Trabajo Final de Licenciatura
en Ciencias de la Computación

Derivación de Contraejemplos para Model Checking Cuantitativo

Autor: *Miguel E. Andrés*[*]
Director: *Pedro D'Argenio*

Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba
Córdoba, Septiembre de 2006

[*] Correo Electrónico: eandres@hal.famaf.unc.edu.ar

Dedico este trabajo a mis queridos padres

Miguel y Bacho.

Agradecimientos

Es mi deseo agradecer a Pedro D’Argenio el haberme dado la oportunidad de ‘‘trabajar’’ y aprender con él.

También es mi deseo agradecerle a Diego Vaggione quién me enseñó a razonar formalmente, lo cual ha resultado ser una herramienta invaluable que me acompañará a lo largo de mi trayectoria académica.

Por supuesto, a los profesores, personal y compañeros de la FaMAF con los cuales aprendí a querer a las Ciencias de la Computación.

Por último, pero quizás en primer lugar, el ‘‘gracias’’ más grande del mundo a mi mamá, papá, Ignacio, Josefina, Augusto, tía Maru, tío Carlos, a los amigos que la vida me ha regalado y a mi ‘‘compañera’’... gracias por estar.

Índice general

Capítulo 1

Introducción

1.1. Necesidad de Verificación

En nuestra vida diaria estamos cada vez más relacionados con la tecnología, en forma explícita o no, a través del uso de computadoras personales, internet, palms, etc o, simplemente mediante televisores, electrodomésticos, celulares, etc. El mal funcionamiento de alguno de ellos si bien no es deseado si es aceptado. Tan es así que hemos aprendido a convivir con él. Sin embargo, errores en sistemas críticos como plantas nucleares o sistemas de control aéreo son inaceptables y poseen un altísimo impacto en la seguridad. Además, los errores pueden ser muy costosos, en particular si los mismos se encuentran durante la etapa de operación del sistema, es decir, después de que el producto ha sido lanzado al mercado. Varias historias dramáticas son bien conocidas. Ejemplos de ellas son, entre otros, el error en la unidad de división de punto flotante de los procesadores Intel Pentium II que causó pérdidas estimadas en alrededdor de U$S 500 millones a lo que debe agregarse el desprestigio que sufriera la compañia; un defecto de software en el sistema de control en la maquina de terapia radioactiva Therac-25 produjo la muerte de 6 pacientes de cáncer entre 1985 y 1987 a raíz de una sobre exposición radioactiva; un solo día de falla en el sistema de reservas on-line de una compañía aérea importante provocaría su bancarrota por perdidas de ventas.

Por otro lado, la complejidad de los sistemas, y por consiguiente su vulnerabilidad a errores, se ve incrementada a pasos agigantados con el paso del tiempo.

Por las razones antes expuestas, el proceso de verificación de la correctitud de un sistema respecto de sus especificaciones, se ha convertido en una actividad extremadamente importante. Investigaciones han demostrados que procedimientos de verificación formal hubiesen revelado defectos, por ejemplo, en el procesador Pentium II y la máquina de terapia radiactiva Therac-25. En la actualidad se invierte más tiempo y esfuerzo en la verificación de sistemas complejos que en su construcción.

Existen muchas técnicas de verificación y validación, entre las cuales podemos destacar *verificación asercional*, *simulación*, *testing* y *model checking*. Esta última -en particular- es en la que nos enfocaremos en este trabajo.

1.2. Model Checking

Model checking [4, 9] es una técnica de verificación que, dado el modelo del sistema bajo estudio y la propiedad requerida, permite decidir automáticamente si la propiedad es satisfecha o no.

El modelo del sistema es frecuentemente generado en forma automática a partir de su descripción, la que se especifica en algún dialecto apropiado de lenguajes de programación tal como C o Java. En tanto, los requerimientos son generalmente formalizados a través del uso de lógicas temporales; tales como la lógica temporal lineal (LTL) o la lógica de cómputo ramificado (CTL). **Model Checker** es la herramienta de software que realiza model checking, examinando todos los estados relevantes del sistema para chequear si los mismos satisfacen la propiedad bajo estudio. Si se encuentra un estado que viole la propiedad considerada, el model checker provee un *contraejemplo* [4, 9] que indica como el modelo puede alcanzar el estado indeseado. El **contraejemplo** tiene como función describir una ejecución del modelo que va desde el estado inicial del sistema al estado que viola la propiedad que está siendo verificada. Con la ayuda de un simulador, el usuario puede reproducir el escenario de la violación, de esta forma obtiene información relevante, a los fines de identificar el o los errores y, consecuentemente, adaptar el modelo -o propiedad- adecuadamente. Este proceso se ilustra en la figura 1.1.

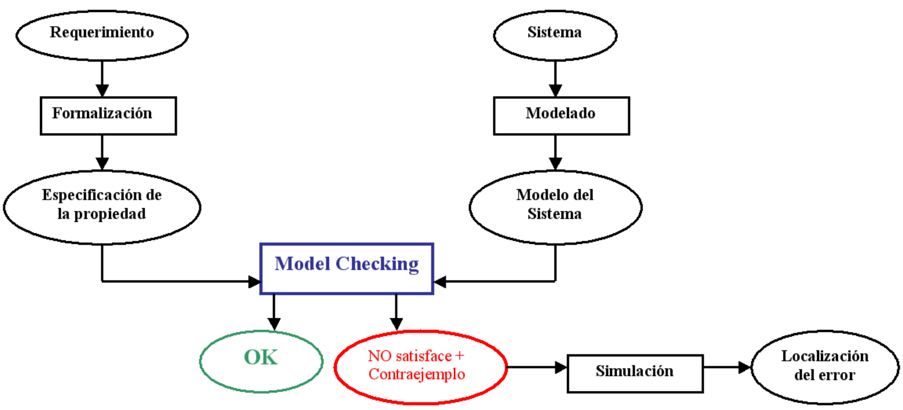


Figura 1.1: Vista esquemática del enfoque Model Checking

Dado el éxito alcanzado en diversos proyectos, hay un interés creciente por parte de la industria en model checking; a punto tal que, varias compañías han comenzado a integrar a sus recursos humanos grupos de investigación e, incluso, desarrollado sus propios model checkers.

Los model checkers usuales verifican propiedades que pueden ser observadas pero no medidas, por ejemplo, *"El resultado generado es correcto"* o, *"El sistema alcanza un estado de deadlock"*. A este tipo de propiedades se las conoce como **propiedades cualitativas** y a los model checkers que las verifican como **model checkers cualitativos**.

Ejemplo 1.1. *En la figura 1.2 se ilustra una situacion en la cual se evalúa la validez de la propiedad:* "El sistema NO alcanza el estado s_3" *($\neg\mathbf{F}\, s_3$ en notación LTL). A la izquierda de la figura se observa el modelo y a la derecha la propiedad. $\mathcal{M} \models \phi$ denota que el modelo $\mathcal{M}$ satisface la propiedad ϕ.*

En este caso, la propiedad no se satisface por lo que el model checker reporta esta situación y entrega el siguiente contraejemplo $s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3$

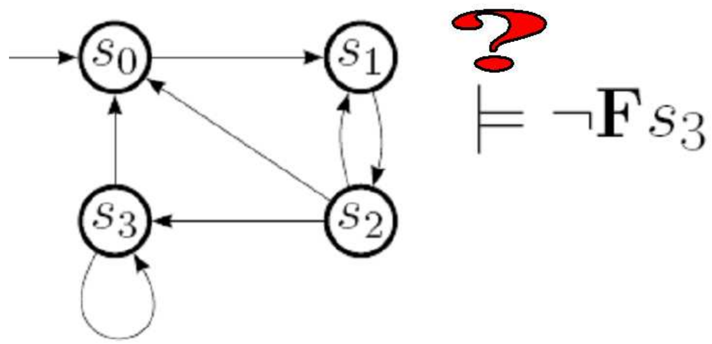


Figura 1.2: Problema de Model Checking

1.3. Necesidad de Probabilidades: Model Checking Cuantitativo

Los algoritmos aleatorios concurrentes y/o distribuidos presentan muchas veces soluciones más veloces que los algoritmos tradicionales [7] e, incluso, soluciones no posibles dentro del dominio de los algoritmos tradicionales. Un factor que contribuye a la aleatoriedad de un sistema es el entorno o medio con el cual deben interactuar las distintas componentes del programa. Este factor se presenta en situaciones tales como la pérdida de un mensaje en la red, la falla de una componente de un sistema, o la disponibilidad de un recurso.

El hecho de considerar probabilidades dentro del comportamiento de los sistemas implica que el conjunto de propiedades asociadas a estos sistemas se sale de la lógica usual. Un ejemplo característico en este sentido se presenta en protocolos con retransmisión limitada donde es imposible establecer que todo mensaje enviado se recibe. A cambio de ello, se podría analizar la validez de la propiedad *"todo mensaje enviado se recibe con probabilidad 0.99"*. A este tipo de propiedades se las denomina **propiedades cuantitativas**, las mismas existen en un rango de magnitudes (usualmente [0,1]) y, por lo tanto, pueden ser medidas. En tanto, a los model checkers que verifican propiedades cuantitativas se los denomina **Model Checkers Cuantitativos** [3, 13, 6, 2].

Dado que la validez o no de una propiedad cuantitativa depende de un conjunto (usualmente infinito) de ejecuciones, los contraejemplos para model chequers cuantitativos son conjuntos (usualmente infinitos) de ejecuciones . Por consiguiente su derivación no es para nada obvia.

Ejemplo 1.2. *En la figura 1.3 se ilustra el problema de model checking cuantitativo. En este caso estamos interesados en virificar si el sistema probabilista y no determinista presentado a la izquierda de la figura satisface la siguiente propiedad:* La probabilidad de alcanzar el estado s_3 es menor o igual que 0,5.

Analizando la situación concluimos que si consideramos la distribución de probabilidades π_1 *el modelo alcanza* s_3 *con probabilidad* 0,4 *con lo cual la propiedad se cumple. Ahora, si analizamos el modelo teniendo en cuenta la distribución* π_2 *observamos que la probabilidad de alcanzar* s_3 *es* 0,6*, por lo tanto la propiedad no se cumple. En este caso el model checker debe reportarlo y presentar un contraejemplo.*

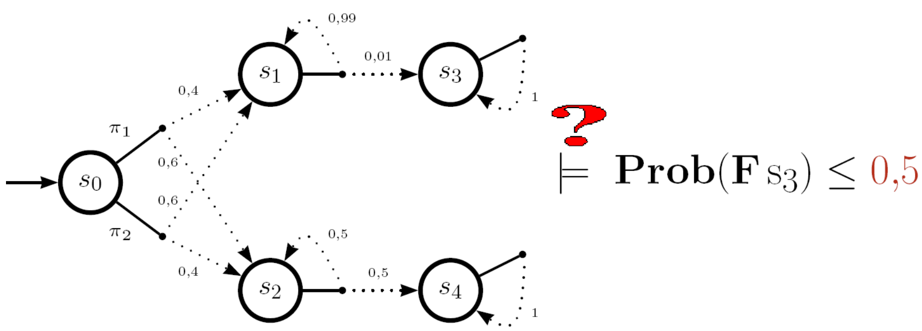


Figura 1.3: Problema de Model Checking Cuantitativo

1.4. Contraejemplos para Model Checkers Cuantitativos

Si bien el fin del model checker es conocer la validez de una propiedad deseada en un modelo dado, es igualmente relevante conocer el *por qué* de las propiedades que no se cumplen.

Como observamos en el ejemplo 1.1, la derivación de contraejemplos en model checkers cualitativos no es compleja. Pero, si analizamos la situación en el ejemplo 1.2, un contraejemplo de dicha violación es un conjunto infinito compuesto de todas las ejecuciones en el que el sistema evoluciona desde s_0 a s_3 a través de la distribución π_2. En la práctica, los conjuntos de ejecuciones que conforman un contraejemplo usualmente evolucionan por distintos sectores del sistema. Todo ello, implica que la derivación de contraejemplos en el caso cuantitativo no es nada sencilla. Quizás esta complejidad sea la razón por la cual este problema casi no haya sido abordado.

1.4.1. Nuestro Enfoque: Carriles

El objetivo de los contraejemplos es brindar información de utilidad para corregir el modelo del sistema, es decir, para "debuguear" el sistema. Puesto que los contraejemplos -en el caso cuantitativo- poseen infinitas ejecuciones su derivación difícilmente ofrezca una pista clara de la ubicación del error en el sistema. De allí la idea de introducir la noción de ***testigos de contraejemplos***. Estos son conjuntos de ejecuciones que en combinación conforman el contraejemplo.

La decisión de *qué* conjunto de ejecuciones presentar como testigos de contraejemplos, constituye un gran desafío y es el corazón de este trabajo.

Para que los testigos de contraejemplo resulten útiles como herramientas a fin de identificar el error en el sistema deben ser similares entre sí. Por ello introducimos la noción de ***carril***, que puede ser pensado como un camino "guía" a través del cual "fluyen" cantidades -usualmente infinitas- de ejecuciones. Al conjunto de dichas ejecuciones lo denominamos ***torrente*** asociado al carril. A continuación especificamos con mayor exactitud estos conceptos

Definición 1.1. *Sea ρ una ejecución finita sin estados repetidos, definimos el conjunto*

$$<\rho> = \{\sigma \text{ ejecucion del modelo} \mid \rho \text{ está "embebido" en } \sigma\}$$

Intuitivamente, un carril está embebido en una ejecución si el mismo puede ser fragmentado de forma tal que, todos sus fragmentos estén en la ejecución y entre los mismos encontremos ciclos (en la ejecución). En la figura 1.4 se ilustra esta idea.

*A ρ lo llamaremos **carril** y a $<\rho>$ lo llamaremos **torrente** asociado a ρ. En tanto, la probabilidad de un carril es la probabilidad de que ocurra alguna ejecución de su torrente* *asociado.*

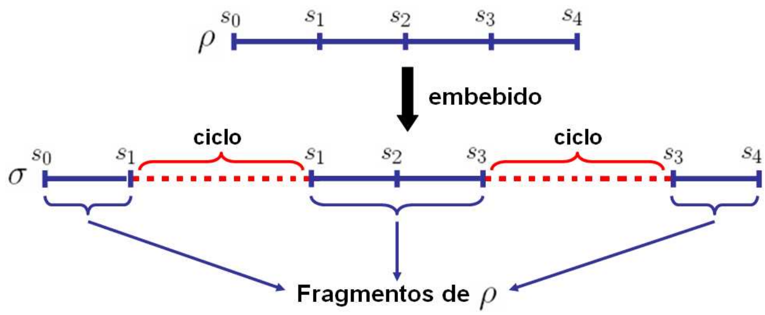


Figura 1.4: ρ embebido en σ

Ejemplo 1.3. *La ejecución $s_0 s_1 s_3$ es un carril del modelo ilustrado en la figura 1.3. El mismo se encuentra embebido en todas las ejecuciones de la forma $s_0(s_1)^i s_3$, con $i \geq 1$.*

Nuestro enfoque consiste en presentar a los torrentes con mayor probabilidad como testigos de contraejemplos. Muchas veces hablaremos del carril con mayor probabilidad (también referido como *máximo*) como testigo del contraejemplo pero debe quedar claro que el testigo es su torrente asociado.

Más adelante, indicamos detalladamente las ventajas de esta técnica por sobre la existente, pero antes veamos cuál es el enfoque del mismo.

1.4.2. Trabajos Relacionados

En la actualidad y hasta donde sabemos, sólo existe un trabajo relacionado: [1]. En el mismo, se plantea presentar la ejecución simple con mayor probabilidad como testigo del contraejemplo. Seguidamente señalamos algunas desventajas inherentes a la selección de ejecuciones simples como herramientas de debugueo

a) Menor precisión en la información reportada para el posterior debugueo: Muchas veces la ejecución con mayor probabilidad no refleja la transición de estados en la que el sistema acumula mayor masa probabilística

b) Grandes similitudes entre resultados distintos: En general las ejecuciones parecidas presentan idéntica información para la tarea de debugueo.

c) Usualmente, la probabilidad de la ejecución con mayor probabilidad que viola la propiedad verificada es insignificante con respecto a la cota de dicha propiedad

d) Infinitas soluciones: Esta es una gran desventaja considerando que se deben evaluar una por una hasta encontrar el error en la etapa de debugueo.

Otra importante carencia en [1] es que se limita a cadenas de Markov de Tiempo Continuo, consecuentemente no contempla comportamientos no deterministas en el sistema.

A continuación presentamos un ejemplo en el que comparamos la elección de carriles como testigos de contraejemplo con la elección de ejecuciones simples.

Ejemplo 1.4. ***Carriles vs Ejecuciones***

La figura 1.5 ilustra una problema usual de model checking cuantitativo. El cuadro 1.1 enumera (columna Rank) las ejecuciones del modelo de la figura 1.5 según su probabilidad asociada. Además enumera los carriles de la cadena que van desde s_0 hasta s_3 o s_4.

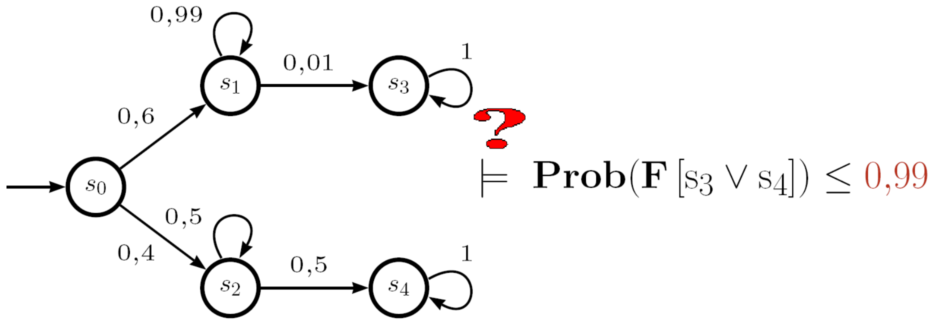


Figura 1.5: Ejemplo de Problema de Model Checking Cuantitativo

A continuación, analizamos las cuatro desventajas previamente citadas.

a) Como observamos en el ejemplo, las ejecuciones con mayor probabilidad nos inducen a pensar que el problema en el sistema original se encuentra cuando el mismo evoluciona hacia el estado s_2. Por el contrario, en realidad la cadena acumula mayor masa probabilista (0.6) cuando evoluciona hacia el estado s_1.

b) Si analizamos las ejecuciones con mayor probabilidad observamos que poseen un prefijo y sufijo común; de allí la idea de agrupar este tipo de ejecuciones como un solo resultado: torrentes asociados a un carril.

c) La cota de la probabilidad de alcanzabilidad cuantitativa es 0.99. Sin embargo la ejecución simple más probable posee probabilidad 0,2.

d) En la tabla 1.1 solo se presentan las 9 ejecuciones con mayor probabilidad pero, dichas ejecuciones son infinitas, podríamos continuar enumerándolas. Como resultado de este proceso obtendríamos infinitas ejecuciones -de la forma $s_0(s_1)^i s_3$ y $s_0(s_2)^j s_4$ para $i, j \geq \mathbb{N}$- intercaladas en la tabla según su probabilidad.

Testigos con Ejecuciones			**Testigos con Carriles**		
Rank	Ejecución	Prob	Rank	Carril	Prob
1	$s_0(s_2)^1s_4$	0.2	1	$s_0s_1s_3$	0.6
2	$s_0(s_2)^2s_4$	0.1	2	$s_0s_2s_4$	0.4
3	$s_0(s_2)^3s_4$	0.05	-	-	-
4	$s_0(s_2)^4s_4$	0.025	-	-	-
5	$s_0(s_2)^5s_4$	0.0125	-	-	-
6	$s_0(s_2)^6s_4$	0.00625	-	-	-
7	$s_0(s_1)^1s_3$	0.006	-	-	-
8	$s_0(s_1)^2s_3$	0.0059	-	-	-
9	$s_0(s_1)^3s_3$	0.0058	-	-	-
$\vdots$	$\vdots$	$\vdots$	-	-	-

Cuadro 1.1: Carriles VS Ejecuciones simples

Luego, en este ejemplo, nuestra técnica reporta el testigo de contraejemplo -carril- $s_0s_1s_4$ *e indica que su probabilidad es* 0,6*, es decir, la probabilidad de su torrente asociado. En tanto la técnica utilizada en [1] reporta la ejecución* $s_0s_2s_4$ *con probabilidad* 0,2.

Ahora, si los testigos de contraejemplos presentados por ambas técnicas no fueran suficientes para localizar el error en el sistema original, se pueden buscar los segundos testigos más probables. En este caso, nuestra técnica reportaría $s_0s_2s_4$ *con probabilidad* 0,4*. Observamos entonces que con nuestra técnica en el segundo intento obtuvimos un contraejemplo puesto que hemos obtenido conjuntos de ejecuciones cuya probabilidad es* $1 > 0{,}99$*. Por otro lado, la técnica utilizada en [1] entregaría la ejecución* $s_0s_2s_2s_4$ *con probabilidad* 0,1*.*

1.5. Alcanzabilidad

Este trabajo se enfoca en la obtención de testigos de contraejemplos para model checkers cuantitativos que verifican **propiedades de alcanzabilidad cuantitativa acotadas superiormente**. Estas, indican que determinada situación puede ser alcanzada con probabilidad menor o igual que un valor probabilista dado. Como ejemplos podemos mencionar: *-Un deadlock es alcanzado con probabilidad menor o igual a 0.05* o, *-Dos procesos están en la región crítica simultáneamente con probabilidad menor o igual a 0.01.* La sintaxis de estas propiedades es la siguiente: $\varphi = \mathcal{P}_{\leq a}(\mathbf{F}\ \psi)$, esto indica que la probabilidad de alcanzar algún estado que satisfaga ψ es menor o igual que a. A ψ la denominamos condición de alcanzabilidad.

Por cada fórmula de alcanzabilidad $\varphi = \mathcal{P}_{\leq a}(\mathbf{F}\ \psi)$ podemos definir el conjunto de estados que satisfacen la condición de alcanzabilidad, es decir, $\mathcal{G}_\varphi = \{s \mid s \models \psi\}$. A los elementos de $\mathcal{G}_\varphi$ los llamamos estados *goal*. Finalmente, el cálculo de alcanzabilidad se reduce a encontrar la ejecuciones que llevan al modelo desde su estado inicial hasta algún estado de $\mathcal{G}_\varphi$. En general omitiremos φ.

Restringir el conjunto de propiedades a verificar a aquéllas de alcanzabilidad no conlleva a una perdida sustancial de generalidad. Dichas propiedades nos permiten especificar gran cantidad de características de un sistema. Un ejemplo de la relevancia de las propiedades de alcanzabilidad es que el problema de Model Checking cuantitativo con propiedades expresadas en LTL se puede reducir a un problema de alcanzabilidad cuantitativo [6, 14].

1.6. El Problema

El problema abordado por el presente trabajo se puede formular de la siguiente manera:

> "Dado un modelo probabilista y no determinista $\mathcal{M}$ y una propiedad de alcanzabilidad cuantitativa $\varphi = \mathcal{P}_{\leq a}(\mathbf{F}\ \psi)$ tal que $\mathcal{M} \not\models \varphi$, derivar el carril -desde el estado inicial del sistema hasta algún estado que satisfaga la condición de alcanzabilidad ψ- con mayor probabilidad."

1.7. Sumario

En el capítulo 2 presentamos las cadenas de Markov de tiempo discreto, hacemos un análisis del cálculo de probabilidades sobre sus ejecuciones y formalizamos la probabilidad de alcanzabilidad sobre las mismas.

En el capítulo 3 se analizan diversos algoritmos de búsqueda que nos resultan de utilidad para hallar del carril más probable.

En el capítulo 4 se desarrolla la técnica utilizada para localizar el carril con mayor probabilidad en una cadena de Markov de tiempo discreto.

En el capítulo 5 analizamos los Procesos de Decisión de Markov y como definir probabilidades sobre sus ejecuciones.

Por último, en el capítulo 6 se extiende la técnica presentada en el capítulo 4 para obtener carriles máximos en cadenas de Markov a procesos de decisión de Markov.

Capítulo 2

Cadenas de Markov

En este capítulo presentamos las cadenas de Markov (CMTD), estructuras introducidas por Andrei Markov en 1906, que son útiles para modelar Sistemas Probabilistas Deterministas. Luego, introducimos conceptos fundamentales de probabilidades necesarios para asociar probabilidades a las ejecuciones de una CMTD. Finalmente definimos la probabilidad de alcanzabilidad y una técnica para calcularla.

2.1. Cadenas de Markov de Tiempo Discreto

Una cadena de Markov de tiempo discreto es un sistema de transiciones probabilistas compuesto por estados y transiciones con probabilidades asociadas. Formalmente definimos una CMTD de la siguiente manera:

Definición 2.1. *Una* ***cadena de Markov de tiempo discreto*** *es una tupla* $\Theta = (S, s_0, \mathcal{M})$*, donde:*

i. *S es el espacio de estados finito del sistema;*

ii. *$s_0 \in S$ es el estado inicial;*

iii. *$\mathcal{M} : S \times S \to [0,1]$ es una matriz de probabilidades que satisface $\sum_{s\in S} \mathcal{M}(t,s) = 1$ para todo $t \in S$;*

Para cada par de estados s y t, $\mathcal{M}(s,t)$ indica la probabilidad de que el sistema pase del estado s al t. Puesto que en el presente trabajo solo trabajamos con cadenas de Markov de tiempo discreto en general las referimos simplemente como cadenas de Markov.

A continuación abordamos los conceptos de caminos y ejecuciones sobre CMTD's.

Definición 2.2 **($Path_s(\Theta)$).** *Sea $\Theta = (S, s_0, \mathcal{M})$ una CMTD y $s_0 \in S$, un s_0-path o camino de Θ sera una secuencia* ***finita*** *de estados de S, $\sigma = s_0 s_1 \dots s_n$, donde para cada*

$0 \leq i < n$ $\mathcal{M}(s_i, s_{i+1}) > 0$. σ_i denota el estado en la i-esima posición de σ, $(\sigma \uparrow i)$ el prefijo finito de σ con longitud n, $|\sigma|$ la longitud de σ, $\mathrm{last}(\sigma)$ el último estado de σ y, $Paths_s(\Theta)$ el conjunto de s-path's de Θ. Cuando $s = s_0$ podremos omitirlo, es decir $Path_{s_0}(\Theta)$=$Path(\Theta)$

Definición 2.3 ($Exec_s(\Theta)$)**.** *Sea $\Theta = (S, s_0, \mathcal{M})$ una* CMTD *y $s_0 \in S$, una ejecución o s_0-exec de Θ sera una secuencia **infinita** de estados de S, $\omega = s_0 s_1 s_2 \ldots$, donde para cada $i \in \mathbb{N}_0$ $\mathcal{M}(s_i, s_{i+1}) > 0$. ω_i denota el estado en la i-esima posición de ω, $(\omega \uparrow n)$ el prefijo finito de ω con longitud n y, $Execs_s(\Theta)$ el conjunto de s-exec's de Θ. Al igual que para el caso de caminos $Exec_{s_0}(\Theta)$=$Exec(\Theta)$.*

Ejemplo 2.1. *La figura 2.1 ilustra una* CMTD $\Theta = (S, s_0, \mathcal{M})$ *con* $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$. *La matriz de probabilidades $\mathcal{M}$ se encuentra a la izquierda de la figura. A la derecha observamos la representación asociada (grafo) a Θ.*

$$\mathcal{M} = \begin{pmatrix} 0 & 0{,}25 & 0{,}5 & 0{,}25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}8 & 0{,}2 & 0 \\ 0 & 0 & 0{,}6 & 0 & 0 & 0{,}4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}5 & 0 & 0{,}5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

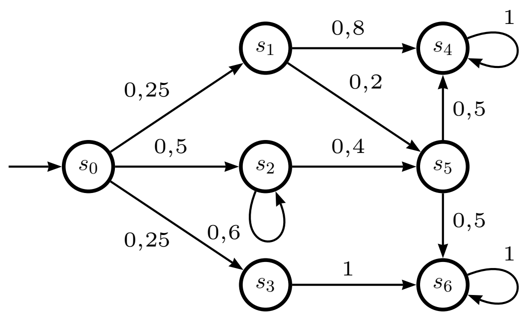


Figura 2.1: Cadena de Markov de tiempo discreto

Además, $s_0 s_2 s_2 s_5$ y $s_0 s_2 s_2 s_2 \ldots$ son ejemplos de caminos y ejecuciones sobre Θ, respectivamente.

2.2. Conceptos Fundamentales de Probabilidades

En esta sección definimos la noción de probabilidad asociada a las ejecuciones en una CMTD. Para ello es necesario acudir a los principales conceptos de σ-Algebras para luego definir una σ-algebra de ejecuciones cuyos elementos son conjuntos de ejecuciones a los que es posible asignarles una probabilidad.

2.2.1. σ-Algebras

Una σ-álgebra sobre Ω es un conjunto $\mathcal{F}$ de subconjuntos de Ω cerrado bajo operaciones contables de conjuntos, es decir, el complemento de elementos de $\mathcal{F}$ es un elemento de $\mathcal{F}$ y la unión o intersección de elementos contables de $\mathcal{F}$ se encuentra en $\mathcal{F}$. Formalizamos estos conceptos de la siguiente manera:

Definición 2.4 (σ-Algebra). *Una σ-algebra sobre Ω es un subconjunto $\mathcal{F}$ de $\mathcal{P}(\Omega)$ con las siguientes propiedades:*

i. $A \in \mathcal{F} \Rightarrow A^c \in \mathcal{F}$

ii. $A_n \in \mathcal{F}, n \geq 1 \Rightarrow \bigcup_{n=1}^{+\infty} A_n \in \mathcal{F}$

El par $(\Omega, \mathcal{F})$ se denomina ***espacio medible****.*

De i. y ii. se deduce que Ω y $\emptyset$ están en $\mathcal{F}$ (puesto que $\mathcal{F}$ es no vacío) y que la σ-algebra es también cerrada bajo intersección (usando las leyes de Morgan).

El concepto de σ-álgebras sirve principalmente para definir medidas sobre un conjunto. Una medida en Ω es una función que le asigna un numero real a los subconjuntos de Ω. Para ello se puede definir una noción precisa de "tamaño" o "volumen" para conjuntos. En este caso nos interesan las medidas probabilistas, las cuales asignan un número real entre 0 y 1 a los subconjuntos. A continuación formalizamos dichos conceptos:

Definición 2.5. *Sea $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ con $\emptyset \in \mathcal{A}$, llamamos* ***medida*** *en $\mathcal{A}$, a cualquier mapeo $\mu : \mathcal{A} \to [0, 1]$ con las siguientes propiedades:*

i. $\mu(\emptyset) = 0$

ii. $A \in \mathcal{A}, A_n \in \mathcal{A}$ *y* $A = \biguplus_{n=1}^{+\infty} A_n \Rightarrow \mu(A) = \sum_{n=1}^{+\infty} \mu(A_n)$

donde $\biguplus$ indica que los conjuntos A_n son disjuntos de a pares.

Además, cuando $\mathcal{A}$ sea una σ-álgebra y se satisfaga $\mu(\Omega) = 1$ diremos que μ es una medida de probabilidad. Formalmente

Definición 2.6. *Sea $\mathcal{F}$ una σ-álgebra sobre Ω, llamamos* ***medida de probabilidad*** *en $\mathcal{F}$, a cualquier mapeo $\mu : \mathcal{F} \to [0, 1]$ con las siguientes propiedades:*

i. $\mu(\emptyset) = 0$

ii. $F \in \mathcal{F}, F_n \in \mathcal{F}$ *y* $F = \biguplus_{n=1}^{+\infty} F_n \Rightarrow \mu(F) = \sum_{n=1}^{+\infty} \mu(F_n)$

iii. $\mu(\Omega) = 1$

En este caso la estructura $(\Omega, \mathcal{F}, \mu)$ se denomina ***Espacio de Probabilidad****.*

Dado un conjunto de subconjuntos siempre es posible obtener la menor σ-álgebra que lo contenga.

Definición 2.7. *Sea $\mathcal{A} \subseteq \mathcal{P}(\Omega)$, se denomina* ***$\sigma$-algebra generada*** *por $\mathcal{A}$, a la σ-algebra en Ω, denotada $\sigma(\mathcal{A})$, igual a la intersección de todas las σ-algebras en Ω que contengan a $\mathcal{A}$. En este caso diremos que $\mathcal{A}$ es el* ***conjunto generador*** *de $\sigma(\mathcal{A})$.*

Ahora veamos que existe una medida de probabilidad sobre σ-álgebras. Para ello introducimos el concepto de *semi anillos*

Definición 2.8. *Sea $\mathcal{S} \subseteq \mathcal{P}(\Omega)$, decimos que $\mathcal{S}$ es un* ***semi anillo*** *sobre Ω si contiene al vacío, es cerrado por intersecciones finitas y, todo conjunto de $\mathcal{S}$ puede escribirse como una unión disjunta finita de elementos de $\mathcal{S}$, es decir:*

i. $\emptyset \in \mathcal{S}$

ii. $A, B \in \mathcal{S} \Rightarrow A \cap B \in \mathcal{S}$

iii. $A, B \in \mathcal{S} \Rightarrow \exists\, n \geq 0, \exists A_i \in \mathcal{S} : A - B = \biguplus_{i=1}^{n} A_i$

La propiedad iii. indica que cuando $A, B \in \mathcal{S}$, existe $n \geq 0$ y $A_1, A_2 \ldots A_n \in \mathcal{S}$ disjuntos de a pares, tales que $A - B = A_1 \biguplus A_2 \biguplus \ldots \biguplus A_n$. Si $n = 0$ se entiende que la correspondiente unión es igual a $\emptyset$ (en cuyo caso $A \subseteq B$).

Finalmente presentamos el teorema de Caratheodory, que asegura la existencia de una medida de probabilidad sobre σ-álgebras.

Teorema 2.1 (Caratheodory). *Sea $\mathcal{S} \subseteq \mathcal{P}(\Omega)$ un semi anillo y sea $\mu : \mathcal{S} \to [0, 1]$ una medida de probabilidad en $\mathcal{S}$. Entonces existe una* ***única*** *medida μ' en $\sigma(\mathcal{S})$ tal que para todo $A \in \mathcal{S}$, $\mu(A) = \mu'(A)$.*

Para facilitar la notación a μ' la llamaremos μ.

Para más detalles de probabilidades y teoría de la medida consultar [10].

2.2.2. σ-Algebra de Ejecuciones

A continuación definimos una σ-álgebra de ejecuciones de una CMTD y la medida de probabilidad sobre ella. A tal fin, introducimos la noción de cilindros básicos, que designa conjuntos de ejecuciones con un prefijo común, formalmente:

Definición 2.9. *Dada una* CMTD *Θ y $\sigma \in Path_s(\Theta)$, definimos el* ***cilindro básico*** *asociado a σ como*

$$\langle \sigma \rangle = \{\omega \in Exec_s(\Theta) \mid \omega \uparrow |\sigma| = \sigma\}$$

Además denotamos con $\mathcal{B}_\Theta$ al conjunto de cilindros básicos de Θ, es decir:

$$\mathcal{B}_\Theta = \{\langle \sigma \rangle \mid \sigma \in Path(\Theta)\}$$

Ahora definimos una medida de probabilidad sobre cilindros básicos

Definición 2.10. *Dada una CMTD* Θ *y el conjunto de sus cilindros básicos* $\mathcal{B}_\Theta$*, definimos la medida de probabilidad* $\mu : \mathcal{B}_\Theta \to [0,1]$ *de la siguiente manera:*

$$\mu\left(\langle\sigma\rangle\right) = \prod_{i=0}^{|\sigma|-2} \mathcal{M}(\sigma_i, \sigma_{i+1})$$

para todo $\langle\sigma\rangle \in \mathcal{B}_\Theta$*.*

Luego hacemos lo propio en relación a σ-álgebra de ejecuciones

Definición 2.11. *Dada una CMTD* $\Theta = (S, s_0, \mathcal{M})$*. Definimos a* $\mathcal{F}_\Theta$ *como la* ***σ-algebra de ejecuciones de*** Θ *generada por el conjunto generador* $\mathcal{B}_\Theta$*.*

No es difícil verificar que $\mathcal{B}_\Theta$ es un semi anillo, luego -por Teorema 2.1- μ tiene una única extensión sobre $\mathcal{F}_\Theta$. Por consiguiente, $(Exec(\Theta), \mathcal{F}_\Theta, \mu)$ forma el espacio de probabilidades de $Exec(\Theta)$.

2.3. Alcanzabilidad

En el presente trabajo el concepto de alcanzabilidad constituye uno de los ejes centrales. En esta sección especificamos la probabilidad de alcanzar estados y luego como calcularla.

2.3.1. Probabilidad de Alcanzabilidad

A continuación definimos la probabilidad de alcanzar un estado t desde otro estado s en una CMTD

Definición 2.12. *Sea* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD*, y* $s, t \in S$*, entonces definimos a la* ***probabilidad de alcanzar*** t ***desde*** s *en* Θ *como:*

$$\mathrm{P_{s \rightsquigarrow t}} = \mu\left(\{\omega \in \mathrm{Exec_s}(\Theta) \mid \exists\, \mathrm{i} \,.\, \omega_\mathrm{i} = \mathrm{t}\}\right) \tag{2.1}$$

Además, definimos la probabilidad de alcanzar un conjunto de estados $S' \subseteq S$ *desde* s *como:*

$$\mathrm{P_{s \rightsquigarrow S'}} = \mu\left(\{\omega \in \mathrm{Exec_s}(\Theta) \mid \exists\, \mathrm{i} \,.\, \omega_\mathrm{i} \in \mathrm{S'}\}\right) \tag{2.2}$$

Proposición 2.1. *Sea* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD*, y* $s, t \in S$ *donde* $s \neq t$*, entonces la probabilidad de alcanzar* t *desde* s *en* Θ *es*

$$\mathrm{P_{s \rightsquigarrow t}} = \mu\left(\bigcup_{\sigma \in \mathrm{C_t^s}} \langle\, \sigma \rangle\right) \tag{2.3}$$

donde el conjunto C_t^s *contendrá los caminos* σ *entre* s *y* t *tales que* t *sea alcanzado solo una vez en* σ*, es decir* $C_t^s = \{\sigma \in Paths_s(\Theta) \mid \mathrm{last}(\sigma) = \mathrm{t} \wedge (\forall 0 \leq \mathrm{i} < |\sigma| - 1 \,.\, \sigma_\mathrm{i} \neq \mathrm{t})\}$.

Ejemplo 2.2. *Ahora supongamos que* s_3 *y* s_5 *satisfacen alguna condición de alcanzabilidad en la cadena de Markov presentada en el ejemplo 2.1.*

Luego, de acuerdo a la ecuación (2.3), la probabilidad de que la cadena alcance dichos estados será:

$$\begin{aligned} \mathrm{P}_{\mathrm{s_0 \rightsquigarrow s_3}} &= \mu\left(\{\langle s_0 s_3\rangle\}\right) \\ &= 0{,}25 \end{aligned} \qquad \begin{aligned} \mathrm{P}_{\mathrm{s_0 \rightsquigarrow s_5}} &= \mu\left(\{\langle s_0 s_1 s_5\rangle, \textstyle\bigcup_{i=0}^{\infty}\langle s_0 (s_2)^i s_5\rangle\}\right) \\ &= 0{,}05 + \textstyle\sum_{i=0}^{\infty}(0{,}5 \cdot (0{,}6)^i \cdot 0{,}4) \\ &= 0{,}55 \end{aligned}$$

Puesto que los conjuntos de ejecuciones $\langle s_0 s_3\rangle$*,* $\langle s_0 s_1 s_5\rangle$ *y* $\bigcup_{i=0}^{\infty}\langle s_0 (s_2)^i s_5\rangle$ *son disjuntos de a pares, la probabilidad* $\mathrm{P}_{\mathrm{s_0 \rightsquigarrow \{s_3, s_5\}}} = \mu\left(\{\langle \mathrm{s_0 s_3}\rangle, \langle \mathrm{s_0 s_1 s_5}\rangle, \bigcup_{\mathrm{i}=0}^{\infty}\langle \mathrm{s_0 (s_2)^i s_5}\rangle\}\right)$ *de que la cadena alcance algún estado que satisface la condición de alcanzabilidad es* 0,8

2.3.2. Cálculo de Alcanzabilidad

Una vez definida la probabilidad de alcanzar un estado, procedemos al cálculo de la misma. Previamente, presentamos algunas definiciones necesarias para obtener una técnica eficiente a tal fin.

Clasificación de Estados

En primer lugar, definimos el concepto de alcanzabilidad entre estados en una cadena de Markov. Un estado será alcanzado desde otro si existe un camino entre ellos. Formalmente

Definición 2.13. *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD *y* $t, s \in S$*, diremos que* t ***es alcanzable desde*** s *si*

$$\{\sigma \in Paths_s(\Theta) \mid \mathrm{last}(\sigma) = \mathrm{t}\} \neq \emptyset$$

Ahora, sea S' un subconjunto de un espacio de estados S, si no hay ninguna transición desde un estado de S' a alguno fuera del mismo entonces diremos que S' es un conjunto cerrado.

Definición 2.14. *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD *y* $S' \subseteq S$*, si* $\mathcal{M}(s,t) = 0$ *para cualquier* $s \in S', t \notin S'$ *entonces diremos que* S' ***es cerrado****.*

Un caso particularmente interesante de un conjunto cerrado es cuando el conjunto S' sólo posee un estado.

Definición 2.15. *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD *y* $s \in S$*, si* $\{s\}$ *es cerrado entonces diremos que* s ***es absorbente****.*

Además distinguimos a los conjuntos de estados mutuamente alcanzables de la siguiente manera:

Definición 2.16. *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD *y* $S' \subseteq S$*, diremos que* S' *es un* ***conjunto fuertemente conexo (CFC)*** *si el estado* t *es alcanzable desde el estado* s *para cualquier* $s, t \in S'$*. Además, si: para todo estado* $u \in S$ *tal que* u *alcanza a todos los estados de* S' *y todos los estados de* S' *alcanzan a* u *implica* $u \in S'$*, diremos que* S' *es un conjunto fuertemente conexo maximal (CFCM).*

A continuación definimos a los conjuntos irreducibles

Definición 2.17 (Conjuntos Irreducibles). *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD *y* $S' \subseteq S$*, diremos que* S' *es irreducible si* S' *es un conjunto cerrado y fuertemente conexo.*

Nótese que si s es absorbente entonces $\{s\}$ es irreducible.

Definición 2.18 (Cadenas de Markov Irreducibles y Reducibles). *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD*, si* S *es irreducible entonces diremos que* Θ *es irreducible. Si* Θ *no es irreducible la llamaremos reducible.*

Ahora supongamos que el estado actual de una CMTD Θ es s, resulta interesante preguntarse si la CMTD volverá a s en una evolución futura del sistema; si la respuesta es "quizás no" el estado s se denomina *transitorio*. Formalizamos dicha idea de la siguiente manera:

Definición 2.19. *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD *y* $t \in S$*, diremos que* t *es un* ***estado transitorio de*** Θ *si*

$$\{\omega \in Exec_t(\Theta) \mid \forall\, i \geq 1 \,.\, \omega_i \neq t\} \neq \emptyset$$

Steady State Analysis

Finalmente, estamos en condiciones de definir una técnica para el cálculo de la probabilidad de alcanzar estados. Para ello utilizamos los resultados enunciados en la teoría de Steady State Analysis [11] para cadenas de Markov reducibles.

Lema 2.1. *Dado* $\Theta = (S, s_0, \mathcal{M})$ *una* CMTD *reducible,* S_t *el conjunto de estados transitorios de* Θ*,* $t \in S_t$ *y* s *absorbente*

$$\mathrm{P_{t \rightsquigarrow s}} = \mathcal{M}(\mathrm{t, s}) + \sum_{\mathrm{u \in S_t}} \mathcal{M}(\mathrm{t, u}) \mathrm{P_{u \rightsquigarrow s}} \tag{2.4}$$

en tanto, si $S_a \subseteq S$ es un conjunto de estados absorbentes

$$\mathrm{P}_{\mathrm{t}\rightsquigarrow\mathrm{S_a}} = \sum_{\mathrm{s}\in\mathrm{S_a}} \mathcal{M}(\mathrm{t},\mathrm{s}) + \sum_{\mathrm{u}\in\mathrm{S_t}} \mathcal{M}(\mathrm{t},\mathrm{u})\mathrm{P}_{\mathrm{u}\rightsquigarrow\mathrm{S_a}} \tag{2.5}$$

Es posible demostrar que si S_t es finito el conjunto de ecuaciones en (2.4) y (2.5) tienen una única solución. Luego, puesto que el espacio de estados S de una CMTD Θ es finito las ecuaciones (2.4) y (2.5) definen correctamente la probabilidad de alcanzar el estado s (respectivamente S_a) desde t en Θ .

Concluimos que para calcular la probabilidad de alcanzar un estado (o conjunto de estados) particular en una CMTD nos basta con resolver un sistema de ecuaciones lineales. Las variables de este sistema serán $\mathrm{P}_{\mathrm{s_t}\rightsquigarrow\mathrm{s}}$ con $s_t \in S_t$.

A continuación ilustramos esta conclusión con un ejemplo.

Ejemplo 2.3. *Supongamos que queremos calcular la probabilidad* $\mathrm{P}_{\mathrm{s_0}\rightsquigarrow\mathrm{s_5}}$ *en la* CMTD *del ejemplo 2.1. Para ello debemos resolver el siguiente sistema de ecuaciones:*

$$\begin{array}{rclcrcl} x_0 & = & 0{,}25 \cdot x_1 + 0{,}5 \cdot x_2 + 0{,}25 \cdot x_3 & \qquad & x_1 & = & 0{,}2 \\ x_2 & = & 0{,}6 \cdot x_2 + 0{,}4 & & x_3 & = & 0 \end{array}$$

x_i representa la probabilidad $\mathrm{P}_{\mathrm{s_i}\rightsquigarrow\mathrm{s_5}}$ y s_5 es transformado en un estado absorbente. Esta técnica de hacer absorbente un estado previo a calcular la probabilidad de alcanzarlo será usada a lo largo del presente trabajo.

Por lo que, para calcular el valor de x_0 (y consecuentemente el de $\mathrm{P}_{\mathrm{s_0}\rightsquigarrow\mathrm{s_5}}$) debemos resolver el siguiente sistema

$$\left(\begin{array}{cccc|c} 1 & -0{,}25 & -0{,}5 & -0{,}25 & 0 \\ 0 & 1 & 0 & 0 & 0{,}2 \\ 0 & 0 & 0{,}4 & 0 & 0{,}4 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right)$$

Para ello podemos usar el método de eliminación Gaussiana, entre otros. El sistema reducido es el siguiente

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0{,}55 \\ 0 & 1 & 0 & 0 & 0{,}2 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right)$$

Luego $\mathrm{P}_{\mathrm{s_0}\rightsquigarrow\mathrm{s_5}} = 0{,}55$ tal como lo anticipamos en el ejemplo 2.2.

Capítulo 3

Algoritmos de Búsqueda

En este capítulo presentamos los algoritmos de búsqueda Depth First Search (DFS), Breadth First Search (BFS), Best First Search (BF) y $Z^\star$. Tanto DFS como $Z^\star$ serán utilizados, posteriormente, para la obtención de carriles sobre una CMTD.

Introducimos las ideas de cada algoritmo y analizamos un método de búsqueda iterativo (MBI) que nos será de gran utilidad para definir DFS y BFS. Seguidamente estudiamos las modificaciones necesarias para que el MBI implemente BF. Asimismo, explicamos como derivar $Z^\star$ a partir de BF. Por último, damos ejemplos de todos los casos.

3.1. Presentación

En esta sección presentamos de manera informal el funcionamiento de los algoritmos analizados en este capítulo.

Depth First Search: DFS es considerado uno de los más populares algoritmos de *búsqueda desinformada*, comúnmente usado para búsquedas sobre árboles o grafos. En nuestro caso, nos resulta útil para las búsquedas sobre cadenas de Markov.

Intuitivamente el comportamiento del algoritmo es el siguiente: comienza en un estado específico s que se convierte en el *estado actual*, en un principio s es el estado inicial de la cadena. Luego, selecciona un estado s' sucesor de s; si s' ya ha sido procesado por el algoritmo, entonces continúa eligiendo sucesores de s. Si, por el contrario, s' aún no ha sido procesado por el algoritmo, éste empieza a procesarlo y lo convierte en el nuevo estado actual. Continuamos con este procedimiento hasta que, o bien alcanzamos un estado *goal*, o bien alcanzamos un estado cuyos sucesores han sido procesados. En este caso, debemos regresar por el camino mediante el cual se alcanzó el estado actual hasta encontrar el último estado visitado que no ha sido *completamente procesado*. Es decir, que tiene algún hijo aún no procesado. El procedimiento termina cuando el estado inicial ha sido completamente procesado.

Breadth First Search: Este es otro algoritmo de búsqueda muy popular y, al igual que DFS, puede realizar búsquedas desinformadas sobre cadenas de Markov.

Intuitivamente el comportamiento del algoritmo es el siguiente: comienza en un estado específico s, que está en el nivel 0. Al comenzar el algoritmo, s es el estado inicial de la cadena. En la primera etapa, se visitan todos los estados en el nivel 1. Es decir, aquellos estados sucesores de s. En la segunda etapa, se visitan todos los estados en el nivel 2, o sea, todos los estados sucesores de algún estado en el nivel 1. Este procedimiento continúa hasta encontrar un estado *goal* o hasta que todos los estados hayan sido visitados.

Best First Search: Este algoritmo optimiza el funcionamiento de DFS y BFS seleccionando el estado más promisorio para procesar en lugar de simplemente seleccionar algún estado sucesor (DFS) o vecino (BFS). Al igual que DFS y BFS el algoritmo concluye al encontrar la primera solución.

Para determinar que tan promisorio es un estado s, se utiliza una *función de evaluación heurística* $f(s)$ que -en general- no sólo depende de s, sino también de la descripción de los estados goal, la información generada por el algoritmo hasta el momento de la evaluación de f sobre s y, cualquier otra información relevante en el dominio del problema.

Muchas estrategias Best-First difieren en el tipo de función de evaluación que utilizan. El algoritmo que describimos en este capítulo es común a todas estas estrategias, puesto que no imponemos restricciones a f.

Z*: El algoritmo Z* es derivado de BF demorando su finalización hasta tanto no se haya encontrado la solución óptima, la que -en general- no es la primera en ser encontrada. Además, se imponen restricciones sobre la función de evaluación f. La definición de esta función determina en gran medida el rendimiento del algoritmo y por ello resulta muy importante.

3.2. Esquema de Método Iterativo de Búsqueda

A continuación describimos el MBI que nos servirá para definir DFS, BFS y -en parte- BF.

En términos generales, el trabajo del algoritmo es recorrer los estados comenzando desde el inicial -aún no especificaremos en que orden lo hace- hasta encontrar algún estado que satisfaga una condición dada, asumiendo que existe al menos uno.

A medida que el algoritmo evoluciona en su ejecución almacena información en los estados indicando el recorrido realizado hasta alcanzarlos. De esta manera, el algoritmo va construyendo un árbol a medida que avanza en su ejecución. Los nodos del árbol son los estados que han sido alcanzados y hay una arista entre s y s' si s' ha sido alcanzado por el algoritmo desde el estado s. Para implementar esta característica basta asignarle a cada estado un enlace hacia su estado predecesor en la ejecución del algoritmo. A dicho árbol lo llamaremos ***árbol de recorrido***.

Luego, si el algoritmo encuentra un estado s que satisfaga la condición de finalidad, se puede obtener a partir del árbol de recorrido -ascendiendo desde s hasta alcanzar la raíz- un único camino entre el estado inicial del sistema y s.

Para la implementación de dicho método se requiere una estructura de datos que referenciamos *Struct*. Esta estructura cuenta con las operaciones *insert*, *get* e *is-notEmpty*. Estas operaciones agregan un estado a la estructura, quitan un estado de la estructura y, nos indican si la estructura está vacía, respectivamente.

Además, son necesarias las operaciones sobre estados *mark*, *not-marked* y *succ* las que son usadas para marcar un estado, distinguir entre estados que han sido marcados de aquéllos que no lo han sido y, obtener la lista de sucesores de un estado; respectivamente.

El pseudo código del algoritmo se encuentra en la figura 3.1

```
MBI(CMTD (S,s0,M), Struct struct)
for (all s ∈ S) s.padre=NULL
insert(s0 , struct)
while is-notEmpty(struct)
    s=get(struct)
    mark(s)
    for (all t ∈ succ(s) ∧ not-marked(t))
        t.padre=s
        if (is-goal(t)) finish-succefully(t)
        else insert(t , struct)
    end-for
end-while
```

Figura 3.1: Pseudo código del Método de Búsqueda Iterativo

3.3. Los Algoritmos

El MBI resulta interesante puesto que nos permite realizar distintos tipos de exploración sobre cadenas de Markov según la estructura empleada. El cuadro 3.1 ilustra dichos tipos de exploración. En el mismo, la función *add* agrega estados a una cola con prioridades. En caso de que el estado a agregar se encuentre en la misma, se actualiza su prioridad y, de ser necesario, su posición en la cola.

Estructura (Struct)	**insert**	**get**	**Técnica de Exploración**
Pila	push	pop	Depth First Search
Cola	enqueue	dequeue	Breadth First Search
Cola de Prioridades	add	remove	$\approx$ Best First Search

Cuadro 3.1: Técnicas de exploración de acuerdo al tipo de estructura

De esta manera quedan definidos los algoritmos DFS y BFS.

En cuanto a BF la situación es distinta, dado que aún usando una cola de prioridades IBM no se comporta exactamente como BF. La diferencia radica en que BF permite desmarcar estados y volverlos a colocar en la cola con una nueva prioridad.

En la figura 3.2 presentamos la versión modificada de MBI que implementa a BF, donde la función f es la función de evaluación heurística mencionada en la sección 3.1.

El algoritmo Z*: En su forma general el algoritmo BF es sólo un esqueleto de estrategias y está lejos de exhibir los detalles necesarios para su implementación. Esto se debe a que la función de evaluación es arbitraria; de este modo el algoritmo no especifica su función heurística, como así tampoco, de donde obtiene la información para decidir que tan promisorio es un estado o como propagar esta información a través de la cadena.

Como dijimos anteriormente, Z* es un caso particular de BF. Una de las modificaciones a BF necesarias para implementarlo radica en la restricción de la función de evaluación, en tanto la segunda modificación está ligada a la condición de finalización del algoritmo BF. El objetivo es asegurar la obtención de una solución óptima.

Seguidamente, explicamos las condiciones necesarias para la satisfacción de las modificaciones antedichas

Primer Condición: f debe ser recursiva Este requerimiento nos indica que para cada par de estados s y s', donde s es el predecesor de s', $f(s')$ es computada por $f(s') = F[\psi(s), f(s), h(s')]$. Donde F es una función de combinación arbitraria, $\psi(s)$ un conjunto de parámetros locales caracterizando a s y h es una función que asocia a cada estado s

```
BF(CMTD (S,s0,M))
PQ⟵ empty Priority Queue
for (all s ∈ S) {s.f=0 ; s.padre=NULL}
add(s0,PQ)
while is-notEmpty(PQ)
    s=remove(PQ)
    mark(s)
    for (all t∈succ(s))
        if (is-goal(t)) finish-succefully(t)
        if (not-marked(t) ∧ t∉PQ)
            t.padre=s
            add(t,PQ)
        else
            if (f(t)>t.f)
                t.padre=s
                if(marked(t)) unmark(t)
                add(t,PQ)
            end-if
        end-if
    end-for
end-while
```

Figura 3.2: Pseudo código de Best First Search

una **sobre** estimación de la máxima probabilidad entre los caminos desde s a algún estado que satisfaga la condición de alcanzabilidad.

Por último, F debe satisfacer la propiedades de *preservación de orden* definida por la ecuación (3.1). La misma nos asegura que si dos caminos σ_1 y σ_2 de s a s' son encontrados, donde σ_1 posee mayor probabilidad que σ_2, los procesos futuros de búsqueda no invertirán esta situación.

$$\begin{aligned}&F[\psi(s_1), f(s_1), h_1(s')] \geq F[\psi(s_2), f(s_2), h_1(s')] \Rightarrow \\ &F[\psi(s_1, f(s_1), h_2(s')] \geq F[\psi(s_2), f(s_2), h_2(s')].\end{aligned} \tag{3.1}$$

para todos los estados s_1, s_2 y $s' \in succ(s_1) \cap succ(s_2)$ y todas las funciones heurísticas h_1 y h_2.

Segunda condición: Demora del test de terminación Esta condición nos impone que el algoritmo demore su finalización hasta tanto encuentre la mejor solución en lugar de finalizar con la primer solución encontrada como lo hace BF.

Para garantizarlo basta con asegurar que el estado que satisface la condición deseada y, por lo tanto, que induce una solución, sea el que mayor prioridad posee entre los estados que se encuentran en la cola de prioridades. Consecuentemente, es suficiente modificar el algoritmo BF de modo tal de realizar el test de terminación al estado obtenido por la operación *remove* (véase línea 5 del algoritmo BF- figura 3.2).

Finalmente presentamos en la figura 3.3 el pseudo código del algoritmo Z*. En el mismo f es una función *recursiva* que satisface la propiedad de *preservación de orden*, mientras que h es una *sobre* estimación de la solución optima.

```
Z*(CMTD (S,s0,M))
PQ⟵ empty Priority Queue
for (all s ∈ S) {s.f=0 ; s.padre=NULL}
add(s0,PQ)
while is-notEmpty(PQ)
    s=remove(PQ)
    if (is-goal(t)) finish-succefully(t)
    mark(s)
    for (all t∈succ(s))
        if (not-marked(t) ∧ t∉PQ)
            t.padre=s
            add(t,PQ)
        else
            if (f(t)>t.f)
                t.padre=s
                if(marked(t)) unmark(t)
                add(t,PQ)
            end-if
        end-if
    end-for
end-while
```

Figura 3.3: Pseudo Código de Z*

Nótese que DFS es un caso particular de Z* configurando a f de la siguiente manera:

$$f(s) = \begin{cases} 0, \text{ si } s \text{ es el estado inicial} \\ f(s.padre) + 1, \ cc \end{cases}$$

3.4. Ejemplo

En la figura 3.4 es posible observar la exploración de estados de una cadena de Markov según los distintos algoritmos de búsqueda presentados en el capítulo. Los estados con doble línea son considerados goal. En tanto los estados s_7, s_8, s_9 y s_{10} son considerados absorbentes. Se encuentran marcados en azul los estados involucrados en las soluciones entregadas por los distintos algoritmos.

Cabe destacar que los algoritmos concluyen una vez encontrada la primer solución (DFS, BFS y, BF) o la solución óptima (Z*). Aún así, indicamos el orden en que los restantes estados son explorados con el objetivo de clarificar el funcionamiento de cada algoritmo.

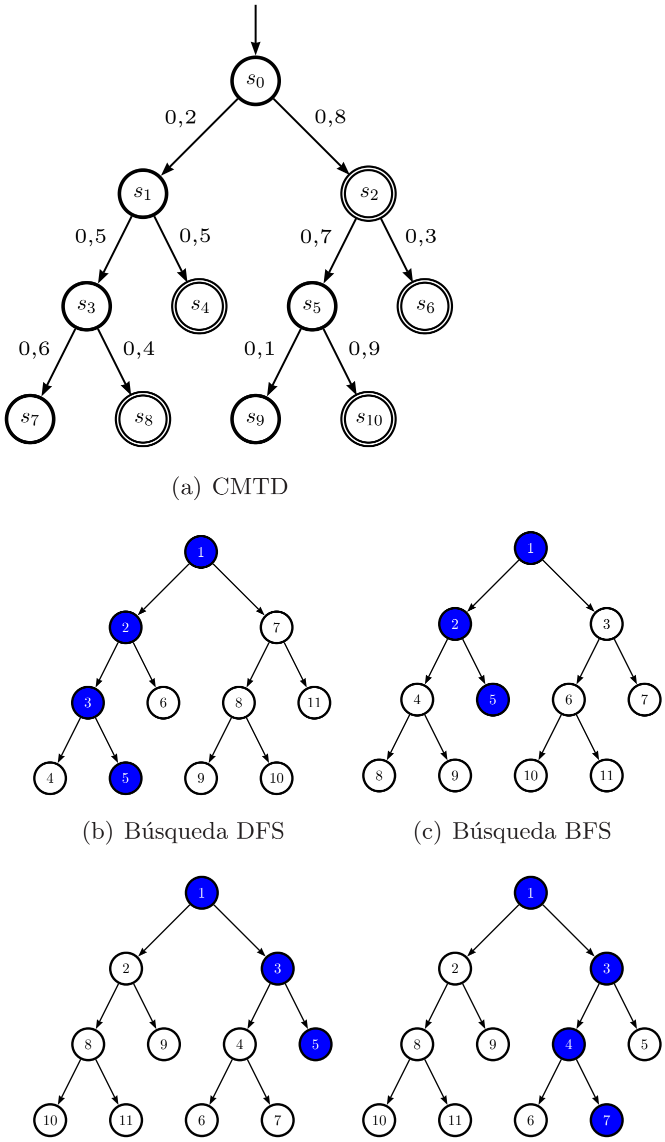
Es importante aclarar que el orden en que hemos decidido procesar estados sucesores está impuesto por la gráfica, dicho orden se define descendentemente de izquierda a derecha.

La función de evaluación utilizada por BF y Z* en este ejemplo es la siguiente:

$$f(s) = \begin{cases} 1, \text{ si } s \text{ es el estado inicial} \\ f(s.padre) \cdot \mathcal{M}(s.padre, s) \cdot h(s), \ cc \end{cases}$$

donde -en este caso- $h(s) = 1$ para todo s en S.

Como se puede observar en la figura 3.4 (e) el algoritmo Z* es el que encuentra la solución óptima $s_0 s_2 s_5 s_{10}$ cuya probabilidad es 0,504.



(a) CMTD

(b) Búsqueda DFS

(c) Búsqueda BFS

(d) Búsqueda BF

(e) Búsqueda Z*

Figura 3.4: Técnicas de búsqueda sobre Cadenas de Markov

Capítulo 4

Derivación de Carriles

En este capítulo presentamos la técnica utilizada para obtener el carril con mayor probabilidad en una cadena de Markov. Para ello, formalizamos los conceptos asociados a carriles. Luego, presentamos la técnica empleada para localizar carriles máximos, la misma se divide dos etapas. En la primera, se realiza un preprocesamiento de la cadena de Markov obteniendo una cadena reducida y, en la segunda, se procede a la búsqueda del carril máximo sobre la cadena reducida. Seguidamente, estudiamos dos técnicas para optimizar el rendimiento del algoritmo. Por último, presentamos un esquema ilustrando el uso de la técnica propuesta.

4.1. Carriles

A continuación formalizamos las nociones asociadas a carriles introducidas en el capítulo 1.

Un camino r es denominado carril si es acíclico, es decir:

Definición 4.1. *Dada una CMTD* Θ *y* $r \in Path(\Theta)$*, diremos que* r *es un* ***carril*** *de* Θ *si* $r_i \neq r_j$ *para todo* $i \neq j$*.*

Ahora podemos pensar en este carril como conductor de un conjunto de ejecuciones, formalmente:

Definición 4.2. *Dada una CMTD* Θ *y un carril* r *de* Θ*, definimos el* ***torrente*** *asociado a* r *en* Θ *como*

$$\lessdot \mathrm{r} \gtrdot = \{\langle\sigma\rangle \mid \sigma \in \mathrm{Path}(\Theta) \textit{ y } \mathrm{r} \textit{ está “embebido” en } \sigma\}$$

donde r *esta* ***embebido*** *en* σ *si existen* $n_0, \ldots, n_k$, $0 = n_0 \leq n_1 \leq n_2 \leq \ldots \leq n_k = |r| - 1$, $0 = m_0, \ldots, m_{k-1}$ *y* $m'_1, \ldots m'_k = |\sigma| - 1$ *con* $m'_i \leq m_i \leq m'_{i+1}$ *tales que:*

$$(\forall\, 0 \leq i < k \,.\, r[n_i \ldots n_{i+1}] = \sigma[m_i \ldots m'_{i+1}]) \wedge (\forall\, 0 < i < k \,.\, \sigma_{m_i} = \sigma_{m_{i'}})$$

Además, $\mathrm{Carril}_s(\Theta)$ es el conjunto de todos los carriles que van desde s hasta algún estado goal de una cadena.

Ejemplo 4.1. *Analicemos la CMTD de la figura 4.2(a), en la misma $r = s_0s_1s_4s_5s_6$ es un carril y $\sigma_1 = s_0s_1s_4s_5s_2s_0s_1s_4s_5s_6$, $\sigma_2 = s_0s_1s_3s_4s_5s_2s_0s_1s_4s_5s_6$ son caminos. Ahora veamos que r esta embebido en σ_1 y en σ_2. Para ello debemos identificar los valores n_i, m_i y $m_{i'}$ que lo atestiguan. Los mismos se ilustran en el cuadro 4.1.*

Camino	n_0	n_1	n_2	m_0	$\mathrm{m}_{1'}$	m_1	$\mathrm{m}_{2'}$
σ_1	0	3	4	0	3	8	9
σ_2	0	1	4	0	1	7	10

Cuadro 4.1: Valores que atestiguan r embebido en σ_1 y σ_2

Dado que $Path(\Theta)$ es un conjunto numerable, $\lessdot \mathrm{r} \gtrdot$ puede escribirse como una unión numerable de cilindros básicos. Por consiguiente, $\lessdot \mathrm{r} \gtrdot$ es medible en $\mathcal{F}_\Theta$. Luego $\mu\,(\lessdot \mathrm{r} \gtrdot)$, la probabilidad de que ocurra alguna ejecución del torrente definido por el carril r, está bien definida.

4.2. Primer Etapa: Preprocesamiento

En esta sección desarrollamos la técnica utilizada para -a partir de una cadena de Markov Θ- obtener una cadena de Markov Θ' acíclica con determinadas propiedades que describiremos a lo largo de la sección. A esta cadena la llamamos cadena *reducida* asociada a Θ y la denotamos $[\Theta]$. La búsqueda de carriles máximos se hará sobre $[\Theta]$ y, a partir de los mismos, derivamos los carriles máximos de Θ.

La derivación de $[\Theta]$ a partir de Θ se obtiene mediante un proceso de dos fases. En la primera, se identifican todos los carriles desde el estado inicial de Θ a algún estado goal y todos los conjuntos fuertemente conexos maximales de Θ.

Formalmente, debemos obtener el conjunto de carriles -$\mathrm{R}^{\mathcal{G}}_{s_0}(\Theta)$- y de componentes fuertemente conexas maximales -CFCM(Θ)- de Θ:

$$\mathrm{R}^{\mathcal{G}}_{s_0}(\Theta) = \{\sigma \in \mathrm{Carril}_{\mathrm{s}_0}(\Theta) \mid \mathrm{last}(\sigma) \in \mathcal{G} \wedge (\forall\, 0 \leq \mathrm{i} < |\sigma| - 1 \,.\, \sigma_{\mathrm{i}} \notin \mathcal{G})\} \tag{4.1}$$

$$CFCM(\Theta) = \{S' \subseteq S \mid S' \text{ es un conjunto fuertemente conexo maximal de } \Theta\} \tag{4.2}$$

Cuando s_0 y $\mathcal{G}$ quedan claros por el contexto podemos omitirlos. En tal caso denotamos $\mathrm{R}_{s_0}^{\mathcal{G}}(\Theta)$ con $\mathrm{R}(\Theta)$. Además, con frecuencia usaremos el término carril para referirnos al carril entre s_0 y algún estado en $\mathcal{G}$.

En la segunda fase, nos encargamos de obtener una cadena acíclica -a partir de Θ- cuyo espacio de estados posea sólo los estados necesarios para el cálculo de $\mu\,(<\mathrm{r}>)$, para todo $r \in \mathrm{R}(\Theta)$.

4.2.1. Primer Fase: DFS Extendido

Con el objetivo de obtener todos los elementos de $\mathrm{R}(\Theta)$ y $\mathrm{CFCM}(\Theta)$ exploramos la cadena Θ desde su estado inicial en busca de estados goal y de ciclos. Para ello utilizaremos una técnica Depth First Search.

El pseudo código del algoritmo utilizado con este fin es el de la figura 4.1, al mismo lo llamaremos DFS Extendido (DFS-E). Entre las principales diferencias con la versión original de DFS (véase figura 3.1) encontramos las siguientes:

1. El algoritmo no concluye hasta no procesar todos los estados.

2. Cada estado s tiene asociado un flag que indica si s está involucrado en algún carril, o ciclo, en ambos o ninguno de ellos.

3. A cada estado se le asocia el estatus *no-alcanzado*, *alcanzado* o *procesado* según su condición en la evolución del algoritmo.

4. En el cuerpo del **for** -línea 6 del algoritmo- no sólo se tienen en cuenta los estados sucesores que el algoritmo aún no ha alcanzado, sino que los evalúa, cualquiera sea su estatus.

Para poder identificar todos los carriles y ciclos almacenamos información en cada uno de los estados explorados por el algoritmo. Cada estado s tiene un campo llamado *flag* que nos indica si $s \in [\mathrm{States}(\mathrm{R}(\Theta)) - \mathrm{States}(\mathrm{CFCM}(\Theta))]$, es decir, que s está involucrado en algún carril; $s \in [\mathrm{States}(\mathrm{CFCM}(\Theta)) - \mathrm{States}(\mathrm{R}(\Theta))]$, es decir, que s está involucrado en algún ciclo; $s \in [\mathrm{States}(\mathrm{R}(\Theta)) \cap \mathrm{States}(\mathrm{CFCM}(\Theta))]$, es decir, que s está involucrado en algún carril y algún ciclo; o $s \notin [\mathrm{States}(\mathrm{R}(\Theta)) \cup \mathrm{States}(\mathrm{CFCM}(\Theta))]$, es decir, que s no está involucrado en ningún carril, ni ciclo. Estas situaciones se identifican con $s.flag\!=\!1$, $s.flag\!=\!2$, $s.flag\!=\!3$ y $s.flag\!=\!0$, respectivamente.

Asimismo, para poder reconstruir los carriles y ciclos a partir de la información suministrada por el flag de cada estado, éstos almacenan información que permite identificar cada uno de los carriles y ciclos a los que pertenecen. La información inherente a los carriles de cada estado s es almacenada en una 4-uplas, una por cada carril en el que se encuentre involucrado. Esta 4-upla consta de un identificador id de carril, el predecesor de s en el carril id, el sucesor de s en el carril id y la distancia (cantidad de estados intermedios) al estado goal del carril id. En tanto, la información inherente a ciclos de s, es almacenada en una lista que contiene los sucesores de s en su componente fuertemente conexa maximal (Véase ejemplo 4.2).

Además, durante la ejecución de DFS-E cada estado s de la cadena de Markov tendrá uno de los siguientes tres estatus asociados: *no-alcanzado* que indica que el estado aún no ha sido alcanzado por el algoritmo; *alcanzado* que indica que el estado ha sido alcanzado por el algoritmo, pero alguno de sus sucesores aún no; y, *procesado* que indica que el estado ha sido alcanzado por el algoritmo y todos sus sucesores también. En este caso, decimos que el estado ha sido *completamente procesado*. Cabe destacar que en su versión original DFS sólo distingue entre estados alcanzados -los estados marcados- y no alcanzados -los estados que no están marcados.

Finalmente, veamos como localiza los carriles y ciclos DFS-E. Para ello supongamos que el algoritmo está procesando un estado s. Es decir que s acaba de extraerse de la pila. La clave está en analizar el estatus de cada uno de los estados t sucesores de s. Las posibilidades son las siguientes:

1. El estatus de t es *no-alcanzado*: En este caso el algoritmo se comporta como lo hacía en su versión original, marcando los estados t, $t.padre$, $(t.padre).padre$, $\cdots$, s_0 como miembros de un mismo carril si t es un estado goal -esta tarea es realizada por la función *marcar-carril*- o, en caso contrario, agregándolo a la pila. Dicha situación se ilustra en la figura 4.2 (b).

2. El estatus de t es *alcanzado*: En este caso, s es un predecesor de t, ante lo cual estamos en presencia de un ciclo. Por consiguiente, el algoritmo marca los estados $t, t.padre, \cdots, s$ como estados pertenecientes a un ciclo. Dicha tarea es realizada por la función *marcar-ciclo*. Ls situación se ilustra en la figura 4.2 (c).

3. El estatus de t es procesado: En este caso se debe verificar lo siguiente:

    - Si t está involucrado en uno o más carriles -según la información obtenida por el algoritmo hasta el momento- estamos en presencia de uno o más nuevos carriles (dependiendo de la cantidad de carriles en los cuales se encuentre involucrado t). Consecuentemente se deben marcar los estados t, $t.padre$, $(t.padre).padre$, $\cdots$, s_0 como miembros de uno o más carriles y actualizar los sufijos de los carriles de t adecuadamente. Esta tarea es realizada por la función *p-marcar-carril*. Dicha situación se ilustra en la figura 4.2 (d).
    - Si alguno de los estados s' predecesores de t forma parte -según la información obtenida por el algoritmo hasta el momento- de la componente fuertemente

conexa maximal de t se deben marcar los estados s, $s.padre$, $(s.padre).padre$, $\cdots$, s' como estados involucrados en un ciclo. Esta situación se ve reflejada en la figura 4.2 (d). Dicha tarea es realizada por la función *p-marcar-ciclo*.

Ejemplo 4.2. *A continuación indicamos la información obtenida por DFS-E de la CMTD* Θ *de la figura 4.2 (a).*

Como ya mencionamos cada estado s posee información de los carriles y ciclos en los que se encuentra involucrado. En el cuadro 4.2.1 se resume dicha información.

A partir de la información suministrada por la ejecución de DFS-E, podemos identificar los carriles y CFCM de una cadena de Markov. En este caso la información derivada es: $R_{s_0}^{\{s_6\}}(\Theta) = \{s_0 s_1 s_4 s_5 s_6, s_0 s_1 s_3 s_4 s_5 s_6\}$, $CFCM(\Theta) = \{\{s_0, s_1, s_2, s_3, s_4, s_5\}\}$.

Estado	**Flag**	**Carriles(id,suc,pred,dist)**	**Ciclos(suc)**
s_0	3	$[(1,4,\emptyset,s_1),(2,5,\emptyset,s_1)]$	$[s_1]$
s_1	3	$[(1,3,s_0,s_4),(2,4,s_0,s_3)]$	$[s_4,s_3]$
s_2	2	[]	$[s_0]$
s_3	3	$[(2,3,s_1,s_4)]$	$[s_4]$
s_4	3	$[(1,2,s_1,s_5),(2,2,s_3,s_5)]$	$[s_5]$
s_5	3	$[(1,1,s_4,s_6),(2,1,s_4,s_6)]$	$[s_2]$
s_6	1	$[(1,0,s_5,\emptyset),(2,0,s_5,\emptyset)]$	[]

Cuadro 4.2: Información suministrada por DFS-E

4.2.2. Segunda Fase: Reducción de CFCM's

Una vez obtenidos los conjuntos R(Θ) y CFCM(Θ) nos centramos en obtener una cadena de Markov acíclica $[\Theta]$ a partir de Θ.

La cadena reducida contendrá todos los estados acíclicos involucrados en algún carril de la cadena, esto es, los estados $s \in States(R(\Theta)) - States(CFCM(\Theta))$. Además, contendrá un subconjunto de todos los estados involucrados en carriles y ciclos; dicho subconjunto está definido como $\{s \in S \mid \exists\, r \in R(\Theta), c \in CFCM(\Theta) \,.\, s = min_r(States(r) \cap c)\}$, donde $r_i = \min_r(A)$ si $r_i \in A$ y s antecede a todos los estados de A en r, es decir para todo j tal que r_j esta en A se tiene que $i \leq j$. Por último, $[\Theta]$ contiene un estado distinguido utilizado para direccionar toda la masa probabilística "perdida" en la reducción, a dicho estado lo denominamos *sink*.

Formalizamos, entonces, la noción de cadena reducida de Θ como:

```
DFS-E(CMTD ⟨S, s0, M⟩)
for (all s ∈ S) {s.estatus=no-alcanzado;s.padre=NULL}
push(s0 , Stack)
while is-notEmpty(Stack)
    s=pop(Stack)
    s.estatus=alcanzado
    for (all t ∈ succ(s))
        switch (t.estatus)
            case no-alcanzado
                t.padre=s
                if (es-goal(t)) marcar-carril(t)
                else push(t , Stack)
            case alcanzado
                marcar-ciclo(s,t)
            case procesado
                if (es-carril(t)) p-marcar-carril(t)
                if (is-ciclo(t)) p-marcar-ciclo(s,t)
    end-for
    s.estatus=procesado
end-while
```

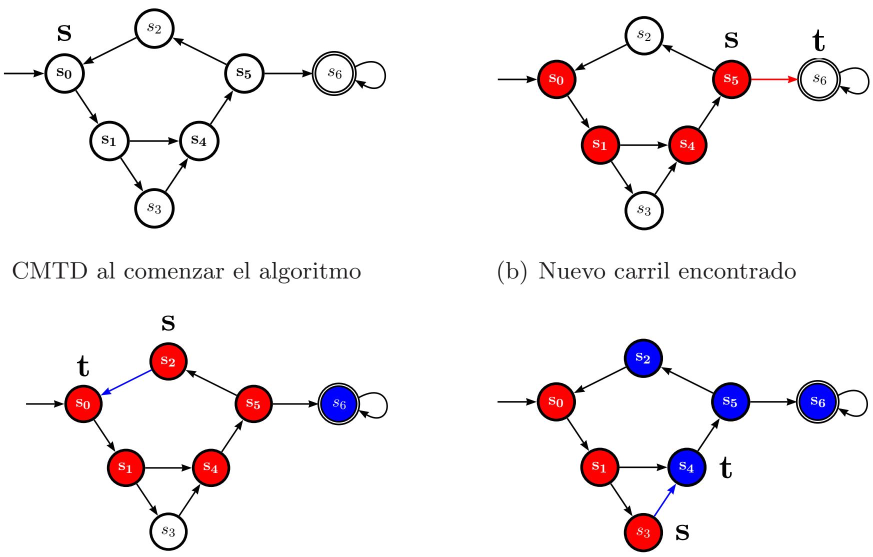
```
marcar-carril(state s)
depth=0
r=new-rail
s.rail←r
s.pred(r)=s.pred
s.depth(r)=depth
mark s as rail
depth++
while(s≠s0)
    t=s.pred
    t.rail← r
    t.pred(r)=t.pred
    t.suc(r)=s
    t.depth(r)=depth
    mark t as rail
    depth++
    s=s.pred
end-while
```

```
p-marcar-carril(state t)
for (all r∈t.rail)
    r′=new-rail
    assign(r′,lasts_{t.depth(r)}(r))
    depth=t.depth(r)+1
    s=t
    while(s≠s0)
        s′=s.pred
        s′.rail← r′
        s′.pred(r′)=s′.pred
        s′.suc(r′)=s
        s′.depth(r′)=depth
        mark s′ as rail
        depth++
        s=s.pred
    end-while
end-for
```

```
marcar-ciclo(state t, state s)
marcar s como ciclo
s.cycle-sun←t
while (s≠t)
    s′=s.pred
    mark s′ as cycle
    s′.cycle-sun← s
    s=s.pred
end-while
```

```
p-marcar-ciclo(state t, state s)
T=CompFuertConexa(t)
flag=0
u=s
while (u≠s0∧flag==0)
    if(u∈T){flag=1}
    else {u=u.pred}
end-while
if(flag==1)
    mark s as cycle
    s.cycle-sun←t
    while(s≠u)
        v=s.pred
        mark v as cycle
        v.cycle-sun←s
        s=s.pred
    end-while
end-if
```

Figura 4.1: DFS Extendido



(a) CMTD al comenzar el algoritmo (b) Nuevo carril encontrado

(c) Nuevo ciclo encontrado (d) Nuevo carril y ciclo encontrado

Figura 4.2: Situaciones en las que DFS-E encuentra carriles o ciclos. Los estados no alcanzados por el algoritmo se encuentran en blanco, los alcanzados en rojo y los completamente procesados en azul

Definición 4.3. *Dada una CMTD* $\Theta = (S, s_0, \mathcal{M})$*, definimos la* ***cadena reducida*** *de* Θ *como* $[\Theta] = ([S], s_0, [\mathcal{M}])$ *donde*

$$- \ [S] = \overbrace{(States(R^{\mathcal{G}}_{s_0}(\Theta)) - States(CFCM(\Theta)))}^{S_{com}} \bigcup \overbrace{\{s \in S \mid \exists r \in R^{\mathcal{G}}_{s_0}(\Theta), c \in CFCM(\Theta) \,.\, s = min_r(States(r) \cap c)\}}^{S_{abs}} \bigcup \{sink\}$$

$$- \ [\mathcal{M}](s,t) = \begin{cases} 1 & si\ s = t = sink \\ 0 & si\ s = sink \wedge t \neq sink \\ \mathcal{M}(s,t) & si\ s \in S_{com} \wedge t \neq sink \\ \sum_{u \in (S-[S])} \mathcal{M}(s,u) & si\ s \in S_{com} \wedge t = sink \\ \mathrm{P}_{s \rightsquigarrow t} & si\ s \in S_{abs} \wedge t \in \overline{succ}((s)^{\circlearrowleft}) \\ 0 & si\ s \in S_{abs} \wedge t \notin \overline{succ}((s)^{\circlearrowleft}) \wedge t \neq sink \\ 1 - \sum_{u \in (\overline{succ}((s)^{\circlearrowleft}))} \mathrm{P}_{s \rightsquigarrow u} & si\ s \in S_{abs} \wedge t = sink \end{cases}$$

Donde $(s)^{\circlearrowleft}$ *denota el conjunto de estados involucrados en la componente fuertemente conexa maximal de* s *y* $\overline{succ}((s)^{\circlearrowleft}) = \{u \in [S] \mid \exists\, t \in (s)^{\circlearrowleft} \,.\, \mathcal{M}(t,u) > 0\}$

A la cadena $[\Theta]$ *la llamamos cadena reducida de* Θ.

A continuación corroboramos que a partir de la información suministrada por DFS-E es posible obtener $[\Theta]$.

En primer lugar, analizamos como obtener el espacio de estados $[S]$ de la cadena reducida a partir de la original. El conjunto S_{com} es el conjunto de estados con flag igual a 1, es decir, $S_{com} = \{s \in S \mid s.flag = 1\}$. En tanto el conjunto S_{abs} se obtiene analizando los estados con flag igual a 3. Formalmente $S_{abs} = \{s \in S \mid s.flag = 3 \wedge \exists\, r\,.\, s.depth(r) = \text{máx}_{t.flag=3 \wedge s \rightsquigarrow t}\, t.depth(r)\}$. Con $s \rightsquigarrow t$ significamos que t es alcanzable desde s.

Por último, $[\mathcal{M}](s,t)$ puede ser fácilmente computada haciendo un análisis por casos sobre s y t. (Notar que $(s)^{\circlearrowleft}$ y $\overline{succ}((s)^{\circlearrowleft})$ son computables)

Ejemplo 4.3. *Veamos como reducir la CMTD Θ de la figura 4.3 (a). Para ello identificamos el conjunto $[S] = \{s_0, s_6, s_9, s_{11}\} \cup \{s_1, s_2\} \cup \{sink\}$. En tanto $[\mathcal{M}](s,t)$ se obtiene según la definición 4.3 haciendo un análisis por casos sobre s y t.*

Es posible derivar a partir de DFS-E los conjuntos $R_{s_0}^{\{s_6,s_9,s_{11}\}}(\Theta) = \{s_0s_1s_5s_8s_9, s_0s_1s_5s_{11}, s_0s_2s_6, s_0s_2s_{11}\}$ y CFCM(Θ)=$\{\{s_1, s_5, s_8, s_4\}, \{s_2\}\}$.

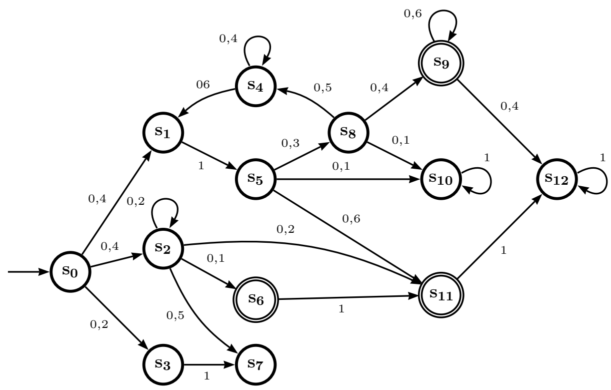
La cadena de Markov reducida resultante de Θ, es decir $[\Theta]$, es la de la figura 4.3(b). En rojo indicamos las probabilidad de transiciones que han cambiado, en tanto con líneas de puntos indicamos las transiciones nuevas. Hemos eliminado las transiciones desde y hacia estados que no pertenecen a $[S]$, dichos estados se encuentran sombreados.

Puentes: Un truco para evitar la pérdida de información

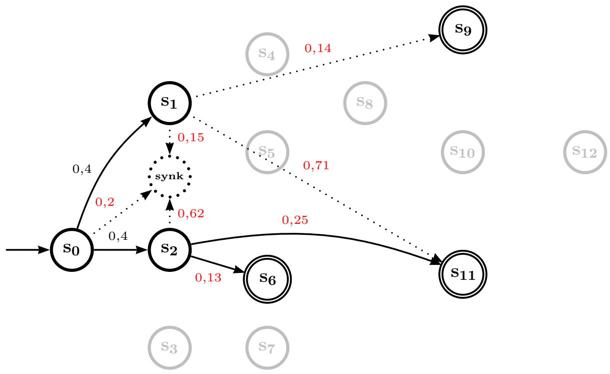
Como resultado de la reducción de cadenas se pierden datos con respecto a los carriles de la cadena original, puesto que los estados que integran las componentes fuertemente conexas maximales son descartados en la cadena reducida y, consecuentemente, en sus carriles. Dicha información nos resulta imprescindible para la obtención del carril con mayor probabilidad y, por lo tanto, debemos almacenarla. Para ello introducimos la noción de puente. Un puente es un camino de la cadena original que conecta los estados s de S_{abs} con sus sucesores en la cadena reducida (es decir, $\overline{succ}((s)^{\circlearrowleft})$). Para formalizar la noción de puentes introducimos la función bridge, la que, dado un estado s y un carril r indica cual es el puente en r desde s hasta su sucesor en la cadena reducida.

Definición 4.4. *Dada una CMTD Θ, $s \in S$ y $r \in R(\Theta)$ definimos*

$$bridge(s,r) = \sigma$$
$$\Updownarrow$$
$$s \in S_{abs} \wedge (s \cdot \sigma \text{ es subpalabra de } r) \wedge (\forall\, t \in States(r)\,.\, t \in (s)^{\circlearrowleft} \Rightarrow t \in (States(\sigma) \cup \{s\}))$$



(a) CMTD



(b) CMTD obtenida luego de la reducción

Figura 4.3: Preprocesamiento

Nótese que dicha función se encuentra bien definida puesto que no puede haber puentes distintos de un carril asociados a un mismo estado.

Ejemplo 4.4. *Los puentes de la CMTD de la figura 4.3. En este caso los elementos de $R(\Theta)$ son $r_1 = s_0 s_1 s_5 s_8 s_9$, $r_2 = s_0 s_1 s_5 s_{11}$, $r_3 = s_0 s_2 s_6$ y $r_4 = s_0 s_2 s_{11}$. Por consiguiente se obtiene $bridge(s_1, r_1) = s_5 s_8, bridge(s_1, r_2) = s_5$ en tanto $bridge(s, r) = \epsilon$ para los restantes s y r.*

Es importante destacar que, localizando el carril r con mayor probabilidad y sus respectivos puentes en la cadena $[\Theta]$ podemos derivar el carril r' con mayor probabilidad en Θ. En la siguiente sección nos enfocamos en la obtención de r.

4.3. Segunda Etapa: Obtención del Carril Máximo

El objetivo de esta etapa es localizar el carril máximo en la cadena reducida $[\Theta]$.

Puesto que $[\Theta]$ es acíclico, el torrente de cada carril r de $[\Theta]$ sólo contiene una única ejecución, es decir $\ll$r$\gg$ = {r}.

Luego, el cálculo de la probabilidad de los carriles r en una cadena acíclica resulta muy sencillo y eficiente, puesto que se reduce al cálculo de $\mu(\langle r\rangle)$.

La forma más sencilla de obtener el carril máximo de $[\Theta]$ es calcular $p_r = \mu(\langle r\rangle)$ para todo $r \in R([\Theta])$. Luego el carril r tal que $p_r = \text{máx}_{r\in R([\Theta])} \mathcal{P}_c(r)$ es el máximo de la cadena reducida. Esta técnica podría ser fácilmente implementada realizando una exploración DFS en búsqueda de todos los carriles y calculando sus probabilidades asociadas a medida que se los va encontrado.

En el presente trabajo se emplea una técnica mas eficiente que la descrita en el párrafo anterior. La misma consiste en guiar el recorrido del grafo de acuerdo a las probabilidades de los prefijos de los carriles. Para clarificar esta idea supongamos una cadena de Markov con tres carriles r^a, r^b y r^c. Inicialmente, se calculan las probabilidad $p_{r^a} = \mathcal{M}(r^a_0, r^a_1)$,$p_{r^b} = \mathcal{M}(r^b_0, r^b_1)$ y $p_{r^c} = \mathcal{M}(r^c_0, r^c_1)$. Luego supongamos que $p_{r^a} = \text{máx}(p_{r^a}, p_{r^b}, p_{r^c})$, en cuyo caso supondremos que r^a es el carril máximo y, consecuentemente, calculamos la probabilidad de la extensión de su prefijo, es decir, $p_{r^a} = \mathcal{M}(r^a_0, r^a_1) \cdot \mathcal{M}(r^a_1, r^a_2)$. El mismo procedimiento empleamos con p_{r^b} si $p_{r^b} = \text{máx}(p_{r^a}, p_{r^b}, p_{r^c})$ y con p_{r^c} si $p_{r^c} = \text{máx}(p_{r^a}, p_{r^b}, p_{r^c})$. Continuamos con este procedimiento hasta calcular la probabilidad de un carril completo r^x (para $x \in \{a, b, c\}$), en cuyo caso r^x sera el carril máximo de la cadena y la búsqueda termina.

Para implementar eficientemente este procedimiento utilizamos el algoritmo Z*. A continuación, indicamos su configuración para que se comporte de este modo.

4.3.1. Configuración de Z*

El corazón del algoritmo Z* es su función de evaluación $f(s) = F[\psi(s.padre), f(s.padre), h(s)]$ que indica cuán promisorios son los estados de la cadena.

En nuestro caso, la función f indica precisamente la probabilidad de los prefijos de los carriles de la cadena. De este modo, la función f asigna a cada estado s la probabilidad del único camino existente entre s_0 y s en el *árbol de recorrido* (véase capítulo 3, sección 3.2) asociado a la evolución de Z*. Así, cuando el algoritmo extrae un estado goal de la cola de prioridades habremos encontrado el carril máximo.

La función f que define dicho comportamiento es la siguiente:

$$f(s) = F[\psi(s.padre), f(s.padre), h(s)] = \begin{cases} 1 & \text{si } s = s_0 \\ f(s.padre) \cdot [\mathcal{M}](s.padre, s) \cdot h(s) & cc \end{cases}$$

donde $h(s) = 1$. En la sección 4.4.1 redefinimos a la función h, de modo tal que provea una sobre estimación mas precisa y así incrementar el rendimiento del algoritmo.

Ahora debemos verificar que la función f satisface las condiciones para ser la función de evaluación de $Z^\star$, es decir: f es *recursiva*, F satisface la propiedad de *preservación de orden* y h es una *sobre*estimación de la solución óptima.

La función f es recursiva por definición, en tanto h es una *sobre*estimación de la solución óptima ya que $\mu(\langle\sigma\rangle) \leq 1$ para cualquier solución σ. Finalmente, veamos que preserva orden:

$$
\begin{aligned}
& F[\psi(s_1), f(s_1), h_1(s')] \geq F[\psi(s_2), f(s_2), h_1(s')] \\
\Rightarrow\ & f(s_1) \cdot \mathcal{M}(s_1, s') \cdot h_1(s') \geq f(s_2) \cdot \mathcal{M}(s_2, s') \cdot h_1(s') \\
\Rightarrow\ & f(s_1) \cdot \mathcal{M}(s_1, s') \geq f(s_2) \cdot \mathcal{M}(s_2, s') \\
\Rightarrow\ & f(s_1) \cdot \mathcal{M}(s_1, s') \cdot h_2(s') \geq f(s_2) \cdot \mathcal{M}(s_2, s') \cdot h_2(s') \\
\Rightarrow\ & F[\psi(s_1), f(s_1), h_2(s')] \geq F[\psi(s_2), f(s_2), h_2(s')]
\end{aligned}
$$

esto se cumple para todos los estados s_1, s_2 y $s' \in succ(s_1) \cap succ(s_2)$ y todas las funciones heurísticas h_1 y h_2. Luego, F preserva orden y, consecuentemente, la función f puede ser utilizada como función de evaluación de $Z^\star$.

Ejemplo 4.5. *Ilustramos el comportamiento de $Z^\star$ para la cadena de Markov de la figura 4.3 (b) mediante snapshots con los valores de sus variables al comenzar el cuerpo del while -línea 4 del algoritmo (véase figura 3.3).*

Antes de ejecutarse el primer bucle del while el algoritmo sólo posee el estado s_0 en la cola con prioridad asociada 1. Además no posee estados marcados ni predecesores asignados. Este situación se ve reflejada en la primera línea de la tabla 4.3.

Luego, el primer bucle comienza a ejecutarse y se procesa el estado s_0, se agregan s_1 y s_2 a la cola y se actualizan sus predecesores y prioridades asociadas. Dichas modificaciones se observan en la segunda línea del cuadro 4.3.

El algoritmo continúa evolucionando hasta remover de la cola de prioridades un estado goal. En este caso, el algoritmo encuentra el carril máximo $s_0s_1s_{11}$ y su probabilidad asociada $f(s_{11}) = \mathcal{M}(s_0, s_1) \cdot \mathcal{M}(s_1, s_{11}) = \mu(\langle s_0s_1s_{11}\rangle) = 0{,}284$. Esta información también se puede inducir del cuadro 4.3.

Luego, con la información aportada el carril máximo $r = s_0s_1s_{11}$ de $[\Theta]$ y la función bridge, es posible identificar al carril $r' = s_0s_1s_5s_{11}$, el cual es máximo en Θ y posee igual probabilidad que r.

Por último, destacamos que el estado sink no es procesado por el algoritmo. El mismo se emplea sólo para que $[\Theta]$ satisfaga las propiedades características de cadenas de Markov. Este comportamiento se implementa de manera sencilla restringiendo los sucesores del estado que el algoritmo esta procesando -$succ(s)$- a aquellos estados sucesores de s que sean distintos de sink.

Cola de Prioridades	mark	padre
$[(1,s_0)]$	-	-
$[(0.4,s_2),(0.4,s_1)]$	s_0	$s_1.padre \rightarrow s_0$, $s_2.padre \rightarrow s_0$
$[(0.4,s_1),(0.1,s_{11}),(0{,}052,s_6)]$	s_0, s_2	$s_6.padre \rightarrow s_2$, $s_{11}.padre \rightarrow s_2$
$[(0.284,s_{11}),(0{,}056,s_9)(0{,}052,s_6)]$	s_0, s_2, s_1	$s_9.padre \rightarrow s_1$, $s_{11}.padre \rightarrow s_1$

Cuadro 4.3: Evolución del Algoritmo Z*

4.4. Optimización

El objetivo de esta sección es optimizar la técnica empleada a fin de hallar el carril máximo. En primer lugar, redefinimos la función h utilizada por Z* de modo tal que sea una sobre estimación más próxima de la probabilidad del camino óptimo asociado a cada estado. De ésta forma incrementamos en gran medida el rendimiento del algoritmo, ya que la elección de los estados más promisorios resulta mas precisa. En segundo lugar, planteamos una técnica alternativa a la reducción de todas las CFCM previo a la búsqueda del carril máximo (segunda fase del preprocesamiento). Esta técnica alternativa optimiza en gran medida el costo del preprocesamiento, en virtud de que la reducción de cada CFCM implica la resolución de al menos un sistema de ecuaciones lineales.

4.4.1. La Función h

Como ya mencionamos h es una función que asocia a cada estado s una sobre estimación de la máxima probabilidad entre los caminos desde s a algún estado que satisfaga la condición de alcanzabilidad. Formalmente

$$h_\Theta(s) \geq h^\star_\Theta(s) = \text{máx}\{\mu\left(\langle\sigma\rangle\right) \mid \sigma \in Path_s(\Theta) \wedge \text{last}(\sigma) \in \mathcal{G}\} \tag{4.3}$$

A los caminos $\sigma \in \{\sigma' \in Path_s(\Theta) \mid \text{last}(\sigma') \in \mathcal{G}\}$ tales que $\mu\left(\langle\sigma\rangle\right) = h^\star_\Theta(s)$ los denominamos *caminos óptimos* asociados al estado s. Además diremos que $h^\star_\Theta(s)$ es el valor *h-asociado* a s. Por último, como es usual, cuando la cadena Θ sea clara en el contexto podremos omitirla.

En la sección 4.3.1 se utilizó $h_\Theta(s) = 1$ como parte de la configuración de Z*. Ahora estudiamos como mejorar esta definición, es decir, encontrar una sobre estimación más cercana a la probabilidad asociada al camino óptimo de cada estado.

Una forma natural de calcular la probabilidad asociada al camino óptimo de cada estado es analizar cada carril r desde atrás hacia adelante. Es decir desde last(r) hasta first(r)) asignando a cada estado intermedio r_i el valor $\mu\left(\lessdot(\text{r} \uparrow \text{i})\gtrdot\right)$, en caso de que, $(r \uparrow i)$ sea un camino óptimo de s o, en caso contrario, no modificar el valor h-asociado a s.

Para ejecutar dicho procedimiento necesitamos identificar todos los carriles de la cadena, por lo que, en primer lugar, debemos ejecutar DFS-E sobre la misma. En la figura 4.4 presentamos el algoritmo compute-h* utilizado para computar dichos valores en una cadena de Markov.

```
compute-h*(CMTD Θ)
DFS-E(Θ)
for (all s ∈ S){s.h=0}
for (all r ∈ s0.carril)
      compute-h*(r)
end-for

compute-h*(carril r)
last(r).h=1
s=last(r)
while (s≠s0)
      u=s.padre
      u.h=máx (u.h, M(u, s)∗s.h)
      s=s.padre
end-while
```

Figura 4.4: Algoritmo para asociar el valor h-asociado a los estados de una cadena

Nótese que:

$$h^\star_{[\Theta]}(s) = \text{máx}\{\mu\,(\lessdot \text{r} \gtrdot) \mid r \in R^{\mathcal{G}}_s(\Theta)\}$$

Por lo que, desafortunadamente, si Θ posee algún ciclo existen estados $s \in [S]$ tales que $s.h < h^\star_{[\Theta]}(s)$, consecuentemente, no podemos utilizar esta técnica para definir una función h válida para el algoritmo Z* utilizado en la sección 4.3.1 (recordemos que el mismo se ejecuta sobre $[\Theta]$).

A continuación vemos un ejemplo en el que se refleja este problema y brindamos una solución.

Ejemplo 4.6. *En la figura 4.5 se presenta una cadena de Markov Θ con ciclos; las líneas de punto representan un camino. Observamos que, luego de ejecutar compute-h sobre Θ, $s_0.h = \mu\,(\langle s_0 s_1 s_4 s_6\rangle) < \mathrm{P}_{s_0 \rightsquigarrow s_6} * \mathrm{s}_6.\mathrm{h} = \mathrm{h}^\star_{[\Theta]}(\mathrm{s}_0)$. Esto se debe a que en $[\Theta]$ consideramos las probabilidades aportadas por la componente fuertemente conexa maximal de s_0.*

A fin de asegurar $s.h \geq h^\star_{[\Theta]}(s)$ asumimos que los ciclos asociados a estados en el carril no pierden masa probabilística. Por ejemplo, si consideramos s_1 en la CMTD de la figura 4.5, debemos asumir que $s_1 \xrightarrow{\mathcal{M}(s_1,s_2)} s_2 \xrightarrow{1} \cdots \xrightarrow{1} s_1$, $s_1 \xrightarrow{\mathcal{M}(s_1,s_3)} s_3 \xrightarrow{1} \cdots \xrightarrow{1} s_1$ y $s_4 \xrightarrow{\mathcal{M}(s_4,s_5)} s_5 \xrightarrow{1} \cdots \xrightarrow{1} s_4$. Siguiendo estas hipótesis los valores h-asociados a s_0, s_1 y s_4 se obtienen de acuerdo a las ecuaciones presentadas a la izquierda de Θ en la figura 4.5

Consecuentemente, es posible definir una cota superior de $h^{\star}_{[\Theta]}$ de la siguiente manera:

$$h_{[\Theta]}(s) = \text{máx}\{\mathrm{P}^{+}(\lessdot \mathrm{r} \gtrdot) \mid \mathrm{r} \in \mathrm{R}^{\mathcal{G}}_{\mathrm{s}}(\Theta)\}$$

donde

$$\mathrm{P}^{+}(\lessdot \mathrm{r} \gtrdot) = \prod_{\mathrm{i}=0}^{|\mathrm{r}|-2} \frac{\mathcal{M}(\mathrm{r_i}, \mathrm{r_{i+1}})}{1 - \sum_{\mathrm{t} \in ((\mathrm{r_i})^{\circlearrowleft} - \{\mathrm{r_{i+1}}\})} \mathcal{M}(\mathrm{r_i}, \mathrm{t})}$$

Por consiguiente, $h_{[\Theta]}(s) \geq h^{\star}_{[\Theta]}(s)$ para todo $s \in [S]$.

$$s_4.h = \frac{\mathcal{M}(s_4,s_6) * s_6.h}{1-\mathcal{M}(s_4,s_5)}$$

$$s_1.h = \frac{\mathcal{M}(s_1,s_4) * s_4.h}{1-(\mathcal{M}(s_1,s_2)+\mathcal{M}(s_1,s_3))}$$

$$s_0.h = \mathcal{M}(s_0, s_1) * s_1.h$$

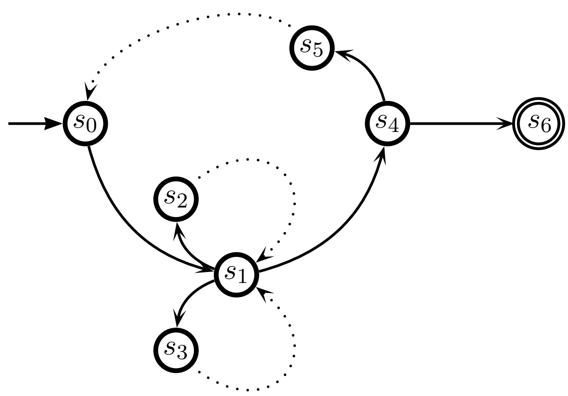


Figura 4.5: CMTD Θ con ciclos

Finalmente, hemos encontrado una técnica para computar una sobre estimación de los valores h-asociados a los estados de una cadena una vez reducida. La misma se presenta en el algoritmo de la figura 4.6.

Es posible implementar esta técnica sin necesidad de procesar todos los carriles. Para ello modificamos DFS-E de forma tal que, calcule los valores $h_{[\Theta]}$. En la figura 4.7 se ilustran las modificaciones que presentan las distintas rutinas del algoritmo original para calcular dichos valores.

4.4.2. Reducción de CFCM‘s por Demanda

En esta sección presentamos una nueva técnica de optimización. Anteriormente (ver sección 4.2.2) reducimos todas las CFCM de la cadena. Dicha reducción resulta extremadamente costosa, puesto que se deben resolver uno o más sistemas de ecuaciones lineales por cada CFCM que se reduce de la cadena. No obstante, analizando detenidamente la técnica usada para la obtención del carril máximo, concluimos que no es necesario reducir todas las CFCM de Θ. Consecuentemente, nos centramos en identificar al subconjunto de CFCM

```
compute-h(CMTD Θ)
DFS-E(Θ)
for (all s ∈ S){s.h=0}
for(all r ∈ s0.carril)
    compute-h(r)
end-for

compute-h(carril r)
last(r).h=1
s=last(r)
while (s≠s0)
    u=s.padre
    if (u es ciclo)
        u.h=máx(u.h, M(u,c)*s.h / (1 − Σ_{t∈(u.cycle_sun−{s})}))
    else
        u.h=máx(u.h, M(u,c)*s.h)
    s=s.padre
end-while
```

Figura 4.6: Algoritmo para asociar una sobre estimación del valor h-asociado a los estados de una cadena reducida

que es necesario reducir. Luego, presentamos una versión modificada de Z* que se ejecuta sobre cadenas no necesariamente acíclicas y reduce sólo las CFCM que sean necesarias reducir para la obtención del carril máximo. Dichas reducciones se realizan durante la ejecución del algoritmo. Además, calcula los puentes cuando ello resulte necesario. De este modo evitamos la segunda fase del preprocesamiento y así optimizamos en gran medida el rendimiento de la técnica de obtención de carriles máximos.

A continuación definimos la cadena de Markov sobre la cual se ejecuta la versión modificada de Z*.

Definición 4.5. *Dada una CMTD* $\Theta = (S, s_0\mathcal{M})$*, definimos la* ***cadena semi-reducida*** *de* Θ *como* $\llbracket\Theta\rrbracket = (\llbracket S\rrbracket, s_0, \llbracket\mathcal{M}\rrbracket)$ *donde*

- $\llbracket S\rrbracket = \overbrace{(States(R^{\mathcal{G}}_{s_0}(\Theta)) \cup States(CFCM(\Theta)))}^{S_{com}} \cup \{sink\}$

```
DFS-h-E(CMTD ⟨S, s0, M⟩)
for (all s ∈ S) {s.estatus=no-alcanzado;s.padre=NULL}
push(s0 , Stack)
while is-notEmpty(Stack)
    s=pop(Stack)
    s.estatus=alcanzado
    for (all t ∈ succ(s))
        switch (t.estatus)
            case no-alcanzado
                t.padre=s
                if (is-goal(t)) marcar-carril-h(t)
                else push(t , Stack)
            case alcanzado
                marcar-ciclo-h(s,t)
            case procesado
                if (es-carril(t)) p-marcar-carril-h(t)
                if (es-ciclo(t)) p-marcar-ciclo-h(s,t)
    end-for
    s.estatus=procesado
end-while
```

```
marcar-carril-h(state s)
depth=0
r=new-rail
s.rail←r
s.pred(r)=s.pred
s.depth(r)=depth
mark s as rail
depth++
s.h=1
while(s≠s0)
    t=s.pred
    t.rail← r
    t.pred(r)=t.pred
    t.suc(r)=s
    t.depth(r)=depth
    mark t as rail
    depth++
    if(t.h<h(t,s))
        t.h=h(t,s)
        t.next=s
    end-if
    s=s.pred
end-while
```

```
p-marcar-carril-h(state t)
for (all r∈t.rail)
    r′=new-rail
    assign(r′,lasts_t.depth(r)(r))
    depth=t.depth(r)+1
    s=t
    while(s≠ s0)
        s′=s.pred
        s′.rail← r′
        s′.pred(r′)=s′.pred
        s′.suc(r′)=s
        s′.depth(r′)=depth
        mark s′ as rail
        depth++
        if(t.h<h(t,s))
            t.h=h(t,s)
            t.next=s
        end-if
        s=s.pred
    end-while
end-for
```

```
marcar-ciclo-h-E(state t, state s)
flag=0
mark s as cycle
s.cycle-sun← t
while (s≠t)
    s′=s.pred
    mark s′ as cycle
    s′.cycle-sun← s
    if(s′ is rail ∧ flag==0)
        e= 1/M(s′,s)
        flag=1
    end-if
    if(flag==1) {s′.h=s′.h * e}
    s=s.pred
end-while
if(flag=1)
    while(t≠s0)
        t=t.pred
        t.h=t.h * e
    end-while
end-if
```

```
p-marcar-ciclo(state t, state s)
T=CompFuertConexa(t)
flag=0
u=s
while (u≠s0∧flag==0)
    if(u∈T){flag=1}
    else {u=u.pred}
end-while
if(flag==1)
    mark s as cycle
    s.cycle-sun←t
    while(s≠u)
        v=s.pred
        mark v as cycle
        v.cycle-sun←s
        if(v==u){e=M(u,s)}
        s=s.pred
    end-while
if (u is rail)
    while(u≠s0){ u.h=u.h * e}
end-if
```

```
h(state t, state s)
if(t is cycle)
    h=máx (t.h, M(t, s)∗s.h)
else
    h=máx ( t.h, M(t,s)∗s.h / (1− Σ_{t′∈(t.cycle−sun−{s})} M(t,t′)) )
end-if
return h
```

Figura 4.7: DFS h Extendido

$$- \ \llbracket\mathcal{M}\rrbracket(s,t) = \begin{cases} 1 & si\ s = t = sink \\ 0 & si\ s = sink \wedge t \in S_{com} \\ \mathcal{M}(s,t) & si\ s,t \in S_{com} \\ \sum_{u \in (S - \llbracket S \rrbracket)} \mathcal{M}(s,u) & si\ s \in S_{com} \wedge t = sink \end{cases}$$

A la cadena $\llbracket\Theta\rrbracket$ *la llamamos cadena semi-reducida de* Θ.

Puesto que en las cadenas semi reducidas no se eliminan las CFCM, la derivación de las mismas a partir de una CMTD resulta mucho mas económica -en términos de costos de procesamiento- que la derivación de las cadenas reducidas.

Ahora vemos cuales son las CFCM de Θ que precisamos absorber. Para ello, definimos por cada CMTD Θ y carril r de la misma, el conjunto de los prefijos de carriles de Θ tal que el torrente asociado a dichos prefijos poseen mayor probabilidad que la del torrente asociado a r. Formalmente:

Definición 4.6. *Dada una CMTD* Θ*, definimos para cada* $r \in R(\Theta)$ *los conjuntos*

$$\overset{\leadsto}{r} = \{\hat{r} \in S^\star \mid \exists\, r' \in R(\Theta)\,.\, \hat{r}\ es\ prefijo\ de\ r' \wedge \mu\,(\lessdot\hat{\mathrm{r}}\gtrdot) > \mu\,(\lessdot\mathrm{r}\gtrdot)\}$$

y

$$S_{abs}(\overset{\leadsto}{r}) = States(\overset{\leadsto}{r}) \cap S_{abs}$$

donde S_{abs} es el conjunto de estados introducidos en la definición 4.3.

En la figura 4.8 presentamos el algoritmo Z*-E el cual se comporta como Z* si el estado que se está procesando no posee CFCM y, en caso contrario, lo reduce. Se puede demostrar que si r es el carril máximo, Z*-E lo encuentra reduciendo solo los estados en $S_{abs}(\overset{\leadsto}{r})$. Luego, como $S_{abs}(\overset{\leadsto}{r}) \subseteq S_{abs}$, hemos conseguido obtener el carril máximo de una CMTD sin necesidad de reducir todas sus CFCM.

El algoritmo Z*-E utiliza la función $\mathrm{Path}_C(s,t)$ que devuelve un camino σ de la cadena tal que $\sigma_0 = s$, $\mathrm{tail}(\sigma) = \mathrm{t}$ y, $\sigma_i \in C$ para $i = 1, 2, \cdots, |\sigma| - 2$. Además la función $\overline{succ}$ se define como lo hicimos anteriormente (véase definición 4.3). La función f que utilizamos para configurar Z*-E se define de acuerdo al estado evaluado, si el mismo no posee CFCM entonces la función se comporta como lo hacía anteriormente (ecuación 4.4). Por el contrario, si el estado evaluado posee CFCM entonces f se comporta según la función f_E definida en la ecuación 4.5. En tanto $\mathrm{add}_E(\mathrm{s})$ agrega s a la cola con probabilidad $\mathrm{f}_E(s)$.

$$f(s) = \begin{cases} 1 & \text{si } s = s_0 \\ f(s.padre) \cdot \llbracket\mathcal{M}\rrbracket(s.padre, s) \cdot h(s) & cc \end{cases} \tag{4.4}$$

$$f_E(s) = \begin{cases} 1 & \text{si } s = s_0 \\ f(s.padre) \cdot \mathrm{P}_{\mathrm{s.padre} \rightsquigarrow \mathrm{s}} \cdot \mathrm{h(s)} & cc \end{cases} \tag{4.5}$$

Nótese que como f_E no es una función recursiva, no podemos considerar a Z⋆-E como miembro de la familia de algoritmos Z⋆.

```
Z⋆-E(CMTD ⟨S,s0,M⟩)
PQ⟵ empty Priority Queue
for (all s ∈ S) {s.f=0 ; s.padre=NULL}
add(s0,PQ)
while is-notEmpty(PQ)
    s=remove(PQ)
    if (is-goal(t)) finish-succefully(t)
    mark(s)
    if (s∉CFCM(⟨ S,s0,M⟩))
        for (all t∈succ(s))
            if (not-marked(t) ∧ t∉PQ)
                t.padre=s
                add(t,PQ)
            else
                if (f(t)>t.f)
                    t.padre=s
                    if(marked(t)) unmark(t)
                    add(t,PQ)
                end-if
            end-if
        end-for
    else
        for (all t∈succ‾(CFCM(s)))
            if (not-marked(t) ∧ t∉PQ)
                s.bridge=tail(Path_CFCM(s)(s,t))
                t.padre=s
                add_E(t,PQ)
            else
                if (f_E(t)>t.f)
                    s.bridge=tail(Path_CFCM(s)(s,t))
                    t.padre=s
                    if(marked(t)) unmark(t)
                    add_E(t,PQ)
                end-if
            end-if
        end-for
    end-if
end-while
```

Figura 4.8: Pseudo código de Z⋆-Extendido

4.5. Esquema de uso de la Herramienta

Por último presentamos un esquema en el cual resumimos el procedimiento a llevar a cabo en caso de la no satisfacción de una propiedad de alcanzabilidad.

1. La técnica resulta necesaria cuando una CMTD Θ no satisface una formula de alcanzabilidad cuantitativa acotada superiormente φ.

2. En dicho caso se ejecuta el algoritmo DFS-h-E (véase figura 4.7) sobre Θ asumiendo como estados goal a los elementos de $\mathcal{G}_\varphi$. De este modo obtenemos:

   - Una CMTD semi reducida $[\![\Theta]\!]$ (véase sección 4.4.2).
   - Todos los carriles de Θ que van desde su estado inicial hasta algún estado en $\mathcal{G}_\varphi$ (véase sección 4.2.1).
   - Todas las componenters fuertemente conexas maximales de Θ (véase sección 4.2.1).

3. Luego, ejecutamos Z*-E (véase 4.8) sobre la cadena semi reducida previamente derivada. El algoritmo reporta el carril r con mayor probabilidad de $[\![\Theta]\!]$, a partir del cual derivamos el carril máximo r' de Θ

4. Luego, analizamos el sistema original en busca del error guiándonos con la información aportada por r', en este caso hay dos posibilidades:

   - Si encontramos el error entonces el proceso termina.
   - De lo contrario, continuamos la ejecución de Z*-E en busca del segundo carril con mayor probabilidad (ir al paso 3).

Este proceso continúa hasta localizar el error en el sistema.

En la figura 4.9 se ilustra el esquema previamente referido.

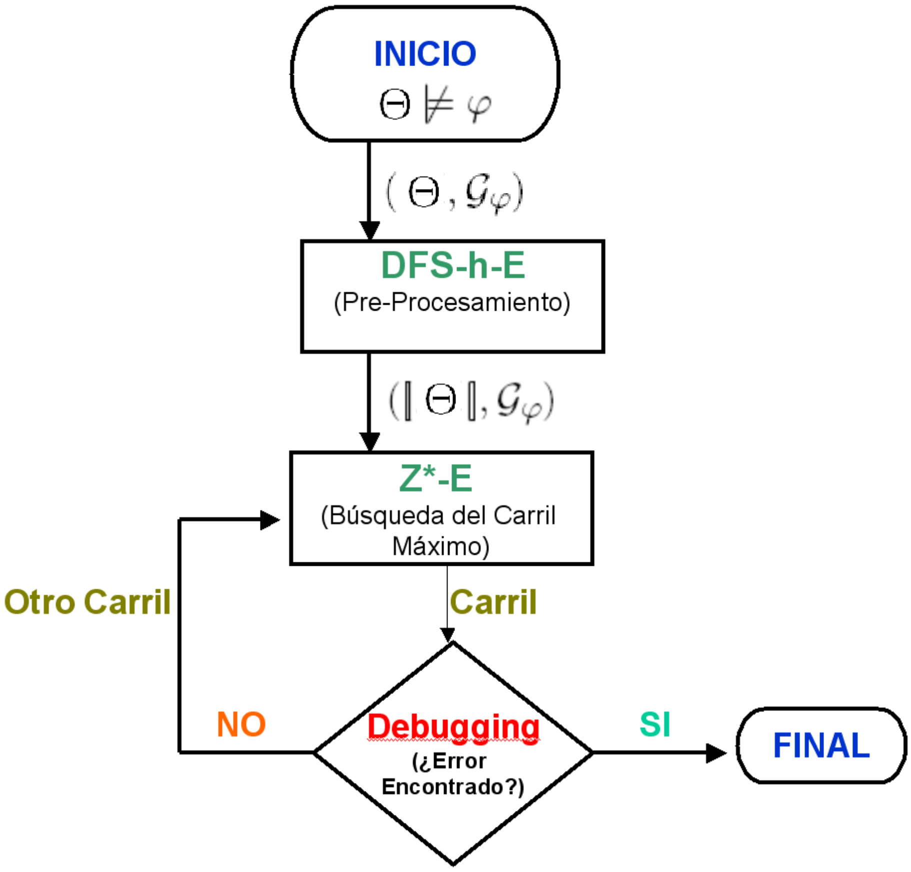


Figura 4.9: Esquema de uso de la Herramienta

Capítulo 5

Procesos de Decision de Markov

En este capítulo presentamos los *Procesos de decisión de Markov* (PDM), estructuras introducidas por Ronald Howard en 1960 y de gran utilidad para modelar sistemas en los cuales coexisten comportamientos probabilistas y no deterministas. Al igual que lo hecho con cadenas de Markov, definimos la σ-algebra de ejecuciones para PDM, además de la noción de schedulers probabilistas y la medida de probabilidad que ésta define.

5.1. Procesos de Decisión de Markov

Un PDM funciona como una cadena de Markov con la diferencia que, en lugar de asociar una distribución de probabilidades a cada estado, le asocia un conjunto -no vacío- de distribuciones de probabilidades. Formalmente

Definición 5.1. *Un* ***Proceso de decisión de Markov*** *es una terna* $\Pi = (S, s_0, \tau)$, *donde:*

1. *S es el espacio de estados finito del sistema;*
2. *$s_0 \in S$ es el estado inicial;*
3. *τ es una función que asocia a cada $s \in S$ un conjunto $\mathcal{D}_s \subseteq \mathcal{P}(Distr(S))$*

Distr(Ω) es el conjunto de todas las distribuciones de probabilidad discretas sobre el espacio Ω.

El sucesor de un estado $s \in S$ es elegido de acuerdo a un proceso de dos fases. Primero, una distribución de probabilidad asociada a s, $d \in \tau(s)$, es seleccionada no deterministicamente entre los elementos de $\tau(s)$; segundo, un estado sucesor $t \in S$ es seleccionado de acuerdo a la distribución de probabilidades d.

Este modelo, permite una codificación simple de los sistemas de composición paralela. Para ver como modelamos el paralelismo a través de un PDM, consideramos como ejemplo la composición paralela de m cadenas de Markov $\Theta_1, \ldots, \Theta_m$. Es posible construir un PDM Π representando $\Theta_1 \| \Theta_2 \| \ldots \| \Theta_m$. Para ello asociamos a cada estado $s \in S$ el conjunto de distribuciones de probabilidad $\tau(s) = \{d_1, \ldots d_m\}$, donde la distribución d_i corresponde a un movimiento hecho por la cadena Θ_i, para $1 \leq i \leq m$. De esta forma, la información probabilista en el comportamiento de cada cadena es preservada en Π y, la elección de la cadena de Markov que realiza la transición es no determinista.

Definición 5.2 (Path$_s(\Pi)$). *Sea $\Pi = (S, s_0, \tau)$ un PDM y $s_0 \in S$, un s_0-path o camino de Π sera una secuencia **finita** de estados de S, $\sigma = s_0 s_1 \ldots s_n$, donde para cada $0 \leq i < n$ existe $d_i \in \tau(s_i)$ tal que $d_i(s_{i+1}) > 0$. σ_i denota el estado en la i-esima posición de σ, $(\sigma \uparrow i)$ el prefijo finito de σ con longitud n, $|\sigma|$ la longitud de σ, $\mathrm{last}(\sigma)$ el último estado de σ y, $Paths_s(\Pi)$ el conjunto de s-path's de Π. Cuando $s = s_0$ podremos omitirlo, es decir $Path_{s_0}(\Pi)$=$Path(\Pi)$*

De manera similar definimos las ejecuciones sobre un PDM

Definición 5.3 (Exec$_s(\Pi)$). *Sea $\Pi = (S, s_0, \tau)$ un PDM y $s_0 \in S$, una ejecución o s_0-exec de Π sera una secuencia **infinita** de estados de S, $\omega = s_0 s_1 s_2 \ldots$, donde para cada $i \in \mathbb{N}_0$ existe $d_i \in \tau(s_i)$ tal que $d_i(s_{i+1}) > 0$. ω_i denota el estado en la i-esima posición de ω, $(\omega \uparrow n)$ el prefijo finito de ω con longitud n y, $Execs_s(\Pi)$ el conjunto de s-exec's de Π. Al igual que para el caso de caminos $Exec_{s_0}(\Pi)$=$Exec(\Pi)$.*

Ejemplo 5.1. *La figura 5.1 ilustra un PDM $\Pi = (S, s_0, \tau)$ donde $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, $\tau(s_0) = \{d_1, d_2, d_3\}$ y $\tau(s_i) = \{d_{i+3}\}$ para $i = 1, \ldots, 6$.*

Además $s_0 s_0 s_1 s_4$ y $s_0 s_3 s_5 s_5 s_5 \cdots$ son ejemplos de un camino y una ejecución sobre Π, respectivamente.

5.2. Probabilidades sobre un PDM

A continuación analizamos -al igual que lo hicimos para el caso de Cadenas de Markov- como asociar probabilidades a ejecuciones de un PDM. Inicialmente presentamos la σ-álgebra para luego definir una medida de probabilidad sobre la misma.

5.2.1. σ-Algebra de Ejecuciones sobre PDM

Dado que el conjunto de ejecuciones Ω de un PDM es un conjunto de secuencias de estados al igual que para el caso de CMTD, la σ-álgebra $\mathcal{F}_\Pi$ de ejecuciones sobre un PDM es exactamente la misma definida en el capítulo 2 (véase sección 2.11). Es decir $\mathcal{F}_\Pi = \sigma(\mathcal{B}_\Pi)$.

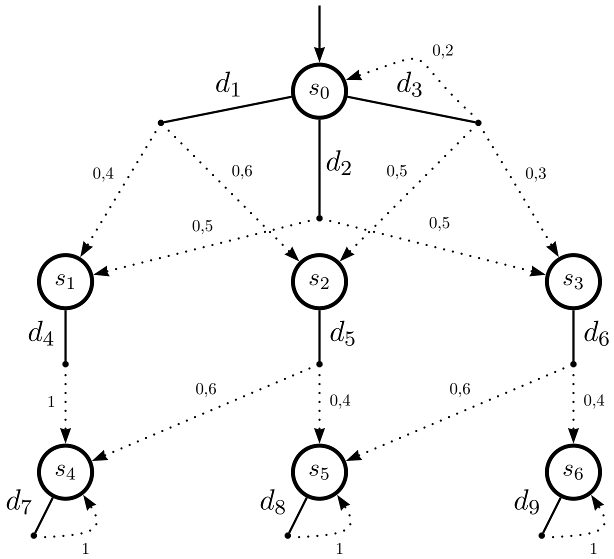


Figura 5.1: Proceso de Decisión de Markov

5.2.2. Medida de Probabilidad para el caso No Determinista

Dada la presencia de no determinismo en un PDM Π no es posible definir una medida de probabilidad en la σ-algebra de ejecuciones $\mathcal{F}_\Pi$. Sin embargo, dado un conjunto de ejecuciones $\Delta \in \mathcal{F}_\Pi$, podemos definir su *probabilidad máxima* $\mu^+(\Delta)$ y su *probabilidad mínima* $\mu^-(\Delta)$. Intuitivamente, $\mu^+(\Delta)$(respectivamente $\mu^-(\Delta)$) representa la probabilidad de que el sistema siga alguna ejecución de Δ suponiendo elecciones no deterministas tan favorables (respectivamente desfavorables) como sea posible.

Para formalizar la idea de elecciones favorables y desfavorables introducimos el concepto de *Schedulers Probabilistas* (similares a los schedulers en [8, 5]). Básicamente, un scheduler probabilista se encarga de "romper el no determinismo" asociando a cada camino σ de un PDM una distribución de probabilidades. De esta manera el PDM queda completamente determinado y, consecuentemente, se puede definir una medida de probabilidad en la σ-Algebra de ejecuciones.

Schedulers Probabilistas

Seguidamente formalizamos el concepto de Schedulers Probabilistas.

Definición 5.4. *Dado un PDM* $\Pi = (S, s_0, \tau)$*, un* ***scheduler probabilista*** *de* Π *es una función* $\eta : Path(\Pi) \to Distr(Distr(S))$ *que satisface*

$$(\eta(\sigma) = p) \quad \Rightarrow \left(\sum_{d \in \tau(\mathrm{last}(\sigma))} p(d) = 1 \right)$$

Con frecuencia escribimos η_σ en lugar de $\eta(\sigma)$ para facilitar la notación. A lo largo del presente capítulo usamos el término scheduler para referirnos a un scheduler probabilista.

Ejemplo 5.2. *La siguiente función es un scheduler del PDM del ejemplo 5.1*

$$\eta(\sigma) = \begin{cases} \{(d_1, \frac{1}{4}), (d_2, \frac{1}{2}), (d_3, \frac{1}{4})\} & si\ last(\sigma) = s_0 \\ \{d_{i+3}\} & si\ last(\sigma) = s_i\ \ con\ i \in \{1, 2, \ldots, 6\} \end{cases}$$

Cabe destacar que en este ejemplo un scheduler solo debe caracterizar las distribuciones asociadas a s_0, puesto que el resto de los estados no poseen no determinismo, es decir, sólo tienen una distribución de probabilidades asociada.

Cuando un sistema se comporta de acuerdo a una scheduler probabilista η en la evolución desde σ_0, y alcanza last(σ) siguiendo el camino σ, elegirá la distribución de probabilidades d asociada a last(σ) con probabilidad $\eta_\sigma(d)$. La probabilidad $\mathbb{P}_\eta(\sigma, t)$ asociada a un scheduler probabilista η de que una vez que el PDM ha seguido la secuencia de estados σ, ocurra una transición directa a t es igual a $\sum_{d \in \tau last(\sigma)} \eta_\sigma(d) \cdot d(t)$.

Luego podemos asociar a cada cilindro básico de un PDM una medida de probabilidad de la siguiente forma:

Definición 5.5 (Probabilidad asociada a Cilindros Básicos de un PDM). *Dada un PDM* Π*,* $\mathcal{B}_\Pi$ *y un scheduler* η *de* Π*. Definimos la medida de probabilidad* $\mu_\eta : \mathcal{F}_\Pi \to [0, 1]$ *como la única medida de probabilidad (por teorema 2.1) tal que*

$$\mu_\eta(\langle\sigma\rangle) = \prod_{i=0}^{|\sigma|-2} \mathbb{P}_\eta((\sigma \uparrow i), \sigma_{i+1}) \tag{5.1}$$

para todo $\langle\sigma\rangle \in \mathcal{B}_\Pi$*.*

Luego, dado un scheduler η de Π, $(Exec(\Pi), \mathcal{F}_\Pi, \mu_\eta)$ es un espacio de probabilidades sobre $Exec(\Pi)$

Ejemplo 5.3. *Para clarificar la ecuación (5.1) calculemos* $\mu_\eta(s_0 s_0 s_1 s_4)$ *para el PDM del ejemplo 5.1 y donde* η *es el scheduler definido en el ejemplo 5.2*

$$\begin{aligned} \mu_\eta(s_0 s_0 s_1 s_4) &= \textstyle\prod_{i=0}^{2} \mathbb{P}_\eta((\sigma \uparrow i), \sigma_{i+1}) \\ &= \mathbb{P}_\eta(s_0, s_0) \cdot \mathbb{P}_\eta(s_0 s_0, s_1) \cdot \mathbb{P}_\eta(s_0 s_0 s_1, s4) \\ &= (0{,}2 \cdot \tfrac{1}{4}) \cdot (0{,}4 \cdot \tfrac{1}{4} + 0{,}5 \cdot \tfrac{1}{2} + 0 \cdot \tfrac{1}{4}) \cdot (1 \cdot 1) \\ &= 0{,}05 \cdot 0{,}35 \cdot 1 \\ &= 0{,}0175 \end{aligned}$$

Finalmente, es posible definir las probabilidades máximas y mínimas como sigue.

Probabilidades Máximas y Mínimas

Las probabilidades $\mu^+(\Delta)$ y $\mu^-(\Delta)$ representan la probabilidad con la cual el sistema sigue una ejecución $ss_1s_2\ldots\in\Delta$ cuando las elecciones no deterministas son tan favorables o tan desfavorables como sea posible, respectivamente.

Formalmente definimos a $\mu^+(\Delta)$ y $\mu^-(\Delta)$ de la siguiente manera

Definición 5.6. *Dado un PDM $\Pi = (S, s_0, \tau)$, $s \in S$ y un conjunto de ejecuciones $\Delta \in \sigma(\mathcal{F}_\Pi)$, la* ***probabilidad máxima*** *$\mu^+(\Delta)$ y la* ***probabilidad mínima*** *$\mu^-(\Delta)$ están definidas como:*

$$\mu^+(\Delta) = \max_\eta \mu_\eta(\Delta) \qquad \mu^-(\Delta) = \min_\eta \mu_\eta(\Delta) \tag{5.2}$$

Ejemplo 5.4. *Ahora calculemos la máxima y mínima probabilidad de que el PDM del ejemplo 6.1 siga el camino $s_0s_0s_1s_4$. Para ello debemos calcular $\mu^+(\langle\, s_0s_0s_1s_4\rangle)$ y $\mu^-(\langle\, s_0s_0s_1s_4\rangle)$. Este último cálculo es más sencillo, puesto que seleccionando un scheduler η tal que $\eta_{s_0}(d_3) = 0$ obtenemos $\mu^-(\langle\, s_0s_0s_1s_4\rangle) = 0$. En tanto $\mu^+(\langle\, s_0s_0s_1s_4\rangle) = 0{,}1$ y se obtiene con algún scheduler η de Π que satisfaga $\eta_{s_0}(d_3) = 1$ y $\eta_{s_0s_0}(d_2) = 1$.*

Luca de Alfaro demuestra en [13] que para el cálculo de alcanzabilidad (es decir cuando $\Delta = \{\omega \in Exec(\Pi) \mid \exists i \geq 0 \,.\, \omega_i \in \mathcal{G}\}$ en (5.2)) existen η y η' deterministas tales que $\mu_\eta(\Delta) = \mu^+(\Delta)$ y $\mu_{\eta'}(\Delta) = \mu^-(\Delta)$, donde un scheduler η es *determinista* si, para todo σ y σ' tal que $\mathrm{last}(\sigma) = \mathrm{last}(\sigma')$, se tiene $\eta(\sigma) = \eta(\sigma')$, es decir que la elección dada por η sólo depende del último estado. En el capítulo siguiente formalizamos adecuadamente los schedulers deterministas.

Capítulo 6

Extracción de la Cadena de Markov

En este capítulo extendemos la técnica utilizada para obtener el carril máximo en CMTD a Procesos de Decisión de Markov. Para ello derivamos a partir de un PDM una CMTD de forma tal que la probabilidad máxima de alcanzar un estado goal en el PDM sea igual a la probabilidad de alcanzar un estado goal en la CMTD derivada. De esta forma, es posible utilizar los resultados obtenidos para identificar el carril máximo (véase Capítulo 4) sobre la cadena derivada y así obtener testigos de contraejemplo en el PDM.

Formalmente, debemos derivar -a partir de un PDM Π- una CMTD Θ^+ tal que:

$$\mu^+(\{\omega \in Exec(\Pi) \mid \exists\, i \geq 0 \,.\, \omega_i \in \mathcal{G}\}) = \mu\left(\{\omega \in Exec(\Theta^+) \mid \exists\, i \geq 0 \,.\, \omega_i \in \mathcal{G}\}\right) \tag{6.1}$$

6.1. ¿Qué cadena extraer?

En esta sección definimos la cadena de Markov Θ^+ de la ecuación (6.1). Para ello comenzamos presentando la noción de schedulers deterministas, los cuales seleccionan siempre la misma distribución para cada estado. Formalmente:

Definición 6.1. *Dado un PDM* $\Pi = (S, s_0, \tau)$, *y un scheduler probabilista* η *de* Π, *diremos que* η *es un* ***scheduler determinista*** *de* Π, *si para todo* $\sigma \in Path(\Pi)$ *y* $s \in S$

- $\eta(\sigma) = \eta(\text{last}(\sigma))$ $\{$*propiedad memoryless*$\}$
- *existe* $p \in \tau(s)$ *tal que* $\eta_s(p) = 1$

Para facilitar la notación escribiremos η_σ *en lugar de* $\eta(\sigma)$.

El conjunto de ejecuciones válidas dado un scheduler determinista se definen de la siguiente manera:

Definición 6.2. *Dado un PDM* $\Pi = (S, s_0, \tau)$ *y* η *un scheduler determinista de* Π*, definimos las* ***η-ejecuciones*** *de* Π *como*

$$Exec_\eta(\Pi) = \{\omega \in Exec(\Pi) \mid \forall\, i \geq 0\,.\, \mathbb{P}_\eta(\omega \uparrow i, \omega_{i+1}) > 0\}$$

A partir de un scheduler determinista es posible definir una Cadena de Markov de la siguiente manera:

Definición 6.3. *Sea* $\Pi = (S, s_0, \tau)$ *un PDM y* η *un scheduler determinista de* Π*, definimos la cadena de Markov* $\Pi_\eta = (S_\eta, s_0, \mathcal{M}_\eta)$ *donde*

- $S_\eta = States(Exec(\Pi))$
- $\mathcal{M}_\eta(s, t) = \eta_s(t)$ *para todo* $s, t \in S_\eta$.

En este caso decimos que Π_η *es una* ***cadena de Markov η-asociada*** *a* Π.

Nótese que, η_s representa la distribución asociada al camino con un único estado s.

Ahora presentamos algunas definiciones que nos serán de gran utilidad para la extracción de la cadena de Markov. En primer lugar, definimos el conjunto de estados que alcanzan algún estado en $S' \subseteq S$ de la siguiente manera:

$$A(S') = \{s \in S \mid \exists\, \omega \in Exec(\Pi)\,.\,\omega_i \in S' \text{ para algun } i \geq 0\}$$

y al conjunto de estados que alcanzan algún estado de S' respetando un scheduler η como

$$A_\eta(S') = \{s \in S \mid \exists\, \omega \in Exec_\eta(\Pi)\,.\,\omega_i \in S' \text{ para algun } i \geq 0\}$$

y la restricción del conjunto S' a η como

$$S'_\eta = S' \cap States(Exec_\eta(\Pi))$$

Además abreviamos $\mu_s^+(S') \triangleq \mu^+(\{\omega \in Exec_s(\Pi) \mid \exists\, i \geq 0\,.\,\omega_i \in S'\})$

Lema 6.1. *[3] Dado un PDM* $\Pi = (S, s_0, \tau)$ *y* $S' \subseteq S$

$$\mu_s^+(S') = \max_{p \in \tau(s)} \left[\sum_{t \in A(S')} p(t)\mu_t^+(S') + \sum_{t \in S'} p(t) \right]$$

Finalmente estamos en condiciones de enunciar el siguiente teorema:

Teorema 6.1 (Schedulers Máximos). *Dado un PDM* $\Pi=(S,s_0,\tau)$ *y* $\mathcal{G}$ *el conjunto de estados goal, existe un scheduler determinista* η *asociado a* Π *tal que:*

i) $\mu_s^+(\mathcal{G})=\sum_{t\in A(\mathcal{G})}\eta_s(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}}\eta_s(t)$

ii) $\mu_s^+(\mathcal{G})=\mu\left(\{\omega\in Exec_s(\Pi_\eta)\mid\exists\, i\geq 0\,.\,\omega_i\in\mathcal{G}\}\right)$

para todo $s\in A_\eta(\mathcal{G})$*. En tal caso diremos que* η *es un* ***scheduler máximo*** *de* Π*.*

Demostración. Sea η un scheduler determinista de Π tal que para todo $s\in S$

$$(\eta_s=p_s)\Leftrightarrow\left(\left[\sum_{t\in A(\mathcal{G})}p_s(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}}p_s(t)\right]=\max_{p\in\tau(s)}\left[\sum_{t\in A(\mathcal{G})}p(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}}p(t)\right]\right) \tag{6.2}$$

i) Sea $s\in A_\eta(\mathcal{G})$, entonces

$$\begin{array}{rcl}
\mu_s^+(\mathcal{G}) & = & \{\text{Por lema 6.1}\}\ \max_{p'\in\tau(s)}\left[\sum_{t\in A(\mathcal{G})}p'(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}}p'(t)\right] \\
& & \{\text{Por definición } p_s \text{ es la distribución máxima}\} \\
& = & \sum_{t\in A(\mathcal{G})}p_s(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}}p_s(t) \\
& & \{\text{Por definición de } \eta\} \\
& = & \sum_{t\in A(\mathcal{G})}\eta_s(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}}\eta_s(t)
\end{array}$$

ii)

$$\begin{array}{rcl}
\mu_s^+(\mathcal{G}) & = & \{\text{Por i)}\}\ \sum_{t\in A(\mathcal{G})}\eta_s(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}}\eta_s(t) \\
& & \{A_\eta(\mathcal{G})\subseteq A(\mathcal{G})\wedge\forall\, t\in(A(\mathcal{G})-A_\eta(\mathcal{G}))\,.\,\mu_t^+(\mathcal{G})=0; \\
& & \ \mathcal{G}\subseteq\mathcal{G}_\eta\wedge\forall\, t\in(\mathcal{G}-\mathcal{G}_\eta)\,.\,\eta_s(t)=0\} \\
& = & \sum_{t\in A_\eta(\mathcal{G})}\eta_s(t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}_\eta}\eta_s(t) \\
& & \{\text{Definición de } \mathcal{M}_\eta\} \\
& = & \sum_{t\in A_\eta(\mathcal{G})}\mathcal{M}_\eta(s,t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}_\eta}\mathcal{M}_\eta(s,t) \\
& & \{A_\eta(\mathcal{G})\subseteq S_t\wedge\forall\, t\in(S_t-A_\eta(\mathcal{G}))\,.\,\mu_t^+(\mathcal{G})=0; \\
& & \ \text{donde } S_t \text{ es el conjunto de estados transitorios de } \Pi_\eta\} \\
& = & \sum_{t\in S_t}\mathcal{M}_\eta(s,t)\mu_t^+(\mathcal{G})+\sum_{t\in\mathcal{G}_\eta}\mathcal{M}_\eta(s,t) \\
& = & \{\text{Los } \mu_t^+(\mathcal{G}) \text{ son solución del sistema de ecuaciones lineales} \\
& & \ \text{asociado a } \Pi_\eta \text{ (véase lema 2.1)}\} \\
& = & \mathrm{P}_{s\rightsquigarrow\mathcal{G}} \\
& & \{\text{Definición 2.12}\} \\
& = & \mu\left(\{\omega\in Exec_s(\Pi_\eta)\mid\exists\, i\geq 0\,.\,\omega_i\in\mathcal{G}\}\right)
\end{array}$$

□

Luego, dado que $s_0 \in A_\eta(\mathcal{G})$, el teorema 6.1 asegura que si η es un scheduler máximo de Π, entonces Π_η satisface la ecuación 6.1.

6.2. ¿Cómo extraer la cadena?

Una vez convencidos de que si η es un scheduler máximo de Π entonces Π_η satisface 6.1 nos enfocamos en el cálculo de η. Para ello enunciamos el siguiente lema:

Lema 6.2. *[13] Para determinar $\mu_s^+(G)$ para todo $s \in A(G)$ es suficiente encontrar el conjunto de valores $\{x_s : s \in A(G)\}$ que minimiza $\sum_{s \in A(G)} x_s$ sujeto a las siguientes restricciones*

$$x_s \geq \sum_{t \in A(G)} p_s^i(t) x_t + \sum_{t \in G} p_s^i(t) \tag{6.3}$$

para todo $s \in A(G)$ y $p_s^i \in \tau(s)$. Además, $\mu_s^+(G) = x_s$ para todo $s \in A(G)$.

Estos valores están bien definidos, pues el problema admite una única solución.

Teorema 6.2. *Dado un PDM Π y el conjunto de estados $\mathcal{G}$, es posible calcular un scheduler máximo de Π.*

Demostración. A través de un método de resolución de problemas lineales, tal como Simplex, es posible resolver el problema de minimización lineal planteado por la ecuación 6.3 y, a partir de allí, definir un scheduler η que satisface:

$$\eta(s) = p \in \tau(s) \text{ tal que } x_s = \sum_{t \in A(\mathcal{G})} p(t) x_t + \sum_{t \in \mathcal{G}} p(t)$$

Por lema 6.2 tenemos que

$$\eta(s) = p \in \tau(s) \text{ tal que } \mu_s^+(\mathcal{G}) = \sum_{t \in A(\mathcal{G})} p(t) \mu_t^+(\mathcal{G}) + \sum_{t \in \mathcal{G}} p(t)$$

Luego por lema 6.1

$$\eta(s) = p \in \tau(s) \text{ tal que } \mu_s^+(\mathcal{G}) = \sum_{t \in A(\mathcal{G})} p(t) \mu_t^+(\mathcal{G}) + \sum_{t \in \mathcal{G}} p(t) = \max_{p \in \tau(s)} \left[\sum_{t \in A(S')} p(t) \mu_t^+(S') + \sum_{t \in S'} p(t) \right]$$

Pero entonces η satisface la ecuación 6.2, luego η es un scheduler máximo de Π. $\square$

Capítulo 7

Conclusion y trabajos futuros

7.1. Lo Aportado

Los principales aportes del presente trabajo son:

- **CARRILES:** El aporte mas destacado del presente trabajo es la presentación del concepto de carriles. Gracias a ello, estamos en condiciones de proveer información relevante para debuguear sistemas donde las componentes probabilistas y no deterministas se combinan. Dicho aporte se complementa con la presentación de una técnica para la derivación de los carriles con mayor probabilidad de un PDM.

- **CADENAS DE MARKOV A PARTIR DE PDM:** Se definió una técnica para derivar a partir de un PDM una cadena de Markov que contiene un contraejemplo de una propiedad de alcanzabilidad acotada superiormente violada en el PDM.

- **HERRAMIENTA PROTOTIPICA:** Se implementó una herramienta prototípica, la cual extrae carriles de una Cadena de Markov de tiempo discreto. La misma no cuenta con la optimización planteada en la sección 4.4.1. Su principal objetivo ha sido probar los resultados obtenidos. De ninguna manera pretende ser una herramienta de uso práctico y real.

7.2. Trabajos Futuros

Es nuestra intención continuar con el desarrollo del presente trabajo. Los principales puntos a desarrollar son:

- Implementar una herramienta de visualización de carriles.

- Extender la herramienta prototípica a una de uso real.

- Mejorar las técnicas de uso de memoria, por ejemplo, a través del uso de Binary Decision Diagrams (BDD‘s).
- Estudiar el uso de algoritmos alternativos a $Z^{\star}$ para la obtención del carril con mayor probabilidad.
- Estudiar el aporte de carriles como testigos de contraejemplo en model checkers cuantitativos sobre propiedades de alcanzabilidad cuantitativa acotadas inferiormente, es decir, aquéllas de la forma $\varphi = \mathcal{P}_{\geq a}(\mathbf{F}\ \psi)$

Bibliografía

[1] H. Aljazzar, H. Hermanns, and Stefan Leue. Counterexamples for Timed Probabilistic Reachability FORMATS 2005, Springer.

[2] C. Baier, M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. Distr. Computing, 11(3), 1996

[3] A. Bianco, L. de Alfaro. Model checking of probabilistic and nondeterministic systems. FST&TCS'95, LNCS 1026. Springer

[4] E. Clarke, O. Grumberg, D. Peled. Model Checking. MIT Press, 1999.

[5] C. Courcoubetis, M. Yannakakis. Verifying temporal properties of finite-state probabilistic processes. 29th FOCS. IEEE, 1988

[6] L. de Alfaro. Temporal logics for the specification of performance and reliability. STACS'97, LNCS 1200. Springer

[7] R. Gupta, S. Smolka, Bhaskar. On randomization in sequential and distributed algorithms. ACM Comp. Surveys, 26(1), 1994

[8] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. 26th FOCS. IEEE, 1985

[9] J-P. Katoen. Concepts, Algorithms, and Tools for Model Checking. Universität Erlangen-Nürnberg, 1999.

[10] www.probability.net

[11] C. Cassandras Discrete-Event Systems: Modeling and Performance Analysis. IL: Irwin, 1993.

[12] J. Pearl. Heuristics - Intelligent Search Strategies for Computer Problem Solving. Addision-Wesley, 1986.

[13] L. de Alfaro. Formal Verification of Probabilistic Systems. PhD Thesis. Stanford University, 1997.

[14] Maximiliano Combina, Matías Lee. Model Checking Cuantitativo con un enfoque de la Teoría de Autómatas. Tesis de Grado FaMAF-UNC, 2006.