Verificación de Sistemas Temporizados utilizando Unfoldings

Pablo Andrés Rubinstein Jorge Alberto Suppicich

Directores: Dr. Victor Adrián BRABERMAN
Dr. Alfredo OLIVERO

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

15 de Marzo de 2002

Resumen

La construcción de sistemas temporizados libres de fallas es un objetivo perseguido por muchas actividades industriales debido al alto costo monetario y en vidas humanas que pueden provocar los desperfectos en los sistemas que éstas utilizan.

A partir de esta necesidad, el área de verificación formal desarrolló la técnica de *model checking*, cuyo objetivo principal es el de constatar, sobre un modelo del sistema a construir, si se verifican sobre este último aquellas propiedades que se desean que satisfaga el sistema original.

Existen varias técnicas de model checking, cada una de las cuales aprovecha características específicas tanto de la propiedad a verificar como del formalismo utilizado para expresar el modelo. Una de estas técnicas es la de *unfolding* que propone tomar el modelo a verificar e ir 'desdoblándolo' con el objetivo de eliminar el comportamiento cíclico del sistema al mismo tiempo que se evita la generación de todas las posibles combinaciones de evoluciones equivalentes (interleavings). El resultado de este proceso es un grafo donde se encuentran representadas (de manera implícita) todas las posibles evoluciones del sistema a partir de su estado inicial.

Si bien esta técnica fue originalmente propuesta para trabajar con modelos descriptos como Redes de Petri, trabajos posteriores fueron extendiendo la gama de formalismos: Redes de Petri Temporizadas, Álgebras de Procesos, Sistemas de Transiciones Etiquetadas, etc. Es llamativa entonces, la ausencia de trabajos que adapten la técnica de unfoldings sobre Autómatas Temporizados, ya que estos constituyen un formalismo ampliamente difundido para el modelado de sistemas de sistemas temporizados. El mismo es utilizado como entrada para herramientas de model checking como Kronos, UPPAAL y HyTech.

En el presente trabajo se presenta una propuesta para la aplicación de la técnica de unfolding sobre Autómatas Temporizados, de esta manera extendiendo la misma a un formalismo sobre el cual no se había aplicado anteriormente. Para esto se enriquece el marco formal existente de manera tal de poder desarrollar un algoritmo que verifique propiedades de alcanzabilidad sobre sistemas modelados como composición de Autómatas Temporizados. Adicionalmente se implementa el prototipo de una herramienta que permite aplicar el método a casos de la literatura, corroborando la viabilidad de la propuesta.

Índice General

Capítulo 1

Introducción

1.1 Verificación de Modelos

Los sistemas concurrentes de tiempo real están presentes en una cantidad creciente de ámbitos de aplicación incluyendo a la industria electrónica, control industrial y la ingeniería aeronáutica, entre otras. En la mayoría de los casos estos sistemas (ya sean elementos de hardware o software) son considerados *críticos*, ya que un defecto en los mismos puede tener un impacto directo sobre la pérdida de vidas humanas o bienes materiales. Por este motivo, los diseñadores de estos sistemas están interesados en obtener un producto final que se encuentre libre errores. El inconveniente que deben superar es el hecho de que la construcción de sistemas libres de fallas es una tarea de una magnitud considerable.

La intuición nos dice que a medida que los sistemas van creciendo en complejidad, más difícil se torna la detección de sus fallas. En la práctica, los sistemas a los que nos referimos están compuestos por una gran cantidad de subsistemas que interactúan en forma conjunta. Esta interacción incrementa notablemente la complejidad del proceso de análisis.

Un enfoque utilizado para sobreponer esta dificultad es el de la *verificación de modelos* o *model checking* (ver [CGP99], por ejemplo). Esta metodología permite utilizar un modelo del sistema para realizar la búsqueda de fallas, posibilitando de esta forma la detección exhaustiva de errores en forma automática. El uso de herramientas de model checking en las etapas iniciales del desarrollo de un sistema por lo general reduce los costos asociados a la presencia de errores.

En particular nos concentraremos en sistemas *control-intensive* donde el interés está puesto sobre la ocurrencia de eventos y en el tiempo en que éstos ocurren, y no en la transformación de datos.

1.1.1 Formalismos

Es importante hacer notar que el proceso de verificación no es trivial. Como primera medida debemos contar con una estructura formal que nos permita definir modelos, expresar las propiedades que son de nuestro interés, y finalmente, determinar la satisfactibilidad de las mismas. Nos referiremos a estas estructuras como *formalismos*. Como ejemplos de formalismos para el modelado de sistemas *control-intensive* podemos citar a las Máquinas de Estados[AHU79], Sistemas de Transiciones Etiquetadas[Arn92], Redes de Petri[Pet77] y Álgebras de Procesos como CCS[Mil89] y CSP[BHR84], entre otros.

Uno de los formalismos más simples y representativos lo constituyen las *máquinas de estados* o *autómatas*. Una máquina de estados puede verse como un grafo, donde los nodos, llamados *locaciones*, representan los estados posibles del sistema. Las locaciones se encuentran vinculadas mediante *transiciones*. Existe una locación distinguida, llamada *locación inicial*, que representa el estado inicial del sistema. Cada cambio en el sistema se representa en la máquina de estados como un cambio de locación a través de la transición correspondiente.

A continuación presentaremos un ejemplo de un sistema modelado con una máquina de estados. El mismo consiste en un tren que se desplaza por una vía circular. Existe una barrera que cruza esa vía, y es de interés conocer la posición del tren respecto a la barrera: si está lejos, si se está acercando, o si está pasando por el cruce.

Figura 1.1: Una Máquina de Estados que modela un tren

La figura 1.1 muestra la máquina de estados que modela un tren sobre una vía circular. La misma tiene 3 locaciones: $Lejos$, $Cerca$, $Cruza$, de las cuales $Lejos$ es la locación inicial. Las tres transiciones permiten la evolución

del sistema en ciclos $Lejos \rightarrow Cerca \rightarrow Cruza \rightarrow Lejos...$

Concurrencia

Algunos formalismos permiten modelar un sistema en función de partes o componentes. De esta forma el comportamiento del sistema original se obtiene como resultado de la interacción de las partes que lo componen. Esto permite modelar la *concurrencia* de los sistemas, ya que se permite que dos partes de un sistema evolucionen en forma simultánea. En algunos casos, para modelar correctamente el comportamiento del sistema, es necesario sincronizar la evolución de algunas de sus componentes. En consecuencia, los formalismos que presenten un enfoque composicional deben definir cuál es el mecanismo de sincronización que utilizan.

Una forma de sincronizar las Máquinas de Estados consiste en asignar etiquetas a las transiciones. De esta forma, para que una componente pueda ejecutar una transición, toda otra componente que tenga alguna transición con la misma etiqueta estará obligada a ejecutarla al mismo tiempo.

Figura 1.2: Máquinas de Estados que modelan un cruce ferroviario

Consideremos nuevamente el ejemplo del tren. Consideremos ahora que se desea agregar al sistema el comportamiento de la barrera que controla el cruce ferroviario. La barrera tiene un sensor que detecta la proximidad del tren, lo que le permite descender cuando el mismo se está acercando al cruce y liberar el paso una vez éste ha pasado por completo. Para modelar este comportamiento se agrega al sistema de la figura 1.1 una nueva componente, y adicionalmente se etiquetan las transiciones para agregar la sincronización entre las componentes.

La figura 1.2 muestra al cruce ferroviario modelado como dos Máquinas de Estados. Como se puede apreciar en la figura ambas componentes sincronizan a través de las transiciones *aproximandose* y *alejandose*. Veamos con un ejemplo las consecuencias de esta sincronización: supongamos que la barrera se encuentra en la locación *Subiendo* y el tren se encuentra en la locación *Lejos*. En este escenario no es posible tomar la transición que lleva al tren de *Lejos* a *Cerca* hasta que la barrera haya evolucionado de *Subiendo* a *Alta*, ya que la sincronización entre las dos componentes nos obliga a tomar las transiciones etiquetadas con *aproximandose* al mismo tiempo.

Interleavings

Dado un modelo, puede suceder que existan eventos que son independientes entre sí. Es decir, eventos que pueden ser tomados por el sistema en cualquier orden evolucionando en todos los casos al mismo estado. Esta posibilidad de ir combinando la ejecución de los eventos es conocida como la *propiedad del diamante* (figura 1.3(b)). La literatura se refiere a estas ejecuciones, formadas por los mismos eventos independientes tomados en distinto orden, como *interleavings* del sistema.

Figura 1.3: Dos Máquinas de Estados (a) y su composición en paralelo (b). Los eventos a y b son independientes entre sí, por lo que luego de la ejecución de ambos el sistema alcanza el estado $(1, B)$.

Es importante hacer notar que los interleavings, por más de tener el mismo efecto sobre las evoluciones del sistema, no son iguales entre sí, ya que el orden en que se toman los eventos los diferencia.

Existen contextos en los que distinguir los interleavings es fundamental para considerar todas las posibles evoluciones del sistema, mientras que en otros es posible relajar esta diferencia y considerar a todos los interleavings

con el mismo resultado final como una única evolución. (Esto será tratado con más detalle en secciones posteriores). El enfoque tradicional de la verificación de modelos utiliza la primera de las dos semánticas mencionadas.

1.1.2 Propiedades de Alcanzabilidad

Una familia de propiedades muy simple, pero a la vez de gran poder expresivo, lo constituyen las *propiedades de alcanzabilidad*. Una propiedad de alcanzabilidad puede verse como una pregunta de la forma *"¿existe alguna forma, partiendo del estado inicial del sistema, de que el mismo evolucione a un cierto estado X?"*. Resulta obvio que para poder plantear una propiedad de este tipo, el formalismo elegido debe permitirnos (además de poder expresar el modelo y la propiedad como ya habíamos mencionado) indicar de qué forma está representado el estado inicial del sistema en el modelo, cómo se van realizando las evoluciones, y cuál es la relación que existe entre un estado del sistema y el modelo.

Las propiedades de alcanzabilidad pueden ser utilizadas para la verificación de otros tipos de propiedades mediante una estrategia que involucra la incorporación de una nueva componente *observadora* o *monitora virtual* al modelo. De esta forma es posible reducir una propiedad a verificar como una propiedad de alcanzabilidad sobre un estado de la nueva componente.

Consideremos nuevamente la máquina de estados que modela el cruce ferroviario (figura 1.2). Supongamos que se quisiera verificar que en ninguna circunstancia sucede que una vez que la barrera ha comenzado a descender inmediatamente comienza a ascender sin haber llegado a estar completamente baja. Para esto agregamos una nueva componente observadora tal como se muestra la figura 1.4.

La componente observadora tiene tres locaciones: OK_1, OK_2 y *Error*. *Error* indica que la barrera comenzó a ascender sin haber estado completamente baja. OK_2 indica que la barrera está descendiendo. OK_1 indica que la barrera no se encuentra en ninguno de los estados anteriores.

En forma sencilla se puede apreciar que el sistema verifica la propiedad mencionada si y sólo si verifica la propiedad de alcanzabilidad *"no es posible llegar a la locación Error"*.

Este esquema de verificación es atractivo por su sencillez y porque puede ser fácilmente incorporado a casi cualquier formalismo. Lo único que requiere es el agregado de una nueva componente que no debe alterar el comportamiento del modelo original respecto a la propiedad a verificar. De esta forma, la verificación utilizando observadores tiene la ventaja de ser un enfoque no intrusivo de verificación.

Figura 1.4: Agregado de una componente observadora al cruce ferroviario.

1.1.3 El "Problema de la Explosión de Estados"

Por lo general, los modelos que se requieren para verificar sistemas presentan el inconveniente de tener un tamaño considerable. Es decir, para los que la cantidad de posibles evoluciones del sistema determina un espacio de búsqueda demasiado grande como para evaluar la satisfactibilidad de una propiedad de manera exhaustiva.

Consideremos el autómata de la figura 1.5. El mismo tiene 7 locaciones y las transiciones correspondientes que le permiten evolucionar entre dos locaciones consecutivas. Supongamos ahora que se desea modelar el sistema que se obtiene de componer tres de estos autómatas. El espacio de estados de este sistema (que se obtiene como resultado de la composición de las tres componentes) tiene un tamaño menor o igual a $7 \times 7 \times 7 = 343$ estados. Es decir que determinar exhaustivamente la satisfactibilidad de una propiedad sobre el sistema puede requerir la evaluación de la misma sobre el producto del espacio de estados de cada componente del sistema.

Se puede observar fácilmente que la complejidad del proceso de verifica-

Figura 1.5: Una máquina de estados con 7 locaciones.

ción utilizando este enfoque exhaustivo crece exponencialmente en la cantidad de componentes del sistema.

Esta situación es mencionada en la literatura como el *problema de la explosión de estados* (por ej: [Val98]). De esta forma hemos revelado un serio escollo que deben sortear las herramientas de verificación. Es importante destacar que si bien esta situación se ilustró con un ejemplo sobre máquinas de estados, el problema de la explosión de estados no depende del formalismo utilizado sino que es esencial a la modelización de procesos concurrentes.

El problema de la explosión de estados continúa abierto, en el sentido de que no se ha encontrado una técnica de verificación que pueda ser utilizada en forma general (para todos los tipos de sistemas, para todos los formalismos y para todas las propiedades a verificar). Sin embargo, se han propuesto técnicas que permiten obtener éxito práctico en casos particulares:

- reducciones: Por lo general los modelos contienen información redundante y/o innecesaria. Las técnicas de reducción intentan despojar al modelo de todos esos excedentes, obteniendo un modelo "equivalente" pero con un espacio de búsqueda menor. Se han propuesto reducciones sobre distintas variables del modelo: cantidad de locaciones, cantidad de componentes que son necesarias para verificar una propiedad, etc.

- symbolic model checking[BCM+92]: En vez de representar el espacio de estados en forma explícita se utilizan funciones booleanas para representarlo de manera implícita. La verificación se realiza aplicando métodos iterativos para el cálculo de punto fijo de estas funciones.

- métodos de recorrida selectiva[YS97, Val94, Pel93]: Parte de la observación de que la concurrencia del sistema produce evoluciones que son en algún sentido equivalentes, y que por lo tanto no es necesario considerar todos los posibles ordenes de ejecución de las transiciones independientes (interleavings). La técnica propone identificar un representante para cada conjunto de ejecuciones equivalentes.

- unfoldings[McM92a]: Se basa en la misma observación que los métodos de recorrida (evitan el interleaving de los eventos), pero el enfoque está puesto en la elección de una representación del espacio de estados que no considere todos los interleavings sino que modele la relación causal de los eventos del sistema (*true concurrency*).

1.1.4 Unfoldings

Como mencionamos anteriormente, una de las propuestas para evitar el problema de la explosión de estados es la técnica de unfolding[McM92b, McM92c]. La idea consiste en ir "desdoblando" el modelo para ir generando todas las posibles evoluciones del sistema, definiendo un orden parcial que representa la relación causal entre los eventos[1]. Esto tiene como consecuencia la eliminación del comportamiento cíclico que el modelo pudiese contener. El resultado de este proceso, llamado unfolding del modelo, contiene todas las posibles evoluciones del sistema, por lo que en consecuencia contiene todos los estados alcanzables del mismo.

Utilizaremos el ejemplo del cruce ferroviario con observador (figura 1.4) para ilustrar el proceso de construcción del unfolding de una máquina de estados.

El primer paso consiste en detectar los "estados iniciales" del sistema. Los mismos serán representados por los nodos iniciales del grafo, es decir, el punto de partida de todas las posibles evoluciones del sistema. En el ejemplo hay tres estados iniciales (uno por cada componente) representados en la figura 1.6(a).

Una vez que hemos identificado los estados iniciales calculamos todos los posibles "eventos" que se pueden tomar a partir de los estados que se encuentran en el unfolding (hasta el momento sólo los iniciales). Para nuestro ejemplo hay un único evento posible que es *aproximandose*. Agregamos un nodo que representa a este evento y nodos que representan a las locaciones a las que el sistema evoluciona a través de dicho evento 1.6(b).

En este punto se presentan dos alternativas: tomar $porCruzar$ o $abajo$. Deberemos optar por alguno de ellos y nuevamente agregaremos nodos al

[1] De allí su nombre. unfold = desdoblar, desplegar.

Figura 1.6: Algunos pasos en la construcción del unfolding del Cruce Ferroviario

grafo que representen a la transición elegida y a sus locaciones asociadas. En 1.6(c) se puede observar el unfolding que resulta de tomar la transición *porCruzar* antes de *abajo*.

Este procedimiento de evaluación → selección → representación se repite hasta que no es posible ejecutar ninguna transición. En la figura 1.6 se encuentran representados las etapas iniciales de la construcción del unfolding del ascensor, cuando se han agregado 0, 1, 2, y 4 eventos.

Como hemos mencionado, el proceso se repite hasta que no hay más eventos para tomar. Tal como se plantea en [McM92b], de esta forma el unfolding de un modelo puede ser infinito[2], lo que claramente no representa una ventaja contra la explosión de estados. Afortunadamente McMillan demostró que utilizando una noción de "corte" adecuada, es posible realizar una poda en la construcción del unfolding obteniendo un prefijo finito del mismo que contiene toda su información. Por lo que la verificación de propiedades puede efectuarse sobre este prefijo. Esto constituye una ventaja importante ya que estamos obteniendo un escenario de tamaño significativamente menor a la composición original, sobre el cual chequear las propiedades. De esta forma la gran ventaja de la técnica de unfolding es que podría representar de manera más compacta que otras técnicas las posibles evoluciones del sistema, al evitar los interleavings del mismo.

Otra ventaja es que una vez construido el prefijo éste puede ser reutilizado durante la verificación de varias propiedades. Recordemos que en el mismo se encuentran representadas todas las locaciones alcanzables, por lo que una vez construido puede ser utilizado para la verificación de un conjunto de propiedades sin necesidad de recalcularlo.

Originalmente esta técnica fue propuesta para ser aplicada sobre Redes de Petri. Trabajos posteriores fueron extendiendo la gama de formalismos sobre los cuales se aplicó el método: Álgebras Composicionales[LB99], Productos Sincrónicos de Sistemas de Transiciones Etiquetadas[ER99], Redes de Petri Temporizadas[Lil99], etc.

Este método puede ser utilizado para verificar distintos tipos de propiedades tales como alcanzabilidad, ausencia de deadlock, persistencia, etc. Un área de aplicación importante la constituye la verificación de circuitos asincrónicos, principalmente en la detección de *hazards*[SYK98, SY95].

[2]La potencial no-finitud del unfolding se aprecia en el ejemplo de la figura 1.6. Desde el paso (d) el único evento que puede ser agregado al unfolding es *arriba*, lo que agrega nuevas condiciones Lejos, Alta y Ok-1, volviendo de esta manera a una situación equivalente a la del paso (a). Como este procedimiento puede repetirse todas las veces que se desee el unfolding que se genera para este ejemplo es infinito.

1.1.5 Sistemas Temporizados

Los formalismos presentados hasta el momento están orientados a modelar principalmente los eventos del sistema, haciendo hincapié en el orden causal entre los mismos y en su concurrencia. Esto no es suficiente para modelar en forma correcta a aquellos sistemas de control para los que el tiempo cumple un rol fundamental. Por ejemplo, en el caso de un equipo de radioterapia deberían valer restricciones como *"una vez presionado el botón de STOP la fuente cesará de irradiar rayos X en menos de 1 microsegundo"*. Llamaremos *sistemas temporizados* a este tipo de sistemas. Estos sistemas consideran al tiempo como parte constituyente de la noción de estado: dado un estado s de un sistema no temporizado, el mismo puede corresponder a los estados $(s, tiempo_0)$ y $(s, tiempo_1)$ de un sistema temporizado.

La posibilidad de especificar este tipo de comportamiento hace que el uso de sistemas con restricciones temporales sea atractivo para una gran cantidad de actividades industriales, como los sistemas de control de tiempo real, protocolos con timers, circuitos asincrónicos, etc.

El problema de la explosión de estados se ve agravado en un orden de magnitud cuando la verificación se realiza sobre sistemas temporizados. Debido a que la nueva dimensión que se agrega al estado del sistema incrementa el tamaño del espacio de estados posibles, **desencadenando una nueva explosión de estados**. De esta forma, las ventajas de utilizar este tipo de sistemas tiene un costo: el precio que se paga por su utilización es el incremento en la complejidad de los mismos y una mayor dificultad para obtener un sistema libre de fallas.

Por los motivos mencionados, los sistemas temporizados se encuentran entre los casos más difíciles de verificación de modelos.

1.1.6 Autómatas Temporizados

Los autómatas temporizados constituyen uno de los formalismos más difundidos para la modelización de sistemas con restricciones temporales.

Los autómatas temporizados pueden verse como una extensión al formalismo de las máquinas de estados finitos, a las que se les ha incorporado un conjunto de *relojes*, *guardas* en sus transiciones e *invariantes* en sus locaciones para poder modelar las restricciones temporales de los sistemas temporizados.

Un autómata temporizado posee un conjunto finito de relojes. Estos relojes son variables reales que se utilizan para representar el tiempo transcurrido entre la ocurrencia de eventos. Los relojes están sincronizados de manera tal que avanzan todos a la par (en otras palabras un reloj no puede

decidir detener su marcha si otros avanzan). Las transiciones del autómata tienen asociadas *guardas* que son condiciones que se deben cumplir para que el autómata evolucione a través de ellas. Estas guardas se expresan como predicados sobre los relojes del autómata. Adicionalmente las transiciones tienen la capacidad de resetear el valor de los relojes poniéndolos nuevamente en cero. Así como las transiciones tienen guardas que se utilizan para determinar la factibilidad de su activación, las locaciones también poseen predicados (también sobre los relojes del autómata) llamados *invariantes* que indican si es posible permanecer en ellas sin necesidad de activar ninguna transición.

Estas características otorgan un gran poder expresivo al formalismo y le permiten modelar con facilidad una amplia variedad de situaciones.

Figura 1.7: Autómata Temporizado para el Cruce Ferroviario

Veamos un ejemplo de como modelar un sistema con restricciones temporales utilizando autómatas temporizados. Consideremos una vez más el sistema del cruce ferroviario (ver figura 1.4), al que se quiere enriquecer incorporándole información sobre el tiempo que tarda el tren en desplazarse y la barrera en cambiar de posición:

- la barrera tarda no menos de 1 y no más de 2 unidades de tiempo en bajar.
- la barrera tarda no más de 1 unidad de tiempo en subir.
- el tren tarda no menos de 2 unidades de tiempo en recorrer la distancia desde que es detectado por la barrera hasta que se encuentra atravesando el cruce.

La figura 1.7 muestra el autómata temporizado que modela estas nuevas condiciones. Los invariantes de las locaciones, al igual que las guardas de la transiciones, se notan como un predicado sobre un reloj entre llaves. Los resets de los relojes se indican encerrando el nombre del reloj entre llaves.

Los autómatas temporizados son usados como entrada en varias herramientas de verificación de sistemas, como por ejemplo Kronos[DOTY96], UPPAAL[BLL+96] y HyTech[HHWT95].

1.2 Objetivo de la tesis (Nuestra propuesta)

El objetivo del presente trabajo es adaptar la técnica de unfolding para utilizarla sobre autómatas temporizados. De esta forma se extiende la técnica a un formalismo sobre el cual nunca se había aplicado anteriormente.

Nos concentraremos exclusivamente en la verificación de propiedades de alcanzabilidad. Esta decisión está motivada por el hecho de que, como hemos mencionado, las mismas son relativamente sencillas de enunciar y verificar, y a que, si se las utiliza conjuntamente con un esquema de observadores son lo suficientemente poderosas como para tener una importancia práctica real.

De esta manera el proceso de verificación que vamos a utilizar a lo largo del trabajo puede enunciarse de la siguiente manera: para verificar una propiedad de alcanzabilidad p sobre un conjunto de autómatas A, se agrega un autómata observador Obs, y se determina si existe un camino en el prefijo del unfolding de $A + Obs$ que satisfaga $\neg p$.

Para poder alcanzar este objetivo se hace necesario extender el marco formal de la técnica de manera tal de poder desarrollar un algoritmo[3] que verifique propiedades de alcanzabilidad sobre sistemas con restricciones temporales modelados como composición de autómatas temporizados.

Nuestro enfoque permitirá realizar la verificación de propiedades de alcanzabilidad realizando la composición de las componentes del sistema *on the fly*. Adicionalmente, el uso de unfoldings permitirá evitar los interleavings del sistema, con lo que se espera obtener una reducción significativa en el tamaño de estados del sistema.

La adaptación de la técnica a un formalismo temporizado nos plantea algunos desafíos que deben ser superados:

- Asegurar la validez respecto a las restricciones temporales de cada paso en la construcción del unfolding (o del prefijo completo).

[3]Como se verá en secciones posteriores, en realidad es un semi-algoritmo.

- Obtener una nueva definición de corte que tenga en cuenta el aspecto temporal además del causal, que permita realizar una poda adecuada para la obtención de un prefijo completo (noción que no es trivial)
- La incorporación del aspecto temporal al modelo introduce "el problema de la urgencia" (ver Apéndice A). Se hace necesario encontrar una solución al mismo que permita garantizar que las decisiones que toma el algoritmo que construye el unfolding son correctas aún cuando estas se realizan considerando sólo algunas componentes de sistema.

Durante el desarrollo del presente trabajo se presentarán las soluciones encontradas para resolver cada una de estos puntos.

Es importante destacar que nuestro enfoque se diferencia del de herramientas de verificación de autómatas temporizados como Kronos, UPPAAL y HyTech, ya que las mismas consideran los interleavings del sistema y utilizan de una representación simbólica del valor de los relojes.

1.3 Trabajos relacionados

Antes de proseguir con el desarrollo de nuestra propuesta, consideramos necesario incorporar un breve resumen de los enfoques explorados en trabajos anteriores.

Una gran parte de la literatura de unfoldings trabaja con Redes de Petri como formalismo, ya que éste fue el que se utilizó en [McM92a], trabajo que sentó las bases de la técnica. En esta misma línea de trabajo se destaca [ERV96] debido al rigor utilizado al formalizar la propuesta original de McMillan.

La aplicación de la técnica a modelos composicionales casi no ha sido explorado. Entre los pocos trabajos que utilizan un enfoque composicional podemos destacar a [ER99] que realiza unfoldings de Sistemas (no temporizados) de Transiciones Etiquetadas.

Trabajos como [YS97, SB97, BJLY98, VdJL96, Lil99] utilizan la exploración selectiva del espacio de estados (no recorrer todos los interleavings [WG93, Pel93, CGP99]) para formalismos temporizados basados en Redes de Petri, generalmente menos expresivos en los aspectos temporales que los autómatas temporizados. Por ejemplo, [VdJL96] analiza Redes de Petri Temporizadas (Time Petri Nets) con delays asociados en los places. [YS97] también trabaja con Redes de Petri Temporizadas y, al igual que [SB97], adapta el método de stubborn de Valmari al marco temporizado. Es interesante que en el trabajo de [Lil99] la noción de unfoldings no temporizados se utiliza en una parte de su método: el cálculo de loops y "persistent sets". El algoritmo

de [BM98] si bien explora todos los interleavings no temporizados, trata de evitar la generación de demasiados estados simbólicos a partir de cada estado no temporizado. Esto se logra creando una sola región simbólica para cualquier secuencia de eventos que sólo difiera en el orden de los eventos concurrentes. El método se basa en *time event/level structures* que son menos expresivas que los autómatas temporizados.

En [HB94] y [PJKP00] se utilizan estructuras de eventos con información sobre delays para calcular máxima separación entre ellos, adaptando el algoritmo de [MD92]. Ambos trabajos apuntan a circuitos asincrónicos. En particular el trabajo de [PJKP00] utiliza estructura para refinar el espacio de estados no temporizado en un proceso de verificación iterativo. Es decir, a partir de una evolución no temporizada que es testigo de una posible violación de una propiedad se construye una estructura de eventos cuya temporización podría confirmarla o descartarla junto con un conjunto de trazas no factibles. Este proceso se repite con el espacio de estados no temporizado donde se removieron las trazas no factibles.

Los únicos trabajos en verificación basados en unfoldings para formalismos temporizados que tenemos noticias son [SY95] y [BF99]. [SY95] trabaja con un tipo particular de Red de Petri Temporizada que ignora el hecho que las restricciones temporales introducen dependencias sobre las elecciones. El trabajo asume que las elecciones son resueltas por el ambiente, solución que se justifica en el área de aplicación del algoritmo: circuitos asincrónicos. Sin embargo, esta restricción en el problema de la urgencia no permite modelar construcciones importantes de los sistemas de tiempo real tales como los timeouts (ver discusión en [Lil99]). Por otro lado no utiliza, a diferencia de nuestra propuesta y la original de McMillan, el concepto de configuraciones locales; en su lugar trabaja de manera global sobre configuraciones que si poseen un evento también contienen todos los eventos que son anteriores temporalmente. [BF99] extiende un método basado en unfoldings para una lógica temporal simple sobre Redes de Petri Temporizadas. Sin embargo, el método se basa en una transformación a *place transition nets* que expande las restricciones temporales usando eventos "tics" que representan el paso del tiempo en una unidad discreta. Esta conversión no es económica en especial cuando las constantes son grandes. Por otra parte, nuestro método funciona para un dominio temporal denso, no requiere introducir el tiempo como un elemento explícito del modelo y no depende necesariamente en el tamaño de las constantes.

También podemos citar [AL97], un trabajo teórico sobre una estructura para representar la semántica de una Red de Petri Temporizada por intermedio de ordenes parciales. Entre otras diferencias, los procesos de este trabajo como los de [SY95] poseen todos los eventos hasta una determinada cota

temporal, requerimiento que no existe en nuestro enfoque y por lo tanto son mas cercanos a la filosofía del enfoque de McMillan. Es importante recalcar que este trabajo no sienta las bases para un algoritmo de verificación.

1.4 Estructura de la tesis

En el capítulo 2 presentaremos la base teórica que sustenta la técnica de unfolding sobre la composición de autómatas no temporizados siguiendo el enfoque presentado en [ERV96]. Se presentarán además algoritmos para la construcción del unfolding y del prefijo completo del mismo. El capítulo 3 es el núcleo del trabajo ya que en el mismo se realiza el desarrollo de nuestra propuesta. Para esto se extiende el marco formal de la técnica para poder aplicarla sobre autómatas temporizados. Al igual que en el caso no temporizado, se obtienen algoritmos para la construcción del unfolding y el prefijo completo. En el capítulo 4 se presentan las demostraciones que avalan la correctitud de estos últimos algoritmos, y se propone un algoritmo que permite verificar propiedad de alcanzabilidad local utilizando la técnica extendida. La implementación de un prototipo funcional que se llevó a cabo con el propósito de obtener resultados empíricos, junto con el análisis de los mismos se presentan en el capítulo 5. El capítulo 6 cierra el trabajo presentando las conclusiones del mismo y definiendo posibles líneas de trabajo futuro.

Capítulo 2

Unfoldings de Sistemas No Temporizados

En este capítulo presentaremos las definiciones básicas sobre las que se sustentará el desarrollo del presente trabajo. Lo haremos de manera incremental, partiendo de la noción de autómata (no temporizado), pasando por el procedimiento de composición en redes de autómatas, hasta finalmente llegar a un algoritmo que construya el unfolding de este tipo de redes.

Los contenidos del presente capítulo constituyen una adapatación de la literatura de unfoldings sobre sistemas de transiciones etiquetadas, principalmente [ER99], al contexto de los autómatas de transiciones etiquetadas.

2.1 Autómatas de Transiciones Etiquetadas

El primer concepto que utilizaremos es el de *autómata de transiciones etiquetadas*. Un autómata de transiciones etiquetadas es una máquina de estados con la particularidad de que cada una de sus transiciones tiene asignado un nombre o etiqueta. Al igual que cualquier otro autómata, los estados de un autómata de transiciones etiquetadas están representados por las locaciones del mismo, y las transiciones vinculan pares de locaciones, permitiendo pasar de un estado del sistema a otro. De esta forma, partiendo de la locación inicial y tomando las sucesivas transiciones que parten de los estados que vamos recorriendo, es posible hacer evolucionar el sistema modelado por el autómata.

Definición 2.1. (Autómata de Transiciones Etiquetadas). *Un autómata de transiciones etiquetadas (LTA = Labeled Transition Automaton) es una tupla* $\langle Q, T, \lambda, sc, tg, q^{in}, \Sigma \rangle$ *donde*[1]*:*

- *Q es un conjunto de locaciones*
- *T es un conjunto de transiciones*
- *λ es una función sobre T que etiqueta cada transición de T con una palabra del alfabeto Σ*
- *$sc : T \to Q$ es una función que permite obtener el origen de las transiciones*
- *$tg : T \to Q$ es una función que permite obtener el destino de las transiciones*
- *$q^{in} \in Q$ es la locación inicial*

■

Permitiremos que nuestros autómatas contengan un tipo especial de transiciones llamadas *transiciones stutter*. Las mismas tienen la particularidad de que al ser ejecutadas no modifican el estado actual del sistema. La motivación para el uso de este tipo de transiciones se apreciará con claridad durante el desarrollo del trabajo.

Definición 2.2. (Transición Stutter). *Dado $A = \langle Q, T, \lambda, sc, tg, q^{in}, \Sigma \rangle$ un autómata de transiciones etiquetadas, consideraremos que cada locación $q \in Q$ tendrá asociada una transición $t_{\epsilon_q} \in T$. Llamaremos* stutter *a estas transiciones. Las mismas tienen la particularidad que $sc(t_{\epsilon_q}) = q = tg(t_{\epsilon_q})$, es decir que parten y llegan a la misma locación. Estas transiciones estarán etiquetadas con el símbolo ϵ que no pertenece al alfabeto Σ, es decir $\lambda(t_{\epsilon_q}) = \epsilon$. Ninguna otra transición que no sea stutter va a estar etiquetada con este símbolo.* ■

Por comodidad, utilizaremos la forma abreviada t_ϵ cuando la locación a la que está asociada la transición stutter se deduzca del contexto, o bien cuando no nos interese distinguir la locación a la que nos estemos refiriendo. La figura 2.1 muestra un autómata temporizado con tres locaciones. De las cinco transiciones que se encuentran representadas, dos de ellas, t_4 y t_5, son stutter. El autómata posee una transición stutter adicional (t_{ϵ_1}), que no ha sido representada.

Hemos hablado de la forma en que evoluciona nuestro sistema, partiendo de la locación inicial y tomando alguna de las transiciones habilitadas en cada paso. Las *ejecuciones* constituyen la formalización de este concepto.

[1]La literatura presenta varias versiones para la definición de autómata (con o sin etiquetas en las transiciones). Nuestra elección se basa en la decisión de utilizar la etiqueta de las transiciones como mecanismo de sincronización entre autómatas.

Figura 2.1: Automata con transiciones stutter

Definición 2.3. (Ejecución de un Autómata de Transiciones Etiquetadas). *Dado un LTA* $A = \langle Q, T, \lambda, sc, tg, q^{in}, \Sigma \rangle$ *diremos que la secuencia de transiciones* $\sigma = (t_1)(t_2)...(t_n)$ *es una ejecución de A si satisface las siguientes condiciones:*

- $\forall i, 1 \leq i \leq n : t_i \in T$
- $sc(t_1) = q^{in}$
- $\forall i, 1 \leq i < n : tg(t_i) = sc(t_{i+1})$

Para cada ejecución σ *definiremos* $|\sigma|$ *como la longitud de la misma. Asimismo definiremos* σ^j*, con* $1 \leq j \leq |\sigma|$ *como la j-ésimo elemento de* σ*.* ■

La secuencia $t_1.t_2.t_5.t_5.t_3.t_1.t_2.t_3.t_4$ es un ejemplo de una ejecución del autómata de la figura 2.1.

2.2 Redes de Autómatas de Transiciones Etiquetadas

Hasta el momento hemos asumido implícitamente que el autómata de transiciones etiquetadas está en relación uno-a-uno con el sistema que modela. Sin embargo esto no se corresponde con el enfoque composicional que deseamos que nuestra propuesta presente. Desearíamos poder aplicar la técnica que se presenta en este trabajo al conjunto de las componentes que conforman el sistema a modelar, en vez de aplicarla sobre el autómata que se obtiene de

la composición de las mismas. Por este motivo introducimos el concepto de *red de autómatas* que formaliza la noción de un grupo de autómatas que se combinan para modelar un sistema.

Definición 2.4. (Red de Autómatas de Transiciones Etiquetadas). *Una red de autómatas de transiciones etiquetadas (LTAN = Labeled Transition Automata Network) es una tupla* $N = \langle A_1, A_2, ..., A_n \rangle$*, donde cada* A_i *es un LTA* $A_i = \langle Q_i, T_i, \lambda_i, sc_i, tg_i, q_i^{in}, \Sigma_i \rangle$*,* $\forall 1 \leq i \leq n$*.* ■

Figura 2.2: LTAN de dos componentes

Ahora que consideramos una red de autómatas debemos revisar la manera en que se llevan a cabo cada una de las evoluciones posibles de nuestro modelo. Al igual que en el caso anterior, en el que considerábamos un único autómata, partiremos de una locación inicial de la red (que ahora agrupará a las locaciones iniciales de cada una de las componentes). Y antes de ejecutar una transición de una componente deberemos constatar, que para todo otro autómata de la red que contenga alguna transición con la misma etiqueta, al menos una de estas transiciones se encuentre habilitada. Adicionalmente, cada vez que ejecutemos una transición en una componente, para cada una de las demás deberemos hacer lo mismo con una de las transiciones habilitadas que sincronicen por la etiqueta (si es que ésta existe). De esta forma, las redes de autómatas obligan a las componentes a "consultarse" entre sí antes de tomar una decisión.

Para lograr esta comunicación, vamos a utilizar *transiciones globales*, que nos permitan evolucionar la LTAN de una manera análoga a las transiciones locales sobre un LTA. Es decir, el resultado de tomar una transición global

en la LTAN es equivalente al de hacer evolucionar a cada uno de los LTA tomando la transición local mencionada en la transición global. (A lo sumo algunas componentes deberán tomar transiciones stutter, para indicar que no cambian de locación).

Definición 2.5. (Transición global de una LTAN). *Dada una LTAN* $N = \langle A_1, A_2, ..., A_n \rangle$ *diremos que la tupla* $t = \langle t_1, t_2, ..., t_n \rangle$ *es una transición global de* N *si se satisfacen las siguientes condiciones:*

- $\forall i, 1 \leq i \leq n \ : \ t_i \in T_i$
- $\exists i, 1 \leq i \leq n \ : \ \lambda_i(t_i) \neq \epsilon$
- $\exists a \in \Sigma : \forall i, 1 \leq i \leq n :$
  $\lambda_i(t_i) \in \{a, \epsilon\} \wedge [(\lambda_i(t_i) = \epsilon) \Rightarrow (\forall t \in T_i : \lambda_i(t) \neq a)]$

Llamaremos $T^n(N)$ *al conjunto de transiciones globales de* N.
Notaremos $\pi_i(t)$, *con* $t \in T^n(N)$, *a la proyección de la i-ésima componente de* t. ■

Por ejemplo, las transiciones globales de la LTAN de la figura 2.2 son: (t_1, t_4) etiquetada con a, (t_2, t_5) etiquetada con b, (t_2, t_6) etiquetada con b, (t_3, t_5) etiquetada con c y (t_3, t_6) también etiquetada con c.

Vamos a definir una relación de equivalencia entre transiciones globales que no tome en consideración la aparición de transiciones stutter. Valiéndonos de esta equivalencia podremos identificar transiciones globales cuyo resultado observable sobre la evolución del sistema es el mismo. La utilidad de la misma se apreciará en secciones posteriores.

Definición 2.6. (Equivalencia Stutter). *Dada una LTAN* N, *diremos que las transiciones globales* $t_1, t_2 \in T^n(N)$ *son stutter equivalentes, notado* $t_1 \equiv_{stutter} t_2$, *si se satisface la siguiente condición*

$$t_1 \equiv_{stutter} t_2 \ \Leftrightarrow \ \forall i, 1 \leq i \leq n, \ Participates(A_i, t_1) \Rightarrow [Participates(A_i, t_2) \ \wedge \ \pi_i(t_1) = \pi_i(t_2)]$$

■

En el caso en que se tome una transición local que no sincroniza con algunas componentes, estas componentes estarán representadas por transiciones stutter en la transición global asociada. De esta manera queda reflejado el hecho de que esas componentes no tomaron ninguna transición "verdadera".

Esta noción introduce el concepto de autómatas que participan en una transición global, es decir, aquellos que son relevantes para la misma.

Definición 2.7. (Autómatas que participan en una Transición Global). *Dada una transición global t de una LTAN N definiremos el predicado Participates como*

$$Participates(A_i, t) \text{ sii } (\pi_i(t) \neq t_\epsilon)$$

■

Si nos basamos nuevamente en la LTAN definida en la figura 2.2, observamos que el autómata A_1 participa de todas las transiciones globales de la misma, mientras que el autómata A_2 sólo participa de la transición global (t_1, t_4). Además podemos ver que $(t_2, t_5) \equiv_{stutter} (t_2, t_6)$ (al igual que (t_3, t_5) y (t_3, t_6)), ya que difieren únicamente en transiciones stutters.

Una consecuencia de utilizar transiciones globales es la necesidad de identificar las locaciones de salida y de llegada en cada una de las componentes de la LTAN. (Se debe hacer notar que sólo tendremos en cuenta aquellas componentes que participan de la transición).

Definición 2.8. (Source y Target). *Dada una transición global $t \in T^n(N)$ definiremos*

$$source(t) = \{sc_i(\pi_i(t)) \ / \ 1 \leq i \leq n \wedge Participates(A_i, t)\}$$
$$target(t) = \{tg_i(\pi_i(t)) \ / \ 1 \leq i \leq n \wedge Participates(A_i, t)\}$$

como los conjuntos de estados que anteceden y preceden a t en cada uno de los autómatas de N que participan de la transición global. ■

Para clarificar los conceptos de source y target veremos como se definen para las transiciones globales de la LTAN de la figura 2.2. Tenemos que $source(t_1, t_4) = \{0, A\}$ y que $target(t_1, t_4) = \{1, B\}$, mientras que $source(t_3, t_6) = \{2\}$ y $target(t_3, t_6) = \{0\}$.

De manera análoga a lo realizado sobre LTA, definiremos el concepto de ejecución de una LTAN. A diferencia de la anterior, donde se consideraban transiciones locales, en las siguientes definiciones las transiciones son globales.

Definición 2.9. (Proyección). *Dada una secuencia de transiciones temporizada σ de una LTAN N, definiremos $\Pi_i(\sigma)$ como la proyección de σ en el i-ésimo LTA de N. Es decir $\Pi_i(\sigma)^j = \pi_i(t_j)$* ■

Definición 2.10. (Ejecución de una LTAN). *Dada una secuencia de transiciones σ diremos que σ es una ejecución de la LTAN N, si $\forall i, 1 \leq i \leq n$, $\Pi_i(\sigma)$ es una ejecución de A_i.* ■

2.3 Branching Processes

El desarrollo realizado en las secciones anteriores es suficiente como base para un procedimiento de verificación de propiedades de alcanzabilidad local sobre un sistema modelado como una LTAN. En teoría, este procedimiento es simple de enunciar: dada la pregunta ¿es posible, partiendo del conjunto de locaciones iniciales de la LTAN N, alcanzar la locación q del i-ésimo autómata que la compone?, la respuesta a la misma está determinada por la existencia de una ejecución tal que su simulación sobre N arribe a la locación q.

Para esto se hace necesario contar con un mecanismo que genere únicamente las ejecuciones del modelo. Es decir, que genere las secuencias de transiciones que respeten la relación causal entre las mismas. De esta manera nos interesa poder definir una relación de orden sobre todas las transiciones de la LTAN. Obviamente, este orden será un *orden parcial*, ya que puede suceder que algunas transiciones no sean comparables entre sí, y que en consecuencia, el resultado de ejecutar primero la transición a y luego la b sea el mismo que se obtiene al tomarlas en el orden inverso (interleaving). Otro aspecto a considerar es el de identificar aquellos grupos de transiciones que son mutuamente excluyentes entre sí, ya que la presencia de éstas determina que la secuencia de transiciones no sea una ejecución.

Sin embargo, estas dos condiciones siguen siendo insuficientes para generar las ejecuciones. Puede darse el caso en que una transición deba ejecutarse antes que otra en una determinada situación, y que en otro momento deba ejecutarse después (este el típico caso de los ciclos). De este hecho se desprende que el contexto condiciona las transiciones que pueden ejecutarse durante una evolución del sistema.

Afortunadamente, en [Eng91] se presenta una construcción llamada *branching process* que satisface nuestras necesidades. Un branching process constituye una representación acíclica de ejecuciones sobre un modelo. Nuestro objetivo será entonces poder tomar una LTAN y construir un branching process asociado en el que se encuentren representadas **todas** las ejecuciones sobre la misma. Llamaremos *unfolding* a este branching process maximal.

De esta manera, aplicar el proceso de verificación que hemos mencionado sobre una LTAN, se reduce a construir el unfolding de la misma y determinar si es posible arribar a la locación deseada tomando una de las ejecuciones por él representadas.

Dedicaremos lo que sigue de esta sección a la tarea de definir formalmente los conceptos sobre los cuales se sustentarán los unfoldings de LTANs. Comenzaremos presentando las definiciones básicas de la literatura, siguiendo la línea de trabajo presentada en [ERV96].

Nuestro punto de partida lo constituyen un tipo particular de grafos lla-

mados *nets*, que son utilizados como sustento de los branching processes.

Definición 2.11. (Net). *Una tupla $\langle S, T, F\rangle$ es una net si $S \cap T = \emptyset$ y $F \subseteq (S \times T) \cup (T \times S)$. Llamaremos places a los elementos de S, y transiciones a los elementos de T (y ambos serán llamados genéricamente nodos).* ■

La figura 2.3 es un ejemplo de una Net donde $0, 1, 2, 3$ y 4 son places, mientras que t_1, t_2, t_3 y t_4 son transiciones.

Figura 2.3: Ejemplo de Net

Dado un nodo definiremos los conceptos de *preset* y *postset* del mismo, que nos permitirán referirnos a la relación causal (de flujo) de las nets.

Definición 2.12. (Preset y Postset de un Nodo). *Dada una net $N = \langle S, T, F\rangle$, y un nodo $x \in S \cup T$, definiremos el preset de x, denotado ${}^\bullet x$, como ${}^\bullet x = \{y \in S \cup T / (y, x) \in F\}$. De manera análoga definiremos el postset de x, denotado $x^\bullet$, como $x^\bullet = \{y \in S \cup T / (x, y) \in F\}$.*

Extenderemos las definiciones de preset y postset para poder aplicarlas sobre conjuntos de nodos. Sea $X \subseteq S \cup T$, definiremos entonces

$$ {}^\bullet X = \bigcup_{x \in X} {}^\bullet x \quad \text{y} \quad X^\bullet = \bigcup_{x \in X} x^\bullet $$

■

Aplicando las definiciones anteriores al place 2 de la figura 2.3 obtenemos que ${}^\bullet 2 = \{t_1, t_2\}$ y $2^\bullet = \{t_3, t_4\}$ tal como se observa en la misma.

También se hace necesario poder identificar las decisiones que se van tomando durante una ejecución del sistema. Esto es, poder determinar cuáles son aquellas evoluciones del sistema que inhabilitan a otras posibles. Por este motivo, introduciremos el concepto de *conflicto*, que utilizaremos para expresar que dos evoluciones posibles son mutuamente excluyentes.

Definición 2.13. (Nodos en Conflicto). *Sea (S, T, F) una net, y sean x_1 y $x_2 \in S \cup T$. Los nodos x_1 y x_2 se encuentran en conflicto, denotado $x_1 \# x_2$, si existen transiciones distintas t_1 y $t_2 \in T$ tales que ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, y $(t_1, x_1), (t_2, x_2)$ pertenecen a la clausura reflexo-transitiva de F.* ■

Figura 2.4: Ejemplo de Preset, Postset y Nodos en Conflicto

Tal como se puede ver en la figura 2.4 los nodos 3 y 4 están en conflicto ($3\#4$) ya que ${}^\bullet t_3 = {}^\bullet t_4$, la transición t_3 está relacionada al place 3 y la transición t_4 está relacionada al place 4.

No estamos interesados en cualquier tipo de net, sino en aquellas que satisfagan algunas condiciones particulares. Es decir, nets que modelen el aspecto estructural del unfolding de la LTAN: ausencia de ciclos, safeness y la condición de que todo nodo de la net sea alcanzable partiendo desde la raíz de la misma. Llamaremos *Ocurrence Nets* a las nets que cumplan con todos estos requisitos.

Definición 2.14. (Ocurrence Net). *Diremos que una net $ON = \langle B, E, F \rangle$ con B un conjunto de condiciones y E un conjunto de eventos es una ocurrence net si se satisface:*

- *para cada $b \in B$, $|{}^\bullet b| \leq 1$*
- *F no tiene ciclos*
- *para cada $x \in B \cup E$, el conjunto de elementos $y \in B \cup E$ tales que (y, x) pertenece a la clausura transitiva de F es finito*
- *ningún evento $e \in E$ está en conflicto con sí mismo*

■

Utilizaremos la función Min(N) para definir formalmente el concepto de raíz de una ocurrence net.

Definición 2.15. (Min). *Dada una ocurrence net $ON = \langle B, E, F \rangle$ llamaremos $Min(ON)$ al conjunto de elementos de $B \cup E$ que es minimal respecto a la clausura transitiva de F.* ■

En la net representada en la figura 2.3 se cumple que $Min() = \{0, 1\}$.

Todas las evoluciones de la LTAN al ser reflejadas sobre una ocurrence net partirán de los nodos contenidos en Min(). Si tomamos el conjunto de los eventos involucrados durante una ejecución hasta un cierto punto habremos definido una *configuración* sobre la ocurrence net. Podemos pensar entonces que cada paso de simulación de una ejecución de la LTAN va definiendo una nueva configuración sobre una ocurrence net, la que registra todos los eventos que han sido tomados desde Min().

Definición 2.16. (Configuración). *Dada $ON = \langle B, E, F \rangle$, una ocurrence net, diremos que el conjunto de eventos C es una configuración de N, si se satisfacen las siguientes condiciones:*

- $C \subseteq E$
- $e \in C \Rightarrow (\forall e' \in E : e' \leq e \Rightarrow e' \in C)$[2]
- $\forall e, e' \in C : \neg(e \# e')$

■

[2]Al indicar $e' \leq e$ se desea expresar que el par (e', e) pertenece a la clausura transitiva de la relación de flujo F. Es decir, que para poder tomar el evento e es necesario tomar antes el evento e'.

Observación: los elementos de un LTA en una configuración están totalmente ordenados.

Como ejemplos de configuraciones de la figura 2.3 podemos mencionar a $\{t_1, t_2\}$, $\{t_1, t_2, t_3\}$ y $\{t_1, t_2, t_3, t_4\}$.

Resulta bastante cómodo poder hablar de la configuración mínima necesaria para alcanzar un evento dado. Es decir, el conjunto de eventos imprescindibles para poder activarlo. Este conjunto de eventos (que constituye una configuración) que llamaremos *configuración local* del evento.

Definición 2.17. (Configuración local). *Dado un evento e de una ocurrence net $ON = \langle B, E, F \rangle$, definiremos la configuración local de e, denotada $\lceil e \rceil$, como la configuración formada por todos los eventos $e' \in E$ tales que $e' \leq e$.*

■

Por ejemplo, la configuración local del evento t_3 de la figura 2.3 es el conjunto formado por los eventos t_1, t_2 y t_3.

Ha llegado el momento de formalizar la relación existente entre una ocurrence net y una LTAN. Esto es necesario ya que en nuestro esquema de verificación, el chequeo sobre la LTAN en realidad lo estaremos realizando sobre la ocurrence net que representa a la LTAN. Para esto utilizaremos una función (homomorfismo).

Definición 2.18. (Homomorfismo). *Sean $ON = \langle B, E, F \rangle$ una ocurrence net y $N = \langle A_1, A_2, ..., A_n \rangle$ una LTAN. Sean $Q = \bigcup_{1 \leq i \leq n} Q_i$ y $T = T^n(N)$. Un homomorfismo de ON a N es un mapping $h : B \cup E \rightarrow Q \cup T$ tal que:*

- *$h(B) \subseteq Q$ y $h(E) \subseteq T$*
- *para cada $e \in E$, la restricción de h a ${}^\bullet e$ es una biyección entre ${}^\bullet e$ (en ON) y $source(h(e))$ (en N), y de manera similar para $e^\bullet$ y $target(h(e))$*

■

Respecto de la definición anterior es necesario hacer una observación concerniente al segundo item de la misma. La definición de source de una LTAN (definición 2.8) permite obtener sólo la locación de origen de las transiciones locales que participan en la transición global. Por lo que la existencia de una biyección entre el preset del evento (${}^\bullet e$) y el source de la aplicación del homomorfismo ($source(h(e))$) se hace posible. Las mismas consideraciones se aplican al target.

Utilizaremos el concepto de *branching process* de una LTAN para agrupar, en una misma entidad, a la ocurrence net y al homomorfismo que la relaciona con la LTAN.

Definición 2.19. (Branching Process). *Un branching process de una LTAN N es un par $\beta = \langle ON, h\rangle$ donde ON es una ocurrence net y h es un homomorfismo de ON a N tal que:*

- *la restricción de h a $Min(ON)$ es una biyección entre $Min(ON)$ y $\bigcup_{1\leq i\leq n} q^i_{in}$*
- *para cada $e_1, e_2 \in ON$, si ${}^\bullet e_1 = {}^\bullet e_2$ y $h(e_1) = h(e_2)$, entonces $e_1 = e_2$*

■

Nota: A partir de este momento, cada vez que nos refiramos a una configuración (local) de un branching process nos estaremos refiriendo a la configuración (local) en la ocurrence net asociada.

Por comodidad, tomaremos la definición de los autómatas que participan de una transición global de la LTAN (definición 2.7), y la extenderemos para poder identificar a los autómatas que participan de un evento del branching process asociado.

Definición 2.20. (Autómatas que participan en un Evento). *Dado un evento e de un branching process $\beta = \langle ON, h\rangle$ de una LTAN N extenderemos el predicado Participates como*

$$Participates(A_i, e) \text{ sii } Participates(A_i, h(e))$$

■

En la ocurrence net tenemos conjuntos de condiciones y de eventos que simulan evoluciones en la LTAN. Para relacionar de alguna forma los eventos involucrados en una evolución hemos definido con anterioridad el concepto de configuración. Lo que quedaría por definir es el conjunto de requisitos que tiene satisfacer un conjunto de condiciones de una ocurrence net, para poder representar un "estado" válido en la LTAN.

Esto se reduce a la restricción de que, tomadas de a pares, las condiciones no sean comparables entre sí.

Definición 2.21. (Relación *co*). *Dada una ocurrence net $ON = \langle B, E, F\rangle$ y dos nodos $x, y \in B \cup E$, decimos que x co y, si no se cumple que $x\#y$, ni que $x < y$, ni tampoco que $y < x$; donde $<$ es la clausura transitiva irreflexiva de F.* ■

Si vemos la figura 2.3 podemos marcar algunos pares de nodos que cumplen con la propiedad *co*. Por ejemplo $t1$ *co* $t2$ y $t1$ *co* 2.

Definición 2.22. (Co-Set y Cut). *Un conjunto B de condiciones de una ocurrence net es un co-set si todos sus elementos tomados de a dos satisfacen la relación* ***co****. Un co-set maximal respecto a la inclusión es llamado cut.* ■

De acuerdo a esta última definición, un cut es un conjunto que puede representar un "estado" válido de la LTAN.

El *cut de una configuración* nos devuelve las condiciones que son alcanzadas como resultado de tomar todos los eventos de una configuración. A su vez el *marking de una configuración* nos permite conocer cuales son las locaciones de la TAN asociadas a esas condiciones.

Definición 2.23. (Cut de una Configuración). *Dada una configuración C de una ocurrence net ON definiremos el cut asociado a C como $Cut(C) = (Min(ON) \cup C^{\bullet}) \setminus {}^{\bullet}C$.* ■

Definición 2.24. (Marking de una Configuración). *Dada una configuración C de un branching process $\beta = \langle ON, h \rangle$ definiremos $Marking(C) = h(Cut(C))$.* ■

2.4 Unfoldings de LTANs

En la presente sección presentaremos un algoritmo que permita construir el unfolding de una LTAN. Contar con el mismo nos permitirá realizar un primer análisis de la viabilidad del procedimiento de verificación de propiedades de alcanzabilidad propuesto. Adicionalmente discutiremos algunas optimizaciones al método, siendo la más importante de ellas la utilización de un prefijo del unfolding que contenga toda la información presente en este último.

Como hemos mencionado anteriormente, el unfolding no es más ni menos que un branching process maximal sobre un formalismo dado (LTAN en este caso). A medida que vayamos construyendo el branching process que representa las posibles evoluciones del sistema como un conjunto parcialmente ordenado de eventos, se hace necesario poder determinar cuál es la manera correcta de ir extendiendo el mismo en cada paso de iteración. Antes de agregar un nuevo evento debemos asegurarnos que el resultado que obtendremos será válido (se verifica la relación causal entre los eventos). Por este motivo introducimos la definición de *extensión posible*.

Definición 2.25. (Extensión Posible). *Sean $\beta = \langle < B, E, F >, h \rangle$ un branching process de una LTAN N, $t \in T^n(N)$ una transición global de N y $X \subseteq B$ un conjunto de condiciones de β. Diremos que el par (t, X) es una extensión posible de β si:*

- $\forall x, y \in X : \neg(x \# y)$ *(X es un co-set de condiciones de β)*
- $h(X) = source(t)$ *(X es el source de la transición global t)*
- *β no contiene ningún evento e tal que $h(e) = t$ y ${}^\bullet e = X$*

Denotaremos $EP(\beta)$ al conjunto de extensiones posibles del branching process β.

■

Observación: Se debe hacer notar que en el segundo item de la definición anterior, de acuerdo a las definiciones de source (defiición 2.8) y homomorfismo (definición 2.18), se están descartando las transiciones stutter.

Ahora que contamos con la posibilidad de referirnos a la extensión de un branching process, es posible analizar cuáles son las transformaciones que se producen sobre un branching process al ser extendido con un nuevo evento.

Definición 2.26. (Branching Process Extendido). *Dado un branching process $\beta = \langle < B, E, F >, h \rangle$ y una extensión posible $e = (t, X)$, definiremos la extensión de β con e, denotada $\beta' = \beta \oplus \{e\}$, como el branching process $\beta' = \langle < B', E', F' >, h' \rangle$ donde*

- $B' = B \cup \{(q, e)/q \in target(t)\}$
- $E' = E \cup \{e\}$
- $F' = F \cup \{x \rightarrow e/x \in X\} \cup \{e \rightarrow (q, e)/q \in target(t)\}$
- $h' = h \cup \{h(e) \rightarrow t\} \cup \{h(q, e) \rightarrow q/q \in target(h(e))\}$

■

Como consecuencia de la modalidad incremental utilizada en la construcción del branching process, resulta cómodo poder expresarnos sobre las configuraciones, sus extensiones y el resultado de la aplicación de estas últimas, lo que introduce las nociones de *extensión y sufijo de una configuración.*

Definición 2.27. (Extensión y Sufijo de una Configuración). *Sea C una configuración finita en un branching process $\beta = \langle < B, E, F >, h \rangle$ y S un conjunto finito de eventos $S \subseteq E$. Diremos que $C \oplus S$ es una extensión de C y que S es un sufijo de C si $C \cup S$ es una configuración en β tal que $C \cap S = \emptyset$.* ■

De esta forma contamos con todos los elementos necesarios para concentrarnos en un algoritmo que construya el unfolding. El mismo es bastante sencillo: partiendo de un branching process inicial donde están representadas las locaciones iniciales de la LTAN, iremos aplicándole extensiones posibles hasta obtener un branching process que ya no pueda ser extendido. De esta forma, el algoritmo obtenido puede considerarse una traducción al contexto de redes de autómatas del algoritmo presentando en [ERV96] sobre Redes de Petri.

Utilizaremos parte de la notación presentada, en particular las referidas a extensiones posibles (EP : definición 2.25) y a su aplicación sobre un branching process ($\oplus$: definición 2.26).

A continuación presentamos ese algoritmo:

Algoritmo 1 (Unfolding LTAN).

input: Una LTAN $N = \langle A_1, A_2, ..., A_n \rangle$.
output: El unfolding Unf de N.

begin
    $Unf := InitialConditions(N)$
    $ep := \mathrm{EP}(Unf)$

    while $ep \neq \emptyset$
        elegir un $e \in ep$
        $Unf := Unf \oplus \{e\}$
        $ep := \mathrm{EP}(Unf)$
    endwhile
end

El algoritmo comienza construyendo un branching process que corresponde al marking inicial de la LTAN utilizando una función auxiliar llamada $InitialConditions()$, y generando el conjunto de extensiones posibles correspondientes al mismo.

A partir de ese momento, va extendiendo el branching process aplicando en cada paso una nueva extensión posible (e). Este procedimiento se repite hasta que no se pueden generar nuevas extensiones posibles.

Las extensiones se van recalculando en cada paso de iteración ya que puede suceder que al extender con el evento e se generen nuevas posibilidades, las que se suman a las previamente calculadas (con la excepción obvia de e).

Debido a que el unfolding contiene todas las posibles extensiones que van siendo calculados por el algoritmo, el criterio de selección del evento e con el cual se extenderá en cada paso de iteración es irrelevante.

Si en algún paso de iteración no existen extensiones posibles para el branching process construido hasta el momento, entonces el algoritmo finaliza.

En el caso en que la LTAN contenga ciclos, en cada paso existirán nuevas extensiones posibles, por lo que el unfolding será infinito y en consecuencia el algoritmo no finalizará. Por este motivo, sería interesante poder contar con un branching process **finito** que contuviese la misma información que el unfolding. Es decir, un *prefijo completo finito*.

Para obtener este prefijo es un requisito imprescindible contar con un criterio de corte que nos permita determinar el momento en que debemos detener la aplicación de extensiones al branching process. Siendo un poco más específicos, lo que queremos es disponer de un criterio que nos permita saber cuándo la aplicación de una extensión no agregará nueva información al branching process en construcción. Es claro que si para todas las configuraciones en las que participa el evento que estamos agregando podemos encontrar otra configuración con el mismo marking que no lo contenga, podemos afirmar que el nuevo evento no agrega nueva información ya, que ambas configuraciones tienen las mismas extensiones posibles (es decir, tienen el mismo "futuro").

Sin embargo, realizar este chequeo en cada paso de iteración tiene un costo computacional excesivo, lo que hace que su inclusión dentro de un algoritmo para la construcción del prefijo completo sea poco deseable. Para remediar esta situación usaremos la misma idea que utilizan tanto [McM92a] como [ERV96], que consiste en realizar un chequeo similar **considerando únicamente las configuraciones locales**. Es decir, antes de extender el branching process con un evento, verificaremos que en el mismo no se encuentre otro evento cuya configuración local sea a la vez *equivalente* y *menor* que la del nuevo evento. Necesitamos entonces, definir formalmente ambas nociones.

Definición 2.28. (Equivalencia entre Configuraciones). *Dado un branching process β de una LTAN N, diremos que dos configuraciones C_1 y C_2 son equivalentes (notado $C_1 \equiv C_2$) si se verifica que $Marking(C_1) = Marking(C_2)$.* ■

Antes de continuar con la defición del orden entre configuraciones necesitamos poder determinar si dos configuraciones presentan las mismas posibilidades de ser extendidas. Por este motivo, se hace necesario contar con un mecanismo que nos permita "viajar" entre configuraciones. Un procedimiento que nos permita relacionar los eventos y las condiciones de una con

los de la otra. Para este fin nos valdremos de la noción de *isomorfismo* entre configuraciones.

Definición 2.29. (Isomorfismo). *Dados $\beta = \langle ON, h\rangle$ y $\beta' = \langle ON', h'\rangle$ dos branching process de una LTAN N, diremos que β y β' son isomorfos si existe un homomorfismo biyectivo $\overline{h}$ tal que $h \circ \overline{h} = h'$.* ■

Intuitivamente dos branching process isomorfos sólo difieren entre sí en los nombres de los eventos y condiciones. Por otra parte, dadas dos configuraciones que tienen el mismo marking, existe un isomorfismo entre sus cuts. Aún más, las posibles extensiones también son isomorfas. Llamaremos $I_{C_1}^{C_2}$ a este isomorfismo inducido sobre extensiones posibles.

Ahora sí podemos definir la noción de orden entre configuraciones. Consideraremos que una configuración es menor que otra si lo es respecto de algún *orden adecuado*.

Definición 2.30. (Orden Adecuado). *Un orden parcial $\prec$ sobre configuraciones finitas de un branching process es un orden adecuado si:*

- *$\prec$ es un orden bien fundado (well-founded)*
- *$\prec$ refina a $\subset$ (es decir, $C_1 \subset C_2 \Rightarrow C_1 \prec C_2$)*
- *$\prec$ es preservado por extensiones finitas (si $C_1 \prec C_2$ y $C_1 \equiv C_2$, dado un conjunto finito de eventos E se verifica $C_1 \oplus E \prec C_2 \oplus I_{C_1}^{C_2}(E)$)*

■

Finalmente, valiéndonos de algún orden adecuado que nos resulte conveniente, estamos en condiciones de presentar el criterio de corte de un algoritmo que construye el prefijo completo. En cada paso evaluaremos una de las extensiones posibles, y sólo en el caso de que ésta no sea "repetida" la agregaremos al branching process. Para ser rigurosos, diremos que sólo aplicaremos una extensión posible si la misma no es un *cut-off*.

Definición 2.31. (Evento Cut-off). *Sea β un branching process y $\prec$ un orden parcial adecuado sobre las configuraciones de β. Un evento e es un cut-off (en relación a $\prec$) si β contiene una configuración local $\lceil e' \rceil$ tal que:*

- $\lceil e' \rceil \equiv \lceil e \rceil$
- $\lceil e' \rceil \prec \lceil e \rceil$

■

A continuación presentamos un algoritmo que construye el prefijo completo de un unfolding de una LTAN. El mismo está basado en el algoritmo 1 al que adicionaremos los siguientes conceptos: configuración local ($\lceil e \rceil$: definición 2.17), orden adecuado ($\prec$: definición 2.30) y cut-off (definición 2.31).

Algoritmo 2 (Prefijo Completo).

input: Una LTAN $N = \langle A_1, A_2, ..., A_n \rangle$.
output: Un prefijo completo Fin del unfolding de N.

begin
$Fin := InitialConditions(N)$
$ep :=$ EP(Fin)
$cutOffs := \emptyset$

while $ep \neq \emptyset$
elegir $e \in ep$ tq $\lceil e \rceil$ es minimal respecto a $\prec$
if $(\lceil e \rceil \cap cutOffs) = \emptyset$ **then**
$Fin := Fin \oplus \{e\}$
$ep :=$ EP(Fin)
if e es un cut-off de Fin **then**
$cutOffs := cutOffs \cup \{e\}$
endif
else
$ep := ep \setminus \{e\}$
endif
endwhile
end

El algoritmo que construye el prefijo completo, también basado en el presentado en [ERV96], difiere respecto del algoritmo que construye el unfolding en la elección de la nueva posible extensión, y en el hecho de que descarta aquellas que son cut-offs.

En cada paso de iteración, la nueva extensión posible e se elige respecto del orden adecuado $\prec$. El requisito de la minimalidad de la configuración local de e se solicita a los fines de generar prefijos más pequeños, ya que al ir considerando las extensiones posibles de acuerdo a su orden se incrementan nuestras chances de detectar cut-offs.

Como ya hemos mencionado en secciones anteriores del trabajo, una vez que hemos identificado que un evento es cut-off se torna innecesario continuar

extendiendo el branching process con aquellas extensiones posibles que se generan a partir del mismo. Sin embargo, es necesario extender el branching process con los cut-offs mismos. Este hecho, que a primera vista es contrario a la intuición, y que supondría la construcción de un prefijo de tamaño mayor al necesario, no lo es tal. Si los eventos cut-offs fueran descartados por el algoritmo de la misma manera en que no se consideran las extensions posibles generadas por los mismos, el prefijo generado podría no ser un prefijo completo. (Puede consultarse un ejemplo en [ERV96]).

Proposición 1. *El prefijo Fin generado por el algoritmo 2 es un prefijo completo finito.*

Demostración. La demostración es equivalente a la presentada en [ERV96], por lo que no será incluida. □

De esta forma damos por finalizada la presentación de la técnica de unfoldings para LTANs. En el capítulo siguiente mostraremos como extender la misma para ser utilizada en el contexto temporizado.

Capítulo 3

Unfoldings de Sistemas Temporizados

En el presente capítulo se formaliza la propuesta de nuestro trabajo, esto es, la presentación del marco formal que permite definir unfoldings de redes de autómatas temporizadas así como la inserción de los mismos como base de un procedimiento de verificación de propiedades de alcanzabilidad.

El desarrollo que se presentará a continuación puede ser considerado como la adaptación al contexto temporizado del trabajo presentado en el capítulo anterior.

3.1 Autómatas Temporizados

Así como los autómatas de transiciones etiquetadas constituyeron el punto de partida para el desarrollo presentado en capítulo anterior, en el presente partiremos de los *autómatas temporizados*.

Un autómata temporizado es un autómata (de transiciones etiquetadas) que impone restricciones temporales a su evolución. Estas restricciones se implementan utilizando *relojes*, los que van registrando los momentos en los que ocurren aquellos sucesos que son relevantes para el sistema. Cada vez que se produce uno de estos eventos, el reloj que se utiliza para registrar el momento en que se éste se ha producido es reseteado. De esta forma, si trazáramos una flecha temporal con la evolución del sistema desde el momento inicial, y fuéramos marcando cada uno de estos sucesos relevantes, los valores de los relojes nos indicarían qué tan lejos o qué tan cerca nos encontramos de la ocurrencia de los mismos.

Estas restricciones temporales se presentan bajo la forma de inecuaciones lineales sobre los relojes. Llamaremos *guardas* a los predicados asociados a

las transiciones e *invariantes* a los asociados a las locaciones. Las guardas restringen la posibilidad de tomar una transición, ya que para que esto sea posible el valor actual de los relojes del sistema debe satisfacer la restricción. Por ejemplo, una guarda podría decir "sólo es posible tomar la transición t que permite pasar de la locación a a la locación b si han pasado menos de 10 unidades de tiempo desde que se ha reseteado el reloj x". De la misma forma, para poder permanecer en una locación determinada, el valor actual de los relojes debe satisfacer el invariante asociado a la misma. Por ejemplo, el invariante de la locación b podría querer decir: "sólo es posible mantenerse en la locación b si han pasado menos de 5 unidades de tiempo desde la última vez que se reseteó el reloj x".

Además de la guarda, las transiciones tienen asociadas información sobre los relojes que deben ser reseteados al momento de ser ejecutadas. De esta forma se cumplen dos objetivos: olvidar el tiempo transcurrido desde un evento que dejó de ser relevante, y al mismo tiempo registrar la ocurrencia de un nuevo evento.

Definición 3.1. (Autómata Temporizado). *Un autómata temporizado (TA = Timed Automaton) es una tupla* $A = \langle Q, T, \lambda, sc, tg, q^{in}, \Sigma, R, \iota, \gamma, \rho \rangle$ *donde:*

- $\langle Q, T, \lambda, sc, tg, q^{in}, \Sigma \rangle$ *es un autómata de transiciones etiquetadas*
- R *es un conjunto finito de relojes que toman valores reales positivos*
- ι *es una función que asigna a cada locación* q *el invariante* $\iota(q)$*, el cual es una conjunción de predicados de la forma "*$x \leq k$*", donde* $k \in \mathbb{N}$ *y* $x \in R$
- γ *es una función que asigna a cada transición* t *la guarda* $\gamma(t)$*, la cual es una conjunción de predicados de la forma "*$x \sim k$*" donde* $x \in R$*,* $\sim \in \{\leq, \geq\}$*, y* $k \in \mathbb{N}$ [1]
- ρ *es una función que asigna a cada transición* t *un subconjunto* $\rho(t) \subseteq R$ *de relojes, los cuales son reseteados por* t

■

Al igual que en el contexto no temporizado, se hace necesario recurrir a la definición del concepto de transición stutter. Respetando la intuición presentada en la definición 2.2, las restricciones asociadas a una transición stutter no deben tener impacto en la evolución observable del sistema.

[1] Se debe hacer notar que se podrían haber considerado guardas de la forma "$x \sim k''$ con $\in \{<, >\}$. No lo haremos para simplificar la presentación.

Definición 3.2. (Transición Stutter [versión temporizada]). *Dado un TA* $A = \langle Q, T, \lambda, sc, tg, q^{in}, \Sigma, R, \iota, \gamma, \rho, \Sigma \rangle$ *consideraremos que para cada locación* $q \in Q$ *existirá una transición stutter* $t_{\epsilon_q} \in T$ *que satisface la definición 2.2. Adicionalmente, las guardas de estas transiciones stutter no imponen ninguna restricción sobre los relojes así como tampoco resetean ningún reloj.* ■

A diferencia del contexto no temporizado, donde para definir una ejecución sobre un LTA (definición 2.3) era suficiente considerar la relación de causalidad entre las transiciones, en el contexto temporizado se hace necesario recurrir a algunas construcciones adicionales para obtener una definición de ejecución sobre un TA. Comenzaremos definiendo un tipo especial de secuencia donde se encuentren representadas transiciones del sistema, cada una de las cuales tiene asociada un *timestamp* que indica un momento en el tiempo relativo al comienzo de la evolución.

Definición 3.3. (Secuencia de Transiciones Temporizada de un TA). *Dado un TA* $A = \langle Q, T, \lambda, sc, tg, q^{in}, \Sigma, R, \iota, \gamma, \rho \rangle$ *diremos que la secuencia de pares* $\sigma = (t_1, \tau_1)(t_2, \tau_2)...(t_n, \tau_n)$ *es una secuencia de transiciones temporizada de A si satisface las siguientes condiciones:*

- $\forall i, 1 \leq i \leq n : t_i \in T$
- $\forall i, 1 \leq i \leq n : \tau_i \in \mathbb{R}_{\geq 0}$
- $\forall i, 1 \leq i < n : \tau_i \leq \tau_{i+1}$

Para cada secuencia de transiciones temporizada σ *definiremos* $|\sigma|$ *como la longitud de la misma. Asimismo definiremos* σ^j*, con* $1 \leq j \leq |\sigma|$ *como el j-ésimo par de* σ*.* ■

Un ejemplo de una secuencia de transiciones temporizadas del automata que modela el tren en la figura 1.7 es: $(Aproximandose, 0).(PorCruzar, 1.5).(Alejandose, 3).(Aproximandose, 6).(PorCruzar, 7)$.

Se debe hacer notar que los requisitos que hemos impuesto a las secuencias de transiciones temporizadas no son suficientes para garantizar el cumplimiento de las restricciones temporales expresadas por las guardas y los invariantes del TA. Se hace necesario contar con un mecanismo que nos permita conocer el valor de los relojes en el momento en que se toma cada una de las transiciones t_i (para determinar de esta manera si la transicón se encuentra habilitada), así como también el valor de éstos una vez que la misma se ha ejecutado (para determinar si se satisface el invariante del target). Para tal fin utilizaremos las funciones de valuación $v^{t_i}_{before}$ y $v^{t_i}_{after}$ respectivamente.

Definición 3.4. (Funciones de Valuación). *Dada una secuencia de transiciones temporizada σ sobre un TA A, definimos las siguientes funciones de valuación sobre los relojes de A antes y después de la ejecución de la i-ésima transición de σ:*

$$v^{t_i}_{before}(x)(\sigma, A) = \begin{cases} \tau_i - \max\limits_{1 \leq j < i}\{\tau_j / x \in \rho(t_j)\} & si\ \exists j, 1 \leq j < i\ tq\ x \in \rho(t_j) \\ \tau_i & en\ otro\ caso \end{cases}$$

$$v^{t_i}_{after}(x)(\sigma, A) = \begin{cases} 0 & si\ x \in \rho(t_i) \\ v^{t_i}_{before}(x)(\sigma, A) & en\ otro\ caso \end{cases}$$

■

Por ejemplo, para la secuencia de transiciones temporizadas mencionada sobre la figura 1.7 vale que $v^{Aproximandose,6}_{before}(x) = 6$ y $v^{Aproximandose,6}_{after}(x) = 0$.

Adicionalmente, utilizaremos la siguiente definición para referirnos a aquellos relojes del TA que deben ser evaluados para determinar si las guardas se encuentran habilitadas y se satisfacen los invariantes.

Definición 3.5. (Relojes de un Predicado). *Dado un predicado P que es conjunción de fórmulas de la forma "$x \sim k''$ con $\sim \in \{\leq, \geq\}$, definiremos $Clocks(P)$ como el conjunto de relojes que aparecen nombrados en las fórmulas de P.* ■

Ahora sí, estamos en condiciones de formalizar la idea de lo que significa una evolución válida del sistema que tome en cuenta el paso del tiempo.

Definición 3.6. (Ejecución de un TA). *Dado un TA A diremos que la secuencia de transiciones temporizadas $\sigma = (t_1, \tau_1)(t_2, \tau_2)...(t_n, \tau_n)$ es una ejecución de A si satisface las siguientes condiciones:*

- $sc(t_1) = q^{in}$
- $\forall i, 1 \leq i < n : tg(t_i) = q = sc(t_{i+1})$
- $\forall t_i, \forall\ x \in Clocks(\gamma(t_i)) : v^{t_i}_{before}(x)(\sigma, A) \vDash \gamma(t_i),\ 1 \leq i \leq n$
- $\forall t_i, \forall\ x \in Clocks(\iota(sc(t_i))) : v^{t_i}_{before}(x)(\sigma, A) \vDash \iota(sc(t_i)),\ 1 \leq i \leq n$
- $\forall t_i, \forall\ x \in Clocks(\iota(tg(t_i))) : v^{t_i}_{after}(x)(\sigma, A) \vDash \iota(tg(t_i)),\ 1 \leq i \leq n$

donde el símbolo $\vDash$ debe interpretarse como "satisface". Es decir, la expresión $v^{t_i}_{before}(x)(\sigma, A) \vDash \gamma(t_i)$ indica que la valuación del reloj x antes de tomar la i-ésima transición de σ satisface la guarda de la transición. ■

La secuencia de transiciones temporizada anteriormente presentada no es una ejecución ya que su último elemento no satisface la guarda de la transición, mientras que $(Aproximandose, 0).(PorCruzar, 1.5).(Alejandose, 3).(Aproximandose, 6).(PorCruzar, 9)$ sí lo es.

Observación: Consideraremos ejecuciones finitas, ya que trabajaremos con autómatas temporizados que son *non-zeno*[2], y por lo tanto cualquier ejecución finita sobre ellos puede ser extendida a una ejecución infinita **divergente**.

Al igual que en el desarrollo presentado en el capítulo anterior, en el presente también trabajaremos sobre modelos que se obtienen como resultado de la composición de partes. Por lo tanto nos vemos en la necesidad de definir el concepto de *red de autómatas temporizados*, que constituye el análogo temporizado de las redes de autómatas de transiciones etiquetadas (definición 2.4)[3].

Definición 3.7. (Red de Autómatas Temporizados). *Sean los autómatas temporizados* $A_1, A_2, ..., A_n$ *con* $A_i = \langle Q_i, T_i, \lambda_i, sc_i, tg_i, q_i^{in}, \Sigma_i, R_i, \iota_i, \gamma_i, \rho_i \rangle$. *Diremos que la tupla* $N = \langle A_1, A_2, ..., A_n \rangle$ *es una red de autómatas temporizados (TAN = Timed Automata Network) si se verifica que* $R_i \cap R_j = \emptyset$, $\forall 1 \leq i < j \leq n$. ■

De la misma manera que en el contexto no temporizado, necesitamos de un mecanismo para poder sincronizar las componentes de una red de autómatas. Al igual que antes utilizaremos para tal fin a las *transiciones globales*, que agrupan transiciones locales de distintas componentes que tengan la misma etiqueta.

Definición 3.8. (Transición Global de una TAN). *Esta definición es análoga a la definición 2.5.* ■

De manera análoga a lo realizado con los TA, definimos el concepto de secuencia de transiciones temporizada de una TAN para poder obtener una noción de ejecución sobre una red de autómatas. A diferencia de las definiciones anteriores donde se consideraban transiciones locales, en las siguientes las transiciones son globales.

[2]Un modelo es non-zeno si todas sus ejecuciones son divergentes. Es decir, si siempre es posible encontrar una ejecución tan larga en el tiempo como uno desee.

[3]A partir de este momento consideraremos a las LTANs como un tipo especial de TAN cuyos predicados son trivialmente verdaderos (sus guardas y sus invariantes son True). De esta forma no se presentarán nuevas definiciones para conceptos como *source*, *target*, *participates*, etc., ya que los mismos son trivialmente extensibles al contexto de las TANs.

Definición 3.9. (Secuencia de Transiciones Temporizada de una TAN). *Dada una TAN* $N = \langle A_1, A_2, ..., A_n \rangle$ *diremos que la secuencia de pares* $\sigma = (t_1, \tau_1)(t_2, \tau_2)...(t_n, \tau_n)$ *es una secuencia de transiciones temporizada de* N *si se satisfacen las siguientes condiciones:*

- $\forall i, 1 \leq i \leq n : t_i \in T^n(N)$
- $\forall i, 1 \leq i \leq n : \tau_i \in \mathbb{R}_{\geq 0}$
- $\forall i, 1 \leq i < n : \tau_i \leq \tau_{i+1}$

■

Definición 3.10. (Ejecución de una TAN). *Dada una secuencia de transiciones temporizada* σ *diremos que* σ *es una ejecución de la TAN* N *si* $\forall i, 1 \leq i \leq n$, $\Pi_i(\sigma)$ *es una ejecución de* A_i. ■

3.2 Redes Asincrónicas

Como hemos mencionado anteriormente, el objetivo principal de nuestra propuesta es permitir la verificación de propiedades de alcanzabilidad, más específicamente propiedades de alcanzabilidad local. Para llevar a cabo esta tarea, es suficiente exhibir una ejecución de la TAN que modela nuestro sistema, con origen en las locaciones iniciales y que durante su recorrido alcance la locación buscada.

La mayor dificultad para obtener esta ejecución radica en el procedimiento utilizado para la construcción del unfolding, ya que el mismo no está orientado a la manipulación ni a la búsqueda de ejecuciones. Es por esto que se utilizará un enfoque alternativo: en vez de buscar una ejecución que alcance la locación deseada se probará que toda locación contenida en el unfolding es alcanzada por una ejecución. De esta forma, bastará con determinar si la locación a alcanzar pertenece o no al unfolding.

Durante la ejecución del procedimiento de unfolding que utilizaremos para chequear las propiedades de alcanzabilidad, cada vez que extendamos el branching process verificaremos la correctitud de la extensión basándonos únicamente en la información local provista por la misma. Como consecuencia de esta decisión no es posible asegurar que exista una ejecución de la TAN que alcance a la locación deseada. Esto debido a que no fueron tomados en consideración elementos que determinan la correctitud de la extensión en el marco global de la TAN. Lo que sí es posible afirmar es la existencia de una secuencia de transiciones temporizada que alcanza la locación deseada[4]. Desafortunadamente no hay garantías de que esta secuencia sea una

ejecución, por lo que definiremos un contexto en el que la secuencia pueda ser transformada en una ejecución.

Para esto vamos a tratar de caracterizar a aquellas secuencias de transiciones temporizadas sobre la TAN tales que las proyecciones sobre cada uno de los TA que la componen sean cubiertas por ejecuciones de los mismos. Es decir, secuencias tales que la parte significativa de su proyección sobre cada uno de las componentes pueda ser extendida a ejecuciones sobre los mismos[5]. Llamaremos *ejecuciones end asynchronous* a las secuencias con estas caraterísticas, y su definición formal será presentada en breve. Antes será necesario presentar algunas definiciones adicionales. La primera de ellas corresponde a la formalización de la noción de *cobertura entre ejecuciones*, que nos permite determinar cuándo una ejecución cubre a otra, lo que informalmente se corresponde con el hecho de que la parte significativa de la última de ellas sea un prefijo de la primera.

Definición 3.11. (Cobertura de una Ejecución de un TA). *Dadas σ_1, σ_2 ejecuciones del TA A diremos que σ_2 cubre a σ_1, notado $\sigma_1 \preceq^{cov} \sigma_2$, si existen secuencias de transiciones temporizadas σ'_1, σ''_1 y σ_3 tales que:*

- $\sigma_1 = \sigma'_1\sigma''_1$
- $\sigma_2 = \sigma'_1\sigma_3$
- $\sigma''_1 = (t_\epsilon, \tau''_1)(t_\epsilon, \tau''_2)...(t_\epsilon, \tau''_n)$

■

De manera similar, extenderemos el concepto de cobertura entre ejecuciones de una TAN.

Definición 3.12. (Cobertura de una Ejecución de una TAN). *Dadas σ_1, σ_2 ejecuciones de la TAN N diremos que σ_2 cubre a σ_1, notado $\sigma_1 \preceq^{cov} \sigma_2$ si $\forall i, 1 \leq i \leq n$, si se cumple $\Pi_i(\sigma_1) \preceq_i^{cov} \Pi_i(\sigma_2)$ (donde $\preceq_i^{cov}$ denota el orden $\preceq^{cov}$ en el i-ésimo TA de N).* ■

Finalmente presentamos la definición de ejecución end asynchronous que habíamos anticipado.

[4]Es más, esta secuencia estará constituida por las transiciones asociadas a los eventos que pertenecen a la configuración local del evento utilizado como nueva extensión del branching process.

[5]Por "parte significativa" nos estamos refiriendo a la secuencia que se obtiene de eliminar el sufijo de transiciones stutter.

Definición 3.13. (Ejecución End Asynchronous). *Una ejecución end asynchronous σ de una TAN N es una secuencia de transiciones temporizada tal que $\forall i, 1 \leq i \leq n$ existe una ejecución σ_i de A_i que cumple $\Pi_i(\sigma) \preceq_i^{cov} \sigma_i$.* ■

Observación: Se debe hacer notar que una ejecución end asynchronous no es necesariamente una ejecución.

Con lo que tenemos hasta este momento, estamos en condiciones de asegurar que en forma local existen ejecuciones con las que es posible alcanzar las mismas locaciones que con una ejecución end asynchronous. Pero esto no es suficiente ya que no estamos obteniendo necesariamente **una ejecución** de la TAN que alcance esas locaciones. Quisiéramos poder contar entonces con un escenario sobre el cual podamos asegurar que para cada ejecución end asynchronous **siempre** existirá una ejecución de la TAN que la cubra. Llamaremos *end asynchronous* a aquellas TAN que presenten estas características.

Si observamos la figura 1.7 la secuencia de transiciones temporizadas $(Aproximandose, 0).(PorCruzar, 3)$ es una ejecución *end asynchronous* pero no es una ejecución válida.

Definición 3.14. (Redes End Asynchronous de Autómatas Temporizados). *Dada una TAN N diremos que N es una red end asynchronous de autómatas temporizados (EATAN = End Asynchronous Timed Automata Network) sii el conjunto de locaciones que son alcanzadas por ejecuciones end asynchronous son también alcanzadas por ejecuciones.* ■

Afortunadamente, dada una TAN, en la práctica siempre es posible obtener una EATAN. Para esto se debe aplicar el procedimiento presentado en el Apéndice A de este trabajo.

Ahora que disponemos de las EATANs, contamos con un contexto adecuado para trabajar con información local con la tranquilidad de saber que la misma también será válida en el contexto global del modelo.

3.3 Time-Feasibility

Para poder manejar el aspecto temporal del modelo se ha decidido agregar nuevos conceptos y definiciones que enriquezcan a las configuraciones del branching process involucrado en la construcción del unfolding con información temporal del mismo. Esta información adicional será utilizada para validar el aspecto temporal del modelo

Antes de comenzar con el análisis temporal, vamos a introducir un evento distinguido que será anterior a cualquier otro. La intuición detrás del mismo es la de representar al conjunto de transiciones ficticias que "llegan" a los estados iniciales de los autómatas temporizados que participan en una TAN. De esta forma, obtenemos una caracterización del primer evento que es disparado en cualquier simulación sobre un branching process.

Definición 3.15. (Evento Bottom). *Dada $ON = \langle B, E, F \rangle$ una ocurrence net, consideraremos la existencia de un evento ficticio al que llamaremos* ***bottom*** *y notaremos $\perp$. Este evento será menor a cualquier elemento de E.* ■

Además de considerar la relación causal entre los eventos del sistema, durante la verificación de propiedades en modelos temporizados es necesario tomar en consideración las variaciones que se van produciendo en los valores de los relojes de los autómatas involucrados. Durante el proceso de construcción del branching process, para cada iteración que agregue un nuevo evento, será necesario constatar que la incorporación del mismo tendrá como resultado final una configuración válida desde el punto de vista temporal. Esto es, que será posible obtener una ejecución end-asynchronous cuya simulación sobre la TAN involucre a todos los eventos pertenecientes a dicha configuración. Llamaremos *time-feasible* a las configuraciones que respeten estas condiciones.

Si revisamos la definición de ejecución de una TAN (definición 3.10), recordaremos que las transiciones que integran las mismas deben satisfacer una serie de restricciones temporales (y otras no temporales). En consecuencia, los eventos de una configuración time-feasible están obligados a satisfacer estas mismas restricciones. Para esto es necesario poder identificar cuáles son los eventos relevantes para determinados relojes y para determinados autómatas. Por ejemplo, poder identificar cuál es el último evento que reseteó el reloj x, y cuál el último evento del i-ésimo autómata. Estas nociones de eventos relevantes se formalizan en los conceptos de *Reset* y *Last* respectivamente.

Definición 3.16. (Reset). *Sea C una configuración finita de un branching process $\beta = \langle ON, h \rangle$ de una TAN $N = \langle A_1, A_2, ..., A_n \rangle$. Definiremos $Reset_x(C)$, con $x \in R_i$ para algún $1 \leq i \leq n$, como el evento*

$$Reset_x(C) = max \ \{ \ \{e' \in C \ / \ x \in \rho(\pi_i(h(e'))) \ \} \cup \{\perp\} \ \}$$

, es decir, el último evento en C tal que su transición asociada resetea el reloj x.

Se debe hacer notar que se definirá $Reset_x(C) = \perp$ *en los casos en que la configuración* C *sea vacía y en aquellos en que todos sus eventos correspondan a transiciones que no resetean el reloj* x. ■

Definición 3.17. (Last). *Sea* C *una configuración finita de un branching process* $\beta = \langle ON, h \rangle$ *de una TAN* $N = \langle A_1, A_2, ..., A_n \rangle$. *Definiremos* $Last_i(C)$, *con* $1 \leq i \leq n$, *como el evento*

$$Last_i(C) = max \ \{ \ \{e' \in C \ / \ Participates(A_i, e')\} \cup \{\perp\} \ \}$$

Se debe hacer notar que se definirá $Last_i(C) = \perp$ *en los casos en que la configuración* C *sea vacía y en aquellos en que todos sus eventos correspondan a transiciones stutter para el i-ésimo autómata.* ■

Como hemos mencionado, las configuraciones time-feasible estarán relacionadas, cada una, con al menos una ejecución end asynchronous sobre la TAN asociada al branching process. De esta forma, en nuestro objetivo de encontrar estas ejecuciones sobre la TAN lo que haremos será intentar determinar si las configuraciones contenidas en el branching process que iremos construyendo son time-feasibles o no.

El análisis de time-feasibility de una configuración lo realizaremos basándonos en información temporal asociada a la misma. Para esto se hace necesario contar con una herramienta que nos permita extraer esta información que se encuentra "contenida" en los invariantes y las guardas relacionados con sus eventos. Para cada uno de estos estamos interesados en obtener el conjunto de restricciones que deben satisfacer sus relojes relevantes (es decir, aquellos relojes que aparecen en la guarda o en el reset de la transición asociada, o bien en el invariante de las locaciones source y target de la misma).

Dado un evento e, consideremos un predicado atómico que pertenece a la guarda de la transición asociada. El mismo será de la forma $x \sim k$, con $\sim \in \{\leq, \geq\}$. Para que el reloj x satisfaga esta restricción debe suceder que k sea un valor posible del conjunto de valores definidos por las funciones de valuación aplicadas a x. De esta forma, podemos pensar esta restricción determinada por el predicado, como una "distancia temporal" que debe existir entre el último evento que resetéo el reloj x y el evento e. Por ejemplo, en el caso que la restricción sea $x \leq 10$, nos interesa representar el hecho de que para poder tomar el evento e deben haber transcurrido a lo sumo 10 unidades de tiempo desde la última vez que se reseteó el reloj x.

En forma similar, la información relativa a un reloj x contenida en los invariantes del source y el target de una transición asociada a un evento e será representada como la distancia temporales entre e y el último evento que reseteó el reloj x.

Dividiremos estas distancias temporales en dos grupos: el primero contendrá aquellas que acotan inferiormente los valores posibles de los relojes, y el segundo aquellas distancias temporales que los acoten superiormente. Llamaremos a estos conjuntos $MinimoDelay$ y $MaximoDelay$ respectivamente.

Podemos imaginar que esta información temporal de una configuración está representada por un grafo de pesos definido sobre sus eventos. En el mismo, la información de mínimo delay está representada utilizando ejes de peso positivo con origen en el *Reset* y destino en el evento, mientras que la información de máximo delay requerirá para su representación ejes de peso negativo con origen en el evento y destino en el *Reset*. El grafo debe ser completado agregando ejes de peso nulo entre aquellos eventos que están relacionando de forma causal. En caso contrario correríamos el riesgo de perder de vista la relación causal al realizar el análisis de factibilidad, lo que podría conducir a deducciones erróneas.

Concretaremos estas ideas mediante la función Δ, la que dada una configuración, nos permitirá extraer de la misma la información temporal requerida para el análisis de time-feasibility.

Definición 3.18. (Información Temporal de una Configuración). *Sea C una configuración de un branching process β de una TAN N. La información temporal asociada a C, denotada $\Delta(C)$, es el conjunto de ejes (es decir, los elementos de $(C \cup \{\bot\}) \times (C \cup \{\bot\}) \times \mathbb{Z}$) definido como:*

$$\Delta(C) = \; Causalidad \cup MinimoDelay \cup MaximoDelay$$

donde

$$\begin{aligned}
Causalidad \quad &= \quad \{(a, b, 0)/a < b)\} \\
MinimoDelay \quad &= \quad \{(Reset_x(\lceil e \rceil \setminus \{e\}), e, k)/ \\
&\qquad \exists 1 \leq i \leq n, x \in R_i : \pi_i(h(e)) \in T_i \wedge \\
&\qquad \text{“}x \geq k\text{”} \in \gamma_i(\pi_i(h(e)))\} \\
MaximoDelay \quad &= \quad \{(e, Reset_x(\lceil e \rceil \setminus \{e\}), -k)/ \\
&\qquad \exists 1 \leq i \leq n, x \in R_i : \pi_i(h(e)) \in T_i \wedge \\
&\qquad (\text{“}x \leq k\text{”} \in \gamma_i(\pi_i(h(e))) \vee \\
&\qquad (\text{“}x \leq k\text{”} \in \iota_i(sc_i(\pi_i(h(e))))) \vee \\
&\qquad (x \notin \rho_i(\pi_i(h(e)))) \wedge \text{“}x \leq k\text{”} \in \iota_i(tg(\pi_i(h(e)))))\}
\end{aligned}$$

■

Observación: $a < b$, significa "menor en 1 paso", lo que significa que en la ocurrence net hay un eje que va de a a b. Se debe hacer notar que la definición incluye en $\Delta(C)$ ternas de la forma $(\perp, a, 0)$ para todos los eventos a que son minimales en C.

Figura 3.1: Una configuración y su grafo de pesos asociado

En la figura 3.1 vemos una configuración del branching process del autómata definido en la figura 1.7 junto con su grafo de pesos.

Ahora que disponemos de la información temporal asociada a una configuración estamos en condiciones de referirnos a la *semántica* de la misma. Será posible etonces, dada una configuración, obtener el conjunto de todas las secuencias de transiciones temporizadas tales que su simulación sobre el branching process tenga como resultado final la configuración original. Obviamente, exigiremos a todas las secuencias que pertenezcan a la semántica

de una configuración que satisfagan, en cada paso, que el tiempo de ejecución de cada transición global pertenezca al intervalo definido por el máximo y mínimo delay de su evento asociado.

Definición 3.19. (Semántica de una Configuración). *Sea C una configuración finita de un branching process β de una TAN N. La semántica de C, denotada como $\|C\|$, es el conjunto de todas las secuencias de transiciones temporizadas σ para las cuales existe una función $f : C \to [1...|\sigma|]$ que satisface las siguientes propiedades:*

1. *f es biyectiva*
2. *$\forall a, b \in C : a < b \Rightarrow f(a) < f(b)$ (orden total compatible con el de β)*
3. *$\forall a \in C : \sigma_{f(a)} \equiv_{stutter} h(a)$ (mismas transiciones sin tener en cuenta stutters)*
4. *$\forall a, b \in C, k \in \mathbb{Z} : (a, b, k) \in \Delta(C) \Rightarrow \tau_{f(b)} - \tau_{f(a)} \geq k$*

■

Observación 1: se debe hacer notar que la función f no es necesariamente la misma para todas las secuencias en $\|C\|$.

Observación 2: Dado un evento e, $f(e)$ indica la posición de la secuencia que contiene a $h(e)$ (por item 3 de la definición anterior).

Nuestro próximo objetivo será demostrar un lema que sostiene que las secuencias de transiciones temporizadas pertenecientes a la semántica de una configuración presentan la ventaja adicional de ser ejecuciones end asynchronous. A los fines de demostrar este lema, demostraremos con anterioridad la siguiente proposición.

Proposición 2. *Dada una configuración C y una secuencia de transiciones temporizada σ tal que $\sigma \in \|C\|$, se verifica que el evento $f^{-1}(|\sigma|)$ es maximal en C.*

Demostración. Lo demostraremos por el absurdo. Sea σ una secuencia de transiciones temporizada de longitud n, tal que $\sigma \in \|C\|$. Supongamos que $f^{-1}(n)$ no es maximal dentro de C. Por lo tanto existe $1 \leq i < n$ tal que $f^{-1}(n) < f^{-1}(i)$. Pero de acuerdo a la definición de f, esto implicaría que $f(f^{-1}(n)) < f(f^{-1}(i))$, lo que es lo mismo que $n < i$, lo cual constituye un absurdo. □

Valiéndonos del resultado anterior, estamos en condiciones de demostrar el lema anteriormente mencionado.

Lema 1 (Semántica de las Configuraciones vs. Semántica de la TAN). *Sea C es una configuración finita de β, un branching process de la TAN $N = \langle A_1, A_2, ..., A_n \rangle$, y sea σ una secuencia de transiciones temporizada. Si $\sigma \in \|C\|$ entonces se cumple que σ es una ejecución end asynchronous de N.*

Demostración. Lo demostraremos haciendo inducción en la longitud de las secuencias de transiciones temporizadas (que equivale a la cardinalidad de la configuración C).

Caso Base: Si la longitud de la secuencia de transiciones temporizada es 0, trivial por vacuidad.

Paso Inductivo: Supongamos que la propiedad vale para toda secuencia de transiciones temporizada de longitud r, queremos probar que también vale para toda secuencia de transiciones temporizada de longitud $r+1$.

Sea $\sigma = \sigma'.(t_{r+1}, \tau_{r+1})$ una secuencia de transiciones temporizada de longitud $r+1$ tal que $\sigma \in \|C\|$.

Veamos primero que $C' = C \setminus \{f^{-1}(r+1)\}$ es una configuración y que $\sigma' \in \|C'\|$. La proposición 2 garantiza que $f^{-1}(r+1)$ es un elemento maximal de C, por lo que al eliminarlo se obtiene una configuración.

Podemos afirmar que $\sigma \in \|C\|$ ya que existe una función f que cumple con los cuatro requisitos para C y para σ, tal que f restringida a C' cumple las propiedades para σ' ya que σ' es prefijo de σ.

Como sabemos que $\sigma' \in \|C'\|$, podemos usar la Hipótesis Inductiva y afirmar que σ' es una ejecución end asynchronous.

Entonces $\forall i, 1 \leq i \leq n$, $\exists \sigma'_i$ ejecución de A_i tal que $\Pi_i(\sigma') \preceq_i^{cov} \sigma'_i$ y nosotros queremos ver que $\forall i, 1 \leq i \leq n$, $\exists \sigma_i$ ejecución de A_i tal que $\Pi_i(\sigma) \preceq_i^{cov} \sigma_i$.

En este punto podemos determinar que para cada A_i puede darse una de las siguientes dos situaciones:

- $\neg Participates(A_i, t_{r+1})$: si esto ocurre basta tomar $\sigma_i = \sigma'_i$, ya que $\Pi_i(\sigma'.(t_{r+1}, \tau_{r+1})) \preceq_i^{cov} \sigma_i$ por definición de $\preceq^{cov}$ y porque $\Pi_i(\sigma') \preceq_i^{cov} \sigma'_i$.

- $Participates(A_i, t_{r+1})$: si esto ocurre sabemos que existe σ'_i, ejecución de A_i, tal que $\Pi_i(\sigma') \preceq_i^{cov} \sigma'_i$.

  De acuerdo a la definición de $\preceq_i^{cov}$ esto significa que $\Pi_i(\sigma') = \sigma_1\sigma_2$ y que $\sigma'_i = \sigma_1\sigma_3$, con σ_2 una secuencia de transiciones stutter. Sabemos que σ_1 es una ejecución de A_i, ya que es un prefijo de σ'_i (que lo es), entonces sabemos por definición de ejecución que todas las transiciones

de σ_1 cumplen las guardas de las mismas y que el source y el target de estas transiciones cumplen también su invariante. Por lo que nos basta con probar que $\sigma_1.\Pi_i((t_{r+1}, \tau_{r+1}))$ es una ejecución de A_i.

Recordemos que las restricciones sobre los relojes son conjunciones de predicados de la forma "$x \sim k$" donde $\sim \in \{\leq, \geq\}$. En lo que sigue de la demostración analizaremos cada predicado por separado (el hecho de que sea una conjunción nos permite hacerlo). Para esto consideramos los siguientes casos:

1. Supongamos que la guarda de $\pi_i(t_{r+1})$, al igual que los invariantes de $sc(\pi_i(t_{r+1}))$ y $tg(\pi_i(t_{r+1}))$ son triviales (True). Sean $e \in C$ y $e_{last} \in C'$ eventos tal que $f(e) = r + 1$, y $e_{last} = Last_i(C')$. Sabemos que $e_{last} < e$, porque e es una extensión de la configuración C'. Por definición de Δ, sabemos que $(e_{last}, e, 0) \in \Delta(C)$. Luego por propiedad 4 de f sabemos que $\tau_{r+1} = \tau_{f(e)} \geq \tau_{f(e_{last})} + 0$, y por lo tanto $\tau_{r+1} \geq \tau_{f(e_{last})}$. Con lo que queda probado que $\sigma_1.\Pi_i((t_{r+1}, \tau_{r+1}))$ es una ejecución de A_i.
2. Consideremos ahora el caso en que la guarda de $\pi_i(t_{r+1})$, el invariante del source o el invariante del target contienen un predicado de la forma "$x \leq k$". Sean $e, e_{reset} \in C'$ eventos tal que $f(e) = r + 1$, y $e_{reset} = Reset_x(C)$. En este caso $\Delta(C)$ contiene la terna $(e, e_{reset}, -k)$, por lo que se sabe que $\tau_{f(e_{reset})} \geq \tau_{f(e)} + (-k)$. Por lo tanto, $k \geq \tau_{f(e)} - \tau_{f(e_{reset})}$, y como $x = \tau_{f(e)} - \tau_{f(e_{reset})}$, se obtiene finalmente que $x \leq k$. Con lo que queda demostrado que se satisface la guarda de $\pi_i(t_{r+1})$ y los invariantes asociados. Con lo que queda probado que $\sigma_1.\Pi_i((t_{r+1}, \tau_{r+1}))$ es una ejecución de A_i.

   Observación: Es fácil ver que la demostración contempla el caso en que $\pi_i(t_{r+1})$ resetee el reloj x.
3. Si la guarda de $\pi_i(t_{r+1})$ contiene un predicado de la forma "$x \geq k$", debemos chequear que la misma se cumple. Sean nuevamente $e \in C$ y $e_{reset} \in C'$ eventos tales que $f(e) = r + 1$, y $e_{reset} = Reset_x(C')$. En este caso $\Delta(C)$ contiene la terna (e_{reset}, e, k), por lo que utilizando la propiedad 4 de f podemos ver que $\tau_{f(e)} \geq \tau_{f(e_{reset})} + k$, que es lo mismo que $\tau_{f(e)} - \tau_{f(e_{reset})} = x \geq k$. Con lo que queda probado que $\sigma_1.\Pi_i((t_{r+1}, \tau_{r+1}))$ es una ejecución de A_i.

De esta manera, hemos probado que $\forall i, 1 \leq i \leq n,\ \exists \sigma_i$ ejecución de A_i tal que $\Pi_i(\sigma) \preceq_i^{cov} \sigma_i$, n lo que ha quedado demostrado que σ es una ejecución end asynchronous de N.

□

Como hemos mencionado anteriormente, la time-feasibility de una configuración está en relación directa con la existencia de una ejecución[6] sobre la TAN asociada al branching process. El lema anterior nos permite revisar nuestra definición informal de time-feasibility y enunciar este concepto formalmente.

Definición 3.20. (Configuración Time-Feasible). *Una configuración finita C de un branching process β de una TAN N es time-feasible si y sólo si $\|C\| \neq \emptyset$.* ■

El concepto de time-feasibility se extiende de manera natural a los branching processes.

Definición 3.21. (Branching Process Time-Feasible). *Un branching process β de una TAN N es time-feasible sii todas sus configuraciones locales son time-feasible.* ■

Volvamos a considerar el grafo de pesos asociado a una configuración. En el mismo está representada la información de máximos y mínimos delays entre un evento y sus eventos relevantes. Quisiéramos extender la utilidad de este grafo para que fuera posible obtener información de los delays **entre dos eventos cualquiera** de una configuración. Es fácil observar que, dados dos nodos, puede suceder que exista más de un camino entre ellos. Sin embargo, para realizar el análisis de time-feasibility nos interesa únicamente conocer el mínimo de los delays máximos, y el máximo de los delays mínimos. Aprovechando el hecho de que los delays máximos están representados en el grafo de pesos como ejes de peso negativo (ver definición 3.18), entonces para calcular la información requerida nos alcanza con utilizar una única función $\widehat{\Delta}$ que calcule el peso máximo para todos los caminos posibles (ya que el máximo delay entre a y b, con a < b, será $|\widehat{\Delta}(b,a)|$ y el mínimo delay será $\widehat{\Delta}(a,b)$).

Si observamos las figuras 1.7 y 3.1 podemos ver que el máximo delay entre *Aproximandose* y *Abajo* es 2 que es $|\widehat{\Delta}(Abajo, Aproximandose)|$. Análogamente vemos que el mínimo delay es 1 que corresponde al valor de $\widehat{\Delta}(Aproximandose, Abajo)$.

[6] A través del lema 1 hemos probado que existen ejecuciones end asynchronous que nos llevan a una configuración dada. Sin embargo podría suceder que las mismas no fueran necesariamente ejecuciones a secas (ver definiciones 3.10 y 3.13). Pero el hecho de estar trabajando sobre EATANs nos garantiza que existen ejecuciones que nos permiten alcanzar todos los eventos de la configuración cumpliendo las restricciones temporales.

Definición 3.22. (Adjusted Delta). *Sea C una configuración de un branching process β de un TAN N. Definimos $\widehat{\Delta}(C) : C \times C \to \mathbb{Z} \cup \{-\infty\}$ como la función total que retorna el peso del camino más largo entre el primer y el segundo parámetro en el grafo asociado a la configuración enriquecido con la información temporal provista por $\Delta(C)$. En caso de que no existiera camino, retornará $-\infty$.* ■

Observación 1: Se debe hacer notar que, si en vez de los pesos originales de $\Delta(C)$ se utilizaran los valores de $\widehat{\Delta}(C)$, se preservaría la semántica original de las configuraciones temporizadas.

Observación 2: El cálculo de $\widehat{\Delta}(C)$ se puede realizar en $\mathcal{O}(n^3)$, con $n = |C|$, utilizando el algoritmo de Floyd-Warshall [PS98].

Estamos en condiciones de obtener un procedimiento de verificación de la time-feasibility de una configuración dada. Este procedimiento estará relacionado con la detección de ciclos de peso positivo en el grafo temporal asociado, ya que en ese caso el máximo delay debe ser menor que el mínimo, indicando que no existe ninguna ejecución que sea capaz de generar la configuración en cuestión. A continuación demostraremos formalmente este razonamiento.

Lema 2 (Time-feasibility). *Una configuración C tiene semántica no vacía (es time-feasible) sii $\nexists a \in C$ tal que $\widehat{\Delta}(C)(a, a) > 0$.*

Demostración. Queremos probar que C es time-feasible $\Leftrightarrow \forall a \in C$ se cumple $\widehat{\Delta}(C)(a, a) \leq 0$.

($\Rightarrow$) C es time-feasible. Esto es que $\|C\| \neq \emptyset$. O lo que es lo mismo, existe un conjunto no vacío de ejecuciones para las que existe alguna función f que satisface las propiedades de la definición 3.19.

Queremos probar que $\forall a \in C$ se cumple $\widehat{\Delta}(C)(a, a) \leq 0$. Lo hacemos por el absurdo. Supongamos que existe un $a \in C$ tal que $\widehat{\Delta}(C)(a, a) > 0$. Consideraremos el caso en que a forma parte de algún ciclo simple[7], ya que en caso contrario $\widehat{\Delta}(C)(a, a) = -\infty < 0$, lo cual constituiría un absurdo. Sea $a \to b_1 \to b_2 \to ... \to b_n \to a$ el ciclo simple de mayor peso de todos los ciclos simples que involucran a a. Por otra parte, sabemos que por definición 3.19 la función f satisface $\forall a, b \in C, k \in \mathbb{Z} : (a, b, k) \in \Delta(C) \Rightarrow \tau_{f(b)} - \tau_{f(a)} \geq k$. Por lo tanto se cumplen las siguientes desigualdades entre los nodos del ciclo:

[7]Para una definición de los conceptos de camino, ciclo y ciclo simple puede consultarse [Har72].

$$\begin{array}{lcl} \tau_{f(b_1)} & \geq & \tau_{f(a)} + k_1 \\ \tau_{f(b_2)} & \geq & \tau_{f(b_2)} + k_2 \\ & \vdots & \\ \tau_{f(b_n)} & \geq & \tau_{f(b_{n-1})} + k_{n-1} \\ \tau_{f(a)} & \geq & \tau_{f(b_n)} + k_n \end{array}$$

de las cuales, haciendo las sustituciones adecuadas se obtiene

$\tau_{f(b_n)} \geq \tau_{f(b_{n-1})} + k_{n-1} \geq \tau_{f(b_{n-2})} + k_{n-2} + k_{n-1} \geq ... \geq \tau_{f(a)} + k_1 + ... + k_{n-1}$
$\tau_{f(a)} \geq \tau_{f(b_n)} + k_n$

o sea

$$\tau_{f(b_n)} \geq \tau_{f(a)} + k_1 + k_2 + ... + k_{n-1}$$
$$\tau_{f(a)} \geq \tau_{f(b_n)} + k_n$$

Sustituyendo nuevamente obtenemos que

$$\tau_{f(a)} \geq \tau_{f(b_n)} + k_n \geq \tau_{f(a)} + k_1 + k_2 + ... + k_{n-1}$$

lo que es equivalente a

$$0 \geq k_1 + k_2 + ... + k_{n-1} + k_n$$

Con lo cual el ciclo simple en el que se encuentra a tiene peso negativo. Debido a que para este razonamiento se utilizó el ciclo simple de mayor peso que involucra a podemos concluir que $\widehat{\Delta}(C)(a, a) \leq 0$, lo que constituye un absurdo.

($\Leftarrow$) Sabemos que $\forall a \in C$ se cumple $\widehat{\Delta}(C)(a, a) \leq 0$. Queremos ver que entonces $\|C\| \neq \emptyset$, o sea que existe una ejecución para la cual existe una f que satisface las propiedades mencionadas.

Proponemos el siguiente algoritmo que construye tanto una ejecución r como una función f que satisfacen las propiedades requeridas:

input: Una configuración C de una TAN N
output: Una función f y una ejecución r de N
var B, D: conjunto de eventos
begin
    $r \leftarrow \langle\rangle$
    $D := \emptyset$

$B := Min(C)$

while $B \neq \emptyset$
para cada $b \in B$
sea $Delay(b) = max\ \{\ \{\tau_{f(a)} + k / (\exists t \in r : h(a) \equiv_{stutter} t) \wedge (a,b,k) \in \Delta(C)\}$
$\cup$
$\{k / (\bot, b, k) \in \Delta(C)\}\ \}$
elegir b tq ${}^\bullet({}^\bullet b) \subseteq D$ y $Delay(b)$ sea mínimo
definir $f(b) := |r| + 1$
definir $\tau_{f(b)} := Delay(b)$
$r := append(r, (h(b), \tau_{f(b)}))$
$D := D \cup \{b\}$
$B := ((B \setminus \{b\}) \cup (b^\bullet)^\bullet) \cap C$
endwhile
end

Algunos comentarios sobre el algoritmo propuesto:

- la función variante del ciclo es $|D| - |C|$
- el invariante del ciclo es: "r es una ejecución de la configuración D y B contiene el postset del $Cut(D)$"
- al tomar b como segunda coordenada de las ternas de $\Delta(C)$ se garantiza que k sea positivo
- se debe hacer notar que tanto B como D son subconjuntos de C

Presentado el algoritmo veamos entonces que la función f así definida satisface para la ejecución r las propiedades mencionadas.

Las propiedades (1), (2) y (3) se satisfacen trivialmente por construcción.

La propiedad (4) se satisface trivialmente por construcción para todos los ejes con $k \geq 0$.

Lo único que faltaría verificar es que se cumpla la propiedad (4) para los k de valor negativo. En este caso sabemos que el k negativo forma un ciclo simple en $\Delta(C)$. Sea $a \to b_1 \to b_2 \to ... \to b_n \to a$ ese ciclo. Es fácil ver que la f generada por el algoritmo cumple las siguientes igualdades:

$$\begin{aligned} \tau_{f(b_1)} &= \tau_{f(a)} + k_1 \\ \tau_{f(b_2)} &= \tau_{f(b_2)} + k_2 \\ &... \\ \tau_{f(b_n)} &= \tau_{f(b_{n-1})} + k_{n-1} \end{aligned}$$

Por hipótesis sabemos que $\widehat{\Delta}(C)(a,a) \leq 0$. Con lo que tenemos que vale también la siguiente desigualdad

$$k_1 + k_2 + ... + k_{n-1} + k_n \leq 0$$

Sumando $\tau_{f(a)}$ a ambos lados de la desigualdad obtenemos

$$\tau_{f(a)} + k_1 + k_2 + ... + k_{n-1} + k_n \leq \tau_{f(a)}$$

con lo que podemos aplicar la sustitución

$$\tau_{f(b_1)} + k_2 + ... + k_{n-1} + k_n \leq \tau_{f(a)}$$

Repitiendo este procedimiento para cada uno de los nodos del ciclo, obtenemos finalmente que

$$\tau_{f(b_n)} + k_n \leq \tau_{f(a)}$$

y por lo tanto la función f propuesta satisface la propiedad (4) $\forall (a,b,k) \in \Delta(C)$, con k negativo. Con lo que queda demostrado que $\|C\| \neq \emptyset$, ya que existe una ejecución r y una función f generadas por nuestro algoritmo tales que $r \in \|C\|$.

□

Damos por finalizada de esta forma nuestra búsqueda de un criterio que nos permita determinar si la extensión de una configuración con un nuevo evento tiene como resultado una nueva configuración time-feasible. El próximo paso es formalizar el mecanismo que nos permita ir construyendo el branching process incrementalmente, lo que nos lleva a adaptar la definición de extensión posible (definición 2.25) a la de *extensión posible time-feasible.*

Definición 3.23. (Extensión Posible Time-Feasible). *Sea $\beta = \langle ON, h \rangle$ un branching process de una TAN N. Diremos que el par $e = (t, X)$ es una extensión posible time-feasible de β si:*

- *e es una extensión posible de β*
- *$\lceil e \rceil$ es time-feasible en $\beta \oplus \{e\}$*

Denotaremos $EPTF(\beta)$ al conjunto de extensiones posibles time-feasibles del branching process β. ■

Observación: se debe destacar que la definición de branching process extendido (definición 2.26) también es aplicable a extensiones posibles time-feasibles.

Al igual que en el contexto no temporizado resulta cómodo poder expresarnos sobre las configuraciones, sus extensiones y el resultado de la aplicación de estas últimas. Por este motivo extendemos las definiciones de extensión y sufijo de una configuración (definición 2.27) a sendas versiones temporizadas.

Definición 3.24. (Extensión y Sufijo de una Configuración [versión temporizada]). *Sea C una configuración finita en un branching process $\beta = \langle < B, E, F >, h \rangle$ y S un conjunto finito de eventos $S \subseteq E$. Diremos que $C \oplus S$ es una extensión time-feasible de C y que S es un sufijo de C si $C \cup S$ es una configuración time-feasible en β tal que $C \cap S = \emptyset$.* ■

Antes de pasar a la próxima sección se considera conveniente hacer un breve balance de lo obtenido hasta el momento. Nos encontramos en el punto donde tenemos los elementos necesarios para plantear un algoritmo que construya el unfolding de una TAN. Sin embargo, aún no podemos encarar la construcción de **un prefijo completo del mismo**, ya que todavía no disponemos de algunas nociones importantes. Entre ellas la equivalencia temporal entre configuraciones, que nos permitiría obtener un criterio de corte para el unfolding. En la siguiente sección presentaremos los elementos necesarios para alcanzar este objetivo.

3.4 Construcción del Prefijo

De la misma manera que en el capítulo anterior, debemos definir el criterio de corte que nos permita detener la aplicación de extensiones al branching process.

Para poder discernir si una extensión agrega información nueva o no vamos a analizar su configuración local. Este análisis va a considerar dos aspectos de la configuración: su marking y las distancias temporales entre sus puntos relevantes. La conjunción de ambos aspectos define una relación de equivalencia entre configuraciones que será formalizada más adelante.

El marking, al igual que en el contexto no temporizado, es necesario para conocer el estado actual del modelo y poder determinar de esta forma cuáles son las transiciones habilitadas (sin considerar las restricciones temporales). La distancias temporales entre los puntos relevantes se requieren para conocer cuáles son las transiciones que satisfacen las restricciones temporales de los invariantes y las guardas. Esto es necesario debido a que puede darse

el caso en que, dado un mismo marking, distintos valores de las distancias temporales determinen que una de las configuraciones verifique las restricciones temporales mientras que la otra no, condicionando de esta manera las posibles evoluciones del sistema.

Sin embargo existe una situación donde la distancia temporal entre dos puntos relevantes temporal podría ser diferente para dos configuraciones, pero en la que esta diferencia no es determinante para que una de ellas verifique las restricciones y la otra no. Este es el caso en el que uno de los eventos satisface que, para todos los relojes para los cuales es *Reset* las valuaciones de estos últimos son mayores a la constante máxima del sistema. En esta situación el paso del tiempo no podrá hacer que los valores de los relojes se vean afectados de manera tal que afecte la satisfactibilidad de guardas e invariantes. Esta idea se formalizará en breve en la definición de *TooFar*. Sin embargo, es necesario que antes presentemos algunas definiciones adicionales.

Comenzaremos definiendo formalmente las nociones de constante máxima de un TA y de una TAN.

Definición 3.25. (Constante Máxima de un TA). *Dado un TA A definiremos Max como*

$$\begin{aligned} Max(A) = \ & \max \{\ k \in \mathbb{N} \ / \ \exists q \in Q, t \in T, x \in R : \\ & (\text{“}x \leq k\text{”} \in \iota(q)) \ \vee (\text{“}x \leq k\text{”} \in \gamma(t)) \ \vee \ (\text{“}x \geq k\text{”} \in \gamma(t))\} \end{aligned}$$

es decir, Max es la máxima constante que aparece en alguna de las restricciones temporales del TA A. ■

Definición 3.26. (Constante Máxima de una TAN). *Dada una TAN N definiremos Max como*

$$Max(N) = \max_{1 \leq i \leq n} \{Max(A_i)\}$$

es decir, Max es la máxima constante que aparece en alguna de las restricciones temporales de la TAN N. ■

Diremos que los relojes que han sobrepasado la constante máxima del sistema se han convertido en $TooLarge$.

Definición 3.27. (Too Large). *Dada una configuración C de un branching process β de una TAN N, definiremos*

$TooLarge_x(C)$ sii $\widehat{\Delta}(C)(Reset_x(C), Last_i(C)) > Max(N)$, donde i tq $x \in R_i$

■

Como hemos mencionado anteriormente, estamos especialmente interesados en aquellas situaciones en las que un evento satisface que, para todos los relojes para los cuales es *Reset*, las valuaciones de estos últimos son mayores que $Max(N)$. Lo que genera la necesidad de extender la noción de $TooLarge$ para todos los relojes para los cuales el evento es *Reset*, necesidad que se formaliza en el concepto de evento $TooFar$.

Definición 3.28. (Too Far). *Dada una configuración C de un branching process β de una TAN N, y dado un evento e que es reset para algún reloj x ($\exists i, 1 \leq i \leq n : \exists x \in R_i$ tq $e = Reset_x(C)$), diremos que e es TooFar en C si se verifica el siguiente predicado:*

$$TooFar(e, C) \text{ sii } \forall x \in R \text{ tq } e = Reset_x(C) : TooLarge_x(C)$$

Es decir, dado un evento e que es $TooFar$ en una configuración, para todos los relojes tal que e es el "último" evento que los resetea, se verifica que los mismos son $TooLarge$ en esa configuración. ■

Queda un último paso antes de poder concretar nuestra búsqueda de una equivalencia entre configuraciones que considere el aspecto temporal. Dadas dos configuraciones, deberíamos poder garantizar que, si son equivalentes, entonces si extendemos a una de ellas con un sufijo time-feasible, y nos desplazamos con el isomorfismo correspondiente hacia un sufijo de la otra, entonces este sufijo va a ser time-feasible. Es decir, dadas dos configuraciones equivalentes deberíamos poder garantizar que es posible extender una de ellas con los sufijos isomorfos a las extensiones de la otra. Esta intuición será formalizada en breve en el concepto de *reachability equivalence* entre configuraciones. Para poder lograr esto, se hace necesario definir la *cobertura de una configuración.*

Definición 3.29. (Cobertura de una Configuración). *Dadas dos configuraciones time-feasible C_1, C_2 de un branching process β, diremos que C_1 cubre a C_2 si se cumple:*

- *Marking(C_1) = Marking (C_2)*
- $\forall$ *sufijo E, si E es sufijo time-feasible de C_2 entonces $I_{C_2}^{C_1}E$ es sufijo time-feasible de C_1*

■

Finalmente, estamos en condiciones de formalizar la noción de equivalencia (respecto a la alcanzabilidad) entre configuraciones de un branching process, la cual constituye la definición más importante de esta sección.

Definición 3.30. (Reachability Equivalence). *Dadas dos configuraciones time-feasible C_1, C_2 de un branching process β, diremos que C_1 es reachability equivalent a C_2 si C_1 cubre a C_2 y C_2 cubre a C_1.* ■

Si bien la definición anterior es correcta desde el punto de vista formal y satisface nuestras necesidades, presenta el inconveniente de que dista mucho de ser un método computable. Por lo que introduciremos una noción de equivalencia más "algorítmica".

Definición 3.31. (Equivalencia entre Configuraciones [versión temporizada]). *Sean C y C' dos configuraciones time-feasible finitas de un branching process β de una TAN N. Diremos que C es equivalente a C', denotado $C \equiv C'$, si se satisfacen las siguientes condiciones*

- $Marking(C) = Marking(C')$
- $\forall i, 1 \leq i \leq n,\ x \in R_i$:
  $TooFar(Reset_x(C), C) \iff TooFar(Reset_x(C'), C')$
- $\forall i, j, 1 \leq i, j \leq n,\ x \in R_i, y \in R_j$:
  - $\widehat{\Delta}(C)(Last_i(C), Last_j(C)) = \widehat{\Delta}(C')(Last_i(C'), Last_j(C'))$
  - $\widehat{\Delta}(C)(Last_i(C), Reset_x(C)) = \widehat{\Delta}(C')(Last_i(C'), Reset_x(C'))$
    $\vee\ TooFar(Reset_x(C), C)$
  - $\widehat{\Delta}(C)(Reset_x(C), Last_i(C)) = \widehat{\Delta}(C')(Reset_x(C'), Last_i(C'))$
    $\vee\ TooFar(Reset_x(C), C)$
  - $\widehat{\Delta}(C)(Reset_x(C), Reset_y(C)) = \widehat{\Delta}(C')(Reset_x(C'), Reset_y(C'))$
    $\vee\ TooFar(Reset_x(C), C)\ \vee\ TooFar(Reset_y(C), C)$

■

A continuación demostraremos que la equivalencia entre configuraciones (definición teórica) implica la reachability equivalence (definición algorítmica).

Teorema 1. *Si $C \equiv C'$ entonces C es reachable equivalent a C'.*

Demostración. Queremos probar que si C y C' tienen el mismo marking y las mismas distancias temporales, entonces tienen exactamente los mismos sufijos time-feasibles.

Tenemos como hipótesis que $C \equiv C'$, por lo que sabemos que los markings son equivalentes, y por lo tanto la primera parte de la definición de reachability equivalence (definición 3.30) se satisface trivialmente.

Veamos entonces que los sufijos time-feasible de una configuración son sufijos de la otra (aplicando el isomorfismo correspondiente). Lo demostramos por inducción en la cantidad de elementos de los sufijos.

Caso Base: $E = \{e\}$. Suponemos que $\{e\}$ es un sufijo time-feasible de C (es decir que $C \oplus \{e\}$ es una extensión time-feasible de C). Queremos probar que $I_C^{C'}\{e\}$ es un sufijo time feasible de C'. Por Lema 2, sabemos que una configuración C es time feasible $\Leftrightarrow \forall a \in C$ vale $\widehat{\Delta}(C)(a,a) \leq 0$. Por definición de $\Delta(C)$ sabemos que, para cada autómata A_i, e va a recibir arcos desde el evento inmediatamente anterior ($Last_i(C)$) y desde/hacia los $Reset_x(C)$ de los relojes involucrados en la guarda de $\pi_i(h(e))$. Sabemos que el peso de estos arcos va a ser el mismo para e que para $I_C^{C'}\{e\}$ (ya que la guarda y el invariante son los mismos porque $\pi_i(h(e)) = \pi_i(h(I_C^{C'}\{e\}))$ para cada $1 \leq i \leq n$). Además sabemos que en C' todos los ciclos tienen peso menor o igual a cero (ya que C' es time-feasible), por lo que si en $C' \oplus I_C^{C'}\{e\}$ hubiera un ciclo de longitud estrictamente positiva ese ciclo tendría que involucrar a $I_C^{C'}\{e\}$.

Figura 3.2: Esquema del ciclo

Puede suceder uno de los siguientes tres casos[8]:

- a' es $Last_i(C')$ y b' es $Reset_x(C')$
- a' es $Last_i(C')$ y b' es $Last_j(C')$
- a' es $Reset_x(C')$ y b' es $Reset_y(C')$

Sean a y b los eventos que cumplen los mismos roles que a' y b' en C. (a' y b' pueden ser el mismo nodo. Lo mismo sucede con a y b).

En cualquiera de estos casos se cumple que $\widehat{\Delta}(C)(b,a) = \widehat{\Delta}(C')(b',a')$. Como además los pesos de k_1 y $-k_2$ son los mismos en C y en C', y sabiendo que $\forall a \in C \oplus \{e\} : \widehat{\Delta}(C)(a,a) \leq 0$, podemos concluir que $\forall a' \in C' \oplus \{e\} : \widehat{\Delta}(C' \oplus I_C^{C'}\{e\})(a',a') \leq 0$, con lo que queda demostrado que $I_C^{C'}\{e\}$ es sufijo time-feasible de C'.

Se debe hacer notar que en el razonamiento anterior se utilizó que las distancias temporales entre a y b en C eran las mismas que las que existen entre a' y b' en C'. Sin embargo puede darse el caso de que estas distancias no sean las mismas en ambas configuraciones pero que la equivalencia entre ellas siga valiendo porque a o b (y análogamente a' o b') sean $TooFar$ cuando cumplen el rol de $Reset$; invalidando de esta forma el razonamiento anterior.

Antes de examinar con mayor detenimiento lo que sucede en este caso, vamos a probar que si $C \equiv C'$ por $TooFar$, entonces los ciclos que involucran a eventos que son $TooFar$ tienen peso positivo.

Figura 3.3: TooFar

El evento r es $Reset_x(C)$. El peso del ciclo que involucra a r es $K + k_1 + (-k_2)$. Como $K > Max(N)$, donde $Max(N)$ es la mayor constante que aparece en las guardas e invariantes de la TAN N, podemos decir que $K > k_1$ y $K > k_2$. De estas desigualdades surge que

[8]El caso restante, a' es $Reset_x(C')$ y b' es $Last_i(C')$, no se toma en cuenta ya que si existe un eje desde e hacia b', entonces b' debe ser necesariamente un $Reset$ (además de ser $Last$), lo que se reduciría al tercer caso.

$$K + k_1 + (-k_2) > K + (-k_2)$$

y por otro lado

$$K > k_2 \Rightarrow K + (-k_2) > 0$$

Por lo que el ciclo $e \to r \to ... \to Last_i(C) \to e$ tiene peso positivo. Pero por hipótesis $C \oplus \{e\}$ era time-feasible, por lo que todos sus ciclos deben tener peso negativo. De lo que se deduce que no puede existir un ciclo en $C \oplus \{e\}$ que involucre a un $Reset_x$ que sea $TooFar$, y por lo tanto en C como en C' se verifican las mismas distancias temporales y la demostración anterior es válida.

Paso Inductivo: Supongamos que la propiedad vale para todo sufijo time-feasible E tal que $|E| \leq k$. Sea $E' = \{e_1, e_2, ..., e_k, e_{k+1}\}$ un sufijo time-feasible de $k+1$ elementos, donde e_{k+1} es maximal. Sean C y C' dos configuraciones tales que $C \equiv C'$. Queremos probar que si $C \oplus E'$ es time-feasible entonces $C' \oplus I_C^{C'} E'$ también lo es.

$$C \oplus E' = (C \oplus \{e_1, e_2, ..., e_k\}) \oplus \{e_{k+1}\}$$

Sabemos que $C \oplus \{e_1, e_2, ..., e_k\}$ es una configuración ya que e_{k+1} es maximal. Aplicando entonces el lema auxiliar de congruencia (que se presenta a continuación del presente teorema) obtenemos que

$$C \oplus \{e_1, e_2, ..., e_k\} \equiv C' \oplus I_C^{C'}\{e_1, e_2, ..., e_k\}$$

Llamando C_1 y C_2 a la configuración de la izquierda y de la derecha respectivamente, la propiedad a demostrar se puede re-escribir como

$$C_1 \oplus \{e_{k+1}\} \text{ es time-feasible} \Rightarrow C_2 \oplus I_C^{C'}\{e_{k+1}\} \text{ es time-feasible}$$

Pero $|\{e_{k+1}\}| = 1 \leq k$, por lo tanto estamos en condiciones de aplicar la hipótesis inductiva, lo que demuestra la validez de la propiedad.

□

Lema 3 (Lema Auxiliar de congruencia). *Sean C y C' dos configuraciones time-feasible. Si $C \equiv C'$, E es un sufijo time-feasible de C y $I_C^{C'} E$ es sufijo time-feasible de C', entonces $C \oplus E \equiv C' \oplus I_C^{C'} E$.*

Demostración. Lo demostramos por inducción en la cantidad de elementos del sufijo E.

Caso Base: Sea $E = \{e\}$. Sabemos que los markings de ambas configuraciones son iguales, ya que $C \equiv C'$ y estamos extendiendo a ambas con sufijos isomorfos. Quedaría probar que las máximas distancias temporales entre los eventos relevantes de ambas configuraciones son iguales entre sí, o que los $Resets$ son $TooFar$.

Sabemos que $h(e) = h(I_C^{C'}\{e\})$, por lo que los arcos que agrega e en $\Delta(C \oplus \{e\})$ están presentes en $\Delta(C' \oplus I_C^{C'}\{e\})$. Estos arcos involucran exclusivamente a eventos que son $Last_i(C)$ o $Reset_x(C)$, para algún reloj x y algún $1 \leq i \leq n$. Y lo que es más, esos arcos relacionan a $I_C^{C'}\{e\}$ con eventos de C' que cumplen el mismo rol que los eventos de C relacionados con e (es decir que si existe un arco entre e y $Reset_x(C)$, existirá un arco equivalente entre $I_C^{C'}\{e\}$ y $Reset_x(C')$; y que si existe un arco entre e y $Last_i(C)$, existirá uno entre $I_C^{C'}\{e\}$ y $Last_i(C')$). Por último, agregaremos que los arcos relacionados entre sí en ambas configuraciones tienen el mismo peso (Notaremos $weight(a, b)$ al peso del arco entre a y b).

Por lo tanto queremos ver que $\widehat{\Delta}(C \oplus \{e\})(p, q) = \widehat{\Delta}(C' \oplus I_C^{C'}\{e\})(p', q')$, para todo p y q tales que son relevantes $C \oplus \{e\}$, y p' e q' que cumplen el mismo rol en $C' \oplus I_C^{C'}\{e\}$ [9].

Consideraremos entonces los siguientes casos:

- $\widehat{\Delta}(C \oplus \{e\})(p, q) = -\infty$: En este caso no existe un camino entre p y q. Supongamos que $\widehat{\Delta}(C' \oplus \{e'\})(p', q') = k > -\infty$, es decir, que en $C' \oplus \{e'\}$ existe un camino entre p' y q'.

  Si e' no pertenece a ese camino entonces $k = \widehat{\Delta}(C' \oplus \{e'\})(p', q') = \widehat{\Delta}(C')(p', q') = \widehat{\Delta}(C)(p, q) = -\infty$, lo cual constituye un absurdo.

  Si e' pertenece a ese camino, el camino es de la forma[10]

  $$p' \rightsquigarrow a' \rightarrow e' \rightarrow b' \rightsquigarrow q'$$

  con a' y b' relevantes en C'. Llamaremos a y b a los eventos de C que cumplen los mismos roles que a' y b'.

  Observación: p' y a' pueden ser el mismo nodo, así como q' y b'. Al mismo tiempo todos ellos pueden ser iguales a e'.

[9] Por comodidad, a partir de este momento denotaremos $I_C^{C'}\{e\}$ como e'.

[10] En lo que sigue de la demostración notaremos $x \rightarrow y$ cuando exista un eje entre x e y (es decir, $(x, y, k) \in \Delta(C)$), mientras que utilizaremos $x \rightsquigarrow y$ para indicar que existe un camino entre x e y (que involucra a uno o más ejes).

En este caso podemos escribir la distancia entre p' y q' como

$$\begin{aligned} k &= \widehat{\Delta}(C' \oplus \{e'\})(p', q') \\ &= \widehat{\Delta}(C' \oplus \{e'\})(p', a') + weight(a', e') + weight(e', b') + \\ &\quad + \widehat{\Delta}(C' \oplus \{e'\})(b', q') \\ &= \widehat{\Delta}(C')(p', a') + weight(a', e') + weight(e', b') + \widehat{\Delta}(C')(b', q') \\ &= \widehat{\Delta}(C)(p, a) + weight(a, e) + weight(e, b) + \widehat{\Delta}(C)(b, q) \\ &= \widehat{\Delta}(C \oplus \{e\})(p, q) \\ &= -\infty \end{aligned}$$

lo cual constituye un absurdo.

- $\widehat{\Delta}(C \oplus \{e\})(p, q) = k > -\infty$: En este caso existe un camino entre p y q. Consideraremos entonces los siguientes subcasos:

  1. $e = p$: Sea $e \rightarrow z \rightsquigarrow q$ un camino entre p y q de longitud k. Llamaremos z al primer nodo que se encuentra en el camino de p a q. Se debe hacer notar que z es un punto relevante en C por definición 3.18. Sea $e' \rightarrow z' \rightsquigarrow q'$ el camino análogo en C', es decir, el camino entre p' y q' donde z' es el nodo que cumple el mismo rol que z en C' (y por lo tanto z' es relevante en C'). Consideramos las siguientes situaciones:

     (a) ni z ni q son $TooFar$ en C:
     En este caso vale que

     $$\begin{aligned} &\widehat{\Delta}(C \oplus \{e\})(e, q) = \\ &weight(e, z) + \widehat{\Delta}(C \oplus \{e\})(z, q) = \\ &weight(e, z) + \widehat{\Delta}(C)(z, q) = \\ &weight(e', z') + \widehat{\Delta}(C')(z', q') \leq \widehat{\Delta}(C' \oplus \{e'\})(e', q') \end{aligned}$$

     Aplicando un razonamiento similar partiendo de C' se obtiene que $\widehat{\Delta}(C' \oplus \{e'\})(e', q') \leq \widehat{\Delta}(C \oplus \{e\})(e, q)$. Con lo que queda demostrada la igualdad, y por lo tanto $\widehat{\Delta}(C' \oplus \{e'\})(e', q') = k$.

     (b) z no es $TooFar$ en C y q es $TooFar$ en C:
     Bajo esta hipótesis se pueden presentar dos situaciones:

     - q es $Last$ en $C \oplus \{e\}$: entonces sabemos que q también es Last de C. Y como tanto q como z son puntos relevantes en C, y por lo tanto es válido el análisis presentado en el item anterior de la demostración.

– q no es Last en $C \oplus \{e\}$: entonces q debe ser Reset de $C \oplus \{e\}$, ya que de otra manera no sería un punto relevante. Sea x uno de los relojes que hacen que q sea $Reset$ de $C \oplus \{e\}$[11], y sea $A_i, 1 \leq i \leq n$, el autómata al cual pertenece x ($x \in R_i$). Valen entonces las siguientes desigualdades:

$$\widehat{\Delta}(C \oplus \{e\})(Reset_x(C \oplus \{e\}), Last_i(C \oplus \{e\})) \geq {}^{12}$$
$$\widehat{\Delta}(C \oplus \{e\})(Reset_x(C \oplus \{e\}), Last_i(C)) =$$
$$\widehat{\Delta}(C \oplus \{e\})(q, Last_i(C)) =$$
$$\widehat{\Delta}(C)(q, Last_i(C)) > Max(N)$$

Como el razonamiento anterior se puede aplicar a cada uno de los relojes para los cuales q es $Reset$ en $C \oplus \{e\}$, podemos concluir que q sigue siendo $TooFar$ en $C \oplus \{e\}$. Adicionalmente, el razonamiento anterior puede ser utilizado para demostrar que q' también es $TooFar$ en $C' \oplus \{e'\}$.

2. $e = q$: Sea $p \rightsquigarrow z \rightarrow e$ un camino entre p y q de longitud k. Llamaremos z al último nodo que se encuentra en el camino de p a q. Se debe hacer notar que z es un punto relevante en C por definición 3.18. Sea $p' \rightsquigarrow z' \rightarrow e'$ el camino análogo en C', es decir, el camino entre p' y q' donde z' es el nodo que cumple el mismo rol que z en C' (y por lo tanto z' es relevante en C').

   Consideramos las siguientes situaciones:

   (a) ni z ni p son $TooFar$ en C:
   Este caso se demuestra de manera análoga al 1.(a).

   (b) z no es $TooFar$ en C y p es $TooFar$ en C:
   Este caso se demuestra de manera análoga al 1.(b).

   (c) z es $TooFar$ en C:
   Sea x uno de los relojes que hacen que z sea $Reset$ de $C(z = Reset_x(C))$, y sea $A_i, 1 \leq i \leq n$ el autómata al cual pertenece x ($x \in R_i$). En este caso, como sabemos que z es $TooFar$, sabemos que $\widehat{\Delta}(C)(z, Last_i(C)) > Max(N)$, por lo tanto se cumple que $\widehat{\Delta}(C)(z, Last_i(C)) > weight(z, e)$, y por lo tanto el camino de longitud máxima entre p y e no va a tener como

[11] Se debe hacer notar que bajo estas suposiciones, el evento q sería al mismo tiempo $Reset_x(C)$.

[12] En en el caso en que e no es $Last_i(C \oplus \{e\})$ la desigualdad vale trivialmente ya que vale la igualdad. En caso contrario, $\widehat{\Delta}(C \oplus \{e\})(q, e) \geq \widehat{\Delta}(C \oplus \{e\})(q, Last_i(C))$ ya que $\Delta(C \oplus \{e\})(Last_i(C), e) \geq 0$ por causalidad.

anteúltimo nodo a z, lo que contradice la suposición de que z era $TooFar$. Lo mismo ocurre con z'.

3. $e \neq p$ y $e \neq q$ ($e' \neq p'$ y $e' \neq q'$):
   Si el camino entre p y q no pasa por e, vale que

   $$\widehat{\Delta}(C \oplus \{e\})(p,q) = \widehat{\Delta}(C)(p,q) = \widehat{\Delta}(C')(p',q')) = \widehat{\Delta}(C' \oplus \{e'\})(p',q')$$

   con lo que queda demostrada la propiedad.
   Si el camino entre p y q pasa por e, utilizando las demostraciones de los casos anteriores vale que

   $$\begin{array}{l} \widehat{\Delta}(C \oplus \{e\})(p,q) = \\ \widehat{\Delta}(C \oplus \{e\})(p,e) + \widehat{\Delta}(C \oplus \{e\})(e,q) = \\ \widehat{\Delta}(C' \oplus \{e'\})(p',e') + \widehat{\Delta}(C' \oplus \{e'\})(e',q') = \\ \widehat{\Delta}(C' \oplus \{e'\})(p',q') \end{array}$$

   con lo que queda demostrada la propiedad (en el caso en que p o q sean $TooFar$, la propiedad sigue siendo válida ya que p' o q' son $TooFar$, de acuerdo a lo demostrado en los casos anteriores).

De esta manera queda probado que $C \oplus \{e\} \equiv C' \oplus I_C^{C'}\{e\}$, con lo que queda demostrado el caso base de la inducción.

Paso Inductivo: Supongamos que la propiedad vale para todo sufijo time-feasible E tal que $|E| \leq k$. Sea $E' = \{e_1, e_2, ..., e_k, e_{k+1}\}$ un sufijo time-feasible de $k+1$ elementos, donde e_{k+1} es maximal. Sean C y C' dos configuraciones tales que $C \equiv C'$, y E' y su isomorfismo son sufijos time-feasibles de C y de C' respectivamente. Veamos que $C \oplus E' \equiv C' \oplus I_C^{C'}E'$. Sabemos que

$$C \oplus E' = C \oplus \{e_1, e_2, ..., e_k, e_{k+1}\} = (C \oplus \{e_1, e_2, ..., e_k\}) \oplus \{e_{k+1}\}$$

Podemos garantizar que $C \oplus \{e_1, e_2, ..., e_k\}$ es una configuración ya que e_{k+1} es maximal). Por lo que aplicando la hipótesis inductiva

$$C \oplus \{e_1, e_2, ..., e_k\} \equiv C' \oplus I_C^{C'}\{e_1, e_2, ..., e_k\}$$

Tomando $C \oplus \{e_1, e_2, ..., e_k\}$ y $C' \oplus I_C^{C'}\{e_1, e_2, ..., e_k\}$ como los nuevos C y C' respectivamente, y $E' = \{e_{k+1}\}$ el paso inductivo queda probado aplicando una vez más la Hipótesis Inductiva.

□

Hasta el momento contamos con dos procedimientos para determinar si dos configuraciones son equivalentes entre sí: el primero, más descriptivo, que utiliza la definición de reachability equivalence entre configuraciones (definición 3.30); y el segundo, más fácil de ser transformado en un algoritmo, basado en la equivalencia entre configuraciones (definición 3.31). Adicionalmente, utilizando el teorema 1 recién enunciado, hemos demostrado que el segundo de ellos implica al primero.

Sin embargo, nuestra búsqueda de un criterio de corte para el algoritmo que construye un prefijo completo del branching process aún no ha finalizado. Al igual que en el contexto no temporizado, se hace necesario contar con un concepto que nos permita indicar que la aplicación de una posible extensión al branching process construido hasta el momento llevaría al mismo a una situación que ya se ha presentado anteriormente. Donde esto último significa que dentro del unfolding existe otra configuración equivalente, y que ésta es menor que la configuración resultante de aplicar la extensión posible.

Siguiendo la misma línea de trabajo presentado en el capítulo anterior utilizaremos la noción de cut-off (definición 2.31), la que extenderemos apropiadamente junto con la definición de orden adecuado (definición 2.30) para adecuarlas al contexto temporizado.

Definición 3.32. (Orden Adecuado [versión temporizada]). *Un orden parcial $\prec$ sobre configuraciones finitas time-feasible de un branching process es un orden adecuado si:*

- *$\prec$ es un orden bien fundado (well-founded)*
- *$\prec$ refina a $\subset$ (es decir, $C_1 \subset C_2 \Rightarrow C_1 \prec C_2$)*
- *$\prec$ es preservado por extensiones finitas (es decir que si $C_1 \prec C_2$ y C_1 es reachability equivalent a C_2, si E es un sufijo time-feasible entonces $C_1 \oplus E \prec C_2 \oplus I_{C_1}^{C_2}(E)$)*

■

Se debe hacer notar que cualquier orden adecuado definido para el caso no temporizado es adecuado según esta definición.

Definición 3.33. (Evento Cut-off [versión temporizada]). *Sea β un branching process y $\prec$ un orden parcial adecuado sobre las configuraciones de β. Un evento e es un cut-off (en relación a $\prec$) si β contiene una configuración local $\lceil e' \rceil$ tal que:*

- *$\lceil e' \rceil$ es reachability equivalent a $\lceil e \rceil$*

- $\lceil e' \rceil \prec \lceil e \rceil$

■

Al igual que en el contexto no temporizado, valiéndonos de algún orden adecuado que nos resulte conveniente seremos capaces de obtener una caracterización de los eventos cut-off, y consecuentemente un criterio de corte para la construcción de un prefijo del unfolding.

A continuación presentaremos un algoritmo que construye un prefijo completo del unfolding de una EATAN. El mismo está basado en el algoritmo 2 presentado en el capítulo anterior.

Algoritmo 3 (Prefijo Completo [versión temporizada]).

input: Una EATAN $N = \langle A_1, A_2, ..., A_n \rangle$.
output: Un prefijo completo Fin del unfolding de N.

begin
$Fin := InitialConditions(N)$
$ep :=$ EPTF(Fin)
$cutOffs := \emptyset$

while $ep \neq \emptyset$
elegir $e \in ep$ tq $\lceil e \rceil$ es minimal respecto a $\prec$
if $(\lceil e \rceil \cap cutOffs) = \emptyset$ **then**
$Fin := Fin \oplus \{e\}$
$ep :=$ EPTF(Fin)
if e es un cut-off de Fin **then**
$cutOffs := cutOffs \cup \{e\}$
endif
else
$ep := ep \setminus \{e\}$
endif
endwhile
end

Existen dos diferencias importantes entre el algoritmo recién presentado y su análogo del capítulo anterior: la primera de ellas corresponde a la verificación de time-feasibility que realiza el nuevo algoritmo en el calculo de las extensiones posibles; la segunda está relacionada con el criterio utilizado

para determinar si el evento e es un cut-off de Fin, ya que en el contexto temporizado cambia la noción de equivalencia entre configuraciones.

Damos por finalizada de esta manera nuestra búsqueda de un algoritmo para la construcción de un prefijo completo del unfolding de una EATAN, así como del marco teórico de nuestro trabajo. En el siguiente capítulo realizaremos el cierre de este desarrollo teórico presentando una demostración de la correctitud del algoritmo 3 junto con algoritmos que realicen los procedimientos auxiliares utilizados por el mismo.

Capítulo 4

Algoritmos y Análisis

En el presente capítulo completaremos el desarrollo teórico de este trabajo demostrando que el algoritmo que construye un prefijo completo del unfolding de una EATAN es un procedimiento correcto. Adicionalmente se presenta un algoritmo para la verificación de propiedades de alcanzabilidad basado en el anterior. Por último, se discutirán posibles cambios que nos permitirán obtener versiones más eficientes del mismo, así como el diseño de una herramienta que lo implemente.

4.1 Demostraciones

Comenzaremos demostrando la correctitud del algoritmo 3, esto es, que realmente construye un prefijo completo de un branching process.

Proposición 3. *El prefijo Fin generado por el algoritmo 3 es un prefijo completo.*

Demostración. [1]Antes que nada, Fin es un prefijo por construcción, ya que puede considerarse como una poda aplicada sobre el algoritmo que genera el unfolding (ver algoritmo 3 en el capítulo anterior).

Es decir, el algoritmo considera los nodos y transiciones del autómata original, y eventualmente los agrega en Fin. Sea C una configuración time-feasible dentro del unfolding time-feasible. Supongamos que C no estuviera en Fin. Si C no está en Fin entonces el unfolding contiene algún cut-off event e, tal que $C = \lceil e \rceil \oplus E$ para algún E adecuado[2].

[1]Esta demostración es una adaptación de la que figura en [ERV96].

[2]Sabemos que el cut-off tiene que estar en Fin ya que de otro modo el algoritmo no podría haberse detenido debido a que existía una extensión posible por agregar.

Por definición de cut-off event sabemos que existe una configuración local $\lceil e' \rceil$ tal que $\lceil e \rceil \prec \lceil e' \rceil$ y tal que ambas configuraciones son reachability equivalent. Consideremos ahora la configuración $C' = \lceil e' \rceil \oplus I_{\lceil e \rceil}^{\lceil e' \rceil}(E)$. Dado que el orden $\prec$ se preserva por extensiones finitas entonces podemos afirmar que $C' \prec C$, además es fácil ver que son reachability equivalent. Si C' no es una configuración de Fin, entonces realizamos una nueva iteración del procedimiento para obtener una C'' tal que $C'' \prec C'$ y con C'' reachability equivalent a C'. El fin de este procedimiento está garantizado por el hecho de que $\prec$ es un buen orden. Por lo que para cada cut C del unfolding es posible encontrar una configuración equivalente dentro de Fin. De esta forma Fin es un prefijo completo.

□

Como se mencionó anteriormente, en el caso de formalismos no temporizados, el algoritmo que construye el prefijo completo del unfolding tiene garantizada su finalización. Esto se debe a que la cantidad de markings distintos que existen es finita, ya que está acotada por el producto cartesiano de las locaciones de sus componentes (que son finitas). En el contexto temporizado, sin embargo, el concepto de cut-off se basa en la reachability equivalence entre configuraciones, que además de considerar los markings toma en cuenta las distancias temporales entre los puntos relevantes de las mismas. Esto puede determinar una relación de equivalencia entre configuraciones que no sea de índice finito, caso en el cual el prefijo podría ser infinito.

A pesar de este hecho, que en principio desalentaría la construcción del prefijo (ya que, como se recordará, tenía como objetivo evitar la generación de un unfolding potencialmente infinito), el uso del prefijo presenta algunas ventajas importantes.

La primera de ellas es que al realizar la poda utilizando la detección de cut-offs se reduce el espacio de estados, lo cual reduce el tiempo de ejecución del algoritmo que verifica alcanzabilidad (que presentaremos en breve). Adicionalmente, esta optimización tiene como consecuencia la reducción del tamaño del branching process generado durante el proceso de verificación. Este hecho es de suma importancia debido a que las herramientas que se utilizan actualmente presentan la desventaja de que en ocasiones el espacio de estados sobre el que trabajan es tan grande que impide la finalización del procedimiento de verificación (el proceso agota la memoria disponible para su ejecución).

Por otra parte, el uso del algoritmo que construye el prefijo en aquellos casos en que el unfolding es infinito y la relación de equivalencia entre configuraciones es de índice finito, asegura la finalización del mismo a diferencia

de un algoritmo que construya el unfolding temporizado .

Demostrada la utilidad del prefijo, se hace necesario constatar que el algoritmo 3 construirá un prefijo completo **finito** en aquellos casos en que esto sea posible. Esto es, cuando la equivalencia entre configuraciones sea de índice finito. Esto será demostrado en la siguiente proposición:

Proposición 4. *Si la relación de equivalencia $\equiv$ es de índice finito entonces Fin es finito.*

Demostración. Nos basamos en la demostración de [ERV96], reemplazando n por la cantidad de clases que determina $\equiv$. Dado un evento e de Fin, se define la altura de e como la longitud de la cadena más larga tal que $e_1 < e_2 < ... < e$; la altura de e está denotada por $d(e)$. Para probar que Fin es finito, probaremos las siguientes 3 propiedades:

1. Para cada evento e de Fin, $d(e) \leq n+1$, donde n es la cantidad de clases de equivalencia generadas por $\equiv$. Dado que los cuts corresponden a markings alcanzables, cada cadena de eventos $e_1 < e_2 < ... < e_n < e_{n+1}$ de Unf contiene dos eventos e_i y e_j, con $i < j$, tales que $[\![e_i]\!] = [\![e_j]\!]$ (es decir, e_i y e_j están en la misma clase de equivalencia). Dado que $\lceil e_i \rceil \subset \lceil e_j \rceil$ (ya que $e_i < e_j$ y las configuraciones son cerradas hacia atrás) y $\prec$ refina a $\subset$, tenemos que $\lceil e_i \rceil \prec \lceil e_j \rceil$, y por lo tanto $\lceil e_j \rceil$ es un cut-off event de Unf. En el caso en que el algoritmo hubiera generado e_j, entonces habría generado e_i antes y e_j también hubiera sido reconocido como un cut-off event.

2. Para cada evento e de Fin, los conjuntos ${}^\bullet e$ y $e^\bullet$ son finitos. Por la definición de homomorfismo, existe una biyección entre $e^\bullet$ y $h(e)^\bullet$, donde h denota el homomorfismo de Fin, y de la misma manera para ${}^\bullet e$ y ${}^\bullet h(e)$. El resultado se obtiene del hecho de que el autómata a partir del cual se generó Fin es finito.

3. Para cada $k \geq 0$, Fin contiene sólo una cantidad finita de eventos e tales que $d(e) \leq k$. Lo probamos por inducción completa en k. El caso base, $k = 0$, es trivial ya que no existe ningún elemento a distancia cero. Sea $E_k = \{e | d(e) \leq k\}$ el conjunto de eventos de altura a lo sumo k. Probaremos entonces que si E_k es finito, entonces E_{k+1} es finito. Definimos $E_k^\bullet = \{b \mid e \in E_k, b \in e^\bullet\}$. Utilizando la propiedad 2 anteriormente demostrada, y la hipótesis inductiva $E_k^\bullet$ es finita. Dado que ${}^\bullet E_{k+1} \subset E_k^\bullet \cup Min(Fin)$, aplicando la segunda parte de la definición de branching process (definición 2.19), y sabiendo que la cantidad de eventos en el autómata original es finita, obtenemos que E_{k+1} es finito.

Aplicando (1) y (3) podemos afirmar que Fin contiene una cantidad finita de eventos. De (2) se obtiene que Fin contiene una cantidad finita de transiciones. Por lo tanto, Fin es finito.

□

Nuestro trabajo está enfocado en la verificación de alcanzabilidad local, pero hasta ahora no hemos mostrado una forma de relacionar alcanzabilidad sobre la EATAN con alcanzabilidad en el prefijo. Para remediar esto, enunciaremos un corolario de las proposiciones anteriores que sostiene que un estado es alcanzable en la EATAN si es alcanzable en el prefijo que genera el algoritmo 3.

Corolario 1 (Alcanzabilidad Local). *Sea $N = \langle A_1, A_2, ..., A_n \rangle$ una EATAN. Un estado $q, q \in Q_k$ para algún $1 \leq k \leq n$ es alcanzable sii es alcanzable en el branching process Fin construido por el algoritmo.*

Demostración. Directamente de la completitud de Fin, del hecho de que sus configuraciones locales son time-feasible, y de que N es una EATAN (es decir, las ejecuciones end asynchronous de N pueden ser extendidas a ejecuciones de N). □

Lo que nos restaría por presentar es un algoritmo capaz de verificar propiedades de alcanzabilidad local que utilice la técnica de unfolding. A continuación mostraremos dicho algoritmo, que está basado en el algoritmo 3.

Algoritmo 4 (Alcanzabilidad).

input: Una EATAN $N = \langle A_1, A_2, ..., A_n \rangle$ y una locación q de N.
output: Un valor booleano que indica si es posible alcanzar la locación q.

begin
$found :=$ false
$Fin := InitialConditions(N)$
if $q \in h(Conditions(Fin))$ **then**
$found :=$ true
endif
$ep :=$ EPTF(Fin)
$cutOffs := \emptyset$

while $ep \neq \emptyset \wedge found =$ false
elegir $e \in ep$ tq $\lceil e \rceil$ es minimal respecto a $\prec$
if $(\lceil e \rceil \cap cutOffs) = \emptyset$ **then**
$Fin := Fin \oplus \{e\}$

```
                    if q ∈ h(Conditions(Fin)) then
                        found := true
                    endif
                    ep := EPTF(Fin)
                    if isCutOff(e, Fin) then
                        cutOffs := cutOffs ∪ {e}
                    endif
                else
                    ep := ep ∖ {e}
                endif
            endwhile

            return found
end
```

Como se observa, la construcción del prefijo se va realizando simultáneamente con la verificación de alcanzabilidad de la locación q. La principal modificación respecto del algoritmo 3 consiste en que luego de la aplicación de la extensión sobre el branching process, se chequea si q se encuentra representada por alguna condición en el mismo. Como consecuencia de esta decisión es posible distinguir cuatro posibles escenarios durante la ejecución del algoritmo 4:

1. q es alcanzable y Fin es finito: el algoritmo retorna true
2. q es alcanzable y Fin es infinito: el algoritmo retorna true
3. q no es alcanzable y Fin es finito: el algoritmo retorna false
4. q no es alcanzable y Fin es infinito: el algoritmo no finaliza

Se podría haber utilizado otro enfoque para realizar el análisis de alcanzabilidad: generar el prefijo completo Fin, y luego chequear si q se encuentra representada entre sus condiciones. Este enfoque, igualmente válido, presenta la desventaja de no finalizar en el segundo, mientras que el algoritmo 4 sí lo hace debido que realiza la verificación de manera incremental.

4.2 Algoritmos Auxiliares

El algoritmo 4 que realiza el análisis de alcanzabilidad utiliza definiciones para las cuales no se han presentado algoritmos, y cuya resolución no es trivial.

A continuación presentaremos algoritmos para el cálculo de las extensiones posibles time-feasible de un branching process y para la determinación de cut-off.

Algoritmo 5 (EPTF).

input: Un branching process $\beta = \langle < B, E, F >, h \rangle$ de una EATAN N.
output: El conjunto $PosExt$ de extensiones posibles time-feasibles de β.

begin
$PosExt := \emptyset$
para cada $X \subseteq \mathcal{P}(B)$
if $co{-}set(X)$ **then**
para cada $t \in T^n(N)$
if $(h(X) = source(t)) \wedge$
$(\nexists e \in E \text{ tq } h(e) \equiv_{stutter} t \wedge {}^\bullet e = X)$ **then**
if $\lceil (t,X) \rceil$ es time-feasible en $\beta \oplus \{(t, X)\}$ **then**
$PosExt := PosExt \ \cup \{(t, X)\}$
endif
endif
endif
end

El algoritmo se basa fuertemente en la definición de extensión posible time-feasible (definición 3.23), por lo que resulta bastante claro. Sin embargo, contiene definiciones para las que no se han presentado algoritmos, pero cuya implementación es directa a partir de su definición.

Para determinar si un conjunto de condiciones X es un co-set, se deben tomar sus elementos de a pares y chequear si satisfacen la relación **co** (definición 2.21), es decir, que no son comparables entre sí ni se encuentran en conflicto.

En el caso en que el conjunto X sea efectivamente un co-set, el algoritmo considerará todas las transiciones globales de la TAN (T^n : definición 3.8). En el caso en que un par (t, X) satisfaga las condiciones necesarias para ser una extensión posible (definición 2.25), se chequerá la time-feasibility de la misma. Esto se realizará verificando la ausencia de ciclos de peso positivo en el grafo de pesos definido por la función $\widehat{\Delta}$ aplicada a la configuración local del evento.

Algoritmo 6 (isCutOff).

input: Un evento e y un branching process $\beta = \langle < B, E, F >, h \rangle$.
output: Un valor booleano que indica si e es cut-off de β.

begin
para cada $e' \in E$
if $\lceil e' \rceil$ es reachability equivalent a $\lceil e \rceil \wedge \lceil e' \rceil \prec \lceil e \rceil$ **then**
return true
endif

return false
end

Nuevamente, el algoritmo para la determinación de cut-off es una reescritura de su definición (definición 2.31). Para determinar si la configuración local de e es reachability equivalent a la de alguno de los eventos e' se puede utilizar el resultado del teorema 1, por lo que alcanzaría con verificar que se satisfacen las condiciones de la definición de equivalencia entre configuraciones (definición 3.31): que ambas configuraciones tienen el mismo marking, que ambas tienen los mismos $TooFar$ y finalmente, que las distancias temporales entre los puntos relevantes de las mismas son equivalentes entre sí (o alguno de esos puntos es $TooFar$).

Capítulo 5

Implementación

Para determinar la viabilidad del desarrollo teórico que hemos presentado, y adicionalmente para agregarle valor, se implementó una herramienta de verificación de propiedades de alcanzabilidad basada en las ideas y algoritmos presentados en los capítulos anteriores.

En el presente capítulo se discutirán algunas decisiones de implementación que fueron tomadas durante el desarrollo de esta herramienta, así como los resultados obtenidos en la aplicación de la misma sobre casos de estudio.

5.1 Estructuras Utilizadas

Se implementaron clases para la mayoría de los conceptos utilizados en el desarrollo teórico (predicados, relojes, transiciones locales y globales, etc.), por lo que sólo se mencionarán las más relevantes y/o cuya implementación no haya sida directa a partir de su definición teórica.

Como lineamiento general se decidió almacenar aquellos valores o datos que son utilizados con frecuencia en las búsquedas, los que se agregaron como miembros de la clase. Esto se realizó teniendo la precaución de que la memoria requerida por la optimización no fuese excesiva.

- La clase más importante es **TimedAutomataNetwork** que se utiliza tanto para representar TANs como EATANs. Hereda de la clase **Array** y puede verse como una tupla de **TimedAutomata**.
  - Integer: size *(cantidad de TAs que componen la TAN)*
  - Set of GlobalTransitions: globalTransitions
  - Integer: max *(máxima constante de la TAN)*
  - Mapping Clock $\leftrightarrow$ Integer: clocks

El miembro *clocks* se utiliza para indexar todos los clocks de la red. Es decir, identificamos a cada reloj a través de un entero, para de esta manera poder tener una representación más compacta a la hora de calcular distancias temporales.

Los métodos más importantes son los utilizados para transformar una TAN en una EATAN, y los necesarios para determinar si una locación es alcanzable.

- **asEATAN**
  Convierte la TAN a una EATAN. Este algoritmo es una implementación del algoritmo 7, presentado en el apéndice A.
- **isReachable:** aLocationName
  Este método determina si la locación con nombre aLocationName es alcanzable. Para esto va construyendo el prefijo completo del unfolding de la EATAN y busca si en este prefijo se encuentra representada la locación buscada. El mismo constituye una implementación del algoritmo 4.
- **buildPETF:** aBranchingProcess **withPE:** aSetOfEvents **lastPostset:** aSetOfLocations
  Este método es usado en **isReachable:** ... y constituye una implementación del algoritmo 5. Devuelve un conjunto con eventos que constituyen extensiones posibles time-feasible del prefijo generaod hasta el momento. El cálculo de extensiones se realiza de manera incremental: sólo considera las transiciones cuyo preset tiene alguna condición en común con la última extension posible agregada al branching process (lastPostset). Primero se calculan las extensiones posibles sin tener en cuenta el aspecto temporal y luego se filtran aquellas que son time-feasibles.
- **findEventWithMinimalConfiguration:** someEvents **in:** aBranchingProcess
  Devuelve el evento de someEvents cuya configuración local es miminal. La relación de orden utilizada es el tamaño de las configuraciones locales[1]. Este método también es utilizado en **isReachable:** ... para elegir la próxima extensión que se agregará al branching process.

- La clase **BranchingProcess** hereda de la clase **AcyclicGraph** y puede verse como un grafo que contiene dos tipos de nodos: **ONEvents**, que representan a los eventos y **ONConditions** que representan condiciones. Los miembros que la componen son

[1]Este orden es el propuesto originalmente por McMillan en [McM92a].

- TimedAutomataNetwok: tan
- Grafo de pesos: g
- Matriz: coMatrix

Donde el grafo de pesos no es más que un grafo al que se le han asociado valores enteros a los ejes entre sus nodos y la matriz coMatrix guarda la relación *co* (ver definición 2.21) entre las condiciones.

Los métodos más destacados son lo que son llamados por **isReachable: aLocationName** de la clase **TimedAutomataNetwork**.

- **addPEToBP:** anONEvent
  Agrega anONEvent al branching process (agregando su postset y relacionándolo con su preset y su postset). Se encarga de actualizar la matriz que contiene la relación co.
- **addPEToWG:** anONEvent
  Agrega anONEvent al Grafo de Pesos del branching process, y agrega ejes con las distancias del evento a sus puntos relevantes.
- **isCutOff:** anONEvent
  Determina si anONEvent es un cut-off. Para esto evalúa la existencia de una configuración menor que sea equivalente a la configuración de anONEvent (utilizando para esto **isEquivalentConfiguration:** ...). El mismo es una implementación del algoritmo 6.
- **isEquivalenConfiguration:** anONEvent **to:** bONEvent
  Retorna si las configuraciones locales determinadas por anONEvent y bONEvent son equivalentes considerando el marking de las mismas y las distancias entre sus puntos relevantes (ver definición 3.31).
- **getDistanceBetween:** aNumber **and:** bNumber **for:** anONEvent **with:** aMatrix
  Devuelve la distancia que hay entre los nodos aNumber y bNumber en la configuración local de anONEvent utilizando los caminos máximos en el grafo de pesos (represenatdos en aMatrix) de la configuración local de anONEvent.
- **isCo:** aPairOfNodes
  Retorna si un par de nodos es *co*, utilizando para esto la matriz coMatrix que guarda la relación *co* entre todos los nodos del Branching Process.

- Los **TimedAutomata** se implementaron como una clase derivada de la clase **Graph**. Contiene los siguientes miembros

  - String: name
  - Location: qin
  - Set of Location: locations
  - Set of Transition: transitions
  - Set of Clock: clocks
  - Integer: max *(Máxima constante que aparece en las restricciones temporales)*

  Los métodos más relevantes de esta clase son los que se utilizan en la ejecución de **asEATAN** de la clase **TimedAutomataNetwork**. Como hemos mencionado este método constituye la implementación del procedimiento presentado en el apéndice A.

  - **getRelevantTransitions**
    Devuelve las transiciones que son relevantes en el autómata.
  - **getTransitionsLabelled:** aLabel
    Devuelve todas las transiciones etiquetadas con aLabel.
  - **includesOutputTransitionWithLabel:** aLabel
    Determina si el autómata contiene una transición de output con label aLabel.
  - **influenceOver:** aLocation
    Devuelve el conjunto de todas las transiciones que tienen influencia sobre aLocation.

5.2 Optimizaciones

Los procedimientos más costosos (desde el punto de vista computacional) los constituyen el cálculo de las extensiones posibles time-feasibles (**buildPETF:** ...) y la determinación de cut-off (**isCutOff:** ...). Por este motivo el esfuerzo estuvo focalizado en tratar de reducir el costo de estos procedimientos.

El procedimiento para el cálculo de extensiones es inherentemente combinatorio ya que debe evaluar todos los posibles subconjuntos que son co-set. Lo que se hizo fue, en vez de calcular en cada paso de iteración todas las posibles extensiones time-feasibles, mantener un conjunto con todas las

calculadas hasta el momento e ir incrementando este conjunto con las extensiones nuevas que se generan luego de la aplicación de una extensión. Esto es posible ya que una extensión time-feasible calculada en una iteración sigue siendo time-feasible en todas las iteraciones posteriores. Es importante hacer notar que este enfoque no encarece significativamente la cantidad de memoria a utilizar, ya que la gran nayoría de las posibles extensiones que se almcenan serán agregados al branching process durante la construcción del prefijo completo.

Ese procedimiento se realiza en tres etapas: el algoritmo calcula primero las extensiones posibles sin considerar las restricciones temporales. Elimina de este conjunto aquellas cuya configuración local contenga algún cut-off. Finalmente chequea time-feasibility para las que no fueron eliminadas.

- Para la primera etapa el uso de la matriz **coMatrix** del **BranchingProcess** es vital, ya que determinar los co-sets de las nuevas posibles extensiones es un procedimiento muy costoso. Esta optimización fue tomada de [ER99].

- Otro cambio que surge para la optimización de este procedimiento consiste en "adelantar" el chequeo de la intersección con los cut-offs antes de la verificación de time-feasibility, ya que este último es un procedimiento mucho más costoso.

- Durante la etapa de chequeo de time-feasibility se hace necesario contar con las distancias temporales entre los puntos relevantes de las configuraciones (para determinar la existencia de ciclos de peso positivo). Para esto se utiliza el grafo de pesos de la clase **BranchingProcess** donde se encuentran representados las distancias temporales **entre todos los eventos del mismo**.

  Esto obedece al hecho de que en nuestra implementación es más económico verificar la existencia de ciclos en todo el branching process que construir el subgrafo de la configuración local y realizar el chequeo en este último. Esto debido a que la detección de ciclos en el branching process se va realizando de manera incremental a medida que se agregan nuevos eventos.

  Se debe hacer notar que la ausencia de ciclos positivos en el grafo implica la ausencia de ciclos positivos en los subgrafos determinados por las configuraciones locales incluidas en el mismo. Pero la presencia de ciclos en el grafo no implica la presencia de ciclos en los subgrafos. Por lo tanto en la ausencia de ciclos en el grafo sólo tiene sentido hasta que se encuentra el primer ciclo de peso positivo en el mismo. A partir

de ese momento es necesario realizar la detección de ciclos positivos en el subgrafo de la configuración local.

Para determinar si un evento es cut-off es necesario evaluar si algún evento del branching process tiene una configuración local menor (respecto a algún orden adecuado) que sea reachability equivalent a la del primero. Como orden adecuado se utilizó el propuesto por McMillan en [McM92a] basado en el tamaño de las configuraciones.

Esta operación es muy costosa debido al alto costo de, por un lado obtener la configuración local de un evento, y por otro el chequeo de equivalencia temporal.

Para el primer punto lo que se hizo fue agregar un miembro a la clase **ONEvent** que almacene la configuración local del mismo. Esto debido a que durante la evaluación de cut-off muchas veces es necesario utilizar el marking de la misma. Al tener precalculada la confiugración se agiliza este procedimiento.

La implementación del algoritmo **isCutOff: ...** primero compara las configuraciones locales (respecto del orden adecuado basado en el tamaño de las configuraciones), después compara los markings, y por último considera las distancias temporales, ordenando estas tareas de acuerdo su orden de complejidad creciente.

Para reducir el costo del cálculo de la equivalencia temporal entre configuraciones se agregaron a cada **anONEvent** sus puntos relevantes (sus *Lasts* y *Resets*) y las distancias temporales entre los mismos. Estas distancias están representadas en una matriz con la siguiente estructura:

		Lasts				Resets			
		1	2	...	n	c_1	c_2	...	c_m
Lasts	1								
	2								
	...								
	n								
Resets	c_1								
	c_2								
	...								
	c_m								

Figura 5.1: Estructura que almacena las distancias entre los puntos relevantes para un evento.

Como se puede apreciar en la figura anterior, dada una configuración local la información temporal de las distancias entre sus puntos relevantes implica el almcenamiento de $(n+m)^2$ valores, para una red de n componentes con m relojes. Los relojes son referenciados a través de los índices definidos en el miembro **clocks** de la **imedAutomataNetwork**. De esta forma es posible acceder eficientemente a la distancia temporal entre cualquier par de puntos relevantes de una configuración.

5.3 Funcionamiento de la Herramienta

El lenguaje elegido para la implementación de la herramienta fue Squeak [Squ] una versión open source del lenguaje de programación Smalltalk. Las razones para esta elección recaen en las facilidades que brinda para la implementación de software (en especial la presencia de un garbage collector), que nos permitieron realizar un desarrollo rápido. Otra característica que fue decisiva fue la posibilidad que brinda este entorno de desarrollo de almacenar una "imagen" de los objetos creados durante uan ejecución, permitiendo de esta forma retomar una ejecución almacenada previamente sin necesidad de comenzarla desde el inicio. Esto fue de vital importancia durante la ejecución de los casos de prueba, ya que agilizó la implementación de correcciones y optimizaciones. De esta forma se decidió privilegiar un tiempo reducido de desarrollo por sobre la performance que se podría haber obtenido si el desarrollo se hubiera realizado utilizando otro lenguaje (C o C++ por ejemplo).

La herramienta utiliza archivos ".aut", que es el formato de entrada de la herramienta *OpenKronos*. El formato de los mismos puede encontrarse en el apéndice B de este trabajo.

Cada archivo .aut (automaton) especifica un TA. Para definir una red se debe suministrar a la herramienta el nombre de un directorio donde se encuentran los archivos .aut que la componen. Un parser se encarga del procesamiento de los archivos y de construir una TAN la que luego transforma en una EATAN utilizando el procedimiento mencionado en el apéndice A.

En este punto es posible realizar la verificación de propiedades de alcanzabilidad. Para esto se invoca al método **isReachable: ...** que implementa el algoritmo 4 con las optimizaciones mencionadas.

Como resultado de cada evaluación de la alcanzabilidad de una locación, si el procedimiento de verificación finaliza (la locación es alcanzable o no es alcanzable pero el prefijo completo es finito), la herramienta indica si la locación es alcanzable o no junto con información asociada a la ejecución del algoritmo. (En el caso en que la locación sea alcanzable muestra una evolución del sistema que permite alcanzarla). En el caso en que la locación

buscada no es alcanzable y el prefijo completo no es finito, la herramienta no finaliza su ejecución.

5.4 Caso de estudio: Latched$n$

Se eligió como caso de prueba un sistema de pipe-line compuesto por una fuente emisora de señales y n procesos, en el cual los procesos deben transformar de manera secuencial la señal emitida por la fuente[2]. La fuente emite la señal a una determinada frecuencia, momento en que es enviada al primer proceso. Si el mismo está en condiciones de procesarla lo hace y la envía al siguiente proceso en el pipe-line. Cada proceso tiene asociado un latch, lo que le permite almacenar el valor de la señal de la fuente en el caso en que no esté preparado para recibir un nuevo valor (porque se encuentra procesando uno anterior).

Figura 5.2: Un pipe-line de n procesos

Cada uno de los componentes del sistema está sujeto a restricciones temporales: la fuente emite un nuevo valor de la señal cada $frec$ unidades de tiempo, el valor señal puede permanecer indefinidamente en los latchs y cada uno de los procesos requiere entre 20 y 100 unidades de tiempo para procesar la señal.

Dado este sistema se desea determinar, dada una frecuencia para la fuente, si es posible que una vez emitida la señal ésta sea evaluada por los n procesos que componen el pipe-line en menos $t = n \times 60$ unidades de tiempo.

Para esto utilizaremos una propiedad de alcanzabilidad local: agregaremos una componente observadora que contenga una locación de error que sólo sea alcanzable en el caso en que hayan transcurrido t unidades de tiempo y alguno de los procesos aún no haya recibido la señal.

La figura 5.3 muestra los autómatas temporizados que se utilizan para la verificación.

[2]Utilizaremos la adaptación presentada en [BGO02].

Figura 5.3: Autómatas Temporizados para Latched-n

En la figura se encuentra modelada la fuente, junto con un latch y un proceso (estos últimos parametrizados en función de su ubicación dentro del pipe-line). También se presenta el autómata que modela el observador para el caso $n = 2$. En este último se puede apreciar que la locación 6 representa el error ya que sólo es posible arribar a la misma si han transcurrido más de $2 \times 60 = 120$ unidades de tiempo desde que la fuente emitió la señal (el reloj R se reseta en la locación 1 y la transición que evoluciona a la locación 6 tiene guarda $\{R > 120\}$) .

5.5 Resultados

En esta sección se presentan los resultados obtenidos de la ejecución de la herramienta aplicada a variaciones de Latched$n$. Se consideraron casos de 2, 3 y 4 procesos para frecuencias de la fuente de 40 y 180 unidades de tiempo.

Name	TA	locs TAN	trans TAN	trans EATAN	global trans	clocks
Latched2 (L2)	6	23	67	78	251	6
Latched3 (L3)	8	32	109	126	460	8
Latched4 (L4)	10	41	159	182	729	10

Name	error	steps	secs[3]	evs	conds	h(fin)	h($\lceil e \rceil$)	cut offs	PE	PETF
L2-40	si	31	46	31	120	11	5	0	669	295
L3-40	si	104	805	104	398	15	7	0	7187	2764
L4-40	si	343	16792	343	1272	19	9	0	64128	21266
L2-180	no	93	556	73	251	35	-	7	2834	305

La primer tabla indica para cada uno de los casos de test la cantidad de autómatas que integran la TAN[4], la cantidad de locaciones y transiciones de la misma, la cantidad de transiciones que tiene la EATAN que se obtiene a partir de esta última y la cantidad de transiciones globales de esta EATAN.

La segunda tabla muestra los resultados obtenidos en la verificación de la alcanzabilidad de la locación de error utilizando la herramienta. Se probaron los casos de pipe-line de 2, 3 y 4 procesos para una fuente de 40 unidades de tiempo (L2-40, L3-40 y l4-40)y un pipe-line de 2 procesos para una fuente de 180 unidades de tiempo (L2-180). Para cada caso se indica si la locación de error es alcanzable o no, la cantidad de iteraciones del algoritmo, el tiempo que llevó este análisis, la cantidad de eventos y transiciones del prefijo que se construyó y altura de este último. En aquellos casos en que la locación fue alcanzada se indica la altura de la configuración local del último evento que se agregó al prefijo. Los últimos valores de la tabla corresponden a la cantidad de cut-offs detectados y a la cantidad de extensiones posibles y extensiones posibles time-feasibles que fueron evaluadas.

A partir de los resultados obtenidos podemos obtener las siguientes conclusiones

- El algoritmo finalizó en todos los casos. En los casos que utilizan una fuente de $frec = 40$ se alcanzó la locación de error y para el caso en que $frec = 180$ no se alcanzó generándose un prefijo completo. Esto último

[3]Los resultados se obtuvieron ejecutando la herramienta en una PC Pentium III, con un procesador de 1.1 Ghz y 256 Mb de memoria RAM ejecutando Windows 2000.

[4]Recordar que cada proceso tiene asociado un latch, por lo que un pipe-line de n procesos se modela con $2 \times n + 1 + 1$ autómatas, siendo los dos últimos la fuente y el observador de la propiedad.

debido a que la relación de equivalencia entre las configuraciones es de índice finito.

- La cantidad de extensiones posibles time-feasible es bastante menor a la cantidad de extensiones posibles. Con lo que una mejora en el proceso de filtrado por time-feasibility tendría un alto impacto en los tiempos de ejecución totales del algoritmo.

- En todos los casos la altura del prefijo generado hasta la finalización del algoritmo no es grande. Para los casos en los que la locación de error es alcanzable esto permite obtener ejecuciones de tamaño reducido que alcanzan el error del sistema, lo que puede simplificar análisis posteriores.

- La técnica es aplicable a casos de la vida real aunque la performance es un aspecto crítico. En su estado actual, la escalabilidad de la herramienta a ejemplos con más participantes se torna inviable.

Se debe destacar que la implementación de la herramienta no se llevó a cabo con el objetivo de obtener una implementación que pudiera competir a nivel de performance con otras herramientas de verificación disponibles (Kronos, UPPAAL, HyTech, etc.), sino con el de obtener un *proof-of-concept* que permitiera corroborar la viabilidad del desarrollo teórico del presente trabajo, así como para identificar las principales dificultades que debe sobreponer una implementación de la técnica.

Capítulo 6

Conclusiones

Antes de dar por finalizado el trabajo presentaremos los resultados más importantes que se han obtenido, para luego mencionar algunas líneas de trabajo futuras que podrían extender el desarrollo presentado.

En primer lugar, el resultado más importante lo constituye la demostración de la factibilidad de la extensión de la técnica de unfoldings al contexto de los autómatas temporizados. Esto constituye un aporte original del trabajo, ya que extiende la técnica a un formalismo sobre el cual nunca se había aplicado anteriormente.

Para lograr este objetivo, fue necesario obtener los siguientes resultados que también se consideran aportes originales del trabajo:

- La obtención de una versión temporizada del marco formal de la técnica, destacándose las nociones de reachability equivalence entre configuraciones y time-feasibility de una configuración.
- La obtención y demostración de correctitud de un semi-algoritmo para la construcción del prefijo completo de un unfolding de una red de Autómatas Temporizados.
- La definición y utilización del concepto de EATAN, que permite resolver el "problema de la urgencia" de manera más elegante que otros enfoques de la literatura ([SY95] por ejemplo).

Un punto importante lo constituye el haber podido incorporar el enfoque composicional presente en otros trabajos ([ER99] por ejemplo). Esto permite evitar la composición de todas las componentes del sistema y el procesamiento de la concurrencia *on-the-fly*.

La viabilidad de la extensión de la técnica no se da únicamente en el plano teórico si no que, adicionalmente, se ha logrado implementar una herramienta que permite aplicarla en casos prácticos. Esto último supeditado a la

optimización de algunos de los procedimientos utilizados, ya que como se ha mencionado en el capítulo anterior, los tiempos de ejecución no pueden competir aún con los de otras herramientas de verificación (Kronos, UPPAAL, etc.). Aunque como hemos mencionado, nuestros esfuerzos no estuvieron concentrados en la optimización de los tiempos de ejecución sino en el análisis de factibilidad de la técnica.

6.1 Trabajo Futuro

En el plano teórico sería interesante emprender la búsqueda de una noción alternativa de equivalencia entre configuraciones que asegure la finitud en los prefijos temporizados, lo que permitiría obtener un algoritmo para la construcción del prefijo y no un semi-algoritmo. En el caso en que esto se pudiese lograr o que existiera un procedimiento que permitiera determinar de antemano la finitud del índice de la relación de equivalencia, sería posible implementar un esquema de verificación que construyera el prefijo completo del unfolding y luego permitiese analizar propiedades (no sólo de alcanzabilidad). De esta forma se evitaría tener que generar el prefijo para cada propiedad a verificar, lo que podría reducir el tiempo de verificación de un sistema.

Otro desarrollo útil sería el de determinar si la información requerida para poder calcular la equivalencia entre dos configuraciones es minimal. Es decir, si para calcular las distancias entre los puntos relevantes de una configuración es necesario utilizar información **de toda** la configuración local (como se ha presentado en el trabajo), o si por el contrario, es posible aprovechar las distancias de los puntos relevantes de sus configuraciones menores.

Sería interesante realizar pruebas sobre nuevos casos de estudio con características distintas (grado de sincronización entre las componentes, tipo de restricciones temporales, etc.) para descubrir si existen tipos de sistemas para los que la aplicación de la técnica de verificación propuesta es particularmente buena. Otra variación interesante sería utilizar un orden adecuado que se ajuste más a las caraterísiticas particulares de la verificación de sistemas temporizados.

La herramienta desarrollada podría evolucionar de un mero *proof-of-concept* hacia una verdadero software de aplicación. Para esto es necesario trabajar en la búsqueda de optimizaciones que reduzcan de manera significativa el tiempo de ejecución de los algoritmos principales. Otra posibilidad sería incorporar el algoritmo optimizado a alguna herramienta de verificación de uso difundido (Kronos, UPPAAL, etc.).

Entre estas optimizaciones sería de gran utilidad contar con un procedi-

miento de conversión a EATAN que no incremente el grado de complejidad de las redes que genere (en relación a la TAN original). Obviamente, bajo la hipótesis de que existe un procedimiento con estas características, caso en el cual la transformación a EATAN presentada en el apéndice A de este trabajo quedaría reducido a un caso particular del anterior.

Bibliografía

[AHU79] A. Aho, J. Hopcroft, and J. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley, 1979.

[AL97] T. Aura and J. Lilius, *Time processes for time petri-nets*, ICATPN, 1997, pp. 136–155.

[Arn92] A. Arnold, *Finite transition systems*, Prentice-Hall, 1992.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, *Symbolic model checking:* 10^{20} *states and beyond*, Information and Computation **98** (1992), no. 2, 142–170.

[BF99] B. Bieber and H. Fleischhack, *Model checking of time petri nets based on partial order semantics.*, Lectures Notes in Computer Science: Proceedings of 10th International Conference on Concurrency Theory, Eindhoven (Baeten, Jos C. M., and Mauw, Sjouke, eds.), vol. 1664, Springer-Verlag, Agosto 1999, pp. 210–225.

[BGO02] V. Braberman, D. Garbervetski, and A. Olivero, *Improving the verication of timed systems using influence information*, Aceptado en TACAS 2002.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, *A theory of communicating sequential processes*, Journal of the ACM **31** (1984), no. 3, 560–599.

[BJLY98] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, *Partial order reductions for timed systems*, International Conference on Concurrency Theory, 1998, pp. 485–500.

[BLL+96] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, *UPPAAL - a tool suited for the automatic verification of real-time systems*, Proceedings of Hybrid Systems III **LNCS 1066** (1996), 232–243.

[BM98] W. Belluomini and C. J. Myers, *Verification of timed systems using POSETs*, Computer Aided Verification, 1998, pp. 403–415.

[Bra00] V. Braberman, *Modeling and checking real-time systems designs*, Ph.D. thesis, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2000.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*, The MIT Press. Cambridge, Massachusetts, 1999.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, *The tool KRONOS*, Proceedings of Hybrid Systems III **LNCS 1066** (1996), 494–510.

[Eng91] J. Engelfriet, *Branching processes of Petri nets*, Acta Informatica **28** (1991), 575–591.

[ER99] J. Esparza and S. Römer, *An unfolding algorithm for synchronous products of transition systems*, Proceedings of the 10th International Conference on Concurrency theory (CONCUR '99) **LNCS 1664** (1999), 2–20.

[ERV96] J. Esparza, S. Römer, and W. Vogler, *An improvement of McMillan's unfolding algorithm*, Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96) **LNCS 1055** (1996), 87–106.

[Gar00] D. Garbervetsky, *Un método de reducción para la composición de sistemas temporizados*, Tesis de licenciatura, Departamento de Computación, Universidad de Buenos Aires, 2000.

[Har72] F. Harary, *Graph theory*, Addison-Wesley, 1972.

[HB94] H. Hulgaard and S. M. Burns, *Bounded delay timing analysis of a class of CSP programs with choice*, Proceedings International Symposium on Advanced Research in Asynchronous Systems and Systems (1994), 2–11.

[HHWT95] T. A. Henzinger, P. H. Ho, and H. Wong-Toi, *A User Guide to HyTech*, Proceedings of International Workshop on Tools and Algorithms for the Construction and Analysis of Systems **LNCS 1019** (1995), 41–71.

[LB99] R. Langerak and E. Brinksma, *A complete finite prefix for process algebra*, Proceedings of 11th International Conference on Computer Aided Verification (CAV '99) **LNCS 1663** (1999), 184–195.

[Lil99] J. Lilius, *Efficient state pace search for Time Petri Nets*, Tech. Report TUCS Technical Report No 255, Turku Centre for Computer Science, December 1999.

[McM92a] K. L. McMillan, *Symbolic model checking: An approach to the state explosion problem*, Ph.D. thesis, Carnegie Mellon University, May 1992.

[McM92b] ———, *A technique of state space search based on unfolding*, Kluwer Academic Publishers, Boston, USA, 1992, pp. 1–22.

[McM92c] ———, *Using unfoldings to avoid the state explosion problem in the verification os asynchronous circuits*, Proc. of the 4th Workshop on Computer Aided Verification **LNCS 663** (1992), 164–174.

[MD92] K. L. McMillan and D. L. Dill, *Algorithms for interface timing verification*, IEEE International Conference on Computer Design (1992), 48–51.

[Mil89] R. Milner, *Communication and Concurrency*, PrenticeHall, 1989.

[OB01] A. Olivero and V. Braberman, *Extending timed automata for compositional modeling healthy systems*, Electronic Notes in Theoretical Computer Science **52** (2001).

[Pel93] D. Peled, *All from One, One for All: on Model-Checking Using Representatives*, Proceedings of CAV'93, 5th International Conference on Computer-Aided Verification (Elounda, Grecia), LNCS, vol. 697, Springer-Verlag, 1993, pp. 409–423.

[Pet77] C. Petri, *Non-sequential processes*, Tech. Report GMD-ISF-77-5, Gesellschaft für Informatik and Datenver-arbeitung, Bonn, Alemania, 1977.

[PJKP00] M. A. Peña, J.Cortadella, A. Kondratyev, and E. Pastor, *Formal verification of safety properties in timed circuits*, Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society Press, 2000, pp. 2–11.

[PS98] X. Papadimitriou and X. Steiglitz, *Combinatorial Optimization*, Dover, 1998.

[SB97] R. H. Sloan and U. A. Buy, *Stubborn sets for real-time Petri nets*, Formal Methods in System Design: An International Journal **11** (1997), no. 1, 23–40.

[Squ] Squeak, *http://www.squeak.org.*

[SY95] A. Semenov and A. Yakovlev, *Verification of asynchronous circuits using Time Petri Net unfolding*, International Workshop on Timing issues in the Specification and Syntehsis of Digital Systems (TAU) (1995), 199–210.

[SYK98] A. Semenov, A. Yakovlev, and A. Koelmans, *Time petri net unfoldings and hardware verification*, Enero 1998.

[Val94] A. Valmari, *State of the art report: Stubborn Sets*, Petri Net Newsletter (1994), no. 46, 6–14.

[Val98] _____, *The state explosion problem*, Lectures on Petri Nets I: Basic Models, LNCS 1491, Springer-Verlag, 1998, pp. 429–528.

[VdJL96] E. Verlind, G. G. de Jong, and B. Lin, *Efficient partial enumeration for timing analysis of asynchronous systems*, Design Automation Conference, 1996, pp. 55–58.

[WG93] P. Wolper and P. Godefroid, *Partial-order methods for temporal verification*, International Conference on Concurrency Theory, 1993, pp. 233–246.

[YS97] T. Yoneda and B. Schingloff, *Efficient verification of parallel real-time systems*, Formal Methods in System Design **11** (1997), no. 2, 197–215.

Apéndice A

Transformando una TAN en una EATAN

El desarrollo presentado a lo largo del trabajo fue realizado bajo la hipótesis de que la TAN a verificar es en realidad una EATAN. Sin embargo no se presentó hasta el momento evidencia de que, dada una TAN, siempre sea posible obtener una EATAN equivalente. En esta sección presentaremos un proceso de transformación económico desde el punto de vista computacional que garantice la validez de esta conjetura[1].

Como se recordará, la verificación de propiedades de alcanzabilidad local se lleva a cabo utilizando una EATAN. De esta forma, existe la certeza de que todas las locaciones que son alcanzadas por ejecuciones end asynchronous también serán alcanzadas por alguna ejecución. Este hecho permite al sistema evolucionar correctamente tomando únicamente decisiones locales. Es decir, que en el momento en que el algoritmo de alcanzabilidad está eligiendo la próxima transición global a ejecutar, no será necesario tomar en consideración las restricciones temporales del resto de las transiciones habilitadas. El hecho de que la red sea una EATAN garantiza la correctitud del enfoque.

Veamos porque no es condición suficiente que la red sea una TAN. Se debe hacer notar que al tomar una transición que sale de una locación el algoritmo está descartando al resto de las transiciones que tienen origen en la misma locación. Por lo tanto, lo que parece ser una decisión local afecta a las demás componentes de la red: las mismas se ven imposibilitadas de tomar el resto de las transiciones de la locación en cuestión. Esto puede implicar que, como consecuencia de la decisión de ejecutar una transición determinada, en

[1]Como se verá en breve, este proceso vale para toda TAN sobre la que se pueda clasificar a las transiciones en controlables y no-controlables, propiedad que en la práctica es satisfecha por la mayoría de las TANs.

otras componentes se deshabiliten transiciones cuyas restricciones temporales obligaban a ser ejecutadas con anterioridad. Provocando de esta manera que el análisis de alcanzabilidad resultara incorrecto. Esta situación recibe el nombre de *el problema de la urgencia*.

La figura A.1 presenta un ejemplo de esta situación. Considerando que el sistema se encuentra en la locaciones q_1 y q_1', existen dos posibles evoluciones que cambian de locación: tomar la transición etiquetada con a o tomar la etiquetada con b. Supongamos la valuación actual de los relojes $x = 0$ e $y = 0$. Una posible evolución es la que toma a luego de 4 unidades de tiempo y toma la transición stutter de la locación q_1' . Respecto de A_1 esta evolución es perfectamente válida (se satisface la guarda de t_1 y se satisface el invariante de q_2). Sin embargo la evolución es inválida para A_2, ya que el invariante de $q_1{}'$ obligaba a tomar antes la etiqueta b. En este tipo de situaciones diremos que b es *más urgente* que a.

Figura A.1: El problema de la urgencia

El enfoque más sencillo para poder realizar la verificación de manera correcta consiste en "consultar" al resto de los autómatas antes de ejecutar cualquier transición. Es decir, antes de ejecutar una transición, revisar en el resto de los autómatas de la red si no existe algún otro que tenga una transición más urgente que afecte la evolución del sistema. Esto se puede llevar a cabo a través del agregado de nuevas transiciones locales que implementen esta sincronización. El inconveniente de este enfoque es que, como consecuencia del agregado de nuevas transiciones locales se produce un incremento en la cantidad de transiciones globales, lo que a su vez incrementa la complejidad de la red. Por este motivo, el agregado de nuevas transiciones debe hacerse evitando el agregado de transiciones innecesarias. De otra forma podría suceder que nuevamente nos enfrentásemos a un problema de

carácter combinatorio similar al de la explosión de estados.

Para poder determinar los casos en los que se hace imprescindible agregar transiciones de sincronización, analizaremos la forma en que las transiciones afectan a los autómatas de una red. Dada una locación podemos dividir al conjunto de transiciones habilitadas con origen en la misma en dos grupos: aquellas transiciones que pueden ser controladas por el autómata al que pertenece la locación, es decir transiciones sobre las que el autómata decide el momento en que se van a ejecutar; y aquellas otras sobre las que el autómata no tiene control, y por lo tanto está obligado a ejecutar en el momento en que así se lo indique alguna otra componente. Llamaremos a las primeras *transiciones de output* y *transiciones de input* a las últimas. Denotaremos *transiciones internals* a aquellas que no se ajusten a ninguna de las dos categorías anteriores. De esta forma estamos definiendo las interfases de Input/Output introducidas por [Bra00].

Definición A.1. (Interfases admisibles de Input/Output). *Dado un TA non-zeno A y los conjuntos de etiquetas $I, O \subseteq Labels(A)$, diremos que (I, O) es una interfase admisible de input/output para A sí y sólo sí:*

- $I \cap O = \emptyset$.
- *Para toda locación alcanzable q y para todo etiqueta $l \in I$ existe una locación q' alcanzable en un paso a través de una transición etiquetada con l.*
- *Para toda locación alcanzable q existe una ejecución divergente r que empieza en q y no contiene ningún label de input.*
- *Para cada $l \in Labels(A)$ tal que $l \in O$ se cumple que cada ejecución que contiene un número infinito de ocurrencias de l es divergente en el tiempo.*

Los conjuntos I, O son llamados Input y Output de A (y los notaremos $Input(A)$ y $Output(A)$). Toda transición cuya etiqueta no pertenezca ni a I ni a O será considerada internal ($Internal(A)$). ■

Tal como se presenta en [Gar00], dada una TAN N se verifican las siguientes propiedades respecto a sus transiciones de Input/Output:

- Sea $t \in T_i \wedge \lambda(t) = a$. Si $t \in Output(A_i) \Rightarrow \nexists t' \in T_j \ (i \neq j)$ tq $\lambda(t') = a \wedge t' \in Output(A_j)$. Dada una transición de output etiquetada con a, ninguna otro autómata puede tener una transición de output con esa etiqueta.

- Sea $t \in T_i \wedge \lambda(t) = a$. Si $t \in Internal(A_i) \Rightarrow \nexists t' \in T_j \ (i \neq j)$ tq $\lambda(t') = a$. Dada una transición internal etiquetada con a ningún otro autómata tiene una transición con esa etiqueta.

- Todas las transiciones de un autómata etiquetadas con el mismo símbolo cumplen el mismo rol (o son de input, o son de output o son internals).

De esto se concluye que las transiciones internals no sincronizan y que las transiciones de output sólo pueden sincronizar con transiciones de input.

Tiene sentido solicitar que, dada una etiqueta utilizada en una TAN, no todas las transiciones etiquetadas con ella sean de input. De otra forma, podría suceder que esas transiciones nunca fuesen ejecutadas debido a que todos los autómatas están esperando la señal para tomarlas. Consideraremos entonces únicamente redes que presenten esta característica. Formalizaremos esta idea en el concepto de *red cerrada* que nos permitirá expresarnos sobre el "dueño" de una etiqueta.

Definición A.2. (Red Cerrada). *Sea $N = \langle A_1, A_2, ..., A_n \rangle$ una TAN. Diremos que N es una red cerrada sii* $\forall a \in Labels(N) \ \exists \ A_i$ tq $a \in Internals(A_i) \vee a \in Output(A_i)$. ■

Definición A.3. (Dueño). *Sea N una TAN cerrada. Dado $a \in Labels(N)$ diremos que $A_i = dueño_N(a)$ sii $a \in Internals(A_i) \vee a \in Output(A_i)$.* ■

Ahora que somos capaces de identificar las transiciones de Input/Output de la red podemos retomar la idea de forzar la sincronización de la red agregando transiciones locales. Como hemos mencionado, estamos interesados en agregar la menor cantidad posible de transiciones que solucione el problema de la urgencia y convierta la TAN en una EATAN. Un análisis más detallado revela que, dada una locación de un autómata, es posible caracterizar al grupo de transiciones locales que tiene "influencia" sobre la misma. Es decir, cuáles son las transiciones para las cuales el autómata no tiene control sobre su ejecución (transiciones de input) y cuya ejecución puede afectar la evolución del autómata. Características que presentan las transiciones que resetean relojes o cambian de locación.

Definición A.4. (Influencia). *Dado un TA A diremos que una transición $t \in T$ tiene influencia sobre $sc(t)$ sii $t \in Input(A) \wedge (tg(t) \neq sc(t) \vee \rho(t) \neq \emptyset)$.* ■

Volviendo a la figura A.1, la transición t_2 tiene influencia sobre la locación q_1 ya que es una transición de input que cambia de locación.

Llamaremos *relevantes* a las transiciones de un autómata sobre la cuales tiene decisión respecto a su ejecución (son de output), o resetean algún reloj, o bien cambian de locación. Nuestro interés en identificar a las transiciones relevantes es que las mismas deben tomarse "con cuidado", es decir, sabiendo que pueden tomarse antes de que se ejecute una transición que tiene influencia sobre la locación de origen.

Definición A.5. (Relevante). *Dado un TA A diremos que $t \in T_i$ es relevante en A sii $t \in Output(A) \vee sc(t) \neq tg(t) \vee \rho(t) \neq \emptyset$.* ■

En la figura A.1 todas las transiciones son relevantes en sus respectivos autómatas: $t1$ y $t3$ porque son de output y adicionalmente cambian de locación, y t_2, porque a pesar de ser de input cambia de locación.

La sincronización buscada se forzará utilizando el siguiente algoritmo que agrega nuevas transiciones "bucle" a la red (transiciones con origen y destino en la misma locación).

Algoritmo 7 (Transformación a EATAN).

input: Una TAN $N = \langle A_1, A_2, .., .A_n \rangle$.
output: N transformada en EATAN.

para cada A_i en N hacer
  para cada $t \in T_i$ tq t es relevante en A_i
    para cada $t' \in T_i$ que influencia sobre $sc(t)$
      if $\lambda(t) \neq Labels(dueño(\lambda(t')))$ **then**
        agregar a cada una de las locaciones del dueño de $\lambda(t')$
        una transición bucle de input etiquetada con $\lambda(t)$

El algoritmo sincroniza los autómatas de la red agregando las transiciones necesarias que impidan que las decisiones locales tomadas por el algoritmo que verifica alcanzabilidad considere alcanzables locaciones que no lo son.

En la figura A.2 se puede observar el resultado de aplicar el algoritmo 7 a la red de la figura A.1. El agregado de las dos transiciones bucle en A_2 soluciona el problema de la urgencia impidiendo que el sistema evolucione incorrectamente tomando la etiqueta a.

En esto consiste la transformación propuesta. La misma aumenta la complejidad de la red, incrementando la cantidad de transiciones globales lo que vuelve más trabajosa la verificación (pero es lo que la hace correcta). La necesidad de construir una EATAN a partir de una TAN es el precio que pagamos por resolver el problema de la urgencia.

Figura A.2: Solucionando el problema de la urgencia

Una vez presentado el proceso de transformación demostraremos la corrección del mismo. La demostración utiliza un lema auxiliar que será introducido en forma posterior.

Teorema 2. *Sea N una TAN y sea N' el resultado de aplicar el algoritmo 7 a N. Se cumple entonces que N' es una EATAN y la alcanzabilidad local en N y en N' es la misma.*

Demostración. Que la alcanzabilidad local es la misma se demuestra trivialmente ya que la transformación sólo agrega transiciones stutter. Resta probar que N' es una EATAN.

Con el lema auxiliar demostraremos que, dada una ejecución end asynchronous sobre N', es posible extender la misma para obtener una nueva ejecución (también end asynchronous) en la que se ha hecho avanzar a los autómatas que se encontraban más rezagados. Notar que al extender la ejecución no se están perdiendo estados alcanzables. Como este proceso puede ser aplicado una cantidad finita de veces hasta lograr que ningún autómata quede rezagado, podemos obtener finalmente una ejecución que alcance todas las locaciones alcanzables por la ejecución end asynchronous original. Luego existe una ejecución que alcanza todas las locaciones alcanzables por ejecuciones end asynchronous y por lo tanto N' es una EATAN. □

A continuación introduciremos algunas definiciones necesarias para la presentación y demostración del lema auxiliar.

Definición A.6. (Time de un Autómata Temporizado). *Dada una secuencia de transiciones temporizada $\sigma = (t_1, \tau_1).(t_2, \tau_2)...(t_n, \tau_n)$ sobre un TA A definiremos*

$$Time(\sigma) = \tau_k \text{ tq } t_k \neq t_\epsilon \wedge \forall\ i, k < i \leq |\sigma| : t_i = t_\epsilon$$

Es decir, $Time(\sigma)$ es el tiempo que corresponde a la ejecución de la última transición no stutter de A. ■

Definición A.7. (Autómatas Rezagados). *Sea σ una ejecución end asynchronous de una EATAN N.*

Definiremos $min(\sigma) = \min_{1 \leq i \leq n} \{Time(\Pi_i(\sigma))\}$ como el tiempo en que se tomó la última transición no stutter del autómata que se encuentra más rezagado.

Definiremos $S_{min}(\sigma) = \{j \ / \ 1 \leq j \leq n \wedge Time(\Pi_j(\sigma)) = min(\sigma)\}$ como el conjunto que contiene los índices de los autómatas que se encuentran más rezagados. ■

Sirviéndonos de las definiciones anteriores enunciaremos y demostraremos el lema auxiliar.

Lema 4 (Lema auxiliar). *Sea σ una ejecución end asynchronous de una EATAN N. Si $|S_{min}(\sigma)| < n$ entonces $\exists$ r ejecución end asynchronous de N tq*

- *$\forall i \notin S_{min}(\sigma)$ se cumple que $\Pi_i(\sigma) \equiv_{stutter} \Pi_i(r)$*
- *$min(r) > min(\sigma)$*
- *$S_{min}(r) \supsetneq S_{min}(\sigma)$*

Demostración. Consideremos la figura A.3 que ilustra un paso de la construcción de r. En la misma tenemos esquematizada la proyección de una ejecución end asynchronous σ sobre los autómatas de N siendo A_n uno de los autómatas más rezagados. Por definición de transiciones Input/Output [OB01], sabemos que es posible hacer que un autómata avance una cantidad finita de unidades de tiempo dada tomando exclusivamente transiciones de output e internals. (Esa cantidad es el tiempo que le falta a A_n para "alcanzar" a A_{n+1}).

Avanzar a través de transiciones internals no provoca conflictos con los autómatas que se encuentran más avanzados, ya que como se ha mencionado las mismas no participan en la sincronización.

El avance utilizando transiciones de output requiere más cuidado ya que podría suceder que al extender la proyección sobre A_n con una transición de output a en tiempo k, otro autómata más adelantado (sea A_1 este autómata),

tuviera una transición de input etiquetada con a que tiene influencia sobre el origen de una transición relevante b que es tomada en un tiempo mayor o igual que k. Con lo que la extensión utilizando a podría no ser válida, ya que A_1 podría verse impedido de ejecutar b. Veamos que esto no puede suceder razonando por el absurdo. Si esto ocurriera, como la transición a de A_1 tiene influencia sobre la locación de origen de la transicíon b, entonces el algoritmo debe haber agregado a b como bucle en el dueño de a (sabemos que A_n es el dueño ya que a es de output en A_n) en un tiempo que es mayor o igual a k. Lo que implica que en la proyección sobre A_n debe estar b en tiempo mayor o igual que k. Lo que lleva al absurdo de que existe una transición no stutter para A_n que se ejecuta en tiempo mayor que $k > min(\sigma)$. El absurdo viene de suponer que existe una transición a que tiene influencia sobre el origen de b en A_1.

Figura A.3: Extendiendo las ejecuciones end asynchronous

De esta forma se concluye que siempre es posible extender con transiciones de output para llevar al autómata más rezagado al $Time$ del resto de los autómatas que están más adelantados en tiempo k. Este procedimiento puede repetirse hasta haber llevado todos los autómatas al $Time$ del más adelantado, con lo que se obtiene una ejecución end asynchronous con las características mencionadas. □

Con esto podemos concluir que la transformación de TAN a EATAN implementada por el algoritmo 7 es válida.

Observación: En el marco del proceso de transformación de una TAN a una EATAN la componente que cumple el rol de observador de las propiedades del sistema tiene una importancia mayor que el resto de las componentes. Esto se debe a que el observador necesita saber si está habilitado para tomar

la/s transición/es que llevan al sistema a la locación de error. En trabajos futuros se podría evaluar la idea de aplicar la transformación a EATAN forzando la sincronización del observador con cada una de las componentes pero evitando la sincronización entre estas últimas. De esta forma se evitarían los problemas causados por la urgencia que pueden producirse entre el observador y los demás autómatas (aunque los problemas entre las componentes no desaparecen). Es decir, debería ser posible demostrar que si la locación de error no es alcanzable aplicando la optimización es porque no lo era utilizando el enfoque presentado en el presente trabajo. Sin embargo, si es posible que el sistema evolucione a la locación de error no sería posible garantizar que esto ocurra realmente.

Apéndice B

Formato de los Archivos

El formato de los archivos .aut tiene cuatro secciones principales

- relojes
- des (estado inicial, ♯ transiciones, ♯ locaciones)
- transiciones
- invariantes

Cada una de las cuales posee la siguiente estructura:

- relojes = clockname (0): clock
- transición = (locación-source, [predicado], label asignación-variables reset-relojes, locación-target)
- invariante = [locación, predicado]
- predicado = predicado con los siguientes conectivos: $\neg, \vee, \wedge, <=>, \geq, \leq$. Los predicados hablan de valores de variable booleanas y de comparaciones de relojes con enteros.
- reset-reloj = reloj := entero
- comentario = (* .... *)