, editors. *Workshop on Theory and Practice of Timed Systems (TPTS-ETAPS)*, volume 65 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.

[Bal96] F. Balarin. Approximate reachability analysis of timed automata. In *RTSS '96: Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96)*, page 52, Washington, DC, USA, 1996. IEEE Computer Society.

[BBD+02] Gerd Behrmann, Johan Bengtsson, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL implementation secrets. In *FTRTFT '02: Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 3–22, London, UK, 2002. Springer-Verlag.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. J. Hu and M. Y. Vardi, editors, *Proc. 10th International Conference on Computer Aided Verification, Vancouver, Canada*, volume 1427, pages 546–550. Springer-Verlag, 1998.

[Beh03] Gerd Behrmann. *Data Structures and Algorithms for the Analysis of Real Time Systems*. PhD thesis, Aalborg University, November 2003.

[BGO04] Víctor A. Braberman, Diego Garbervetsky, and Alfredo Olivero. Obsslice: A timed automata slicer based on observers. In *CAV*, pages 470–474, 2004.

[BLL+96] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL – a tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid Systems III: Verification and Control*, pages 232–243, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[BLN03] D. Beyer, C. Lewerentz, and A.Ñoack. Rabbit: A tool for BDD-based verification of real-time systems. In *Proc. CAV*, LNCS 2725, pages 122–125. Springer, 2003.

[BLP+99] Gerd Behrmann, Kim Guldstrand Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using Clock Difference Diagrams. In *Computer Aided Verification*, pages 341–353, 1999.

[BPO03] Victor Braberman, Carlos G. Lopez Pombo, and Alfredo Olivero. On improving backwards verification of timed automata. In Asarin et al. [AMY03].

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[BTY97] A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *RTSS '97: Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, page 25, Washington, DC, USA, 1997. IEEE Computer Society.

[BY] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools.

[Daw98] Conrado Daws. Optikron: A tool suite for enhancing model-checking of real-time systems. In *Computer Aided Verification*, pages 542–545, 1998.

[Dil90] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 197–212, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 208–219, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS '98: Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 313–329, London, UK, 1998. Springer-Verlag.

[DWT95] David L. Dill and Howard Wong-Toi. Verification of real-time systems by successive over and under approximation. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 409–422, London, UK, 1995. Springer-Verlag.

[EMA+97] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, M. Pnueli, and A. Rasse. Data structures for the verification of timed automata. In O. Maler, editor, *Hybrid and Real-Time Systems*, pages 346–360, Grenoble, France, 1997. Springer Verlag, LNCS 1201.

[HHWT95] Thomas A. Henzinger, Pei Ho, and Howard Wong-Toi. A user guide to HyTech. Technical report, Cornell University, Ithaca, NY, USA, 1995.

[Hol97] Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[KL98] K. Strehl and L. Thiele. Symbolic model checking using Interval Diagram techniques. Technical report, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), 1998.

[Oli94] Alfredo Olivero. *Modélisation et analyse de systèmes temporisés et hybrides.* PhD thesis, Institute National Polytechnique de Grenoble, September 1994.

[PSS00] P. Niebert, S. Tripakis, and S. Yovine. Minimum-time reachability for timed automata. In *MED 2000: Proceedings of the 8th IEEE Mediterranean Conference on Control and Automation*, Patros, Greece, 2000. IEEE Computer Society.

[REC+93] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and Their Applications. In *IEEE /ACM International Conference on CAD*, pages 188–191, Santa Clara, California, 1993. IEEE Computer Society Press.

[Sch02] Fernando P. Schapachnik. *Verificación distribuida y paralela de sistemas de tiempo real.* Tesis de licenciatura, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina, Mayo 2002.

[Tri98] Stavros Tripakis. *L'Analyse Formelle des Systèmes Temporisés en Pratique.* PhD thesis, Université Joseph Fourier, Grenoble, France, December 1998.

[TXJS92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *7th. Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[Wan00] Farn Wang. Efficient data structure for fully symbolic verification of real-time software systems. In *TACAS '00: Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 157–171, London, UK, 2000. Springer-Verlag.

[Wan03] Farn Wang. Efficient verification of timed automata with BDD-like data-structures. In *VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 189–205, London, UK, 2003. Springer-Verlag.

[Yov96] Sergio Yovine. Model checking timed automata. In *European Educational Forum: School on Embedded Systems*, pages 114–152, 1996.