VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"ESPECIFICACIÓN, VERIFICACIÓN Y CÁLCULO DE ESTRATEGIAS EN JUEGOS NO COOPERATIVOS USANDO VERIFICACIÓN DE MODELOS"

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

DOCTOR EN CIENCIAS
(COMPUTACIÓN)

P R E S E N T A :

PEDRO ARTURO GÓNGORA LUNA

DIRECTOR DE TESIS:	DAVID ARTURO ROSENBLUETH LAGUETTE INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS UNAM
COMITÉ TUTOR:	ATOCHA ALISEDA LLERA INSTITUTO DE INVESTIGACIONES FILOSÓFICAS UNAM
	FRANCISCO HERNÁNDEZ QUIROZ FACULTAD DE CIENCIAS UNAM

México D.F. Diciembre 2014.

Resumen

El objetivo general de esta tesis es encontrar un método que utilice la verificación de modelos para caracterizar y calcular equilibrios de Nash en juego finitos no cooperativos. Se proponen tres aproximaciones a dicho objetivo general. La primera aproximación extiende la lógica temporal probabilística PCTL y sus algoritmos de verificación de modelos para caracterizar equilibros de Nash en estrategias mixtas. La segunda aproximación extiende la lógica dinámica proposicional con operadores de la lógica híbrida para caracterizar equilibros de Nash de juegos en forma extensiva. Además, en esta segunda aproximación, el uso del verificador de modelos puede calcular los equilibrios, y no solamente caracterizarlos. En la tercera aproximación, se propone utilizar un algoritmo de ruta más corta como parte del algoritmo de etiquetamiento de estados. Se generalizan los árboles de juego a gráficas de juego. Se extiende el algoritmo Bellman–Ford para encontrar rutas más cortas en dichas gráficas de juego. Adicionalmente, se provee una versión simbólica de la extensión de Bellman–Ford, lo que lo hace apto para su uso práctico en un verificador de modelos. Una vez obtenidos los algoritmos eficientes, se extiende la lógica temporal CTL para incluir nuevos operadores que permitan describir y razonar sobre las gráficas de juego. Utilizando fórmulas sencillas de este CTL extendido, el verificador de modelos es capaz de calcular eficientemente los equilibros de Nash.

Agradecimientos

Agradezco a la Universidad Nacional Autónoma de México por darme el tiempo y el espacio para desarrollar una vez más un proyecto académico que, además del interés personal que tiene para mi, espero también sea una contribución a la universidad y a mi país. Agradezco al CONACyT y al proyecto PAPIIT IN113013 por el poyo económico. Agradezco a mi tutor David A. Rosenblueth por su guía, consejos, apoyo, e infinita paciencia. Agradezco a Francisco y Atocha, miembros de mi comité tutor por el tiempo y empeño dedicado a dirigir mi trabajo. De la misma forma, también agradezco a mis sinodoales Favio y Sergio por sus comentarios que hicieron de esta tesis un mejor trabajo.

Agradezco a mi padres y a mis hermanas por su apoyo en estos años, por los que han pasado mucho más cosas que la vida académica que se reporta aquí. Agradezco a mis amigos por ser compañía, apoyo y fuente de inspiración.

Le dedico este trabajo a mis padres, en el sentido tanto biológico como espiritual, que al ponerme en el camino me permitieron encontrar el mío.

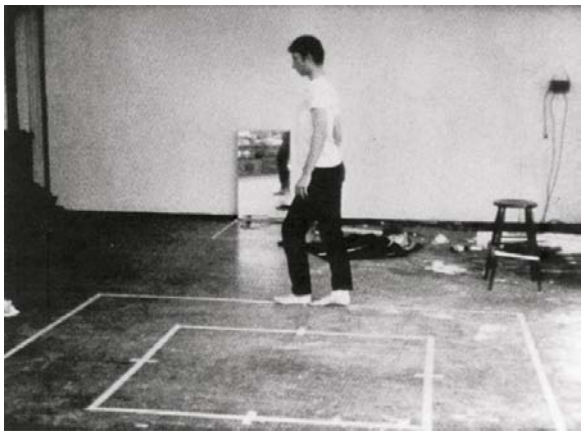
```
You have not convinced mighty Ziltoid
I am so omniscient
If there were to be two omnisciences
I would be both!
```

– Devin Townsend, *Ziltoid the Omniscient*, 2007

Índice general

1

Introducción



– Bruce Nauman, *Walking in an Exaggerated Manner Around the Perimeter of a Square*, 1967-68

Desde el fructífero *Theory of Games and Economic Behavior* de John von Neumann y Oskar Morgenstern [vNM44], y el célebre *Equilibrium Points in n-Person Games* de John F. Nash [Nas50], la teoría de juegos se mantiene en investigación activa en diversas disciplinas. El estudio de la teoría de juegos se extendió de las matemáticas y la economía hacia la filosofía, las ciencias sociales, la biología, y desde luego, a las ciencias de la computación. El que los juegos tengan implicaciones en tantas áreas nos habla de la relevancia de hallar nuevos métodos para razonar y automatizar tareas en sus aplicaciones.

En el estudio de la lógica hay actualmente un gran interés por describir y razonar sobre la estructura lógica de los juegos. Podemos encontrar numerosos ejemplos de esto en las reseñas de Johan van Benthem [vB05, vB12].

Esta tesis trata sobre verificación de modelos (cf. [BK08]) y juegos, y por lo tanto, también de lógica y juegos. Parto desde la línea de investigación de los

lógicos, pero con la visión de obtener resultados para una práctica computacional eficiente. Presento aquí tres acercamientos que reúnen a la verificación de modelos con la teoría de juegos, y que gradualmente se acercan a dicha visión.

En los últimos años podemos observar en la tradición lógica un ímpetu por encontrar, por un lado, la estructura formal de los juegos, y por otro lado nuevos lenguajes que describan mejor dicha estructura. Esto es, se trabaja con la descripción de la forma de un juego usando lenguajes lógicos tradicionales, tanto clásicos como no clásicos. Por otra parte, se desarrollan nuevos formalismos para dar descripciones más acertadas de dicha estructura, a la vez que se indaga su relación con las diversas posturas y perspectivas de pensamiento. Por ejemplo: desde la lógica epistémica tradicional, pero también desde lógicas *ceteris paribus*; desde la lógica dinámica, y también desde lógicas de coaliciones.

Desde un punto de vista computacional, puedo resaltar que la mayoría de estos trabajos se basan en el concepto de *validez deductiva*. Esto es, tenemos un conjunto de axiomas fundacionales que nos describen algún aspecto del juego. Así, para llevar dichos axiomas a la práctica computacional debemos encontrar una descripción lógica del problema, y después auxiliarnos de un demostrador automático de teoremas para decidir si se cumplen las propiedades deseadas.

La verificación de modelos tiene un enfoque complementario a la demostración automática de teoremas. En lugar de responder si un conjunto de fórmulas y axiomas se satisfacen en todo modelo posible, la verificación de modelos simplemente decide si un conjunto de fórmulas se satisface en *un* modelo en particular. En este enfoque, pasamos la descripción del problema de las fórmulas al modelo, y en el lenguaje lógico sólo dejamos descritas las propiedades. Aunque a primera vista el enfoque de la verificación de modelos parece más limitado, en realidad se han desarrollado técnicas para trabajar con problemas muy grandes, tan grandes que no podrían tratarse con un demostrador automático de teoremas.

Un juego describe la interacción de agentes racionales que buscan maximizar su utilidad, sea ésta de cualquier índole, monetaria o no. En su interacción, los agentes pueden competir (mientras unos ganan otros pierden) pero también cooperar. En algunos escenarios, la reacción del agente no pareciese la mejor en lo inmediato. Pero una vez tomada en cuenta la racionalidad de los demás agentes, y sus posibles alternativas de acción, el agente racional tomará las decisiones que le den mayor utilidad en el desenlace del juego. O bien, si el juego es repetido varias veces, las decisiones que en promedio le beneficien más.

Partiendo de esto, una herramienta que automatice o provea predicciones de dicho razonamiento, deberá encontrar siempre las alternativas que maximicen la utilidad de todos los agentes involucrados en el juego. Estas mejores

alternativas se llaman *soluciones* del juego, y en teoría de juegos se analizan múltiples variantes de ellas. Una de las soluciones más conocidas es la del *equilibrio de Nash*. Un equilibrio de Nash es un conjunto de estrategias, es decir, un plan de acciones tal que todos los agentes no pueden incrementar su ganancia desviándose de dicho plan.

Aquí me concentro en los equilibrios de Nash, tanto en su caracterización lógica, como en un su cómputo. En las tres aproximaciones que presento en la tesis utilizo la verificación de modelos como base para estas tareas, por lo que tienen, potencialmente, utilidad práctica.

Mi acercamiento es paulatino. En primera instancia trabajo con una caracterización lógica de equilibrios de Nash. Pero, a diferencia de otros trabajos previos de mi conocimiento, aquí presento una caracterización de equilibrios mixtos. Es decir, equilibrios donde el plan de acción dicta a los agentes actuar con una guía probabilística.

En este primer acercamiento, el juego y la solución potencial se describen en el modelo, una cadena de Markov. Con estos modelos, y utilizando una lógica temporal probabilística, es posible caracterizar un equilibrio de Nash mixto. De esta forma, la tarea del verificador de modelos será decidir si la solución potencial es en realidad una solución del juego.

En el segundo acercamiento exploro la relación que guarda una familia de axiomas de la lógica modal con la estructura de un equilibrio de Nash. Partiendo de estos axiomas, propongo el uso de lógica dinámica híbrida para llevar la caracterización lógica de la validez deductiva a la satisfacción, y con esto, de la demostración automática de teoremas a la verificación de modelos. Como en el acercamiento anterior, el juego se describe en el modelo, una estructura de Kripke, pero aquí no es necesario proponer una posible solución. Al decidir si el juego satisface las fórmulas que propongo, el verificador de modelos calculará automáticamente la solución del juego.

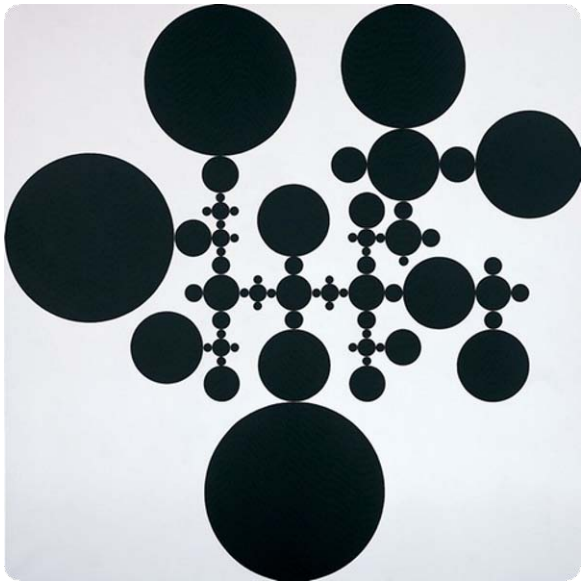
En el tercer acercamiento parto desde una perspectiva distinta. Exploro primero la relación que tiene un problema clásico de la computación con los problemas de optimización. El problema de encontrar rutas más cortas en gráficas dirigidas es un problema de optimización, o programación matemática. Propongo entonces una generalización con agentes y pesos de las gráficas dirigidas, así como una extensión de los algoritmos de ruta más corta. Particularmente, extiendo el algoritmo Bellman–Ford para encontrar rutas más cortas en dichas gráficas generalizadas. Por otra parte, estas gráficas también son una generalización de una clase de juegos, y el problema de encontrar una ruta más corta se vuelve equivalente al de encontrar un equilibrio de Nash. Una vez obtenidos los algoritmos eficientes, extiendo también el lenguaje lógico para describir las propiedades de modelos basados en estas generalizaciones.

Finalmente, también adecúo los procedimientos de verificación de modelos para aprovechar los algoritmos propuestos.

La estructura de la tesis es cronológica, y obedece también a la presentación de los acercamientos como los describí aquí. En el siguiente capítulo hago una breve reseña de algunos trabajos relacionados, además de una presentación muy breve de las propuestas de esta tesis. En los capítulos 3, 4 y 5 presento detalladamente los tres acercamientos. En el último capítulo ofrezco algunas reflexiones finales.

2

Lógica modal, agentes, juegos y conocimiento



– Gabriel Orozco, *The Eye of Go*, 2005

2.1. Conceptos básicos de la teoría de juegos y la lógica modal

Esta sección está dedicada a presentar los conceptos y herramientas más elementales de la teoría juegos y la lógica modal. El objetivo no es proveer de

una introducción exhaustiva. La finalidad es facilitar la lectura del resto del texto, acotando definiciones y unificando la nomenclatura.

Para una introducción concisa a la teoría de juegos, se puede consultar [OR94]. También, [BdRV01, Pop94] son buenas introducciones a la lógica modal.

2.1.1. Juegos y equilibrios

De manera muy general, podemos decir que un juego es la descripción formal de la interacción estratégica entre agentes racionales. A continuación acotaremos los elementos de esta definición general.

El elemento central es el *agente* o *jugador* (ambos términos suelen usarse indistintamente). Un agente puede ser un individuo o un conjunto de individuos. Los agentes pueden ser humanos, computadoras o, en aplicaciones modernas a la evolución, hasta poblaciones de animales. El requisito indispensable para un agente de teoría de juegos es la *racionalidad*.

La racionalidad del agente se refiere al imperativo de maximizar su *utilidad* o *ganancia*. Esta utilidad, que representamos con un valor numérico o un orden de preferencias, es el resultado de la interacción de todos los agentes del sistema. La utilidad obtenida por un agente no sólo es función de sus propias acciones, sino también de las posibles acciones emprendidas por los demás jugadores (característica fundamental que distingue a la teoría de juegos de otras teorías de la decisión).

De esta manera, se vuelve más clara la importancia de la planificación de las acciones. Esto es, la elección de las mejores *estrategias* por parte de cada jugador. La mejor estrategia para un agente es, sin duda, aquella que maximiza su utilidad. Sin embargo, las preferencias de los agentes pueden ser contrarias, lo que beneficia a uno puede ser adverso para otros. Por esta razón, dando por hecho que todos buscan maximizar su propia ganancia, y que este hecho es *conocimiento común* entre todos los agentes, buscar la mejor estrategia no es una tarea fácil.

Un *equilibrio* es una selección de estrategias, una para cada agente, tales que ningún agente podrá tener mayor ganancia desviándose unilateralmente de esta selección. El término *equilibrio de Nash* se da en honor a John Forbes Nash, quien en su célebre teorema de 1950 demostró que todo juego finito tiene al menos un equilibrio (cf. [Nas50]).

Los elementos de esta selección pueden ser simples planes de acción: si sucede esto, entonces haz esto otro. Pero también pueden introducir *incertidumbre*: si sucede esto, entonces elige una acción de acuerdo con una distribución de probabilidad determinada.

A las estrategias simples, las llamamos *puras*, mientras que a las que involucran distribuciones de probabilidad las llamamos *mixtas*. De la misma manera, cuando un equilibrio sólo consta de estrategias puras se le llama *equilibrio en estrategias puras*, y análogamente, cuando contienen estrategias mixtas se les llama *equilibrio en estrategias mixtas*.

Hay dos maneras tradicionales de formalizar estos conceptos. Los juegos en *forma extensiva* tienen información detallada de la secuencia en que suceden las acciones. Los juegos en *forma estratégica* hacen una abstracción mayor y se concentran en las implicaciones inmediatas de todas las acciones. Aunque no vamos a detallarlo aquí, conviene mencionar que existe una equivalencia entre estos dos formalismos. Todo juego extensivo tiene un equivalente estratégico, y todo juego estratégico tiene un número infinito de equivalentes extensivos.

Para delimitar mejor el alcance del texto, cabe aclarar que sólo trataremos con juegos finitos. También, sólo trataremos con los juegos llamados *no cooperativos*, donde cada jugador es, o puede modelarse como, una sola entidad.

Forma extensiva

Esta representación formaliza un juego como un árbol. Los nodos son instantes de tiempo, y cada instante corresponde al turno de exactamente un jugador. En ese momento, el jugador puede seleccionar una de varias posibles acciones. La selección de una acción lleva al siguiente instante de tiempo, donde puede ser turno del mismo u otro jugador. Como tenemos un árbol, las acciones disponibles en un momento dependen de la secuencia de acciones que se ha desarrollado hasta el momento. Decimos que el juego es finito, si el árbol que lo representa tiene anchura y profundidad finita.

Una estrategia es un plan de acción, dicta una sola acción para cada posible situación en el juego. Una estrategia está asociada a un solo jugador. Un perfil de estrategias es una colección de estrategias, una para cada jugador. De esta forma, si comenzamos el juego desde el inicio, un perfil de estrategias determina un único desenlace para el juego.

Definición 2.1. Un *juego en forma extensiva* G es la tupla:

$$G = \langle S, s_0, \prec, N, P, \{u_i\}_{i \in N} \rangle$$

Donde:

- S es un conjunto finito de nodos
- La relación intransitiva $\prec \subseteq S \times S$ ordena S en un árbol con raíz s_0 y hojas en $Z \subseteq S$

- $N = \{1, \ldots, n\}$ es el conjunto de jugadores
- $P : (S - Z) \to N$ es la función que asigna turnos a los nodos no finales
- $u_i : Z \to \mathbb{Q}$ es la función de utilidad del jugador i

Dado un juego G, podemos definir lo siguiente:

- Una *estrategia a_i para el jugador $i \in N$* es una función $a_i : (s \in (S - Z) \mid P(s) = i) \to \{s' \in S \mid s \prec s'\}$, que asigna un sucesor a los nodos con turno i, denotamos como A_i al *conjunto de todas las posibles estrategias de i*
- Un *perfil de estrategias* es una tupla $a \in \times_{i \in N} A_i$, denotamos como A al *conjunto de todos los posibles perfiles de estrategias.* ∎

Dado un perfil de estrategias $a = (a_1, \ldots, a_i, \ldots, a_n)$, usamos a_{-i} para denotar a la tupla que tiene las mismas estrategias que a, menos a_i. Dada una estrategia $a'_i \in A_i$, usamos (a_{-i}, a'_i) para denotar al perfil cuyo i-ésimo componente es a'_i, pero todos sus demás componentes son los mismos que en el perfil a.

Es fácil mostrar que un perfil de estrategias induce un único recorrido de la raíz hacia una hoja. También, la función de utilidad u_i induce una relación de preferencias sobre las hojas, que llamaremos $\leq_i$. Sean z y z' las hojas inducidas por los perfiles a y a', respectivamente. Extendemos la relación de preferencia a los perfiles como sigue: $a \leq_i a'$ sii $u_i(z) \leq_i u_i(z')$.

Definición 2.2. Sea a un perfil de estrategias para el juego $G = \langle\ S,\ s_0,\ \prec,\ N,\ P,\ \{u_i\}_{i \in N}\ \rangle$. Decimos que a es *una mejor respuesta para el jugador $i \in N$* si para toda estrategia $a'_i \in A_i$, se cumple que $(a_{-i}, a'_i) \leq_i a$. También, decimos que a es *un equilibrio de Nash* si a es una mejor respuesta para todo jugador $i \in N$. ∎

Inducción hacia atrás

Un método que siempre encuentra un equilibrio para este tipo de juegos es el de *inducción hacia atrás*. El algoritmo de inducción hacia atrás selecciona la mejor respuesta para los subjuegos más pequeños[1] de un árbol y la lleva hacia la raíz. Vamos a presentar el método por medio de un ejemplo.

La figura 2.1 muestra un juego para dos jugadores (1 y 2). Arriba de cada nodo se muestra el jugador en turno. Debajo de las hojas se muestran las

[1] Subárboles no vacíos cuya raíz sólo tiene descendientes hojas

utilidades. El algoritmo comienza por los subárboles con raíz s_1 y s_2. En s_1 es el turno del jugador 2, por lo que su mejor respuesta es elegir z_2 (marcado con una línea gruesa). Similarmente, en s_2 se elige la acción que lleva a z_4. Como todos los subjuegos de s_0 están resueltos, en la raíz podemos ver que la mejor respuesta es la opción derecha.

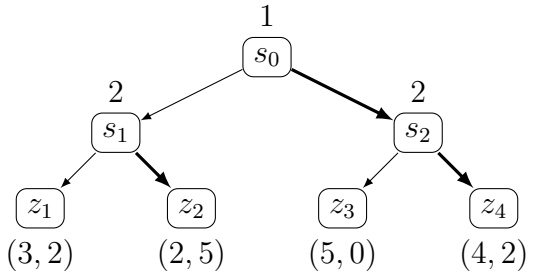


Figura 2.1: Inducción hacia atrás para un juego de 2 jugadores

Antes de continuar, hacemos una aclaración. Las definiciones anteriores se refieren sólo a una subclase de los juegos finitos donde los jugadores carecen de incertidumbre: los juegos de información perfecta y completa. En el siguiente apartado presentamos una representación más abstracta que permite modelar varias formas de incertidumbre. Sin embargo, remitimos al lector al texto de Osborne y Rubinstein [OR94] para una discusión profunda sobre las diferencias entre estos juegos.

Forma estratégica

La forma estratégica (también llamada forma normal o matricial) no toma en cuenta la secuencia de las acciones. En un juego estratégico, cada agente tiene asociado un conjunto de estrategias. Un desenlace es simplemente una tupla que contiene una acción por cada jugador. Así, estos juegos prestan atención a los efectos inmediatos de las estrategias. Esta abstracción también vuelve directa la definición de equilibrio para este tipo de juegos.

Definición 2.3. Un *juego en forma estratégica* G es la tupla:

$$G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$$

Donde:

- $N = \{1, \ldots, n\}$ es un conjunto de jugadores

- A_i es el conjunto finito de las estrategias del jugador i
- $u_i : A \to \mathbb{Q}$ es la función de utilidad del jugador i, donde $A = \times_{i \in N} A_i$.

Sea a un perfil de estrategias de un juego estratégico $G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$. Decimos que a es *una mejor respuesta para el jugador* $i \in N$ si para toda estrategia $a'_i \in A_i$, se cumple que $(a_{-i}, a'_i) \leq_i a$. También, decimos que a es *un equilibrio de Nash* si a es una mejor respuesta para todo jugador $i \in N$. ∎

Hasta el momento sólo hemos presentado equilibrios en estrategias puras. Sin embargo, existen juegos muy simples que no tienen equilibrios en estrategias puras. Por ejemplo, el juego *piedra, papel y tijera* no puede tener equilibrio en estrategias puras: ¡un jugador siempre perdería!

La extensión mixta de un juego permite describir escenarios donde los jugadores emplean estrategias probabilísticas. En estos juegos, las estrategias son distribuciones de probabilidad sobre las estrategias "puras", y la ganancia se convierte en la esperanza matemática de la utilidad. La ventaja es que se puede asociar al menos un equilibrio con cualquier juego finito.

Definición 2.4. Sea $G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$ un juego estratégico y $\Delta(X)$ el conjunto de todas las posibles distribuciones de probabilidad sobre un conjunto X. La *extensión mixta de* G es la tupla:

$$\hat{G} = \langle N, \{\Delta(A_i)\}_{i \in N}, \{U_i\}_{i \in N} \rangle$$

Donde:

- $\Delta(A_i)$ es el conjunto de todas las estrategias mixtas de i (distribuciones de probabilidad sobre las estrategias puras del jugador i)
- $U_i : \hat{A} \to \mathbb{Q}$ es la esperanza matemática de la utilidad, donde $\hat{A} = \times_{i \in N} \Delta(A_i)$.

Sea G un juego estratégico, $\hat{G}$ su extensión mixta y $\alpha \in \hat{A}$ un perfil de estrategias mixtas. Decimos que α *es un equilibrio en estrategias mixtas de* G si α es un equilibrio de Nash de su extensión mixta $\hat{G}$. ∎

Por ejemplo, considera el juego *Bach o Stravinsky* (BoS, fig. 2.2). Los jugadores deben acordar asistir sólo a un concierto. La decisión es simultánea, esto es, no saben la decisión del otro al tomar la suya. El jugador 1 (renglones) prefiere lo doble a Bach que a Stravinsky. El jugador 2 (columnas) prefiere lo doble a Stravinsky que a Bach.

Es fácil verificar que BoS tiene dos equilibrios en estrategias puras: (B_1, B_2) y (S_1, S_2). Pero, también tiene un equilibrio en estrategias mixtas: $\left(\left(\frac{2}{3}, \frac{1}{3}\right), \left(\frac{1}{3}, \frac{2}{3}\right)\right)$ (e.g., el primer jugador elige Bach con $\frac{2}{3}$ de probabilidad y elige Stravinsky con $\frac{1}{3}$ de probabilidad, cf. el teorema 2.10 más adelante).

	B_2	S_2
B_1	$2, 1$	$0, 0$
S_1	$0, 0$	$1, 2$

Figura 2.2: Juego estratégico BoS

2.1.2. Lógica modal

Antes de presentar algunos ejemplos de formalizaciones de equilibrios de Nash, en esta sección presentamos un lenguaje modal simple y su semántica formal. El propósito es establecer un punto de partida presentando las nociones sintácticas y semánticas generales de la lógica modal, y así tener una base común para las diferentes aproximaciones que presentamos más adelante en el capítulo.

La lógica modal extiende a la lógica clásica con características intencionales, permitiendo denotar objetos por sus propiedades. Veamos entonces un lenguaje modal que podemos extender posteriormente.

Posibilidad, necesidad y marcos de Kripke

El lenguaje que presentamos extiende a la lógica proposicional con dos operadores nuevos: de posibilidad y necesidad.

Definición 2.5. Sean I un conjunto numerable de índices y Φ_0 un conjunto numerable de símbolos de proposición. El *conjunto* Φ *de todas las fórmulas de la lógica multimodal* es el menor conjunto construido de acuerdo con la siguiente gramática:

$$\varphi, \psi ::= p \mid \neg\varphi \mid (\varphi \wedge \psi) \mid \Box_i\varphi \mid \Diamond_i\varphi$$

Donde $p \in \Phi_0$ e $i \in I$. ∎

Los conectivos $\neg$ y $\wedge$ heredan su interpretación intuitiva de la lógica clásica. También, definimos los demás conectivos usuales ($\vee$, $\Rightarrow$ y $\Leftrightarrow$) como secundarios (e.g., usando las leyes de De Morgan).

A las fórmulas de tipo $\Box_i\varphi$ les daremos el significado intuitivo: "*de acuerdo con i, necesariamente* φ *es verdadera*". Similarmente, a las fórmulas de tipo $\Diamond_i\varphi$ las leeremos como: "*de acuerdo con i, posiblemente* φ *es verdadera*". Además, estos operadores satisfacen la dualidad: $\Box_i\varphi \Leftrightarrow \neg\Diamond_i\neg\varphi$.

La satisfacción de las fórmulas modales es local. Utilizaremos la semántica formal más común: la semántica relacional o de mundos posibles. Los modelos

de esta semántica se basan en estructuras relacionales llamadas marcos de Kripke (en honor a Saul Kripke por su influyente artículo [Kri63]).

Definición 2.6. Sean I un conjunto numerable de índices y W un conjunto numerable de mundos posibles. Un *marco de Kripke* F es la tupla:

$$F = \langle W, \{R_i\}_{i \in I} \rangle$$

Donde $R_i \subseteq (W \times W)$ es una *relación de accesibilidad para el índice* $i \in I$. Dado un conjunto φ_0 de símbolos de proposición, definimos una *valuación* como una función $V : (\Phi_0 \times W) \to \{1, 0\}$. Dado un marco de Kriple $F = \langle W, \{R_i\}_{i \in I} \rangle$ definimos un *modelo para las fórmulas en* Φ como la tupla:

$$M = \langle W, \{R_i\}_{i \in I}, V \rangle$$

Llamamos *modelo apuntado* a una pareja (M, w), con M un modelo y w un mundo posible de dicho modelo. Finalmente, definimos la *relación de satisfacción* $\models$ entre modelos apuntados y fórmulas en Φ como la menor relación tal que:

1. $(M, w) \models p$ sii $V(p, w) = 1$
2. $(M, w) \models \neg\varphi$ sii $(M, w) \not\models \varphi$
3. $(M, w) \models (\varphi \wedge \psi)$ sii $(M, w) \models \varphi$ y $(M, w) \models \psi$
4. $(M, w) \models \Box_i \varphi$ sii para todo $w' \in W$, $(w, w') \in R_i$ implica que $(M, w') \models \varphi$
5. $(M, w) \models \Diamond_i \varphi$ sii existe $w' \in W$ tal que $(w, w') \in R_i$ y $(M, w') \models \varphi$. ∎

En la siguiente sección vamos a revisar varias extensiones a este lenguaje. Para más discusión sobre la lógica modal se pueden consultar [Pop94, BdRV01].

2.2. Formalización de equilibrios de Nash

Una vez repasadas algunas nociones importantes de la lógica modal y la teoría de juegos, en esta sección haremos un repaso a la caracterización de los equilibrios de Nash usando diversas variantes de lógica modal. Todas las caracterizaciones corresponden al trabajo de distintos autores, y en cada caso referimos al lector a la fuente original. La lista que presentamos no es exhaustiva, sin embargo, presentamos los trabajos que consideramos más relevantes para el desarrollo posterior de la tesis.

2.2.1. Estrategias puras

En lógica temporal

La lógica temporal fue una de las primeras aplicaciones modernas de la lógica modal (v. [Pri57]). Modelamos el tiempo discreto, ya sea como una sucesión infinita o como un árbol infinito de instantes que representan los distintos futuros posibles. La modalidad $\Diamond\varphi$ (o $F\varphi$) afirma: "φ *sucederá eventualmente*". Mientras que $\Box\varphi$ (o $G\varphi$) afirma: "φ *sucede a partir de ahora*".

El trabajo de Bonanno [Bon01] fue el primero en considerar fructífero comparar la estructura de un juego extensivo con los modelos de tiempo ramificado de la lógica temporal. En este trabajo, Bonanno crea un modelo a partir de un juego y un perfil de estrategias. Utilizando un esquema de axiomas de la lógica temporal, es posible decidir por deducción si el perfil empleado para construir el modelo corresponde con el resultado de la aplicación del principio de inducción hacia atrás y, por lo tanto, es un equilibrio de Nash.

Para el marco del modelo usamos un juego extensivo G y un perfil de estrategias a (def. 2.1). Además, el lenguaje modal de Bonanno tiene los siguientes elementos:

- Proposiciones atómicas de tipo $q_1 \leq q_2$ (con $q_1, q_2 \in \mathbb{Q}$), con su interpretación intuitiva
- Proposiciones atómicas de tipo $u_i = q$ (con $i \in N$ y $q \in \mathbb{Q}$), verdaderas sólo en las hojas donde el jugador i obtiene utilidad q
- Fórmulas $F\varphi$: "*eventualmente,* φ *será verdadera en el futuro*", que interpretamos con $\prec$
- Fórmulas $F_p\varphi$: "φ *será verdadera en un futuro predecible*", que interpretamos con la relación de accesibilidad inducida por el perfil a
- Fórmulas $\Box_i\varphi$: "φ *será verdadera sin importar la acción que realice* i", que interpretamos con $\prec$ restringida al dominio $\{s \mid P(s) = i\}$

Definimos el siguiente esquema de *consistencia interna* en este lenguaje modal:

$$IC \stackrel{\text{def}}{=} F_p(u_i = q) \Rightarrow \Box_i(((u_i = r) \vee F_p(u_i = r)) \Rightarrow (r \leq q))$$

Sea q la utilidad que se predice para i dado un perfil de estrategias determinado. Sin importar qué acción realice i, si obtiene una utilidad r o se puede predecir una utilidad r, entonces r es a lo más q

Lo que nos dice el esquema *IC* es que *i no puede incrementar su utilidad desviándose de la predicción.*

Así, tenemos nuestra caracterización deseada ([Bon01]):

Teorema 2.7. *Sean M_G un modelo para el juego G y a un perfil de estrategias. Se cumple que a es un equilibrio de Nash de G sii para todo $s \in S$ y para toda instancia φ de IC, entonces el modelo apuntado (M_G, s) satisface φ (v. [Bon01] para las definiciones precisas de satisfacción).*

En lógica dinámica

Inicialmente pensada como un "cálculo de programas" (v. [HKT00]), la lógica dinámica es una extensión del lenguaje multimodal. Se permite construir modalidades nuevas utilizando algún lenguaje apropiado, usualmente expresiones regulares. Las modalidades del tipo $[a \cup b]\varphi$ nos dicen "*después de ejecutar el programa a o el programa b, necesariamente se cumple* φ". La sintaxis nos indica la interpretación. Por ejemplo, la interpretación de $[a \cup b]$ es la unión de las interpretaciones de $[a]$ y $[b]$. Con estos constructores pueden definirse nuevos programas complejos, como ciclos `while` y condicionales `if-then-else`.

Esta caracterización por Harrenstein et al. [HvdHMW03] se basa en la lógica dinámica. Esto tiene la finalidad de describir las estrategias del juego, además de su estructura temporal.

Dado un juego extensivo G y un perfil de estrategias a, construimos un marco de Kripke F_G con los siguientes elementos (v. [HKT00] para una exposición detallada de esta lógica):

- Un programa atómico a_i para cada jugador i. De esta forma, un nodo s está relacionado con otro s' por el programa a_i sii el jugador i tiene una estrategia que lo lleva de s a s'
- Un programa a para el perfil de estrategia que queramos analizar, interpretado por la relación inducida por a
- Un programa atómico i para cada jugador i. El programa i relaciona una hoja z con otra z' sii la utilidad del jugador i en z es menor o igual a su utilidad en z'

Dado un conjunto de jugadores $\{i_1, \ldots, i_k\}$, definimos el siguiente programa:

$$\pi(a, \{i_1, \ldots, i_k\}) \stackrel{\text{def}}{=} \texttt{while}\ \langle a \rangle \top\ \texttt{do}\ (a_{i_1} \cup \cdots \cup a_{i_k} \cup a)$$

Particularmente nos interesan dos instancias del programa anterior:

$$\pi(a, \{i\}) = \texttt{while } \langle a \rangle \top \texttt{ do } (a_i \cup a)$$
$$\pi(a, \varnothing) = \texttt{while } \langle a \rangle \top \texttt{ do } a$$

El programa $\pi(a, \{i\})$ representa los posibles resultados que puede forzar el jugador i, si suponemos que los demás jugadores siguen el perfil a. El programa $\pi(a, \varnothing)$ representa el resultado en donde *todos* los jugadores siguen el perfil a.

Tomamos el siguiente caso especial de confluencia modal (cf. [Pop94, cap. 6]):

Teorema 2.8. *Sean F un marco, w un mundo posible de F, y φ cualquier fórmula. Entonces, la afirmación:*

$$F, w \models \langle a \rangle [b] \varphi \Rightarrow [c] \varphi$$

se cumple sii para todo w' y w'': si $(w, w') \in R_a$ y $(w, w'') \in R_c$, entonces $(w', w'') \in R_b$.

De este resultado de confluencia se sigue que si un perfil a es la mejor respuesta para un jugador i, entonces:

$$F_G, s_0 \models \langle \pi(a, \{i\}) \rangle [i] \varphi \Rightarrow [\pi(a, \varnothing)] \varphi$$

Esto es, a representa la mejor respuesta para i sii toda posible salida forzada por i es menor o igual, en términos de utilidad, a la salida determinada por el perfil a.

Finalmente, tenemos que si lo anterior se cumple para todos los jugadores, entonces a es un equilibrio de Nash ([HvdHMW03]).

En lógica de preferencias *ceteris paribus*

La lógica de preferencias interpreta modalidades directamente como relaciones de orden sobre los mundos posibles. La fórmula $\Diamond^{<} \varphi$ afirma que ϕ es verdadera en al menos un mundo preferible al actual. El lenguaje de van Benthem et al. [vBGR08] extiende esta lógica para describir preferencias *ceteris paribus*.

El latín *ceteris paribus* puede traducirse como "siendo todo lo demás igual". Así, por ejemplo, la preferencia por el vino tinto generalmente es *ceteris paribus*, pues con un menú sin carne roja posiblemente se prefiera vino blanco. En el lenguaje de [vBGR08] esta preferencia puede expresarse como: $[carne]^{<}$ *tinto* (estrictamente prefiero vino tinto siendo que no cambia el menú de carne roja).

Para esta caracterización vamos a usar fórmulas de tipo $\langle\Gamma\rangle_i^{<}\varphi$, donde i es un jugador y Γ es un conjunto de fórmulas sin operadores modales. Interpretamos la fórmula como: "*restringido a mundos* Γ*-equivalentes y a las preferencias de* i*, existe un mundo preferible al actual donde* φ *es verdadera*"[2].

Consideremos un juego estratégico G para dos jugadores. Supongamos que las estrategias de los jugadores son $A_1 = \{a_1, \ldots, a_{m_1}\}$ y $A_2 = \{b_1, \ldots, b_{m_2}\}$.

Con este juego construimos el siguiente modelo M_G:

- Un marco de Kripke cuyo conjunto de mundos posibles son los perfiles de estrategias de G (i.e., $W = A_1 \times A_2$)
- Las relaciones de accesibilidad que interpretan a las modalidades $\langle\Gamma\rangle_i^{<}\varphi$ son directamente el orden estricto de las utilidades de i sobre $W = A_1 \times A_2$
- Tenemos proposiciones atómicas de tipo a_k y b_l, que se satisfacen sólo en el mundo posible $(a_k, b_l) \in W$

Supongamos que este G tiene un equilibrio de Nash $a = (a_k, b_l)$. Consideremos la afirmación:

$$(M_G, a) \models \neg\langle b_l\rangle_1^{<}\top$$

donde $\top$ es cualquier tautología. Esta afirmación nos indica que para 1, no existe un mundo estrictamente mejor a a siendo que 2 no cambia su estrategia b_l. En otras palabras, la estrategia a_k es una mejor respuesta de 1 al perfil a.

De esta manera, podemos expresar que a es un equilibrio de Nash de este juego como:

$$(M_G, a) \models \neg\langle b_l\rangle_1^{<}\top \wedge \neg\langle a_l\rangle_s^{<}\top$$

Para el caso general de n jugadores es fácil extender el método con el que se construye el modelo M_G. Con esto tenemos la caracterización deseada ([vBGR08]):

Teorema 2.9. *Sea* M_G *un modelo para el juego* G*. Un perfil de estrategias* a *es equilibrio de Nash de* G *sii*

$$(M_G, a) \models \bigwedge_{i \in N} \neg\langle a_{-i}\rangle_i^{<}\top$$

[2]Es posible definir modalidades con órdenes no estrictos o duales, como en el ejemplo del vino tinto

En lógica de coaliciones

Las lógicas de coaliciones describen las capacidades de grupos de agentes. Estas lógicas utilizan modelos donde existen múltiples agentes, cada uno capaz de ejecutar sus propias acciones. En estas lógicas se tienen modalidades de coalición. Por ejemplo, si C es una coalición (un conjunto de agentes), la fórmula $\langle\langle C\rangle\rangle\varphi$ afirma que "*los agentes en C tienen la capacidad de forzar que se satisfaga φ*".

Esta caracterización por van der Hoek et al. [vdHJW05] utiliza una lógica llamada CATL (*Coalitional ATL*). Este lenguaje extiende la lógica temporal y de coaliciones ATL [AHK02] con un operador de contrafactuales.

El lenguaje de [vdHJW05] tiene fórmulas del tipo:

- $\langle\langle C\rangle\rangle\tau$: "*la coalición C puede forzar τ*"
  Donde τ puede ser:
  - $\mathcal{X}\varphi$: "*en el siguiente estado se cumple φ*"
  - $\mathcal{F}\varphi$ "*en el futuro se cumple φ*"
- $\mathcal{C}_i(a_i, \varphi)$: ''*si el agente i se sujeta a la estrategia a_i, entonces se cumple φ*"

Dado un juego estratégico G de dos jugadores construimos un modelo M_G como sigue:

- Los mundos posibles corresponden a las posibles asignaciones de ganancia (i.e., a las celdas de la matriz de utilidades)
- Tenemos proposiciones atómicas de tipo $(q \leq u_i)$ tal que $i \in N$, y q está en la imagen de alguna función de utilidad (aparece en alguna celda de la matriz), llamamos al conjunto de estos valores U

Definimos una fórmula que afirme que la estrategia a_i es la mejor respuesta a la estrategia a_j:

$$BR_i(a_i, a_j) \stackrel{\text{def}}{=} \mathcal{C}_j\left(a_j, \bigwedge_{q\in U}(\langle\langle i\rangle\rangle\mathcal{X}(q \leq u_i) \rightarrow \mathcal{C}_i\left(a_i, \langle\langle\rangle\rangle\mathcal{X}(q \leq u_i)\right))\right)$$

Si j se apega su estrategia a_j, entonces la utilidad que puede forzar i no puede ser mayor a la utilidad que obtiene apegándose a su estrategia a_i

Finalmente, un equilibrio de Nash se caracteriza por:

$$NE(a_i, a_j) \stackrel{\text{def}}{=} BR_i(a_i, a_j) \wedge BR_j(a_j, a_i)$$

2.2.2. Estrategias mixtas

En lógica temporal probabilística

En algunas situaciones describir sólo posibilidad y necesidad no es suficiente. Por ejemplo, la posibilidad de que un sistema falle puede tolerarse si la probabilidad con que ocurre la falla es suficientemente pequeña.

Una extensión natural a los marcos de Kripke es asignar probabilidades a las transiciones entre mundos posibles. El objeto matemático resultante es una *cadena de Markov*. Un trabajo seminal es el de Hansson y Jonsson [HJ94], donde definen una extensión para la lógica temporal CTL [CES86] para adecuarla a sistemas probabilísticos. Esta lógica, llamada PCTL, sustituye los operadores $\Box$ y $\Diamond$ por modalidades probabilísticas. Por ejemplo:

- $\mathcal{P}_{\leq m}[\tau]$: "*con probabilidad no mayor a m se satisface* τ"
- $\mathcal{E}_{\leq m}[\varphi]$: "*la utilidad esperada por alcanzar un mundo posible donde se satisface* φ *es a lo más m*"

donde φ es cualquier fórmula y τ es una fórmula temporal como:

- $\mathcal{X}\varphi$: "*en el siguiente estado se satisface* φ"
- $\mathcal{F}\varphi$: "*en el futuro se satisface* φ"

El trabajo de Góngora y Rosenblueth [GR09] utiliza PCTL para caracterizar equilibrios de Nash en estrategias mixtas. En [GR09] se toman un juego estratégico y un perfil de estrategias mixtas para construir una cadena de Markov. Utilizando una fórmula de PCTL (con una breve extensión) se puede decidir si el perfil es un equilibrio en estrategias mixtas del juego.

Recordemos que una estrategia mixta α_i para el jugador i es una distribución de probabilidad sobre el conjunto de sus estrategias puras A_i. Así, decimos que i puede elegir una estrategia pura a_i con probabilidad $\alpha_i(a_i)$. Al conjunto de estrategias puras con probabilidad mayor a cero le llamamos el soporte de α_i (denotado por $supp(\alpha_i)$). También, decimos que una estrategia mixta α_i *degenera* en una estrategia pura a_i sii $\alpha_i(a_i) = 1$.

Antes de presentar la caracterización en [GR09] es conveniente recordar el siguiente resultado de teoría de juegos:

Teorema 2.10. *Sea G un juego estratégico finito. Un perfil de estrategias mixtas* α *es un equilibrio de Nash de G sii se cumplen las siguientes dos condiciones:*

1. *La ecuación* $U_i(\alpha_{-i}, a_i) = U_i(\alpha_{-i}, a_i')$ *se satisface para toda* a_i *y* a_i' *en* $supp(\alpha_i)$

2. *La desigualdad* $U_i(\alpha_{-i}, a_i) \leq U_i(\alpha)$ *se satisface para toda* a_i *en* $A_i - supp(\alpha_i)$.

El teorema anterior nos dice que cuando α es equilibrio, los jugadores obtienen la misma utilidad *en promedio* para cualquier estrategia pura en el soporte de su estrategia mixta. Además, los jugadores no pueden incrementar su utilidad *promedio* desviándose de ese soporte.

Dado un juego G y una estrategia mixta α construimos un modelo M_G con las siguientes características:

- Un nodo inicial s_0
- Debajo de s_0, para cada agente $i \in N$ y cada estrategia $a_i \in supp(\alpha_i)$:
  - Un árbol con raíz a_i representado la elección de i por la estrategia pura a_i y donde se satisface la proposición atómica homónima a_i
  - Debajo de esta nueva raíz se desenvuelven todas las posibles jugadas tales que, las hojas de este árbol representen perfiles de estrategias puras que tienen fija a_i
  - Las probabilidades de las transiciones de este árbol se asignan de acuerdo con α, y los costos de acuerdo con las funciones de utilidad de los jugadores
- Un nodo final debajo de todo el árbol donde se satisface la proposición atómica *end*

Un ejemplo de esta construcción para el juego *Bach o Stravinsky* y su equilibrio mixto $\alpha = \left(\left(\frac{2}{3}, \frac{1}{3}\right), \left(\frac{1}{3}, \frac{2}{3}\right)\right)$ está en la figura 2.3. En este modelo, podemos verificar los siguiente hechos (v. [BK08, cap. 10] para una explicación detallada de este procedimiento de verificación):

$$(M, s_{(1,B_1)}) \models B_1 \wedge \mathcal{E}_{=\frac{2}{3}} end \qquad (M, s_{(2,B_2)}) \models B_2 \wedge \mathcal{E}_{=\frac{2}{3}} end$$
$$(M, s_{(1,S_1)}) \models S_1 \wedge \mathcal{E}_{=\frac{2}{3}} end \qquad (M, s_{(2,S_2)}) \models S_2 \wedge \mathcal{E}_{=\frac{2}{3}} end$$

Cuantificando la utilidad esperada podemos dar una sola fórmula:

$$\begin{aligned}(M, s_0) \models \exists x.\, (\mathcal{P}_{>0}[\mathcal{X}(B_1 \wedge \mathcal{E}_{=x} end)] \wedge \mathcal{P}_{>0}[\mathcal{X}(S_1 \wedge \mathcal{E}_{=x} end)]) \\ \wedge \exists x.\, (\mathcal{P}_{>0}[\mathcal{X}(B_2 \wedge \mathcal{E}_{=x} end)] \wedge \mathcal{P}_{>0}[\mathcal{X}(S_2 \wedge \mathcal{E}_{=x} end)])\end{aligned}$$

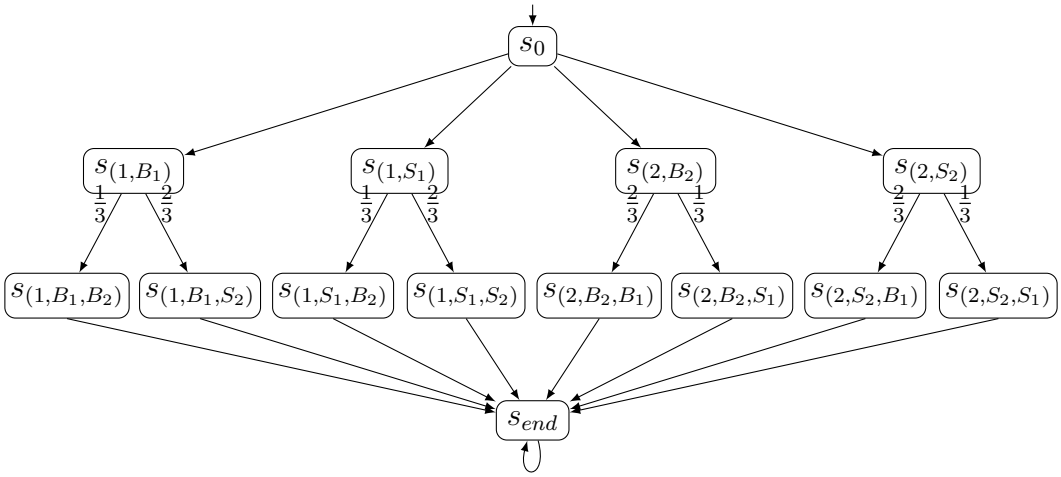


Figura 2.3: Cadena de Markov para el juego BoS y su equilibrio de Nash mixto

lo que nos confirma que el teorema 2.10 se cumple para este caso.

Para el caso general, dados un juego G y un perfil de estrategias mixtas α, la caracterización $NE_{G,\alpha}$ se define como:

$$NE_{G,\alpha} \stackrel{\text{def}}{=} \bigwedge_{i \in N} \exists x.\, \left(f_{supp(\alpha_i)} \wedge f_{\overline{supp}(\alpha_i)} \right)$$

$$f_{supp(\alpha_i)} \stackrel{\text{def}}{=} \bigwedge_{a_i \in supp(\alpha_i)} \mathcal{P}_{>0}[\mathcal{X}(a_i \wedge \mathcal{E}_{=x} end)]$$

$$f_{\overline{supp}(\alpha_i)} \stackrel{\text{def}}{=} \bigwedge_{a_i \in \overline{supp}(\alpha_i)} \mathcal{P}_{>0}[\mathcal{X}(a_i \wedge \mathcal{E}_{\leq x} end)]$$

donde $\overline{supp}(\alpha_i)$ denota al complemento de $supp(\alpha_i)$.

Finalmente, el siguiente teorema establece la relación entre estas fórmulas y el equilibrio de Nash (v. [GR09] para la demostración del teorema).

Teorema 2.11. *Sea M_G un modelo para el juego G y el perfil α. El perfil α es un equilibrio de Nash de G sii se cumple que $(M_G, s_0) \models NE_{G,\alpha}$.*

2.3. Conclusiones

En [vBar] puede consultarse con mayor detalle la relación histórica entre juegos y lógica, además de otros aspectos contemporáneos. Los textos

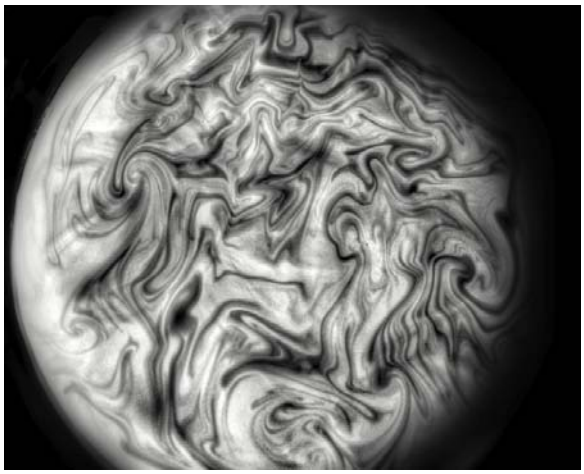
[vB05, vB12] contienen un resumen de trabajo reciente, abarcan varios aspectos de teoría de juegos, y hacen énfasis en presentar retos a la investigación y problemas abiertos. Otros trabajos no mencionados son [Ven05] y [TvdHW08]. En estos trabajos se hacen caracterizaciones similares utilizando lógica temporal y lógica de preferencias, respectivamente.

A pesar de su larga convivencia, la unión de teoría de juegos y lógica aún es fértil. La cantidad de trabajos recientes nos indican que aún existen muchos problemas por resolver. Por ejemplo, siguiendo la misma línea del capítulo, si las mismas técnicas y herramientas computacionales de la lógica pueden aplicarse a juegos. También vale la pena preguntarse si las herramientas desarrolladas para la teoría de juegos pueden llegar a aplicarse a otros problemas computacionales que surgen de la lógica. También, como apunta van Benthem [vB05] es alentador esperar nuevos problemas y retos, tal vez inexistentes en estas áreas por separado.

En los siguientes capítulos, estudiaremos con más detalle la caracterización por Góngora y Rosenblueth [GR09]. Además, presentamos dos métodos para la síntesis de equilibrios, uno utilizando lógica híbrida, y otro método que combina técnicas y algoritmos de verificación de modelos con algoritmos de rutas más cortas.

3

Caracterización de estrategias probabilísticas



– Pedro A. Gongora, *Turbulencia*, 2014

3.1. Introducción

Como un acercamiento a la teoría de la decisión para la escenarios multi-agente, la teoría de juegos está, sin duda, dentro de los intereses de las ciencias computacionales y de la inteligencia artificial. Algunos trabajos recientes han incorporado estos intereses en la literatura de la verificación de modelos, caracterizando diversas nociones de teoría de juegos en lógica temporal y la lógica dinámica (cf. [Bon01, HvdHMW03, vdHJW05]). Estos trabajos se centran en

juegos con estrategias puras, donde los agentes racionales actúan de forma determinista, guiados por sus funciones de utilidad. La atención se concentra en la caracterización de nociones tales como equilibrios de Nash, óptimos de Pareto, y estrategias dominantes/dominadas. En este capítulo, construyo sobre esta tradición incorporando acciones estocásticas, y enfocándome en la caracterización de equilibrios de Nash en estrategias mixtas, y para juegos estratégicos finitos. El contenido de este capítulo está basado en el artículo [GR09].

Otros trabajos previos incluyen, pero no se limitan a, las caracterizaciones de equilibrios de Nash. En [Bon01], Bonanno da una caracterización de las predicciones por inducción hacia atrás (es decir, equilibrios de Nash para juegos en forma extensiva) utilizando una lógica de tiempo ramificado. En [HvdHMW03], Harrenstein et al. proceden de una manera similar, pero utilizando lógica dinámica proposicional (PDL). Otro enfoque similar es el de van der Hoek et al [vdHJW05], donde los autores introducen la lógica de tiempo alternante ATL aumentada con un operador contrafáctico. Esta extensión a ATL permite expresar propiedades tales como "*si el jugador* 1 *se compromete con la estrategia a, entonces se sigue que* φ". Este razonamiento contrafáctico se utiliza para caracterizar equilibrios de Nash de juegos en forma estratégica. Otros trabajos hacen énfasis en diversas nociones de teoría de juegos, como el diseño de mecanismos automatizado (cf. [PW03, vdHRW07]). Sin embargo, ninguno de estos trabajos trata con estrategias mixtas. En [BFW06], Ballarini et al hacen un análisis cuantitativo de un juego de negociación, pero no proporcionan una caracterización de equilibrios de Nash. En [Jam08], Jamroga proporciona una caracterización de equilibrios de Nash utilizando una lógica temporal multi-valuada.

Parto de la lógica temporal probabilística PCTL [HJ94] aumentada con costos como el marco de trabajo subyacente y procedo como sigue. En primer lugar, presento una extensión de PCTL para cuantificar valores en fórmulas de costo esperado (por ejemplo, en $\mathcal{E}_{\bowtie x}[\varphi]$, x, puede cuantificarse de forma existencial o universal). A continuación, doy una codificación de juegos estratégicos finitos como cadenas de Markov de tiempo discreto. La codificación consiste en desplegar los resultados de un juego, bajo un perfil de estrategias mixtas, en una estructura arbórea que modela las posibilidades de acción para cada agente. Por último, doy una fórmula simple de la lógica extendida para caracterizar los equilibrios de Nash bajo esta codificación.

3.2. Juegos estratégicos

La teoría de juegos estudia la interacción entre agentes racionales. Aquí, la racionalidad está directamente relacionada con la maximización de utilidades. Un juego es simplemente una descripción formal de dicha interacción. Nos ocuparemos de juegos en los que los conjuntos de posibles acciones son las de los jugadores individuales, a veces llamados juegos no cooperativos. Por razones de brevedad, en este capítulo, me referiré a los juegos no cooperativos simplemente como juegos.

Entre las dos formalizaciones para juegos, juegos en forma estratégica y juegos en forma extensiva, voy a utilizar la primera. Existen varios conceptos de soluciones para juegos, de las que, sin duda, la más conocida es la de equilibrios de Nash. En términos generales, un equilibrio de Nash se caracteriza por las decisiones tomadas por todos los jugadores del juego, de manera tal que ningún jugador puede aumentar su utilidad tomado otra decisión, suponiendo que todos los demás jugadores se apegan a su decisión.

En esta sección presento algunas definiciones basadas en los primeros capítulos de [OR94], a donde refiero al lector para una discusión más detallada.

Definición 3.1 (Juego estratégico finito)**.** Un juego estratégico finito es una estructura:

$$G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$$

donde $N = \{1, \ldots, n\}$ es un conjunto finito de n agentes, A_i es un conjunto finito de las estrategias puras del agente i, $u_i : A \to \mathbb{R}$ es la función de utilidad del agente i, y $A = \times_{i \in N} A_i$ es el conjunto de todos los perfiles de estrategias puras de G.

Ejemplo 3.2 (Bach o Stravinsky)**.** Consideremos el juego conocido como *Bach o Stravinsky* (BoS) para los jugadores 1 y 2. Ambos jugadores quieren decidir a qué concierto ir, a uno de Bach o de Stravinsky. El jugador 1 prefiere Bach dos veces más que a Stravinsky, mientras que el jugador 2 prefiere a Stravinsky el doble que a Bach. Ambos jugadores prefieren un acuerdo a dejar de asistir a un concierto. Cada jugador toma su decisión de forma independiente, pero tomando en cuenta las preferencias del otro. Los juegos estratégicos finitos de dos jugadores pueden describirse usando matrices de utilidades. La matriz de la Figura 3.1 define las funciones de utilidad para BoS, e.g., $u_1(B_1, B_2) = 2$, $u_2(B_1, B_2) = 1$.

Utilizaremos las siguientes convenciones de notación. Utilizaremos las letras latinas a y a' para nombrar elementos del conjunto A de perfiles de estrategias. Si a es un perfil de estrategia, utilizaremos a_i para referirnos a la estrategia

	B_2	S_2
B_1	$2, 1$	$0, 0$
S_1	$0, 0$	$1, 2$

Figura 3.1: Matriz de utilidades para el juego estratégico BoS

del agente i especificada en A. También, con un poco de abuso de notación, denotamos como a_{-i} al perfil de que especifica las estrategias de todos los demás agentes excepto i. De esta forma, si $a_i \in A_i$, entonces $(a_{-i}, a_i) \in A$. También suponemos que los conjuntos A_i son disjuntos entre sí. Cuando es claro, identificamos a un perfil de estrategia $a \in A$ con otra n-tupla a' si y sólo si contienen exactamente los mismos elementos, independientemente del orden de los elementos de las tuplas.

Definición 3.3 (Mejor respuesta y equilibrio de Nash)**.** Dado un juego estratégico finito $G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$, decimos que la estrategia a_i es una mejor respuesta al perfil a sii $u_i(a_{-i}, a_i) \geq u_i(a_{-i}, a_i')$ para cada $a_i' \in A_i$. Decimos que el perfil a es un equilibrio de Nash de G sii toda estrategia a_i tal que $a = (a_{-i}, a_i)$ es una mejor respuesta al perfil a.

Consideremos la definición previa y la matriz de la Figura 3.1. Podemos fácilmente comprobar que los dos perfiles de estrategia (B_1, B_2) y (S_1, S_2), son equilibrios de Nash de BoS (Ejemplo 3.2).

Definición 3.4 (Extensión mixta de un juego)**.** Sea $\Delta(B)$ el conjunto de todas las distribuciones de probabilidad sobre el conjunto finito B. Para cualquier juego estratégico finito:

$$G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$$

definimos su extensión mixta como la estructura:

$$\widehat{G} = \langle N, \{\Delta(A_i)\}_{i \in N}, \{U_i\}_{i \in N} \rangle$$

donde $\Delta(A_i)$ es el conjunto de todas las estrategias mixtas del jugador i, $U_i : \widehat{A} \to \mathbb{R}$ es la esperanza matemática de la utilidad con respecto a la medida de probabilidad inducida por el perfil de estrategias mixtas. Finalmente, $\widehat{A} = \times_{i \in N} \Delta(A_i)$ es el conjunto de todos los perfiles de estrategias mixtas de $\widehat{G}$.

Usamos letras griegas α y α' para nombrar a los elementos $\widehat{A}$. Esto es, $\alpha = (\alpha_1, \ldots, \alpha_n)$ es un perfil de estrategias mixtas. Todas las convenciones

de notación para los juegos de estrategias puras se utilizan también para sus extensiones mixtas. Como α_i es una distribución de probabilidad sobre A_i, utilizamos $\alpha_i(a_i)$ para denotar a la probabilidad asignada por α_i al evento en el que se selecciona la estrategia pura a_i. Para una estrategia mixta α_i, el conjunto de elementos A_i a los que α_i asigna probabilidad mayor que 0 lo llamamos el *soporte* de α_i. Denotamos con $supp(\alpha_i)$ al subconjunto de A_i cuyos elementos se encuentran en el soporte de la estrategia mixta α_i. Decimos que *una estrategia mixta α_i degenera a una estrategia pura a_i* si y sólo si se asigna la probabilidad 1 al evento a_i (i.e., $\alpha_i(a_i) = 1$). Por último, decimos que el perfil de estrategias mixtas α es un equilibrio de Nash de un juego G si es un equilibrio de Nash de su extensión mixta $\widehat{G}$.

La utilidad esperada bajo algún perfil de estrategias mixta es el valor promedio de dicha utilidad. Para algún perfil de estrategias mixtas α y un jugador i la función de utilidad se determina por:

$$U_i(\alpha) = \sum_{a \in A} p_\alpha(a) u_i(a)$$
$$p_\alpha(a) = \prod_{j \in N} \alpha_j(a_j)$$

El siguiente teorema proporciona una caracterización útil de los equilibrios de Nash. Ver el Lema 33.2 en [OR94, p. 33] para una caracterización similar y una prueba para el caso de la dirección *si* de la implicación.

Teorema 3.5. *Dado cualquier juego estratégico finito $G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$, un perfil de estrategias mixtas $\alpha \in \widehat{A}$ es un equilibrio de Nash de G sii las siguientes dos condiciones se satisfacen para cada jugador $i \in N$:*

1. *Se satisface la igualdad $U_i(\alpha_{-i}, a_i) = U_i(\alpha_{-i}, a_i')$ para cada estrategia degenerada $a_i \in supp(\alpha_i)$.*

2. *Se satisface la desigualdad $U_i(\alpha) \geq U_i(\alpha_{-i}, a_i)$ para cada estrategia degenerada $a_i \in A_i - supp(\alpha_i)$.*

Demostración. Para la primera parte supongamos que la ecuación $U_i(\alpha_{-i}, a_i) = U_i(\alpha_{-i}, a_i')$ no se cumple para algún i. Entonces algún lado debe ser mayor que el otro, pero eso contradice la hipótesis de que α es un equilibrio de Nash, pues i puede incrementar su utilidad esperada asignando mayor probabilidad a la estrategia pura que incrementa su utilidad. La segunda parte se sigue de la definición de un equilibrio de Nash. El converso es directo: si ambas partes se cumplen para cada i, entonces es imposible incrementar la utilidad de algún

agente aumentando la probabilidad de alguna estrategia (ambas partes muestran la probabilidad 1 del peor caso para cada estrategia y cada agente), por lo tanto el perfil es una mejor respuesta a sí mismo. ∎

Consideremos de nuevo la matriz en la Figura 3.1. Podemos utilizar el Teorema 3.5 para verificar que el perfil de estrategias mixtas $\alpha = \left(\left(\frac{2}{3}, \frac{1}{3}\right), \left(\frac{1}{3}, \frac{2}{3}\right)\right)$ es un equilibrio de Nash para BoS. Por ejemplo, para el jugador 1, reemplazamos α_1 con una estrategia mixta degenerada que asigne probabilidad 1 a B_1 o a S_1, y comparar la utilidad esperada en ambos casos. Para $B_1 = (1, 0)$ y $S_1 = (0, 1)$ tenemos: $U_1\left(B_1, \left(\frac{1}{3}, \frac{2}{3}\right)\right) = U_1\left(S_1, \left(\frac{1}{3}, \frac{2}{3}\right)\right) = \frac{2}{3}$. Podemos seguir el mismo procedimiento para el jugador 2 y concluir que α es un equilibrio de Nash para BoS.

3.3. Cadenas de Markov y PCTL

Las fórmulas de PCTL describen propiedades cualitativas y cuantitativas de sistemas probabilísticos, a veces modelados con cadenas de Markov. Estas fórmulas tratan propiedades tales como "*la probabilidad de satisfacer p es por lo menos un medio*", o "*el costo esperado (o recompensa) de satisfacer p es a lo más* 10". Esta sección tiene el propósito de introducir las cadenas de Markov y PCTL. En primer lugar, introduzco las cadenas de Markov, que servirán de modelo semántico para las fórmulas de PCTL. A continuación, presento la sintaxis de PCTL y su satisfacción. Para más detalles sobre el material presentado en esta sección, referimos al lector al documento original [HJ94], así como al libro [BK08].

Definición 3.6 (Cadena de Markov de tiempo discreto)**.** Una cadena de Markov de tiempo discreto (DTMC) es una estructura:

$$M = \langle S, s_{init}, \mathbf{P}, \mathbf{C}, \mathit{AtProp}, \ell \rangle$$

donde S es un conjunto finito a cuyos elementos llamamos estados. s_{init} es un elemento distinguido de S que llamamos estado inicial, $\mathbf{P} : S \times S \to [0, 1]$ es una función de transiciones probabilística, tal que para cualquier estado $s \in S$, $\sum_{s' \in S} \mathbf{P}(s, s') = 1$, $\mathbf{C} : S \to [0, \infty)$ es una función de costos, AtProp es un conjunto contable de proposiciones atómicas, y $\ell : S \to 2^{\mathit{AtProp}}$ es una función de etiquetamiento que marca cada estado en S con un subconjunto de AtProp.

Por brevedad, en adelante llamamos simplemente "cadena de Markov" o "DTMC" a una cadena de Markov de tiempo discreto.

$Post_M(s) = \{s' \mid \mathbf{P}(s, s') > 0\}$ es el conjunto de estados que son posibles de visitar desde s en un sólo paso. Una *ruta* en una cadena de Markov M es una secuencia posiblemente infinita de estados $\pi = s_0 s_1 \cdots$ tal que para cualquier s_i y s_{i+1}, $\mathbf{P}(s_i, s_{i+1}) > 0$. Una ruta es finita si la secuencia es finita. Denotamos como $Paths_M$ al conjunto de todas las rutas infinitas de M, y con $Paths_M^{\text{fin}}$ al conjunto de todas las rutas finitas de M. Dada una ruta $\pi = s_0 s_1 \cdots s_i \cdots$, usamos $\pi[i] = s_i$ para referirnos al i-ésimo elemento de π, y $\pi[0, i]$ para referirnos al prefijo $s_0 \cdots s_i$ de π. El conjunto $Paths_M(s) = \{\pi \mid \pi \in Paths_M \text{ y } \pi[0] = s\}$ es el conjunto de todas las rutas infinitas de M que comienzan con s. Similarmente, el conjunto $Paths_M^{\text{fin}}(s) = \{\pi \mid \pi \in Paths_M^{\text{fin}} \text{ y } \pi[0] = s\}$ denota al conjunto de todas las rutas finitas de M que comienzan con s.

Para cualquier ruta finita π, el *conjunto cilindro* de π es el conjunto $Cyl(\pi) = \{\pi' \in Paths_M \mid \pi' \text{ tiene el prefijo } \pi\}$. La *medida de probabilidad* Pr_s asociada a una DTMC M y un estado s es la de la menor σ-álgebra Σ_s que contiene a todos los conjuntos cilindro $Cyl(\pi)$, para $\pi \in Paths_M^{\text{fin}}(s)$. Para las rutas finitas $\pi = s_0 \cdots s_n$, la probabilidad de π se define como $\mathbf{P}(\pi) = \prod_{i<n} \mathbf{P}(s_i, s_{i+1})$. La probabilidad de $Cyl(\pi)$ bajo Pr_s se determina por $Pr_s(Cyl(\pi)) = \mathbf{P}(\pi)$. Sea $\{C_i\}_{i \in I}$ una colección de conjuntos cilindro disjuntos entre si para un índice contable I. La probabilidad de la unión contable $\bigcup_{i \in I} C_i$ está determinada por $Pr_s\left(\bigcup_{i \in I} C_i\right) = \sum_{i \in I} Pr_s(C_i)$.

$\mathbf{C}(s)$ denota el costo (o recompensa, dependiendo del modelo en consideración) obtenido al *salir* del estado s. Entonces, para cualquier $\pi = s_0 \cdots s_n$ finita en $Paths_M^{\text{fin}}$, el *costo acumulado de* π se define por $Cost_M(\pi) = \sum_{0 \leq i < n} \mathbf{C}(s_i)$. Tenga en cuenta que el costo de dejar el último estado de una ruta no está en la sumatoria, y que para rutas que consistan en un solo estado s, $Cost_M(s) = 0$.

Para una ruta infinita $\pi \in Paths_M(s)$ y $A \subseteq S$, definimos el *costo acumulado de alcanzar un estado en* A como:

$$Cost_M(\pi, A) = \begin{cases} Cost_M(\pi[0, n]) & \text{si } \exists n \geq 0 : \pi[n] \in A \wedge \forall 0 \leq i < n : \pi[i] \notin A \\ \infty & \text{en cualquier otro caso} \end{cases}$$

Para cierto estado s y $A \subseteq S$, $\{s \models \mathcal{F} A\}$ denota al conjunto de todos las rutas finitas $\pi = s_0 \cdots s_n$, tal que $s_0 = s$, $s_n \in A$ y $\forall 0 \leq i < n : s_i \notin A$. Tenga en cuenta que el conjunto $\{s \models \mathcal{F} A\}$ es medible, por lo tanto, $Pr_s(\{s \models \mathcal{F} A\})$ es la *probabilidad de alcanzar un estado en* A *desde el estado* s. Definimos ahora el *costo acumulado esperado de llegar a algún estado en* A *desde* s como:

$$ExpCost_M(s, A) = \begin{cases} \sum_{\pi \in \{s \models \mathcal{F} A\}} \mathbf{P}(\pi) Cost_M(\pi) & \text{si } Pr_s(\{s \models \mathcal{F} A\}) = 1 \\ \infty & \text{en cualquier otro caso} \end{cases}$$

Teniendo los conceptos principales de cadenas de Markov, ahora presento la sintaxis de la lógica PCTL.

Definición 3.7 (Fórmulas de PCTL)**.** El conjunto de las fórmulas φ de PCTL para algún conjunto contable *AtProp* de proposiciones atómicas se define de acuerdo con la siguiente gramática BNF:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{P}_{\bowtie a}[\tau] \mid \mathcal{E}_{\bowtie c}[\varphi]$$
$$\tau ::= \mathcal{X}\varphi \mid \varphi \mathcal{U} \varphi$$

donde $p \in \mathit{AtProp}$, $a \in [0,1]$, $c \in [0,\infty)$ y $\bowtie \in \{<, >, \leq, \geq\}$.

Las fórmulas de PCTL describen propiedades de los cómputos infinitos de un sistema probabilístico. Podemos estudiar dos tipos de fórmulas: las fórmulas de rutas o temporales y las fórmulas de estados. Las fórmulas de rutas heredan su significado de LTL. $\mathcal{X}\varphi$ se satisface en rutas donde el siguiente estado satisface φ. $\varphi\mathcal{U}\psi$ se satisface en rutas en las que existe un estado futuro o actual que satisface ψ, mientras que todos los estados anteriores satisfacen φ. Las fórmulas de estados heredan sus significados de CTL. La fórmula $\top$ se satisface en todas las cadenas de Markov en todos sus estados. La fórmula $\neg\varphi$, para la negación, y $(\varphi \wedge \psi)$, para conjunción, tienen sus significados habituales. Los cuantificadores de trayectorias de CTL son reemplazados por el operador $\mathcal{P}$. Una fórmula $\mathcal{P}_{\bowtie a}[\tau]$ especifica que la probabilidad de que la fórmula temporal τ se satisfaga es $\bowtie a$. $\mathcal{E}_{\bowtie c}[\varphi]$ se satisface en los estados donde el costo esperado de llegar a un estado que satisfaga φ es $\bowtie c$.

Los otros conectivos de la lógica proposicional se definen como de costumbre:

$$\begin{aligned}
\bot &= \neg\top \\
(\varphi \vee \psi) &= \neg(\neg\varphi \wedge \neg\psi) \\
(\varphi \to \psi) &= (\neg\varphi \vee \psi) \\
(\varphi \leftrightarrow \psi) &= ((\varphi \to \psi) \wedge (\psi \to \varphi))
\end{aligned}$$

donde $\bot$ no se satisface en ninguna DTMC en ningún estado, $(\varphi \vee \psi)$ es una disyunción, $(\varphi \to \psi)$ es una implicación $(\varphi \leftrightarrow \psi)$ es un bicondicional.

También definimos las siguientes fórmulas derivadas:

$$\begin{aligned}
\mathcal{P}_{\bowtie a}[\mathcal{F}\varphi] &= \mathcal{P}_{\bowtie a}[\top \mathcal{U} \varphi] \\
\mathcal{P}_{\bowtie a}[\mathcal{G}\varphi] &= \mathcal{P}_{\overline{\bowtie} 1-a}[\mathcal{F}\neg\varphi] \\
\mathcal{P}_{=a}[\tau] &= (\mathcal{P}_{\geq a}[\tau] \wedge \mathcal{P}_{\leq a}[\tau]) \\
\mathcal{E}_{=a}[\varphi] &= (\mathcal{E}_{\geq a}[\varphi] \wedge \mathcal{E}_{\leq a}[\varphi])
\end{aligned}$$

donde $\overline{<} = >$, $\overline{>} = <$, $\overline{\leq} = \geq$ and $\overline{\geq} = \leq$. Las fórmulas de rutas también heredan sus significados de LTL. $\mathcal{F}\varphi$ se satisface en rutas donde existe un

estado presente o futuro que satisface φ. $\mathcal{G}\varphi$ se satisface en rutas donde φ se satisface en todos los estados presentes y futuros de la ruta.

Definición 3.8 (Satisfacción en PCTL)**.** Sea $M = \langle S, s_{init}, \mathbf{P}, \mathbf{C}, AtProp, \ell\rangle$ una cadena de Markov. La relación de satisfacción $\models$ entre parejas (M, s) con $s \in S$ y fórmulas con proposiciones atómicas en $AtProp$ se define como la menor relación tal que:

$$\begin{aligned}
&(M,s) \models \top \\
&(M,s) \models p && \Leftrightarrow p \in \ell(s)\ (p \in AtProp) \\
&(M,s) \models \neg\varphi && \Leftrightarrow (M,s) \not\models \varphi \\
&(M,s) \models (\varphi \wedge \psi) && \Leftrightarrow (M,s) \models \varphi \text{ y } (M,s) \models \psi \\
&(M,s) \models \mathcal{P}_{\bowtie a}[\tau] && \Leftrightarrow p_s(\tau) \bowtie a \\
&(M,s) \models \mathcal{E}_{\bowtie c}[\varphi] && \Leftrightarrow e_s(\varphi) \bowtie c
\end{aligned}$$

donde las funciones $p_s(\tau)$ and $e_s(\varphi)$ son como sigue:

$$\begin{aligned}
p_s(\tau) &= Pr_s(\{\pi \in Paths_M(s) \mid \pi \models \tau\}) \\
e_s(\varphi) &= ExpCost_M(s, \{s' \mid (M,s') \models \varphi\})
\end{aligned}$$

Pr_s es la medida de probabilidad descrita anteriormente y la relación $\models$ entre parejas en $Paths_M$ y fórmulas temporales se define como:

$$\begin{aligned}
&\pi \models \mathcal{X}\varphi \ \Leftrightarrow \pi[1] \models \varphi \\
&\pi \models \varphi\mathcal{U}\psi \Leftrightarrow \exists n \geq 0 : \forall i < n : \pi[i] \models \varphi \wedge \pi[n] \models \psi
\end{aligned}$$

Si existe algún φ tal que $(M, s_{init}) \models \varphi$, entonces decimos que φ se satisface inicialmente, y escribimos $M \models \varphi$.

Tomemos en cuenta que el conjunto $\{\pi \in Paths_M(s) \mid \pi \models \tau\}$ es un conjunto medible. El caso $\tau = \mathcal{X}\varphi$ es directo. Cuando $\tau = \varphi\mathcal{U}\psi$, el conjunto coincide con la unión numerable de conjuntos de cilindro $Cyl(\pi')$, para el prefijo finito π' de π tal que sólo su último estado s_n satisface ψ, y todos sus estados s_i anteriores satisfacen φ.

Ejemplo 3.9 (Protocolo simple)**.** Sea M la cadena de Markov de la Figura 3.2. Este ejemplo modela un protocolo simple para enviar mensajes a través de un canal no confiable [BK08]. Después de enviar un mensaje, una falla puede ocurrir con probabilidad 0.1. En tal caso, el protocolo sólo dicta que se vuelva a intentar. Consideremos lo siguiente:

- Hay un número infinito de rutas posibles desde el estado inicial s_0 hacia el estado s_3 (que representa la recepción exitosa del mensaje). Por ejemplo:

$$\pi_0 = s_0 s_1 s_3$$
$$\pi_1 = s_0 s_1 s_2 s_0 s_1 s_3$$
$$\pi_2 = s_0 s_1 s_2 s_0 s_1 s_2 s_0 s_1 s_3$$

- Cada prefijo abarca un conjunto cilindro. Por lo tanto es posible medir su probabilidad. Por ejemplo:

$$Pr_{s_0}(Cyl(\pi_0)) = 0.9$$
$$Pr_{s_0}(Cyl(\pi_1)) = 0.1 \cdot 0.9$$
$$Pr_{s_0}(Cyl(\pi_2)) = (0.1)^2 \cdot 0.9$$

- La probabilidad de que se reciba el mensaje, $p_{s_0}(\mathcal{F}\,delivered)$, es la suma (infinita) de las probabilidades de cada cilindro:

$$p_{s_0}(\mathcal{F}\,delivered) = 0.9 + (0.1)^1 \cdot 0.9 + (0.1)^2 \cdot 0.9 + \cdots = 1$$

  Este hecho se expresa en PCTL como sigue:

$$(M, s_0) \models \mathcal{P}_{=1}[\mathcal{F}\,delivered]$$

- Para este modelo, el costo acumulado de una ruta donde el mensaje se recibe eventualmente (i.e., la ruta alcanza al estado s_3) cuenta el número de intentos. Por ejemplo:

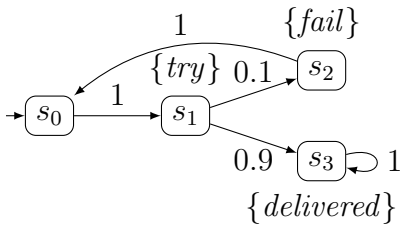
$$Cost_M(\pi_0) = 1$$
$$Cost_M(\pi_1) = 2$$
$$Cost_M(\pi_2) = 3$$

- El costo esperado de alcanzar s_3 se calcula con la siguiente suma:

$$e_{s_0}(delivered) = 1 \cdot 0.9 + 2 \cdot (0.1)^1 \cdot 0.9 + 3 \cdot (0.1)^2 \cdot 0.9 + \cdots = 1\frac{1}{9}$$

  Este valor representa el número promedio de intentos por enviar el mensaje. También, este hecho se expresa en PCTL como:

$$(M, s_0) \models \mathcal{E}_{=1\frac{1}{9}}[delivered]$$



s	$\mathbf{C}(s)$
s_0	1
s_1	0
s_2	0
s_3	0

Figura 3.2: Enviando un mensaje a través de un canal no confiable

Si reducimos una cadena de Markov a una estructura de Kripke, la fórmula de PCTL $\mathcal{P}_{>0}[\tau]$ es equivalente a la fórmula $\exists\tau$ de CTL. Por el contrario, la fórmula de PCTL $\mathcal{P}_{=1}[\tau]$ *no es equivalente* a la fórmula de CTL $\forall\tau$. Veamos el ejemplo anterior: hay un camino infinito que no llega nunca a un estado *delivered*, aún cuando satisface $\mathcal{P}_{=1}[\mathcal{F}\,delivered]$.

Dada una DTMC M, un estado s de M y una fórmula φ de PCTL, el problema de decidir si $(M, s) \models \varphi$ se llama el problema de verificación de modelos PCTL. El algoritmo básico para resolver el problema de verificación de modelos consiste en calcular el conjunto $Sat(\varphi) = \{s \in S \mid (M, s) \models \varphi\}$ de forma recursiva. El cálculo de *Sat* para las fórmulas atómicas está dado por la función de etiquetamiento ℓ. Sólo necesitamos operaciones básicas de conjuntos para computar *Sat* para las fórmulas con conectivos lógicos básicos. El cómputo de *Sat* para las fórmulas $\mathcal{P}_{\bowtie}[\tau]$ y $\mathcal{E}_{\bowtie}[\varphi]$ consiste en el cálculo de probabilidades de alcanzabilidad y de costos esperados para todos los estados. Estas tareas se pueden reducir al problema de encontrar una solución a un sistema de ecuaciones lineales. Para una explicación detallada de estos algoritmos remitimos al lector a [HJ94, BK08].

3.4. Un cuantificador de costos para PCTL

En esta sección, presento el lenguaje de PCTL con costos cuantificados (CQ-PCTL). CQ-PCTL extiende a su antecesor con la posibilidad de cuantificar los valores del operador de costo esperado. El algoritmo de verificación de modelos, sin embargo, se limita a las fórmulas que satisfacen una restricción sintáctica: *las variables cuantificadas no se pueden anidar*. Primero vamos a definir la sintaxis del lenguaje modificado, seguido definimos el algoritmo para la verificación de modelos.

La sintaxis de CQ-PCTL es casi la misma que la de PCTL. Modificamos la definición de fórmulas de costo esperado y añadimos una cláusula adicional

a la gramática que define la sintaxis de las fórmulas PCTL.

Definición 3.10 (Fórmulas de CQ-PCTL)**.** Para algún conjunto contable de proposiciones atómicas *AtProp* o algún conjunto contable Var de nombres de variable, el conjunto de las fórmulas φ de CQ-PCTL se define como el conjunto generado por la siguiente gramática BNF:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathcal{P}_{\bowtie a}[\tau] \mid \mathcal{E}_{\bowtie c}[\varphi] \mid \exists x.\varphi$$
$$\tau ::= \mathcal{X}\varphi \mid \varphi \mathcal{U} \varphi$$

donde $p \in AtProp$, $a \in [0,1]$, $c \in ([0,\infty) \cup Var)$, $x \in Var$ y $\bowtie \in \{<, >, \leq, \geq\}$.

De la sintaxis básica podemos derivar el cuantificador universal:

$$\forall x.\varphi = \neg\exists x.\neg\varphi$$

Además, decimos que *una variable x ocurre libre en φ* si x no ocurre en el ámbito de un cuantificador existencial o universal; de lo contrario decimos que x está *acotada*. Para una fórmula φ, decimos que no tiene *variables anidadas* si para cualquier subfórmula $\mathcal{E}_{\bowtie x}[\psi]$ de φ: (i) el conjunto de variables libres de ψ contiene a lo sumo a x, y (ii) el conjunto de variables acotadas de ψ está vacío. Una fórmula sin variables libres se llama un *enunciado*.

Observación 1 *En el resto del capítulo supondremos que las fórmulas son enunciados sin variables acotadas.*

Definimos ahora la relación de satisfacción para CQ-PCTL. Debido a la observación 1, es suficiente con incorporar una nueva cláusula a la definición de satisfacción de PCTL para tratar con las nuevas fórmulas existenciales.

Definición 3.11 (Satisfacción en CQ-PCTL)**.** La relación de satisfacción para las nuevas fórmulas se define como sigue:

$$(M,s) \models \exists x.\varphi \Leftrightarrow \text{existe } c \in [0,\infty) \text{ tal que } (M,s) \models \varphi[x := c]$$

donde $\varphi[x := c]$ es la abreviatura que reemplaza todas las ocurrencias libres de la variable x en φ con el real no negativo c. La satisfacción para el resto de las fórmulas se define como en PCTL.

Antes de presentar el algoritmo de verificación de modelos para CQ-PCTL, es necesario introducir una transformación de las subfórmulas de $\exists x.\varphi$ con el fin de eliminar las subfórmulas negativas. Esto lo hacemos mediante transformando la fórmula a su Forma Normal Positiva (PNF) [BK08].

Definición 3.12 (Forma Normal Positiva)**.** Una fórmula está en Forma Normal Positiva (PNF, por su siglas en inglés) si el operador de negación $\neg$ aparece a lo más al frente de proposiciones atómicas o la constante $\top$.

Tomemos en cuenta que es posible transformar cualquier fórmula a otra fórmula equivalente en PNF. Esto puede hacerse (i) introduciendo la constante $\bot$, la disyunción, y el cuantificador universal en la sintaxis básica (ver las definiciones en la subsección anterior); (ii) aplicando las leyes de doble negación de De Morgan; y (iii) aplicando de las siguientes equivalencias adicionales:

$$\neg\mathcal{P}_{\bowtie a}[\tau] \Leftrightarrow \mathcal{P}_{\neg\bowtie a}[\tau] \tag{3.1}$$

$$\neg\mathcal{E}_{\bowtie c}[\varphi] \Leftrightarrow \mathcal{E}_{\neg\bowtie c}[\varphi] \tag{3.2}$$

donde $\neg< = \geq$, $\neg> = \leq$, $\neg\leq = >$ and $\neg\geq = <$.[1] También, usamos $PNF(\varphi)$ para denotar la fórmula en PNF equivalente de φ.

El lector familiarizado con CTL pueden notar que el operador $\mathcal{R}$ ("*Release*", dual de $\mathcal{U}$) no está incluido en el lenguaje básico. El operador $\mathcal{R}$ es necesario para definir la Forma Normal Positiva de CTL, pero no para (CQ-) PCTL. La razón es que en (3.1) la negación se absorbe por el predicado $\bowtie a$. Dejando de lado el operador $\mathcal{R}$ no alteramos la expresividad (CQ-) PCTL en PNF. Sin embargo, el algoritmo de verificación de modelos de CQ-PCTL debe tomar en cuenta estas negaciones implícitas.

Para las fórmulas con la misma sintaxis, el algoritmo de verificación de modelos de CQ-PCTL es esencialmente el mismo que para PCTL. En el resto de esta sección sólo presento el método para calcular el conjunto $Sat(\exists x.\varphi)$ para las nuevas fórmulas cuantificadas.

El algoritmo para computar $Sat(\exists x.\varphi)$ consta de dos pasos. El primer paso calcula un conjunto $I(\exists x.\varphi)$ de intervalos. Estos intervalos son restricciones que un valor c asignado a x debe cumplir para que $\varphi[x := c]$ se satisfaga en algún estado de S. El segundo paso consiste en varios intentos de calcular $Sat(\varphi[x := c])$, donde cada intento utiliza un valor de c tomado de uno de los intervalos obtenidos previamente.

El cómputo de $I(\exists x.\varphi) = i(x, \varphi)$ de la Definición 3.13 construye un conjunto que contiene intervalos de números reales. Los valores c dentro de estos intervalos pueden hacer que $\varphi[x := c]$ se satisfaga. Aún más, este conjunto se construye de tal manera que si hay una c satisfactoria (i.e., $\varphi[x := c]$ es satisfactible en algún estado), entonces hay un intervalo A tal que $c \in A \in i(x, \varphi)$. En

[1] Observe que $\neg\bowtie$ niega a $\bowtie$, mientras la notación $\overline{\bowtie}$ de la Sección 3 inverte la dirección de $\bowtie$.

tal caso, también es importante que el intervalo contenga sólo valores satisfactorios (Teorema 3.15), por lo que basta con elegir sólo uno de los posiblemente infinitos valores en el intervalo.

Definición 3.13 (Conjunto $I(\exists x.\varphi)$)**.** Dada una a DTMC $M = \langle S, S_{init}, \mathbf{P}, \mathbf{C}, AtProp, \ell\rangle$ y una fórmula existencial de CQ-PCTL en PNF $\exists x.\varphi$, el conjunto $I(\exists x.\varphi) = i(x,\varphi)$ de intervalos de reales no negativos se construye inductivamente de acuerdo con la siguiente definición:

$$
\begin{aligned}
i(x,l) &= \{[0,\infty)\} \qquad (\text{donde } l \in AtProp \cup \{\top,\bot\})\\
i(x,\neg p) &= \{[0,\infty)\} \qquad (\text{donde } p \in AtProp)\\
i(x,(\psi \vee \psi')) &= i(x,\psi) \cup i(x,\psi')\\
i(x,(\psi \wedge \psi')) &= \{A \cap B \mid A \in i(x,\psi), B \in i(x,\psi')\}\\
i(x,\mathcal{E}_{\bowtie x}[\psi]) &= \{i(s,x,\mathcal{E}_{\bowtie x}[\psi]) \mid s \in S\}\\
i(x,\mathcal{E}_{\bowtie a}[\psi]) &= \|(i(x,\psi)) \cup \|(i(x,PNF(\neg\psi)))\\
i(x,\mathcal{P}_{\bowtie a}[\mathcal{X}\psi]) &= \|(i(x,\psi)) \cup \|(i(x,PNF(\neg\psi)))\\
i(x,\mathcal{P}_{\bowtie a}[\psi\mathcal{U}\psi']) &= \{A \cap B \mid A \in \|(i(x,\psi)), B \in \|(i(x,\psi')\})\\
&\quad \cup \{A \cap B \mid A \in \|(i(x,PNF(\neg\psi))), B \in \|(i(x,PNF(\neg\psi'))\})
\end{aligned}
$$

donde $i(s,x,\mathcal{E}_{\bowtie x}[\psi]) = \{r \in [0,\infty) \mid e_s(\psi) \bowtie r\}$ y $\|(\mathcal{I}) = \left\{\bigcap X \mid X \in 2^{\mathcal{I}}\right\}$ para el conjunto de intervalos $\mathcal{I}$ y donde por definición $\bigcap \varnothing = [0,\infty)$.

El conjunto $i(x,\varphi)$ se construye inductivamente. En la base de la inducción están los átomos y las fórmulas $\mathcal{E}_{\bowtie x}\psi$. Los átomos no establecen restricciones sobre los valores asignables a x. Para $\mathcal{E}_{\bowtie x}\psi$, el computo de las cotas requeridas para los intervalos es directo usando el algoritmo de verificación de modelos de PCTL. Para las disyunciones, el conjunto intervalo puede estar en la unión de los conjuntos computados para ambos disyuntos. El caso de conjunción es más complicado: si hay una c satisfactoria, entonces c debe estar a la vez en un intervalo calculado para cada uno de los conyuntos. Para las fórmulas $\mathcal{E}_{\bowtie a}[\psi]$ (resp. $\mathcal{P}_{\bowtie a}[\tau]$), hacemos un razonamiento similar al de las conjunciones. Si hay un c tal que $\psi[x := c]$ (resp. $\tau[x := c]$) satisface el predicado dado en cada estado de algún subconjunto de S, entonces c puede estar contenido en varios de los intervalos computados para las subfórmulas inmediatas de ψ (resp. τ).

Tengamos en cuenta que, debido a que (3.1) y (3.2), las fórmulas $\mathcal{E}_{\bowtie a}[\psi]$ y $\mathcal{P}_{\bowtie a}[\tau]$ pueden presentar una negación implícita contenida en su predicado $\bowtie a$. Por esta razón, el algoritmo debe buscar valores que pueden satisfacer las rutas complementarias cuando las subfórmulas inmediatas están negadas.

Además, observemos que la intersección $\bigcap \varnothing$ de la Definición 3.13 no es la misma que $\varnothing \cap \varnothing$. Por una parte, $\bigcap \varnothing$ representa una restricción impuesta por

el conjunto de estados vacío. Por la otra parte, $\varnothing \cap \varnothing$ es la intersección del conjunto vacío consigo mismo.

Ejemplo 3.14 (Cómputo de intervalos)**.** Sea M la DTMC mostrada en el panel izquierdo de la Figura 3.3. Usando el algoritmo de verificación de modelos de PCTL es posible computar los costos esperados de alcanzar s_3 y s_4 (caracterizados por los átomos p y q, resp.). Estos valores se muestran en la tabla en el panel derecho de la misma figura. Consideremos los siguientes ejemplos:

- Dados los valores en la tabla podemos computar los siguientes conjuntos:

$$i(x, \mathcal{E}_{\geq x}[p]) = \{[0, 10], [0, 5], [0, 0], [0, \infty)\}$$
$$i(x, \mathcal{E}_{\leq x}[q]) = \{[20, \infty), [15, \infty), [10, \infty), [0, \infty)\}$$

  Los intervalos en estos conjuntos dan una solución directa a la pregunta de si $(M, s) \models \exists x.\mathcal{E}_{\geq x}[p]$ o $(M, s) \models \exists x.\mathcal{E}_{\leq x}[q]$ se cumplen para algún s. Por ejemplo, sustituyendo x por 0, que pertenece a todos los intervalos, podemos verificar que se satisfacen ambas fórmulas en todos los estados, pues *existe* una x que hace que se cumplan las desigualdades.

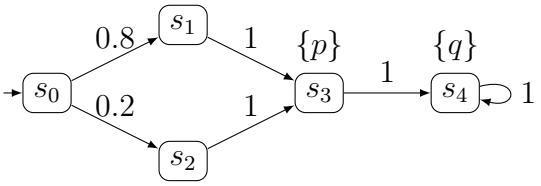
- Sea $\varphi = \mathcal{P}_{=0.2}[\mathcal{E}_{\geq x}[p]\mathcal{U}\mathcal{E}_{\leq x}[q]]$ (por simplicidad, supongamos que $\mathcal{P}_{=a}[\tau]$[2] es una primitiva del lenguaje base). El único valor asignable a x que hace que se cumpla $(M, s_0) \models \exists x.\varphi$ es 10. Este valor hace que la ruta $\pi = s_0 s_2 s_3 \ldots$ satisfaga la subfórmula *until*. Los primeros dos estados de π satisfacen la parte izquierda del *until*, mientras que el tercer estado de π satisface la parte derecha. El valor satisfactorio se computa como la intersección $[10, 10] = [0, 10] \cap [10, \infty)$ donde:

$$[0, 10] \in \|(i(x, \mathcal{E}_{\geq x}[p])) \quad (\text{por } s_0 \text{ y } s_2)$$
$$[10, \infty) \in \|(i(x, \mathcal{E}_{\leq x}[q])) \quad (\text{por } s_3)$$

- Consideremos el problema de si se cumple $(M, s_0) \models \exists x.\neg\varphi$. Convirtiendo a PNF tenemos: $PNF(\neg\varphi) = \mathcal{P}_{\neq 0.2}[\mathcal{E}_{\geq x}[p]\mathcal{U}\mathcal{E}_{\leq x}[q]]$. La solución anterior hace que la probabilidad de la ruta satisfactoria sea igual a 0.2, pero el problema requiere otros valores complementarios de x. Siguiendo este razonamiento, el algoritmo trata de calcular intervalos para las rutas complementarias que satisfagan $\mathcal{E}_{<x}[p]\mathcal{U}\mathcal{E}_{>x}[q]$. Una solución es el intervalo $(20, \infty) = [0, \infty) \cap (20, \infty)$ tal que:

$$[0, \infty) \in \|(i(x, \mathcal{E}_{<x}[p])) \quad (\text{por el conjunto vacío})$$
$$(20, \infty) \in \|(i(x, \mathcal{E}_{>x}[q])) \quad (\text{por } s_0)$$

[2]Podemos definir $\mathcal{P}_{=a}[\tau] \stackrel{\text{def}}{=} \mathcal{P}_{\geq a}[\tau] \wedge \mathcal{P}_{\leq a}[\tau]$



s	$\mathbf{C}(s)$	$e_s(p)$	$e_s(q)$
s_0	4	10	20
s_1	5	5	15
s_2	10	10	20
s_3	11	0	10
s_4	0	∞	0

Figura 3.3: DTMC para el Ejemplo 3.14 y algunos costos esperados

Un valor c dentro de este intervalo nunca satisface el until original, luego $p_{s_0}(\mathcal{E}_{\geq c}[p]\mathcal{U}\mathcal{E}_{\leq c}[q]) = 0 \neq 0.2$, lo que satisface el predicado.

El siguiente teorema establece la propiedad necesaria para usar el conjunto $I(\exists x.\varphi)$ en el algoritmo de verificación de modelos

Teorema 3.15. *Sean M una cadena de Markov, s un estado de M, y $\exists x.\varphi$ una fórmula de CQ-PCTL en PNF. Entonces, para todo $c \in [0, \infty)$ se cumplen las siguientes dos condiciones:*

1. *si $(M, s) \models \varphi[x := c]$, entonces existe $A \in i(x, \varphi)$ tal que $c \in A$ y para todo $c' \in A$, $(M, s) \models \varphi[x := c']$*

2. *Si $(M, s) \not\models \varphi[x := c]$, entonces existe $A \in i(x, PNF(\neg\varphi))$ tal que $c \in A$ y para todo $c' \in A$, $(M, s) \models PNF(\neg\varphi)[x := c']$.*

Demostración. Sólo se muestra el caso cuando x ocurre en φ. La prueba es por inducción sobre φ.

- Caso $\varphi = \psi \vee \psi'$. Condición (1): por hipótesis de inducción el intervalo requerido A está en $i(x, \psi) \cup i(x, \psi')$. Condición (2): por la hipótesis de inducción tenemos los intervalos correspondientes $A \in i(x, PNF(\neg\psi))$ y $B \in i(x, PNF(\neg\psi'))$. Por lo tanto el intervalo requerido $A \cap B$ está en $i(x, PNF(\neg\psi) \wedge PNF(\neg\psi'))$.

- Caso $\psi \wedge \psi'$. Condición (1): por la hipótesis de inducción tenemos los intervalos correspondientes $A \in i(x, \psi)$ y $B \in i(x, \psi')$. Por lo tanto el intervalo requerido $A \cap B$ está en $i(x, (\psi \wedge \psi'))$. Condición (2): por la hipótesis de inducción tenemos los intervalos correspondientes $A \in i(x, PNF(\neg\psi))$ y $B \in i(x, PNF(\neg\psi'))$. Por lo tanto el intervalo requerido está en $i(x, PNF(\neg\psi)) \cup i(x, PNF(\neg\psi'))$.

- Caso $\mathcal{E}_{\bowtie x}[\psi]$. Condición (1): directo por definición. Condición (2): también por definición y la equivalencia $\neg\mathcal{E}_{\bowtie x} \Leftrightarrow \mathcal{E}_{\neg\bowtie x}$.

- Caso $\mathcal{E}_{\bowtie a}[\psi]$ $(a \neq x)$. Condición (1): hay dos subcasos: (a) $e_s(\psi) \in [0,\infty)$ y (b) $e_s(\psi) = \infty$. (a) Hay una ruta de s a un estado en el conjunto no vacío $Sat(\psi[x := c])$. Por la hipótesis de inducción, para cada estado s_j en $Sat(\psi[x := c])$ existe un intervalo correspondiente A_j. Entonces el intervalo requerido para $\mathcal{E}_{\bowtie a}[\psi]$ debe ser la intersección de algunos intervalos A_j (contenidos en $\|(i(x,\psi)))$. (b) El conjunto $Sat(\neg\psi[x := c])$ es no vacío. De nuevo por la hipótesis de inducción, para cada $s_j \in Sat(\neg\psi[x := c])$ hay un intervalo correspondiente A_j (contenido en $\|(i(x, PNF(\neg\psi))))$. Condición (2): se cumple por la equivalencia $\neg\mathcal{E}_{\bowtie a} \Leftrightarrow \mathcal{E}_{\neg\bowtie a}$.

- Caso $\mathcal{P}_{\bowtie a}[\mathcal{X}\psi]$. Condición (1): hay dos posibilidades: (a) $p_s(\mathcal{X}\psi[x := c]) \bowtie a$ se cumple cuando $\psi[x := c]$ es satisfactible en estados alcanzables desde s en un paso, y (b) $p_s(\mathcal{X}\psi) \bowtie a$ se cumple cuando $\psi[x := c]$ no es satisfactible en estados alcanzables desde s en un paso. Para (a) el intervalo requerido está en $\|(i(x,\psi))$. Para (b) el intervalo requerido está en $\|(i(x, PNF(\neg\psi)))$. Condición (2): se cumple por la equivalencia $\neg\mathcal{P}_{\bowtie a} \Leftrightarrow \mathcal{P}_{\neg\bowtie a}$.

- Caso $\mathcal{P}_{\bowtie a}[\psi\mathcal{U}\psi']$. Condición (1): nuevamente, $p_s(\psi\mathcal{U}\psi') \bowtie a$ puede cumplirse cuando alguna de las subfórmulas es satisfactible o no. La primera posibilidad se incluye en $\{A \cap B \mid A \in \|(i(x,\psi)), B \in \|(i(x,\psi')\})$. La segunda, y complementaria, posibilidad está incluida en $\{A \cap B \mid A \in \|(i(x, PNF(\neg\psi))), B \in \|(i(x, PNF(\neg\psi'))\})$. Condición (2): se cumple por la equivalencia $\neg\mathcal{P}_{\bowtie a} \Leftrightarrow \mathcal{P}_{\neg\bowtie a}$.

∎

El Teorema 3.15 sugiere el último paso del algoritmo. Dada una fórmula $\exists x.\varphi$ de CQ-PCTL en PNF, construimos el conjunto de la siguiente manera:

$$Sat(\exists x.\varphi) = \bigcup_{A \in I(\exists x.\varphi)} \{Sat(\varphi[x := c]) \mid c \in A\}$$

Observemos que el Teorema 3.15 también implica que basta con elegir una única c de cada intervalo A.

El algoritmo básico que aquí presento se puede extender fácilmente al caso donde los valores de las fórmulas $\mathcal{P}_{\bowtie a}$ también aparecen cuantificadas.

Además, es posible extender los resultados de esta sección para el caso general de fórmulas, no sólo enunciados, enriqueciendo los modelos con funciones de interpretación de variables. Por claridad, sin embargo, nos restringimos a los enunciados. Algunas restricciones de anidamiento (Observación 1) pueden suavizarse, siempre y cuando no existan dependencias circulares entre las variables cuantificadas. Sin embargo, la restricción de anidamiento arbitrario es imposible quitarla. Por ejemplo, considere la siguiente frase simple:

$$\exists x.\,(\varphi[x] \wedge \exists y.(\psi[x,y]))$$

donde $\varphi[x]$ es una fórmula con ocurrencias libres de x y $\psi[x,y]$ es una fórmula con ocurrencias libres de tanto x como y. En este ejemplo, los problemas de encontrar los valores adecuados para x y y pueden ser mutuamente dependientes. Para niveles de anidamiento arbitrario el problema puede ser aún más complicado.

3.5. Verificando equilibrios de Nash en juegos

En esta sección, muestro cómo construir una cadena de Markov para un juego estratégico finito G y su estrategia mixta α. Aunque la construcción es para juegos en forma estratégica se basa la forma extensiva.

Los juegos en forma extensiva difieren de los juegos en forma estratégica en que la secuencialidad de las acciones es importante. Una juego extensivo puede describirse con una estructura de árbol. En un árbol de juego cada nodo representa el cambio de un solo jugador, y para cada acción posible, el árbol tiene un arco dirigido al turno del siguiente jugador. En un juego de estratégico, suponemos que cada agente realiza sus acciones independientemente de, y sin saber, las acciones de los otros jugadores. Para modelar esto en juego extensivo, los estados se agrupan de tal manera que representan la incertidumbre del siguiente jugador acerca de las acciones anteriores (ver la Figura 3.4 para la forma extensiva de BoS, las líneas punteadas agrupan los movimientos del jugador 1 como un sólo estado, pues el jugador 2 no sabe qué acciones se han realizado).

Dados el juego y el perfil en estrategias mixtas, construimos una estructura similar a un árbol del juego en forma extensiva. En la estructura construida cada arco, excepto los arcos que salen de la raíz, está etiquetada con la probabilidad de que el perfil de estrategias mixtas asigna a esa acción en particular. Como no podemos agrupar estados en una DTMC, construimos un subárbol para cada jugador y cada estrategia pura. Cada uno de estos subárboles mo-

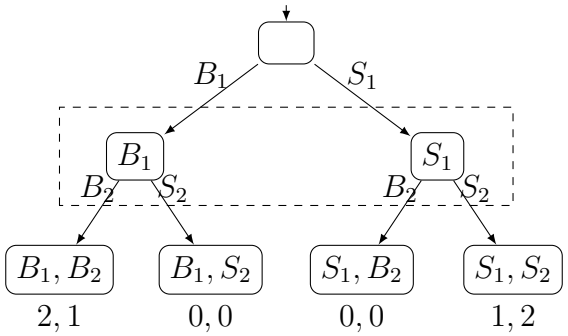


Figura 3.4: Forma extensiva de BoS; las utilidades se muestran debajo de las hojas

delan la situación donde el jugador i elige una cierta estrategia a_i, pero los otros jugadores siguen la estrategia mixta.

Procediendo de esta manera, cada nodo hoja corresponde a un perfil de estrategia del juego en forma estratégica. En consecuencia, cada nodo de la hoja se asocia con su utilidad a través de la función de costos $\mathbf{C}$. Como la función de costos modela el costo de *abandonar* el estado, tenemos que añadir un nodo ficticio absorbente debajo de las hojas, lo que representa el final del juego.

La Figura 3.5 ilustra uno de los subárboles descritos. Tengamos en cuenta que hay exactamente una ruta desde el estado final, y que pasa por cada perfil de estrategias. Los arcos de dicha ruta son las probabilidades asignadas por el perfil de estrategia mixta a esa acción. Por lo tanto, el costo esperado coincide con la utilidad esperada. Por tanto, podemos utilizar una fórmula de costo cuantificado para comparar los costos esperados y verificar si el Teorema 3.5 es aplicable.

Definición 3.16 (Modelo DTMC para un juego)**.** Para un juego:

$$G = \langle N, \{A_i\}_{i \in N}, \{u_i\}_{i \in N} \rangle$$

y un perfil de estrategias mixtas α de su extensión mixta $\widehat{G}$, definimos la DTMC $M_{G,\alpha}$ como la estructura:

$$M_{G,\alpha} = \langle S, s_{init}, \mathbf{P}, \mathbf{C}, AtProp, \ell \rangle$$

donde el conjunto de estados es:

$$S = \{s_{init}\} \cup \{s_{end}\} \cup \{s_x\}_{x \in Idx}$$

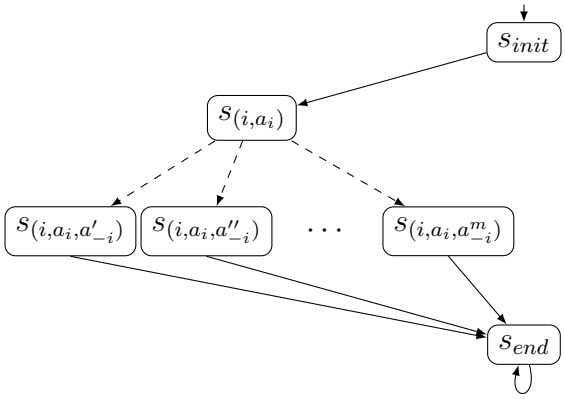


Figura 3.5: Después de que el jugador i elija la estrategia a_i los demás jugadores hacen su propia elección, creando así varios perfiles de estrategias

Idx es el siguiente conjunto de índices:

$$Idx = \bigcup_{\substack{i \in N \\ a_i \in A_i}} \{(i, a_i), (i, a_i, a_{j_1}), \ldots, (i, a_i, a_{j_1}, \ldots, a_{j_m}) \mid j_k \in N - \{i\}, j_k < j_{k+1}, \text{ y } (a_i, a_{j_1}, \ldots, a_{j_m}) \in A\}$$

La función de transiciones probabilística se define por casos:

$$\begin{aligned}
\mathbf{P}(s_{init}, s_{(i,a_i)}) &= 1/n && \text{para } i \in N, a_i \in A_i, n = |\bigcup_{j \in N} A_j| \\
\mathbf{P}(s_{(x)}, s_{(x,a_j)}) &= \alpha_j(a_j) && \text{para } j \in N, x \in Idx \\
\mathbf{P}(s_{(i,a)}, s_{end}) &= 1 && \text{para } i \in N, a \in A \\
\mathbf{P}(s_{end}, s_{end}) &= 1 \\
\mathbf{P}(s, s') &= 0 && \text{en cualquier otro caso}
\end{aligned}$$

La función de costos se define como:

$$\begin{aligned}
\mathbf{C}(s_{(i,a)}) &= u_i(a) && \text{para } a \in A \\
\mathbf{C}(s) &= 0 && \text{en cualquier otro caso}
\end{aligned}$$

Finalmente, el conjunto de proposiciones atómicas y la función de etiqueta-

miento son las siguientes:

$$\begin{aligned} AtProp &= \{end\} \cup \bigcup_{i \in N} A_i \\ \ell(s_{end}) &= \{end\} \\ \ell(s_{(i,a_i)}) &= \{a_i\} && \text{para } i \in N, a_i \in A_i \\ \ell(s) &= \varnothing && \text{en cualquier otro caso} \end{aligned}$$

Observación *La función de costos de una DTMC requere valores no negativos. Por lo tanto suponemos que las funciones de utilidad de los juegos sólo asignan valores no negativos. Si este no es el caso, es posible sumar una constante suficientemente grande a cada valor asignado por las funciones u_i para volverlas no negativas. Sumando tal constante no afecta ningún resultado, pues solamente comparamos los valores promedio de las utilidades.*

Ejemplo 3.17. (Modelo para BoS) El modelo DTMC M construido para el juego BoS y el perfil de estrategias mixtas $\alpha = \left(\left(\frac{2}{3}, \frac{1}{3}\right), \left(\frac{1}{3}, \frac{2}{3}\right)\right)$ se muestra en la Figura 3.6. Podemos verificar lo siguiente:

$$\begin{aligned} (M, s_{(1,B_1)}) &\models B_1 \wedge \mathcal{E}_{=\frac{2}{3}} end & (M, s_{(2,B_2)}) &\models B_2 \wedge \mathcal{E}_{=\frac{2}{3}} end \\ (M, s_{(1,S_1)}) &\models S_1 \wedge \mathcal{E}_{=\frac{2}{3}} end & (M, s_{(2,S_2)}) &\models S_2 \wedge \mathcal{E}_{=\frac{2}{3}} end \end{aligned}$$

Para cada jugador, todas las estrategias puras en el soporte de α producen la misma utilidad. Entonces, por el Teorema 3.5 α es un equilibrio de Nash. Podemos caracterizar este hecho con la siguiente fórmula de CQ-PCTL:

$$\begin{aligned} (M, s_{init}) \models \exists x. (\mathcal{P}_{>0}[\mathcal{X}(B_1 \wedge \mathcal{E}_{=x} end)] \wedge \mathcal{P}_{>0}[\mathcal{X}(S_1 \wedge \mathcal{E}_{=x} end)]) \\ \wedge \exists x. (\mathcal{P}_{>0}[\mathcal{X}(B_2 \wedge \mathcal{E}_{=x} end)] \wedge \mathcal{P}_{>0}[\mathcal{X}(S_2 \wedge \mathcal{E}_{=x} end)]) \end{aligned}$$

El ejemplo anterior muestra cómo es posible caracterizar un equilibrio de Nash en estrategias mixtas en un juego con CQ-PCTL. Aunque no es el caso en BoS, por el Teorema 3.5 debemos verificar que el costo esperado es efectivamente una mejor respuesta. Esto se logra verificando que el costo esperado de desviarse del perfil no supere al valor de las estrategias en el soporte. La siguiente definición de captura esta restricción.

Definición 3.18 (Caracterización de un equilibrio de Nash en estrategias mixtas)**.** Para un modelo DTMC de un juego $M_{G,\alpha}$, la caracterización en CQ-PCTL de un equilibrio de Nash en estrategias mixtas es la fórmula $NE_{G,\alpha}$

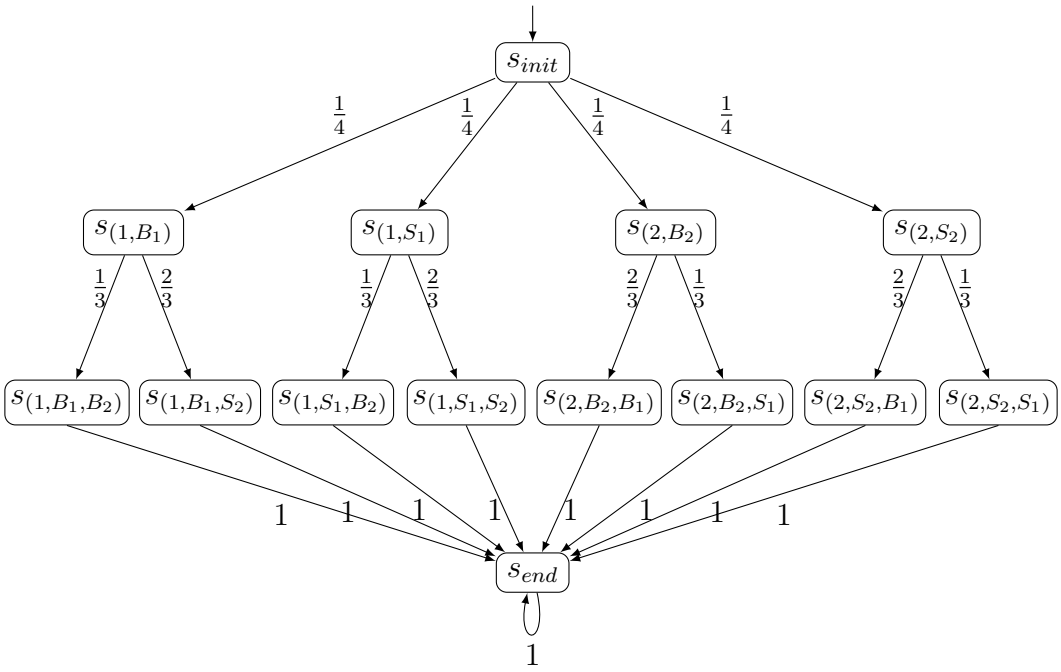


Figura 3.6: DTMC para BoS y su equilibrio de Nash mixto

definida como sigue:

$$NE_{G,\alpha} = \bigwedge_{i \in N} \exists x.\, \left(f_{supp(\alpha_i)} \wedge f_{\overline{supp}(\alpha_i)}\right)$$

$$f_{supp(\alpha_i)} = \bigwedge_{a_i \in supp(\alpha_i)} \mathcal{P}_{>0}[\mathcal{X}(a_i \wedge \mathcal{E}_{=x} end)]$$

$$f_{\overline{supp}(\alpha_i)} = \bigwedge_{a_i \in \overline{supp}(\alpha_i)} \mathcal{P}_{>0}[\mathcal{X}(a_i \wedge \mathcal{E}_{\leq x} end)]$$

donde $\overline{supp}(\alpha_i)$ denota el complemento de $supp(\alpha_i)$.

Finalmente, termino el capítulo con un lema y un teorema que muestran la corrección de esta construcción.

Lema 3.19. Sea $M_{G,\alpha}$ un modelo DTMC para un juego. Para cualquier jugador $i \in N$ y cualquier estrategia $a_i \in A_i$, se cumple la ecuación $U_i(a_i, \alpha_{-i}) = ExpCost_{M_{G,\alpha}}(s_{(i,a_i)}, s_{end})$.

Demostración. Sea $a = (a_i, a_{j_1}, \ldots, a_{j_m}) \in A$ un perfil tal que sus componentes siguen las restricciones del índice Idx. De la definición de S y de $\mathbf{P}$ tenemos que existe una única ruta $\pi = s_{(i,a_i)} s_{(i,a_i,a_{j_1})} \ldots s_{(i,a_i,a_{j_1},\ldots,a_{j_m})} s_{\{end\}}$. Para dicha ruta tenemos que:

$$\begin{aligned}
Pr_{s_{(i,a_i)}}(\pi) &= \mathbf{P}(\pi) \\
&= \mathbf{P}(s_{(i,a_i)}, s_{(i,a_i,a_{j_1})}) \cdots \mathbf{P}(s_{(i,a_i,a_{j_1},\ldots,a_{j_m})}, s_{end}) \\
&= \prod_{j \in N} \alpha_j(a_j) \\
&= p_\alpha(a) \\
Cost_{M_{G,\alpha}}(\pi) &= \mathbf{C}(s_{(i,a_i)}) + \cdots + \mathbf{C}(s_{(i,a)}) \\
&= u_i(a)
\end{aligned}$$

Más aún, el conjunto de todas esas rutas es igual a $P_{(i,a_i)} = \{s_{(i,a_i)} \models \mathcal{F}\{s_{end}\}\}$. Por lo tanto:

$$\begin{aligned}
ExpCost_{M_{G,\alpha}}(s_{(i,a_i)}, \{s_{end}\}) &= \sum_{\pi \in P_{(i,a_i)}} \mathbf{P}(\pi) Cost_{M_{G,\alpha}}(\pi) \\
&= \sum_{a \in A} p_\alpha(a) u_i(a) \\
&= U_i(\alpha)
\end{aligned}$$

∎

Teorema 3.20. *sea $M_{G,\alpha}$ un modelo DTMC de un juego. El perfil de estrategias mixtas α es un equilibrio de Nash de G si y sólo si se cumple que $M_{G,\alpha} \models NE_{G,\alpha}$.*

Demostración. Muestro sólo en una dirección (*si*); la prueba para el converso es similar. Supongamos como contradicción que el consecuente no se cumple. Por lo tanto, existe un jugador $i \in N$ para el cual $\exists x.\left(f_{supp(\alpha_i)} \wedge f_{\overline{supp}(\alpha_i)}\right)$ no se satisface inicialmente. Se sigue del Lema 3.19 que para cualquier $a_i \in A_i$, si $u = U_i(a_i, \alpha_{-i})$, entonces se cumple que $(M_{G,\alpha}, s_{(i,a_i)}) \models a_i \wedge \mathcal{E}_{=u}end$ (el primer conyunto por la definición de ℓ y el segundo por el Lema 3.19). Sea $c = U_i(a_i, \alpha_{-i})$ para algún $a_i \in supp(\alpha_i)$. Entonces, por el hecho anterior y el Teorema 3.5, las fórmulas $f_{supp(\alpha_i)}[x := c]$ and $f_{\overline{supp}(\alpha_i)}[x := c]$ son ambas satisfechas inicialmente. Una contradicción. ∎

3.6. Conclusiones

En este capítulo, abordamos el problema de la caracterización de un equilibrio de Nash en estrategias mixtas usando PCTL enriquecido con un cuantificador costo esperado: CQ-PCTL. En los trabajos anteriores [Bon01, HvdHMW03, vdHJW05], los autores dan caracterizaciones de equilibrios de Nash en estrategias puras y otras nociones de teoría de juegos usando lógica temporal y lógica dinámico. Este capítulo también difiere de [Bon01, HvdHMW03] en que su caracterización es se basa en el concepto de validez, mientras que la caracterización mostrada en este capítulo se basa en el concepto de satisfacción, por lo que este último enfoque es directamente aplicable a la verificación de modelos. En [BFW06], Ballarini incorpora acciones estocásticas. Ballarini proporciona un modelo para un juego de negociación (el juego *alternating offers negotiation protocol* de Rubinstein, ver [OR94]). Con este modelo, el autor utiliza fórmulas PCTL para hacer un análisis cuantitativo para varias estrategias mixtas del juego. El autor, sin embargo, no proporciona caracterizaciones de los equilibrios de Nash.

Hay dos rutas generales para futuras investigaciones: una relativa a CQ-PCTL y la otra respecto a otros conceptos de la teoría de juegos.

En cuanto a la primera ruta, recordemos que en la Sección 4 se presenta un algoritmo para la verificación de modelos de un fragmento de CQ-PCTL. El lenguaje completo incluye fórmulas con variables anidadas. Las variables anidadas introducen dependencias circulares que el algoritmo presentado no puede tratar. No sé si tal algoritmo existe. En cuanto a la complejidad de el algoritmo presentado, sé que es exponencial en el tamaño de la fórmula. Es importante para mejorar esta cota, si es posible.

También sería deseable, en el espíritu de este capítulo, contemplar otros conceptos de solución de juegos, tales como los equilibrios evolutivos y los equilibrios correlacionados (cf. [OR94]). Más allá de los juegos estratégicos finitos, sería interesante tratar con otras clases de juegos, como los Bayesianos y los juegos iterados. Por último, se necesita mayor investigación para determinar si las herramientas de verificación de modelos pueden usarse para calcular soluciones mixtas, además de la caracterizarlas.

4

Síntesis de estrategias con lógica híbrida

– Joseph Kosuth, *Three Color Sentence*, 1965

4.1. Introducción

Una parte de la investigación reciente sobre la combinación de la teoría de juegos y las lógicas modales se centra en temas fundacionales, tales como resultados de correspondencia entre marcos y soluciones de juego (ver [Bon01, HvdHMW03] y la reseña [vB12]). Sin embargo, aún es tema de investigación si la solución de problemas en la teoría de juegos, como el cómputo de equilibrios, puede beneficiarse de las técnicas y herramientas desarrolladas para las lógicas modales. La verificación de la validez formal requiere demostrar que una

propiedad se conserva en todos los posibles modelos. La verificación de modelos, por el contrario, es una técnica para verificar mecánica y eficientemente la satisfacción de un modelo en particular. Harrenstein et al. [HvdHMW03] caracterizan soluciones de juego como un caso especial de la confluencia modal en modelos arbóreos. La automatización de estas caracterizaciones depende verificar la validez formal, por lo que no pueden utilizarse directamente en un enfoque de verificación de modelos. La propuesta de este capítulo es utilizar extensiones de lógica híbrida [AtC07] como un puente entre estas caracterizaciones fundacionales y la verificación de modelos. Específicamente, el propósito de este enfoque es calcular la solución de un juego verificando las caracterizaciones basadas en validez de dicha solución, pero usando verificación de modelos.

Presento un ejemplo en la Lógica dinámica proposicional PDL con extensiones de lógica híbrida. El método que presento en este capítulo requiere la construcción de un modelo del juego que consiste en su estructura de árbol y las preferencias de los jugadores. Utilizo programas PDL para definir recursivamente la relación entre cada nodo no final y el mejor resultado alcanzable desde tales nodos. Esta relación es un caso especial de confluencia modal, y utilizo una fórmula con operadores de lógica híbrida para caracterizar los estados locales que satisfagan dicha propiedad de confluencia. Entonces, un verificador de modelos puede recorrer ese modelo y reconstruir las estrategias de una solución.

El contenido de este capítulo se basa en material aún no publicado fuera de ésta tesis.

4.2. Juegos en forma extensiva e inducción hacia atrás

Los juegos son descripciones formales de las interacciones de agentes racionales. Estas interacciones pueden formalizarse como un conjunto de secuencias de eventos (forma extensiva). La naturaleza del juego (por ejemplo, de conflicto o de acuerdo) se describe numéricamente, asignando unidades de utilidad para cada agente en cada posible resultado del juego. Aunque hay varias clases de juegos, en este capítulo nos ocuparemos únicamente de los juegos finitos, no cooperativos, con información perfecta, y en forma extensiva. Remito al lector a [OR94] para una introducción más completa a los conceptos que aparecen en esta sección.

Un juego en forma extensiva puede representarse como un árbol. Un nodo no final, o estado, representa el turno de un jugador, y tiene un sucesor para

cada acción posible disponible al jugador en ese momento. En los juegos finitos, las hojas representan los resultados posibles del juego. En un árbol de juego, la estrategia de un jugador i es una función que elige un único sucesor para cada nodo que represente el turno de i. Un perfil de la estrategias es un conjunto formado por una estrategia para cada jugador.

Definición 4.1 (Juegos finitos en forma extensiva, estrategias y perfiles de estrategias)**.** Un juego finito en forma extensiva G es un árbol definido como la tupla:

$$G \stackrel{\text{def}}{=} \langle S, \prec, s_0, N, P, \{u_i\}_{i \in N} \rangle$$

donde:

- $S \stackrel{\text{def}}{=} \{s_0, s_1, \ldots, s_n\}$ es el conjunto de los nodos del árbol, con $Z \subseteq S$ denotando al conjunto de las hojas;
- $\prec \subseteq (S \times S)$ ordena S como un árbol con raíz $s_0 \in S$;
- $N \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$ es el conjunto de los jugadores o agentes;
- $P : (S - Z) \to N$ es una función de turnos;
- $u_i : Z \to \mathbb{R}$ es la función de utilidad del jugador i.

Una estrategia a_i para el jugador i, mapea un elemento de $\{s \in S \mid P(s) = i\}$ con alguno de sus $\prec$-sucesores. A_i denota el conjunto de todas las estrategias del jugador i. Un perfil de estrategias es una tupla $a \in \times_{\{i \in N\}} A_i$. A denota el conjunto de todos los perfiles de estrategias.

Ejemplo 4.2. Consideremos el juego de la Figura. 4.1. Tenemos dos jugadores, 1 y 2. Los resultados de P se muestran debajo de los nodos no finales, y los resultados de las funciones u_i se muestran debajo de las hojas. El jugador 1 se mueve primero, entonces el jugador 2 se mueve y el juego termina. Supongamos que 1 se mueve a la derecha, y que después 2 también se mueve, a la derecha, entonces los jugadores obtendrán 4 y 2 unidades de utilidad, respectivamente. desde la raíz, este desenlace está determinado si todos los jugadores siguen el perfil de estrategias enfatizado con las flechas gruesas en la figura.

Una característica clave del enfoque de teoría de juegos a la toma de decisiones es que todos los jugadores se suponen racionales. La racionalidad de un jugador dicta que siempre elija las acciones que maximicen su propia utilidad. En consecuencia, una solución para un juego es un método para de elección de estrategias que maximice la utilidad de todos los jugadores. Un concepto

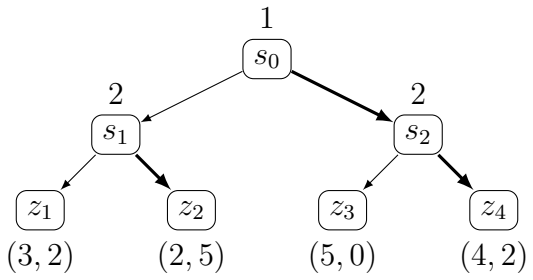


Figura 4.1: Juego de dos jugadores y su perfil de estrategias por inducción hacia atrás.

de la solución bien conocido es el de *equilibrios de Nash*. Un perfil de estrategias es un equilibrio de Nash si cada jugador no puede aumentar su utilidad desviándose unilateralmente del perfil.

Una solución por *inducción hacia atrás* es un caso especial de equilibrio de Nash: equilibrio de Nash perfecto por subjuegos. El algoritmo de inducción hacia atrás se desprende de la prueba inductiva por Kuhn que dice que cada juego finito en forma extensiva está determinado[1].

La inducción hacia atrás puede explicarse como el razonamiento contrafáctico que un jugador utiliza para la elección de sus acciones: "*Si yo elijo esta acción, a continuación, el siguiente jugador elegirá su mejor acción y mi recompensa será . . .*". Este razonamiento hipotético se aplica de forma recursiva hasta que se alcance un posible desenlace del juego.

Ejemplo 4.3. Consideremos nuevamente el juego de la Figura 4.1. El jugador 1 debe decidir entre s_1 y s_2. Si 1 elige s_1, entonces 2 seguramente elegirá z_2. Si 1 elige s_2, entonces 2 seguramente elegirá z_4. Luego, 1 elegirá s_2 para maximizar su ganancia. Las flechas gruesas muestran el resultado de esta inducción hacia atrás. También, observemos que este resultado puede identificarse con el conjunto $\{s_2, z_2, z_4\}$, cuyos elementos corresponden con las mejores decisiones tomadas en un nodo no final.

[1]Como lo presentamos aquí el resultado de Kuhn sólo aplica a juegos con información perfecta. Se puede consultar el artículo original [Kuh53].

4.3. Lógica dinámica proposicional con extensiones de lógica híbrida

La Lógica dinámica proposicional (PDL) es una extensión de la lógica multimodal. En PDL, podemos definir nuevas modalidades a partir de un conjunto base determinado. Definimos estas nuevas modalidades, llamadas *programas*, por medio de un lenguaje apropiado. En este capítulo, trabajaremos con PDL con *programas regulares*.

Primero presento la sintaxis y la semántica de PDL puro, a continuación, presento una extensión utilizando un operador de la lógica híbrida. Para más detalles sobre PDL y las extensiones híbridas que presento aquí, remito al lector a [HKT00] y los artículos originales [PT91] y [FdR06].

Definición 4.4 (Sintaxis de PDL)**.** Sean $AtProp = \{p_0, p_1, \ldots\}$ un conjunto contable de proposiciones atómicas y $\Pi_0 = \{a_0, a_1, \ldots\}$ un conjunto contable de programas atómicos. El conjunto de todas las fórmulas de PDL es el conjunto generado por la gramática:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid [\pi]\varphi$$
$$\pi ::= a \mid (\pi;\pi) \mid (\pi \cup \pi) \mid (\pi^*) \mid \varphi?$$

donde $p \in AtProp$ y $a \in \Pi_0$.

La sintaxis de los programas PDL está estrechamente relacionada con su semántica. Antes de presentar el significado intuitivo de dichos programas, conviene presentar la definición de su semántica.

Definición 4.5 (Modelos para PDL, satisfacción, denotación de programas y fórmulas)**.** Un modelo para PDL $\mathfrak{M}$ es la tupla:

$$\mathfrak{M} \stackrel{\text{def}}{=} \langle S, \{R_a\}_{a\in\Pi_0}, \ell\rangle$$

donde:

- $S \stackrel{\text{def}}{=} \{s_0, \ldots, s_n\}$ es el conjunto de estados;
- $R_a \subseteq (S \times S)$ es una relación de accesibilidad para los programas atómicos $a \in \Pi_0$;
- $\ell : S \to 2^{AtProp}$ es una función de etiquetamiento de estados.

Dado un estado s de un modelo $\mathfrak{M}$ llamamos a la pareja $(\mathfrak{M}, s)$ un modelo apuntado. Definimos la relación de satisfacción $\models$ entre modelos apuntados y las fórmulas de PDL como la menor relación tal que:

1. $(\mathfrak{M}, s) \models p$ sii $p \in \ell(s)$;
2. $(\mathfrak{M}, s) \models \neg\varphi$ sii $(\mathfrak{M}, s) \not\models \varphi$;
3. $(\mathfrak{M}, s) \models (\varphi \wedge \psi)$ sii $(\mathfrak{M}, s) \models \varphi$ and $(\mathfrak{M}, s) \models \psi$;
4. $(\mathfrak{M}, s) \models [\pi]\varphi$ sii para todo $s' \in W$, $(s, s') \in [\![\pi]\!]$ implica $(\mathfrak{M}, s') \models \varphi$.

Donde la relación de denotación de programas de PDL $[\![\pi]\!] \subseteq (S \times S)$ se define como:

$$\begin{aligned}
[\![a]\!] &\stackrel{\text{def}}{=} R_a \\
[\![\pi_1; \pi_2]\!] &\stackrel{\text{def}}{=} [\![\pi_1]\!] \circ [\![\pi_2]\!] \\
[\![\pi_1 \cup \pi_2]\!] &\stackrel{\text{def}}{=} [\![\pi_1]\!] \cup [\![\pi_2]\!] \\
[\![\pi^*]\!] &\stackrel{\text{def}}{=} ([\![\pi]\!])^* \\
[\![\varphi?]\!] &\stackrel{\text{def}}{=} \{(s, s) \mid (\mathfrak{M}, s) \models \varphi\}
\end{aligned}$$

Finalmente, definimos la denotación de fórmulas $[\![\varphi]\!]$ como el conjunto de estados que satisfacen φ:

$$[\![\varphi]\!] \stackrel{\text{def}}{=} \{s \in S \mid (\mathfrak{M}, s) \models \varphi\}$$

Un modelo puede ser considerado como un sistema de transiciones etiquetado (LTS), y los programas como reglas sobre cómo atravesar dichas transiciones. Un programa atómico a especifica tomar todas las transiciones marcadas con a. El constructor ";" denota *composición secuencial*, "$\cup$" denota *elección no determinista*, y " *" denota *iteración no determinista* (o estrella de Kleene). Al programa "$\varphi?$" lo llamamos *prueba* de si φ se cumple en el estado actual.

La lectura de la fórmula $[\pi]\,\varphi$ es: "*φ necesariamente se cumple después de cada posible ejecución de π*". También podemos definir la siguiente modalidad dual: $\langle\pi\rangle\varphi \stackrel{\text{def}}{=} \neg\,[\pi]\,\neg\varphi$. Así, podemos leer la fórmula como: "*hay una posible ejecución de π que alcanza un estado φ*". Utilizaremos π^n para denotar n iteraciones del programa π, por lo tanto $\pi^1 = \pi$, $\pi^2 = \pi; \pi$ y así sucesivamente. El resto de los conectivos lógicos ($\vee$, $\Rightarrow$ y $\Leftrightarrow$) se definen como de costumbre, en términos de $\neg$ y $\wedge$ (por ejemplo, mediante el uso de las leyes de De Morgan). Además, definimos la fórmula $\top \stackrel{\text{def}}{=} (p \vee \neg p)$, para alguna proposición atómica fija p, y la fórmula $\bot \stackrel{\text{def}}{=} \neg\top$.

Ahora presento las extensiones híbridas a PDL que usaremos después. Las lógicas híbridas extienden a lógicas modales con varios operadores nuevos.

Aquí sólo usaremos un subconjunto de estas extensiones, en concreto, vamos a utilizar los *nominales* y el *ancla de variables de estados* $\downarrow$. Para una explicación detallada de estas y otras extensiones se pueden consultar los artículos mencionados [PT91] y [FdR06].

Definición 4.6 (Nominales, variables de estados, y ancla de variables)**.** Sea *Nom* un conjunto de *nominales* tales que cada nominal se identifica con al menos un estado, y cada estado se identifica con al menos un nominal. Por simplicidad, identificamos el conjunto de los nominales con S, suponiendo que S y *AtProp* son disjuntos. Entonces, para cualquier nominal t, definimos:

- $(\mathfrak{M}, s) \models t$ sii $s = t$.

Sea *Var* un conjunto numerable de *variables de estados* con σ, ρ, etc. denotando elementos de *Var*. Para cualquier variable de estado σ y cualquier fórmula φ, agregamos dos constructores de fórmulas, las variables y el ancla de variables:

- σ;
- $\downarrow\sigma.\ \varphi$.

Definimos las nociones de variable *libre* y *acotada* usando reglas análogas a las de la lógica de predicados clásica. Entonces, definimos:

- $(\mathfrak{M}, s) \models \downarrow\sigma.\ \varphi$ sii $(\mathfrak{M}, s) \models \varphi\{s/\sigma\}$.

Donde $\varphi\{s/\sigma\}$ es la fórmula φ con todas las instancias libres de la variable σ sustituidas con el nominal s. Por simplicidad, sólo vamos a considerar fórmulas sin variables libres.

4.4. Computando estrategias por inducción hacia atrás

Podemos utilizar lógicas modales para describir propiedades de sistemas de transiciones. Una fórmula de lógica modal puede satisfacerse en varios estados de un sistema o modelo de este tipo. El problema de decidir si una fórmula φ se satisface en un estado s de un modelo $\mathfrak{M}$ (viz. $(\mathfrak{M}, s) \models \varphi$) se llama el problema de la *verificación de modelos*. El *algoritmo de etiquetamiento de estados* para la verificación de modelos computa el conjunto $[\![\varphi]\!]$ de todos los

estados que satisfacen una fórmula φ. El algoritmo hace este cómputo extendiendo inductivamente la función de etiquetamiento ℓ, desde las proposiciones atómicas hacia las fórmulas.

El objetivo es calcular las estrategias de inducción hacia atrás utilizando la verificación de modelos. Un método posible es la definición de una fórmula modal que caracterice a todos los estados que representen una mejor decisión en nodos no finales. Por ejemplo, ver el juego en la Figura 4.1. El resultado esperado sería el conjunto $\{s_2, z_2, z_4\}$. Primero presento una explicación intuitiva del método a través de este ejemplo.

Procedemos inductivamente por la profundidad del árbol de juego. Para el caso base, considere el subárbol con raíz s_2. Siguiendo el algoritmo, el jugador 2 elige z_4 porque es el *mejor entre los hermanos*: la utilidad asociada a z_4 es mayor o igual que la utilidad asociada a los otros hermanos.

Establezcamos que $\xrightarrow{sib}$ relacione a una hoja consigo misma y con sus hermanos. Establezcamos que $\xrightarrow{2}$ represente un orden sobre las hojas de tal forma que $z \xrightarrow{2} z'$ sii $u_2(z) \leq u_2(z')$. La Figura 4.2 muestra los subconjuntos de estas dos relaciones: en el panel (a) para el análisis de z_3, y en el panel (b) para el análisis de z_4. Observemos que sólo para el mejor hermano z_4, todos las rutas que salen con $\xrightarrow{sib}$ tiene un posible regreso con $\xrightarrow{2}$. Restringiendo nuestra atención a estas porciones de las relaciones, sólo para z_4 se cumple que $\xrightarrow{sib}$ está contenido en el converso de $\xrightarrow{2}$.

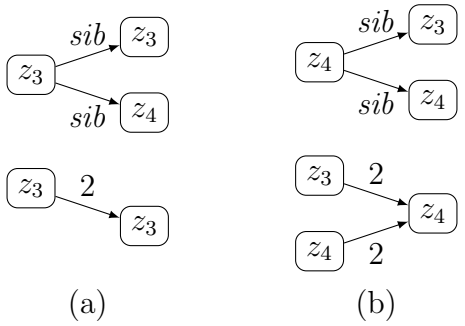


Figura 4.2: (a) $\xrightarrow{sib}$ no está contenida en el converso de $\xrightarrow{2}$, (b) para el mejor hermano z_4, $\xrightarrow{sib}$ sí está contenida en el converso de $\xrightarrow{2}$.

Recordemos que el siguiente esquema de axioma de la lógica multimodal:

$$\varphi \rightarrow [a]\langle b\rangle\varphi \tag{4.1}$$

caracteriza a todos los marcos multimodales tales que la relación $\xrightarrow{a}$ está contenida en el converso de la relación $\xrightarrow{b}$ (cf. los capítulos 5 y 6 de [Pop94]). Esta caracterización se basa en la validez: es necesario demostrar que (4.1) se cumple para toda φ y para cada posible valuación de las proposiciones atómicas. Por lo tanto, (4.1) no se puede utilizar directamente en un enfoque de verificación de modelos basado en la satisfacción en un modelo en particular.

Tengamos en cuenta, sin embargo, que en nuestro ejemplo, si hubiera una fórmula σ identificando exactamente la hoja deseada, bastaría con comprobar que (4.1) se cumple para $\varphi = \sigma$. Esta es la razón que motiva el uso de extensiones de lógica híbrida.

Para el caso base en nuestro ejemplo, usando el operador de ancla de la lógica híbrida, una primera aproximación a la caracterización deseada sería la siguiente:

$$\downarrow\sigma.\ [sib]\langle 2\rangle\sigma \tag{4.2}$$

Así (4.2) caracteriza a todos los estados σ que satisfacen localmente esta propiedad deseada.

Para el caso inductivo utilizamos un razonamiento similar. En el ejemplo, en la raíz del árbol 1 tiene que elegir entre s_1 y s_2. De hecho, la decisión es elegir entre z_2 y x_4. Esto es porque se asume que el algoritmo se aplica de forma recursiva en s_1 y en s_2. Por lo tanto, decimos que z_2 y z_4 son *hermanos virtuales*, y utilizamos el mismo método que antes.

Presento ahora las definiciones formales de modelos de juegos y la caracterización de mejores hermanos.

Definición 4.7 (Modelos de juegos)**.** Sea $G = \langle S, \prec, s_0, N, P, \{u_i\}_{i\in N}\rangle$ un juego como en la Definición 4.1. Un modelo de juego $\mathfrak{M}_G$ es la tupla:

$$\mathfrak{M}_G \stackrel{\text{def}}{=} \langle S, \{R_a\}_{a\in\Pi_0}, \ell\rangle$$

donde:

- $\Pi_0 \stackrel{\text{def}}{=} \{pred, succ\} \cup N$;
- $(s, s') \in R_{pred}$ sii s es un predecesor-$\prec$ inmediato de s';
- $(s, s') \in R_{succ}$ sii s' es un sucesor-$\prec$ inmediato de s;
- $(z, z') \in R_i$ sii $z, z' \in Z$, $i \in N$, y $u_i(z) \leq u_i(z')$;
- para todo $i \in N$, existen proposiciones atómicas de la forma $turn_i$, tales que $turn_i \in \ell(s)$ sii $s \in (S - Z)$ y $P(s) = i$.

Un modelo de juego consta de los siguientes componentes. En primer lugar, el modelo describe la estructura del árbol de juego usando las relaciones de sucesor y predecesor entre los nodos. En segundo lugar, el modelo a su vez codifica la información de los jugadores con las variables proposicionales. Por último, el modelo incluye los órdenes de preferencia en las hojas inducidas por las funciones de utilidad de los jugadores.

Definición 4.8 (Caracterización de mejor hermano)**.** Definimos la fórmula $BS(h)$ que caracteriza a los *mejores hermanos* en un árbol de juego con profundidad h:

$$BS(1) \stackrel{\text{def}}{=} \bigvee_{i \in N} \downarrow\sigma.\ (\langle pred \rangle turn_i \wedge \langle pred \rangle maxDepth(1) \wedge [sib]\langle i \rangle \sigma)$$

$$BS(h{+}1) \stackrel{\text{def}}{=} \bigvee_{i \in N} \downarrow\sigma.\ (\langle pred \rangle turn_i \wedge \langle pred \rangle maxDepth(h{+}1) \wedge [toBest(h)]\downarrow\rho.\ [vsib(\sigma, h)]\langle i \rangle \rho)$$

donde:

$$maxDepth(h) \stackrel{\text{def}}{=} \langle succ^h \rangle [succ]\bot \wedge [succ^{h+1}]\bot$$

$$sib \stackrel{\text{def}}{=} pred; succ$$

$$vsib(\sigma, h) \stackrel{\text{def}}{=} toAncestor(\sigma); sib; toBest(h)$$

$$toAncestor(\sigma) \stackrel{\text{def}}{=} pred^*; \sigma?$$

$$toBest(h) \stackrel{\text{def}}{=} (leaf?) \cup \bigcup_{1 \leq j \leq h} ((succ; BS(j)?); \cdots ; (succ; BS(1)?))$$

La caracterización de mejor hermano depende de la profundidad del árbol de juego, y proveemos de dos fórmulas: una para árboles de profundidad 1 (caso base), y otra para árboles de profundidad arbitraria (caso inductivo).

El principio de la caracterización es el mismo para ambas fórmulas. Utilizamos el operador $\downarrow$ para anclar la evaluación a algún nodo s. La disyunción exterior y la fórmula $\langle pred \rangle turn_i$ afirman que es una decisión tomada por algún jugador i. La fórmula $\langle pred \rangle maxDepth(\cdot)$ asegura que el árbol de juego bajo consideración es de la profundidad deseada.

Para el caso base, el resto de la caracterización es como ya se describió. Cualquier hoja accesible desde s utilizando el programa sib, tiene una posible transición de regreso con el programa i. Esto indica que s es tan deseable como cualquier otro hermano de s.

Para el caso inductivo, recordemos que la relación de preferencia se define sólo para los nodos finales. Por esta razón, utilizamos el programa *toBest* de PDL para relacionar el nodo no final s con una hoja z, de tal manera que z represente el desenlace del subjuego con la raíz r (predecesor de s). Utilizamos de nuevo el operador $\downarrow$ para anclar la evaluación a la hoja z, y luego empleamos la misma caracterización, pero para comparar a z con sus hermanos virtuales.

La naturaleza recursiva del algoritmo de inducción hacia atrás hace patente en los programas *vsib* y *toBest*. El programa *vsib* es muy directo: este programa se relaciona la hoja z con sus hermanos virtuales volviendo a la raíz r, y aplica el programa *toBest* a todos los sucesores de r. Si la profundidad del árbol desde r es $h+1$, entonces, el programa *toBest* reconstruye la ruta de las mejores decisiones para los subjuegos de profundidad $j < h+1$ (el árbol no tiene que estar balanceado).

Por último, caracterizamos el conjunto de nodos que representan las mejores decisiones tomadas en los nodos no finales en un árbol de juego con de profundidad h como el siguiente conjunto:

$$NE(h) \stackrel{\text{def}}{=} \bigcup_{1 \leq j \leq h} [\![BS(j)]\!]$$

4.5. Conclusiones

En [Bon01], Bonanno propone utilizar lógica temporal para formalizar soluciones por inducción hacia atrás. En [HvdHMW03], Harrenstein et al. proporcionan resultados de correspondencia entre marcos de juego, que codifican soluciones de inducción hacia atrás, y un caso especial de confluencia modal. Ambos [Bon01] y [HvdHMW03] son caracterizaciones basadas en validez. Hay otras caracterizaciones basadas satisfacción. Van der Hoek et al. [vdHJW05] usan una variante de la lógica de tiempo alternante. Troquard et al. [TvdHW08] usan una lógica de preferencias con extensiones híbridas. Góngora y Rosenblueth [GR09] utilizan lógica temporal probabilística para la caracterización de soluciones con estrategias mixtas.

Van Benthem et al. [vBGR08] utilizan una lógica de preferencias *ceteris paribus* para caracterizar equilibrios de Nash. Esta caracterización también es local, y un verificador de modelos para esa lógica también puede calcular las soluciones de un juego. La diferencia con nuestro enfoque es que en [vBGR08] la caracterización es para juegos en forma normal. Por lo tanto, el enfoque en [vBGR08] incluye todas las soluciones posibles, mientras que la inducción hacia atrás calcula sólo soluciones perfectas por subjuego.

Passy y Tinchev [PT91] introducen por primera vez PDL con nominales y otras extensiones. Franceschet y de Rijke [FdR06] introducen PDL con varias extensiones híbridas, y también presentan resultados de complejidad para la verificación de modelos de estas lógicas. Aquí, sólo usamos un subconjunto de los idiomas que se presentan en [FdR06].

En este capítulo, se propone el uso de extensiones de lógica híbrida para establecer un puente entre las caracterizaciones de equilibrios de Nash basadas en validez y la verificación de modelos práctica. Usando estas extensiones, se muestra que un verificador de modelos puede calcular los equilibrios, además de sólo caracterizarlos. Se proporciona un ejemplo de este cálculo usando PDL híbrido. Se utiliza una fórmula PDL, que corresponde a un caso especial de confluencia modal, para la caracterización de la mejor decisión tomada en nodos no finales en un árbol de juego. Un verificador de modelos puede calcular el conjunto de todos los nodos que se caracterizan por esta fórmula, y por lo tanto reconstruir el perfil de estrategias de una solución por inducción hacia atrás.

Concluyo señalando algunas direcciones de investigación futura que considero deseable. Para el ejemplo que aquí se presenta, es importante investigar qué tipo de optimizaciones son posibles en implementaciones prácticas. Al utilizar el operador $\downarrow$, la complejidad de la verificación de modelos se vuelve exponencial en la profundidad de las ocurrencias de $\downarrow$ en la fórmula. En definiciones del capítulo, sin embargo, podemos reutilizar resultados y mantener al operador $\downarrow$ en profundidades bajas. Además, mediante la reescritura de algunas fórmulas con algunos tipos de guardas, es posible evitar muchos cálculos (e.g.,, en el programa *toBest*). Otra dirección de investigación es explorar la posibilidad de utilizar las lógicas de uso más común en la comunidad de verificación de modelos como, por ejemplo, CTL. Por último, sería interesante estudiar si este enfoque puede ser aplicado a juegos con sólo soluciones de estrategias mixtas.

5

Síntesis de estrategias con lógica temporal

```
PIEZA DE RELOJ

Adelantar todos los relojes del mundo
dos segundos sin que nadie se entere.


PIEZA DE RELOJ

Quizás el reloj principal del mundo se haya adelantado
o atrasado un segundo sin que nadie
lo sepa.
Pero mientras la gente no lo sepa
nada se altera.
```

– Yoko Ono, *Pomelo*, 1964

5.1. Introducción

En este capítulo exploramos las conexiones entre (i) el cómputo de equilibrios de Nash y (ii) la verificación de propiedades de alcanzabilidad (del inglés *reachability*) en verificación de modelos. En una representación arbórea de juegos, un equilibrio de Nash perfecto por sub-juegos puede pensarse como una colección óptima de rutas (i.e., sólo tiene rutas de peso mínimo o máximo)

yendo de todo nodo interno hacia alguna hoja. Similarmente, en la verificación de modelos, el comportamiento de los sistemas no determinísticos puede representarse con estructuras arbóreas. Al verificar propiedades de alcanzabilidad en estos sistemas, el mecanismo más usual es ir hacia atrás, desde los nodos destino hacia todo posible origen en el árbol. Al ir hacia atrás, este cómputo siempre visita primero los nodos más cercanos. De esta manera, tal cómputo puede pensarse como la construcción de rutas óptimas, yendo desde todo nodo interno hacia el destino previamente designado. Proponemos primero usar los beneficios de los algoritmos de verificación de modelos simbólica en el cómputo de equilibrios de Nash, y segundo extender la aplicabilidad de la verificación de modelos a situaciones representables como juegos. Más específicamente, abordamos estas metas (i) presentando una versión simbólica del algoritmo Bellman–Ford para juegos con múltiples jugadores y turnos, y (ii) extendiendo los procedimientos de la verificación de modelos para verificar dichos juegos.

Por medio de árboles, podemos representar gráficamente juegos cooperativos, finitos, y de información perfecta, esto es, representamos juegos en forma extensiva. En este capítulo proponemos y trabajamos con una generalización directa de los árboles de juego: gráficas de juego. Las dos diferencias principales son que las gráficas de juegos (i) pueden tener ciclos, y (ii) tienen pesos –o unidades de utilidad– asociados a cada transición. Definimos también un concepto de optimalidad para estas gráficas de juegos, partiendo de una generalización del concepto de equilibrio de Nash perfecto por subjuegos. Siguiendo esta definición de optimalidad, mostramos cómo usar una versión extendida del algoritmo de Bellman–Ford para calcular rutas óptimas, y por lo tanto, también calcular equilibrios de Nash perfectos por subjuegos.

Es relevante mencionar que el método que presentamos puede adaptarse para usarse también con árboles de juego. Haciendo una generalización a gráficas de juego añadimos poca complejidad matemática. Por el contrario, facilitamos la conexión con la verificación de modelos, y la aplicabilidad del algoritmo Bellman–Ford.

En verificación de modelos, podemos representar sistemas reactivos y no deterministas como autómatas finitos. Podemos expandir el comportamiento de dichos autómatas a árboles infinitos. Para describir algunas propiedades de estos árboles podemos usar la lógica CTL (*Computation Tree Logic*). Un algoritmo usado comúnmente para la verificación de propiedades en CTL es el algoritmo de etiquetamiento de estados (*state-labeling algorithm*). Como mencionamos antes, cuando verificamos propiedades de alcanzabilidad con el algoritmo de etiquetamiento de estados, realmente calculamos una ruta óptima que va desde cualquier estado del sistema hacia los estados destino dados. Co-

mo las transiciones en los modelos de CTL no tienen pesos, la optimalidad de tales rutas se define sólo en términos del número de transiciones. Proponemos usar modelos basados en gráficas de juegos en la verificación de propiedades CTL. Desarrollamos una extensión a CTL para describir no sólo propiedades de alcanzabilidad, sino también propiedades relativas a los costos o utilidades obtenida por los jugadores.

Una de las principales fortalezas de la verificación de modelos es la posibilidad de verificar sistemas grandes (e.g., sistemas con millones de estados). Esta fortaleza se debe principalmente al uso de algoritmos *simbólicos*. Los algoritmos simbólicos utilizan estructuras de datos eficientes, como los BDD (*Binary Decision Diagrams*), para representar y manipular grandes conjuntos de datos. En particular, usamos una generalización de los BDD llamada MTBDD (*Multi-Terminal BDD* [FMY93]).

Nos interesa la representación y el producto de matrices con MTBDD. La razón es que el algoritmo Bellman–Ford puede formularse como una sucesión de multiplicación de matrices. Extendemos una versión simbólica del algoritmo Bellman–Ford presentada en [FMY93]. El algoritmo en [FMY93] calcula el producto de matrices y, por lo tanto, la ruta más corta para gráficas pesadas, donde los arcos están asociados a un valor o peso (ver también [BFG$^+$97]). Nuestra extensión del algoritmo calcula rutas más cortas para gráficas de juegos, esto es, gráficas con pesos donde los arcos están asociados a múltiples pesos, y los estados con turnos de jugadores.

En las siguientes secciones presentamos una versión simbólica del algoritmo Bellman–Ford para calcular rutas más cortas en gráficas de juegos. Después, usamos este algoritmo para extender el algoritmo de etiquetamiento de estados, y así verificar modelos basados en gráficas de juegos. Así mismo, también presentamos algunos resultados experimentales comparando las versión símbólica versus la versión no símbólica de nuestra extensión del algoritmo Bellman–Ford.

Nuestros resultados muestran que el uso de MTBDD, comparados con los métodos no simbólicos, no resultan en la posibilidad de manipular modelos astronómicamente grandes, como sucede en algunos casos con la verificación de modelos CTL tradicional. En un sentido, esta reducción del beneficio se puede anticipar, pues la presencia de múltiples agentes involucra más pasos de cómputo, ausentes en un verificador de modelos CTL. El uso de los MTBDD, sin embargo, requiere significativamente menos memoria que los métodos no simbólicos, y permite resolver problemas comparativamente más grandes.

Este capítulo se basa en el artículo por Góngora y Rosenbueth [GR14] próximo a publicarse.

5.1.1. Trabajo relacionado

Nuestras gráficas de juegos son similares a las redes dinámicas de Lozovanu [LP09]. En las redes de [LP09], cada jugador selecciona de manera independiente un parámetro de control. Cada estado de la red es entonces determinado por la selección de todos los jugadores. Lozovanu define para sus redes equilibrios de Nash además de otros conceptos de solución de juegos. Posteriormente, el autor muestra una variante del algoritmo de Dijkstra para encontrar dichas soluciones.

En el contexto de algoritmos simbólicos y algoritmos para juegos, podemos mencionar el trabajo de Bolus y Berghammer [Bol11, BB12]. Bolus y Berghammer emplean BDD quiasi ordenados para calcular coaliciones ganadoras y para resolver otros problemas en juegos simples –juegos cooperativos con sólo dos posibles casos de coaliciones (cf. los artículos mencionados para las definiciones precisas de estos conceptos). Nuestro algoritmo simbólico de ruta más corta es una extensión del algoritmo simbólico de Bellman–Ford presentado en [FMY93], y después optimizado en [BFG+97]. Una aproximación diferente del cómputo simbólico de rutas más cortas la presenta Sawitzki en [Saw04]. Sawitzki usa BDD ordinarios, en lugar de BDD multivaluados, y presenta versiones simbólicas del algoritmo de Dijkstra y del algoritmo Bellman-Ford. Tanto los equilibrios de Nash como las rutas más cortas son casos particulares de la optimización multicriterio. Aunque no presenta algoritmos simbólicos, recomendamos al lector la reseña de Garroppo [GGT10], donde se exponen diversas aproximaciones al problema de optimización multicriterio de rutas .

Del lado de la verificación de modelos, en [DCDS01] los autores introducen *min-max CTL*. Dasgupta et al. usan el lenguaje min-max CTL para verificar y consultar propiedades cuantitativas de sistemas con relojes. Comparado con nuestra aproximación, en [DCDS01] no se utilizan ni algoritmos de ruta más corta ni algoritmos simbólicos para la verificación de sistemas con min-max CTL. Otra aproximación a la verificación simbólica de sistemas multi-agente, es el trabajo de [RL07]. Raimondi and Lomuscio introducen un marco de trabajo lógico para describir propiedades temporales, epistémicas, y deónticas. Los autores de [RL07] también proveen de algoritmos basados en BDD para la verificación automatizada.

Por último, en un enfoque más teórico, existen varias trabajos que usan lógicas modales para caracterizar soluciones de juego, como los equilibrios de Nash. En comparación, los otros trabajos *sólo se caracterizan* equilibrios de Nash, mientras que el método de este capítulo los *calcula*. Es importante también mencionar el trabajo seminal de Bonanno [Bon01]. El autor introduce el uso de lógica temporal para el análisis de la estructura lógica juegos con

información perfecta. Otra obra representativa de esta línea es el de Harrenstein et al. [HvdHMW03]. En [HvdHMW03], los autores también presentan una caracterización lógica de perfectos juegos de información. Mediante el uso de la lógica dinámica en lugar de la lógica temporal, el trabajo de Harrenstein et al. añade un sabor más operativo a la caracterización equilibrios de Nash.

5.2. Juegos y gráficas de juegos

Este sección tiene como objetivo presentar los conceptos de juego de información perfecta, inducción hacia atrás, y su generalización a gráficas de juego. Además, de proponer la relación existente entre los conceptos de equilibrio de Nash y de ruta más corta.

5.2.1. Juegos e inducción hacia atrás

Los juegos, en pocas palabras, son descripciones formales de las interacciones de los agentes racionales. Podemos formalizar tales interacciones como colecciones, ya sea de posibles acciones disponibles para cada jugador (forma estratégica), o como conjuntos de secuencias de eventos (forma extensa). La naturaleza del juego (por ejemplo, de conflicto o de acuerdo) es a veces descrita numéricamente, la asignación de unidades de utilidad de cada agente en cada posible resultado del juego. En este trabajo, nos ocupamos únicamente con los juegos de información finitos, no cooperativas, completos y perfectos en forma extensiva. Así remitimos al lector a [OR94] para una introducción más completa a la Teoría de Juegos.

Un juego en *forma extensiva* puede ser representado como un árbol. Un nodo no final, o estado, representa el cambio de un jugador, y tiene un sucesor para cada acción posible a disposición del jugador en ese estado. Las hojas, en los juegos finitos, representan los resultados posibles del juego. En un árbol de juego, una *estrategia* para el jugador i es una función de la elección de un sucesor para cada nodo que representa turno de i. Un *perfil de estrategias* es un conjunto formado por una estrategia para cada jugador. Por lo tanto, un perfil de estrategias induce una sola trayectoria de cada nodo no final hacia alguna hoja.

Una característica clave del enfoque de teoría de juegos para la toma de decisiones es que todos los jugadores se suponen racionales. La racionalidad dicta que un jugador elija siempre las acciones que maximizan la propia utilidad del jugador. En consecuencia, una *solución del juego* es un método para la elección de las estrategias de maximización de la utilidad para todos los

jugadores. Un concepto de la solución conocida es la de *equilibrio de Nash*. Un perfil de estrategias es un equilibrio de Nash si cada jugador no puede aumentar su utilidad desviándose unilateralmente del perfil.

La solución por *inducción hacia atrás* es un caso especial de equilibrio de Nash: un equilibrio de Nash perfecto por subjuegos. El algoritmo de inducción hacia atrás se desprende de la prueba inductiva por Kuhn que muestra que cada juego finito en forma extensiva[1] está determinado.

La inducción hacia atrás puede explicarse como el razonamiento contrafáctico que un jugador puede utilizar para la elección de sus acciones: "Si yo elijo esta acción, a continuación el siguiente jugador elegirá su mejor actuación y mi recompensa será ...". Este razonamiento hipotético se aplica de forma recursiva hasta que se alcance un posible final del juego.

Considere el juego que se muestra en la figura 5.1. El juego tiene dos jugadores, 1 y 2. Los turnos de los jugadores se muestran encima de los nodos que no son finales, y las ganancias de utilidad se muestran debajo de las hojas. El jugador 1 mueve primero, entonces el jugador 2 se mueve y el juego termina. Supongamos que 1 se mueve a la derecha y luego 2 también se mueve a la derecha, a continuación, los jugadores ganarán 4 y 2 unidades de utilidad, respectivamente. Desde el nodo raíz, este resultado se determina si todos los jugadores siguen el perfil de estrategia enfatizado con flechas gruesas en la figura.

Si agregamos un estado ficticio s_{end}, tal que cada hoja z en la Figura 5.1 tiene un único arco sin costo que llega a s_{end}. La solución que describimos define una única ruta que va desde cualquier estado a s_{end}. Observe que para el caso de un único agente, este problema se reduce a encontrar una *ruta más larga* desde cualquier estado al destino s_{end}, y por lo tanto, se puede resolver con un algoritmo de ruta más corta (ver [CLRS09]).

Por último, para convertir esta solución para ser perfecta en subjuegos, requerimos que todas las decisiones adoptadas en todos los estados sean óptimas. Por lo tanto, el jugador 2 debe elegir derecha en el estado s_1.

5.2.2. Gráficas de juego

En esta sección, se amplía el concepto de árboles de juego a gráficas dirigidas. Básicamente, permitimos al juego tomar la forma de una gráfica y asociamos un costo o unidades de utilidad en cada arco de la gráfica.

Se define una gráfica de juego como una tupla $G = (V, E, N, P, C)$ tal que:

- V es un conjunto finito de nodos

[1] el resultado de Kuhn sólo aplica para juegos de información perfecta y completa.

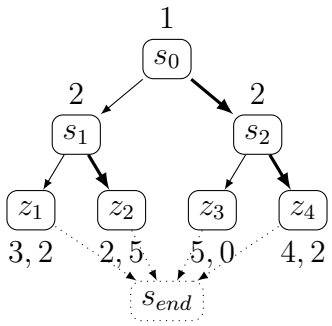


Figura 5.1: Un juego de dos jugadores y su perfil de estrategias por inducción hacia atrás.

- $E \subseteq V^2$ es un conjunto de arcos
- $N = \{1, \ldots, n\}$ es un conjunto de jugadores
- $P : V \to N$ es una función de turnos, y
- $C : V^2 \to (\mathbb{R}^+)^n \cup \{\infty\}^n$ es una función de costos tal que para todo $(v, v') \in V^2$, (i) si $v = v'$, entonces $C(v, v') = \overline{0}$, (ii) si $v \neq v'$ y $(v, v') \in E$, entonces $C(v, v') \in (\mathbb{R}^+)^n$, y (iii) si $v \neq v'$ y $(v, v') \notin E$, entonces $C(v, v') = \overline{\infty}$

Dados dos nodos distintos, v y V', de una gráfico de juego G, definimos una *ruta* de longitud k, de v a v', como una secuencia de nodos distintos $v_1 v_2 \cdots v_k$ de tal manera que $v = v_1$, $v' = v_k$,$(v_i, v_{i+1}) \in E$ para $1 \leq i < k$. También definimos el *costo acumulado* de una ruta como la suma de vectores $Cost(v_1 \cdots v_k) = \sum_1^{k-1} C(v_i, v_{i+1})$. Si $Cost(v_1 \cdots v_k) = x$ y $p \in N$, definimos $Cost_p(v_1 \cdots v_k) = x_p$, donde x_p es el i-ésimo componente de x.

Para los árboles de juego, el equilibrio de Nash perfecto por subjuegos es una función que define una trayectoria de cada nodo del árbol hacia alguna hoja. Por otra parte, una trayectoria definida por un equilibrio de Nash es, de hecho, una *ruta más corta*: un camino que optimiza la utilidad de todos los jugadores. En este apartado nos centraremos en la minimización de una función de costos, y más tarde vamos a discutir sobre cómo hacer frente a problemas de maximización.

Observe que los árboles de juego son un caso especial de las gráficas de juego. Para estos árboles, un equilibrio de Nash perfecto por subjuegos es una función que define una ruta de todos los nodos internos a alguna de las hojas.

Más aún, tales rutas son *rutas más cortas*, pues optimizan la utilidad para todos los agentes.

Debido a que nuestras gráficas de juego pueden no tener hojas, para definir una ruta más corta, debemos elegir un único nodo de destino. De esta manera, nuestra solución debe encontrar una ruta más corta de todos los nodos de la gráfica hacia el destino seleccionado.

Un camino $v_1 \cdots v_k$ de una gráfica G es una *ruta más corta* de v_1 hacia v_k si para todo $i \in \{1, \ldots, k-1\}$, y para todo u tal que $(v_i, u) \in E$, cada ruta $u \cdots v_k$ satisface $Cost_{P(v_i)}(v_i v_{i+1} \cdots v_k) \leq Cost_{P(v_i)}(v_i u \cdots v_k)$.

5.3. Algoritmo Bellman–Ford con agentes y turnos

Ya vimos que podemos generalizar las nociones de árbol de juego a gráficas y la de equilibrio de Nash a ruta más corta. En esta sección presentamos una generalización de un algoritmo de ruta más corta para las gráficas de juegos. Primero, repasamos el algoritmo original y las propiedades que lo vuelven correcto. Después, generalizamos a el caso de gráficas de juego, y usamos argumentos análogos al caso original para demostrar la corrección del algoritmo extendido.

5.3.1. Algoritmo Bellman–Ford

Existen varios algoritmos eficientes para la búsqueda de caminos más cortos en gráficas con pesos. Véase, por ejemplo, Cormen et al. [CLRS09] para una explicación detallada de algunos de estos algoritmos. En esta sección extendemos el algoritmo de Bellman-Ford para encontrar las rutas más cortas en nuestros gráficas de juego.

Por lo general, el algoritmo Bellman-Ford se presenta para resolver el problema de calcular las rutas más cortas desde un origen dado hacia todos los vértices de una gráfica. En nuestro caso, sin embargo, deseamos calcular los caminos más cortos desde todos los vértices hacia un destino determinado. El algoritmo Bellman-Ford también resuelve este otro problema mediante la inversión de los arcos.

El algoritmo de Bellman-Ford (Algoritmo 1) calcula la ruta más corta desde cada nodo hacia un único destino en tiempo $O(|V| \cdot |E|)$. En pocas palabras, el algoritmo consiste en mantener, para cada vértice v, (i) una aproximación $dist(v)$ al costo acumulado de la ruta más corta desde v hacia un destino dado d, y (ii) un mejor sucesor aproximado $succ(v)$ que lleva a v al siguiente nodo en

una ruta más corta. Estas aproximaciones se actualizan iterativamente hasta converger a los valores óptimos. Las actualizaciones se realizan mediante el refinado (RELAX y TRIANGLE) de la aproximación del primer nodo de cada arco en la gráfica. RELAX refina un solo arco, y TRIANGLE refina todos los arcos.

Algoritmo 1 Algoritmo Bellman–Ford

```
1: BELLMAN–FORD(G, d):
2: for all v ∈ V do
3:     succ(v) ← ⊥
4:     dist(v) ← ∞
5: dist(d) ← 0
6: for x ← 1 to |V| − 1 do
7:     TRIANGLE()
8: TRIANGLE():
9: for all (v, v′) ∈ E do
10:    RELAX(v, v′)
11: RELAX(v, v′):
12: if dist(v) > C(v, v′) + dist(v′) then
13:    succ(v) ← v′
14:    dist(v) ← C(v, v′) + dist(v′)
```

Antes de presentar la versión extendida del algoritmo, vamos a recordar dos propiedades de relajación cruciales para la corrección del algoritmo Bellman-Ford:

R1 Para cada arco (v, v'), RELAX(v, v') no puede incrementar la aproximación $dist(v)$.

R2 Si $v_1 \cdots v_k$ es una ruta más corta desde v_1 hacia v_k, entonces $dist(v_1)$ es óptima después de relajar cada arco de la ruta en el siguiente orden: $(v_{k-1}, v_k), (v_{k-2}, v_{k-1}), \ldots, (v_1, v_2)$.

La propiedad **R1** se sigue inmediatamente de la definición de RELAX. Para **R2**, observe que justo después de relajarse (v_{k-1}, v_k) la aproximación $dist(v_{k-1})$ es óptima, ya que se inicializa $dist(v_k) = 0$ con su valor óptimo. Siguiendo un argumento inductivo sencillo, podemos afirmar que después de relajar $(v_{k-(i+1)}, v_{k-i})$, la aproximación $dist(v_{k-(i+1)})$ es óptima.

La corrección del algoritmo se sigue de estas dos propiedades. El algoritmo ejecuta el procedimiento TRIANGLE() $|V| - 1$ veces. Por lo tanto, todos los arcos se relajan $|V|-1$ veces, la longitud máxima posible de una ruta más corta. En la i-ésima ejecución de TRIANGLE(), el primer arco (v, v') de una ruta más corta de longitud I se relajó en el orden requerido por **R2**, y seguirá siendo óptima a partir de entonces (por **R2**).

5.3.2. Extensión de Bellman–Ford

En el Algoritmo 2 presentamos una versión extendida del algoritmo de Bellman-Ford a gráficas de juegos. En las gráficas de juego, la aplicación de la función de turnos $P(v)$ representa una limitación para el nodo v. Durante la visita a v, el costo acumulado a optimizar debe ser el del jugador $P(v)$. Por consiguiente, en el procedimiento de relajación, la aproximación que debe refinarse es $dist_{P(v)}(v)$ (Algoritmo 2, línea 12). Las decisiones tomadas por el jugador $P(v)$ afectan el costo acumulado de todos los demás jugadores. Por lo tanto, si una mejora es posible, el costo acumulado de *todos* los jugadores deberá actualizarse (Algoritmo 2, línea 14).

Para explicar las últimas líneas del algoritmo, observe que un camino más corto puede no ser único. Durante la búsqueda de un camino más corto desde v a v', optimizamos la función de costo acumulado del jugador $P(v)$. El costo acumulado de los demás jugadores puede, sin embargo, variar entre los caminos más cortos. En algunas aplicaciones, como la que se presenta en la Sección 5, también estamos interesados en encontrar, de entre los caminos más cortos, el mejor camino para algún jugador distinguido i^*. Por esta razón, comprobamos, sin deteriorar el costo acumulado de jugador $P(v)$, si hay una mejor opción para el jugador i^* (Algoritmo 2, líneas 15–18).

Para establecer la corrección de este algoritmo, podemos contar con dos propiedades análogas a **R1** y **R2**:

R1′ Para cada arco (v, v'), RELAX(v, v', i^*) no puede incrementar la aproximación $dist_{P(v)}(v)$.

R2′ Si $v_1 \cdots v_k$ es una ruta más corta de v_1 a v_k, entonces $dist_{P(v_1)}(v_1)$ es óptimo después de relajar todos los arcos de la ruta en el siguiente orden: $(v_{k-1}, v_k), (v_{k-2}, v_{k-1}), \ldots, (v_1, v_2)$.

El Algoritmo 2 preserva **R1′** y **R2′**. Por lo tanto, se puede utilizar el mismo argumento inductivo, como en el caso de la versión estándar (i.e., por **R1′** y **R2′**), para demostrar la corrección del algoritmo extendido.

Algoritmo 2 Bellman–Ford para múltiples jugadores con turnos

```
1: BELLMAN–FORD(G, d, i*):
2: for all v ∈ V do
3:     succ(v) ← ⊥
4:     dist(v) ← ∞̄
5: dist(d) ← 0̄
6: for x ← 1 to |V| − 1 do
7:     TRIANGLE()
8: TRIANGLE():
9: for all (v, v′) ∈ E do
10:    RELAX(v, v′, i*)
11: RELAX(v, v′, i*):
12: if dist_{P(v)}(v) > C_{P(v)}(v, v′) + dist_{P(v)}(v′) then
13:    succ(v) ← v′
14:    dist(v) ← C(v, v′) + dist(v′)
15: else if dist_{P(v)}(v) = C_{P(v)}(v, v′) + dist_{P(v)}(v′) then
16:    if dist_{i*}(v) > C_{i*}(v, v′) + dist_{i*}(v′) then
17:        succ(v) ← v′
18:        dist(v) ← C(v, v′) + dist(v′)
```

Por último, sólo mencionamos que es posible utilizar dos propiedades similares a **R1′** y **R2′** para demostrar que si hay diferentes rutas más cortas de v a v', el Algoritmo 2 encontrará la ruta más corta con el menor $dist_{i^*}(v)$.

5.3.3. Maximización de funciones de utilidad

A menudo, los juegos se formulan como un problema de la maximización de utilidad, en lugar de un problema de minimización de costos. Para resolver este tipo de problemas de maximización basta con invertir el signo de las funciones de utilidad. Hay, sin embargo, un detalle que debemos tener en cuenta: la ocurrencia de ciclos negativos en la gráfica.

En la optimización de funciones existe una simetría entre los problemas de minimización y maximización. La razón es que maximizar una función con valores positivos es equivalente a minimizar el inverso aditivo de la misma función. Esta simetría se rompe con la ocurrencia de ciclos negativos. Esto se debe a que siempre es posible minimizar aún más el valor resultante si se

atraviesa un ciclo negativo.

En nuestras gráficas de juego, los pesos negativos sólo afectan arcos (v, v') tales que $C_{P(v)}(v, v') < 0$. Afortunadamente, el algoritmo de Bellman-Ford es capaz de detectar este tipo de ciclos.

Después de ejecutar $|V| - 1$ iteraciones del procedimiento TRIANGLE, *dist* necesariamente converge a los valores óptimos. Si ejecutamos de nuevo el algoritmo (es decir, otras $|V| - 1$ iteraciones de TRIANGLE), en cierta nueva iteración, cuando relajemos los arcos (v, v') de un ciclo con pesos negativos, los valores necesariamente cambiarán. La razón es que el jugador $P(v)$ puede minimizar aún más su valor $dist_{P(v)}(v)$ restando $C_{P(v)}(v, v')$. Por lo tanto, después de esta nueva ejecución, podemos detectar los nodos que forman parte de un ciclo negativo comparando los valores de sus aproximaciones con los de la ejecución del anterior. Si para algún v, cambia $dist(v)$, entonces simplemente establecemos $dist(v) = \overline{\infty}$. Por último, después de terminar esta fase de detección de ciclos negativos, debemos restaurar el signo de los otros valores de *dist* para obtener el resultado deseado.

5.4. Algoritmo Bellman–Ford simbólico con múltiples agentes y turnos

En esta sección, presentamos una versión simbólica del algoritmo Bellman-Ford extendido. Este algoritmo es una adaptación del algoritmo de multiplicación de matrices por Fujita et al. [FMY93]. Primero, revisaremos la relación entre la multiplicación de matrices y el algoritmo de Bellman-Ford.

5.4.1. Producto matricial y el algoritmo Bellman–Ford

Nuestro algoritmo se basa en el hecho de que la multiplicación de matrices y TRIANGLE son procedimientos equivalentes (ver [BFG+97] para una explicación de esta equivalencia). Por lo tanto, es posible adaptar una implementación del producto matricial para calcular rutas más cortas en una gráfica.

Para ejemplificar la relación entre estos procedimientos observemos la siguiente multiplicación de matrices:

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \times \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} a \cdot \alpha & + & b \cdot \beta & + & c \cdot \gamma & + & d \cdot \delta \\ e \cdot \alpha & + & f \cdot \beta & + & g \cdot \gamma & + & h \cdot \delta \\ i \cdot \alpha & + & j \cdot \beta & + & k \cdot \gamma & + & l \cdot \delta \\ m \cdot \alpha & + & n \cdot \beta & + & o \cdot \gamma & + & p \cdot \delta \end{pmatrix}$$

En este producto, combinamos las filas de la matriz cuadrada de la izquierda con cada elemento del vector de la derecha, utilizando los operadores + y ×. Si seguimos el mismo procedimiento, pero sustituyendo + por mín y × por +, podemos calcular TRIANGLE:

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \triangle \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} \text{mín}\{a+\alpha, & b+\beta, & c+\gamma, & d+\delta\} \\ \text{mín}\{e+\alpha, & f+\beta, & g+\gamma, & h+\delta\} \\ \text{mín}\{i+\alpha, & j+\beta, & k+\gamma, & l+\delta\} \\ \text{mín}\{m+\alpha, & n+\beta, & o+\gamma, & p+\delta\} \end{pmatrix}$$

Aquí, para el operador $\triangle$, la matriz izquierda es la matriz de adyacencia de la gráfica, y el vector de la derecha es un vector de aproximaciones que contiene los valores $dist(v)$ para cada $v \in V$.

5.4.2. Multi-Terminal BDD

Los diagramas de decisión binaria multi-terminales (MTBDD; cf. [CMZ+93, FMY93]) son una extensión de los diagramas de decisión binaria reducidos (BDD para abreviar, ver [Bry86]).

Los BDD son una representación gráfica de un mapeo funcional $\mathbb{B}^n \to \mathbb{B}$. Los MTBDD extienden estos mapeos al caso más general $\mathbb{B}^n \to R$, donde R es un conjunto arbitrario, por lo general $R \subseteq \mathbb{R}$.

Un MTBDD que representa una función $f : \mathbb{B}^n \to R$ es una gráfica con pesos, dirigida y acíclica. En una gráfica de esta clase, hay dos tipos de nodos: los nodos terminales y los nodos no terminales. Cada nodo no terminal se asocia con una única variable booleana de entrada $x_i \in \{x_1, \ldots, x_n\}$. Además, cada nodo no terminal tiene dos nodos descendientes: un nodo *lo*, y un nodo *hi*. Un nodo *lo* representa el caso cuando la variable del nodo padre tiene el valor 0, y un nodo *hi* representa el caso cuando la variable tiene el valor 1. Cada nodo terminal está asociado a un único valor en R, y no tiene descendientes.

En un camino desde la raíz a un terminal, no necesariamente ocurren todas las variables de entrada. En estos caminos, sin embargo, todas las variables que ocurren deben aparecer ordenadas. Esto es, un nodo asociado a una variable x debe estar más cerca de la raíz que un nodo asociado a una variable x_j si y sólo si $i < j$, y decimos que $x_i < x_j$ en tal caso. También, un MTBDD es reducido (no hay nodos ni arcos redundantes) y único (a esta propiedad se le llama *canonicidad*).

Como ejemplo, observemos el MTBDD en la Figura 5.2. Este MTBDD representa una función ternaria, y llamamos a las variables de entrada x_1, x_2, y x_3. De los nodos no terminales, las flechas punteadas conducen a nodos *lo*, y las flechas continuas conducen a los nodos *hi*. Para la evaluación de la función,

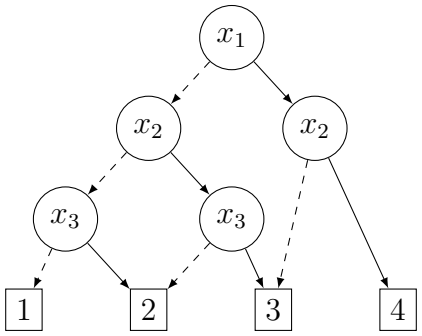


Figura 5.2: MTBDD de una función $f : \mathbb{B}^3 \to \{1, 2, 3, 4\}$

podemos seguir un camino desde la raíz hasta cierto terminal, eligiendo el valor booleano deseado en cada nodo no terminal. Por ejemplo, $f(0, 0, 0) = 1$, $f(0, 1, 1) = 3$, y así sucesivamente. Podemos pensar que un MTBDD se obtiene de un árbol de decisión binario en el que se han eliminado los nodos redundantes. Por ejemplo, debajo del nodo de más a la derecha etiquetado con x_2, podría haber dos nodos descendientes etiquetados con x_3, uno apuntando sus nodos descendientes al terminal '3', y el otro apuntando sus nodos descendientes al terminal '4'. Estos nodos, sin embargo, serían redundantes, y por lo tanto los eliminamos. Además, tenga en cuenta que hay, a lo más, un terminal 'r' para cada $r \in R$.

Por último, antes de detallar nuestro algoritmo simbólico, definimos un poco de notación. Sean A y B dos MTBDD, entonces:

- Nos referimos como $top(A)$ a la variable que etiqueta a la raíz de A (suponiendo que A es no terminal).

- Definimos $top(A, B) = \text{mín}\{top(A), top(B)\}$.

- Si $x = top(A)$, entonces $A|_{\neg x}$ y $A|_x$ se refieren a la rama encabezada por el nodo *lo* y a la rama encabezada por el nodo *hi*, respectivamente.

- Sea x una variable que no aparece ni en A ni en B, y que satisface $x < x'$, para cada variable x' que sí ocurre en A o en B. Llamamos $newNode(x, A, B)$ al MTBDD tal que: (i) tiene el nodo raíz etiquetado con x, (ii) tiene al MTBDD A como su rama *lo*, y (iii) tiene al MTBDD B como su rama *hi*.

5.4.3. Triangle simbólico

El procedimiento TRIANGLE se basa en la definición recursiva de la multiplicación de matrices. En esta definición recursiva, dividimos las matrices originales en cuatro cuadrantes, o dos mitades en el caso de vectores. Para implementar el algoritmo de Bellman–Ford, sólo estamos interesados en la multiplicación de una matriz cuadrada por un vector vertical. El algoritmo, sin embargo, se puede extender fácilmente a casos más generales.

La multiplicación de matrices (por un vector) se define recursivamente como sigue:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} A_{11} \times B_1 + A_{12} \times B_2 \\ A_{21} \times B_1 + A_{22} \times B_2 \end{pmatrix}$$

donde las submatrices cuadradas A_{ij} son los cuatro cuadrantes exactos del operando izquierdo, y los subvectores son las dos mitades exactas del operando derecho.

Una vez más, si sustituimos las operaciones de semi-anillo de la matriz por el las operaciones mín y + del triángulo, obtenemos una definición recursiva de $\triangle$:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \triangle \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} \text{mín}\{A_{11} \triangle B_1, A_{12} \triangle B_2\} \\ \text{mín}\{A_{21} \triangle B_1, A_{22} \triangle B_2\} \end{pmatrix}$$

En el algoritmo Bellman–Ford simbólico, cada matriz se representa con un MTBDD. Utilizamos las variables del MTBDD codificar las filas y columnas de las matrices, y los nodos terminales del MTBDD para almacenar los valores de las celdas. Codificamos las filas y columnas como números binarios. Seguimos además la heurística conocida de alternar las variables MTBDD para minimizar el espacio necesario para almacenar el diagrama (cf. [EFT92, DB95, HMS99]). De esta manera, utilizamos las variables impares para codificar las filas, y las variables pares codificar las columnas.

Siguiendo esta codificación alternada, en la secuencia de bits $x_1 \cdots x_n$, x_1 es el bit más significativo que codifica a una fila de la matriz, y x_2 es el bit más significativo que codificación una columna de la matriz, y así sucesivamente.

Por ejemplo, un MTBDD que representa una matriz de 4×4, digamos A, tendría, como máximo, cuatro variables: dos que codifican las filas, y dos que codifican las columnas. Por lo tanto, la posición de la celda en la cuarta fila (fila 11) y la primera columna (columna 00) está representada por la secuencia de bits 1010. En la siguiente sección presentamos un ejemplo completo de esta codificación.

El algoritmo simbólico para TRIANGLE utiliza el procedimiento *Apply* estándar de los MTBDD para la implementar mín y la adición. El procedimiento *Apply* aplica término a término operaciones binarias. Una operación binaria de matrices $\square$ es *término-a-término*, si para cualquier par de matrices A y B, $(A \square B)_{ij} = A_{ij} \square B_{ij}$.

Cuando ejecutamos TRIANGLE simbólico, atravesamos los MTBDD desde la raíz hasta las hojas. Cada nodo no terminal divide la matriz en dos mitades. Los nodos con variables impares dividen la matriz horizontalmente, y los nodos con variables pares dividen la matriz verticalmente. A continuación, aplicamos TRIANGLE de forma recursiva a ambas mitades. Por un lado, podemos ver en la definición recursiva anterior de TRIANGLE que cuando dividimos la matriz horizontalmente, sólo tenemos que unir las mitades superior e inferior en un único vector resultante. Por otro lado, cuando dividimos la matriz verticalmente, tenemos que aplicar mín (o mín^i en el algoritmo extendido, ver más adelante) para combinar los resultados parciales, izquierdo y derecho.

Es importante mencionar que este algoritmo requiere que siempre podamos dividir recursivamente la matriz en dos mitades exactas. Es decir, el algoritmo sólo funciona para matrices de tamaño $2^n \times 2^m$. A pesar de esto, podemos aplicar el algoritmo para matrices de tamaño arbitrario, adjuntando una submatriz de identidad (cf. [FMY93]):

$$\begin{pmatrix} A & 0 \\ 0 & 1 \end{pmatrix}$$

para ajustar el tamaño de las matrices al requerido.

Observe, sin embargo, que estamos trabajando con una instancia diferente del semi-anillo del producto. Por lo tanto, una matriz de identidad es aquella que contiene ceros en la diagonal, y valores ∞ llenando el resto de la matriz.

5.4.4. Bellman–Ford simbólico extendido

Presentamos una versión simbólica del algoritmo de Bellman-Ford extendido. Este algoritmo encuentra caminos más cortos en una gráfica de juego $G = (V, E, N, P, C)$. Representamos la gráfica con su matriz de adyacencia, y usamos una extensión del algoritmo simbólico de multiplicación de la matrices.

Un MTBDD que representa una gráfica de juego es un mapeo $\mathbb{B} \to (\mathbb{R}^+)^n$. Sean A la matriz de adyacencia y D el primer vector de aproximación (ver el

Algoritmo 2) de alguna gráfica de juego G:

$$A = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \qquad D = \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

Cada componente es ahora un vector, y cada vector está indexado por la codificación binaria de los estados o nodos de la gráfica.

Esta gráfica de juego tiene cuatro nodos, digamos de s_0 a s_3. Después de ejecutar el procedimiento TRIANGLE tenemos el siguiente vector de vectores:

$$\begin{pmatrix} \text{mín}^{P(s_0)}\{a+\alpha, & b+\beta, & c+\gamma, & d+\delta\} \\ \text{mín}^{P(s_1)}\{e+\alpha, & f+\beta, & g+\gamma, & h+\delta\} \\ \text{mín}^{P(s_2)}\{i+\alpha, & j+\beta, & k+\gamma, & l+\delta\} \\ \text{mín}^{P(s_3)}\{m+\alpha, & n+\beta, & o+\gamma, & p+\delta\} \end{pmatrix}$$

Aquí, el operador $+$ es la suma de vectores término-a-término. Tenga en cuenta, sin embargo, que el operador mín^i es relativo a los turnos de los jugadores.

Si X es un conjunto de vectores de longitud mayor que o igual a i, entonces $\text{mín}^i X$ denota a un $x \in X$ con el menor i-ésimo componente. Por ejemplo, sea $x = \text{mín}^i\{(5,10),(8,3),(20,15)\}$. Para $i = 1$ tenemos que $x = (5,10)$, y para $i = 2$ tenemos que $x = (8,3)$.

En la matriz resultante de la aplicación de TRIANGLE, las filas dictan los turnos de los jugadores. Por lo tanto, $\text{mín}^{P(s)}$ corresponde a una elección óptima para el jugador $P(s)$.

La operación mín^i no es término-a-término, y no podemos implementarla mediante el procedimiento estándar *Apply*. Observe que nuestro operador es relativo a la posición de los operandos. Por lo tanto, debemos definir una forma de implementar operaciones binarias, término-a-término, sensibles a la posición.

Una operación binaria de un MTBDD es *término-a-término y sensible a la posición* si el valor resultante depende a lo más de (i) los valores de los dos operandos, y (ii) la trayectoria que va desde la raíz del diagrama a los nodos terminales.

El algoritmo de *Apply* recorre los MTBDD hasta llegar a los nodos terminales, y luego aplica el operador requerido para los valores del nodo. Si la operación es término-a-término y sensible a la posición, entonces el valor resultante puede cambiar si dichos nodos terminales se alcanzan siguiendo distintas

trayectorias. De manera equivalente, si el MTBDD representa una matriz, una trayectoria corresponde a la posición de la celda en dicha matriz.

Nuestro algoritmo simbólico de Bellman–Ford extendido consta de dos subprogramas: los procedimientos *RelTriangle* y *RelApply*. Estas operaciones extendidas difieren de las originales en que son sensibles a la posición de sus operandos, y por lo tanto son capaces de tratar con los turnos de los jugadores en una gráfica de juego.

A primera vista, podría parecer que podemos extender el algoritmo de Bahar et al. [BFG$^+$97] para obtener *RelTriangle* y *RelApply*. En el algoritmo de Bahar et al., a diferencia del algoritmo original de Fujita et al. [FMY93], se aprovecha que hay variables que no aparecen en los MTBDD. Sin embargo, en *RelTriangle* y *RelApply*, después de recorrer de arriba hacia abajo los dos MTBDD, sólo tomaríamos en cuenta las variables que aparecen en cualquiera de las rutas, lo que sería incorrecto. La razón es que las operaciones que deseamos implementar son sensibles a la posición (sensibles a la posición de las celdas en la matriz, o sensibles a las trayectorias de los MTBDD). Al omitir variables, perderíamos esa información de la posición. Esta restricción nos obliga a extender el algoritmo de Fujita et al. [FMY93] en lugar del algoritmo más eficiente de Bahar et al. [BFG$^+$97].

Para que los procedimientos *RelTriangle* y *RelApply* sean sensibles a la posición, debemos registrar los valores de las variables que nos llevan a un nodo terminal. Para esto usamos, además de los otros parámetros, un vector de bits $\overline{b}$. El i-ésimo bit de $\overline{b}$ representa el valor que asignamos a la variable x_i del MTBDD para llegar al nodo terminal. Para ambos procedimientos, establecemos los valores de $\overline{b}$ en cada paso de la recursión. Como el procedimiento se hace recursivamente para cada variable, al alcanzar un nodo terminal cada bit de $\overline{b}$ tiene un valor asignado previamente.

Procedimiento RelTriangle Dada una gráfica de juego $G = (V, E, N, P, C)$, sean (i) A un MTBDD que representa la matriz de adyacencia de G, (ii) D un MTBDD que representa al vector de aproximación de costos, (iii) $\overline{b}$ un vector de bits de longitud $2 \cdot |V|$, (iv) x_i la menor variable asociable con un nodo no terminal, y (v) $i^* \in N$ un agente distinguido.

Definimos el procedimiento *RelTriangle*$(A, D, \overline{b}, x_i, i^*)$ como sigue:

1. Si x_i es impar (i.e.,divide horizontalmente a la matriz):

$$\begin{aligned}&\mathit{RelTriangle}(A, D, \overline{b}, x_i, i^*)\\ &\quad = \mathit{newNode}(x_i,\\ &\qquad \mathit{RelTriangle}(A|_{\neg x_i}, D|_{\neg x_i}, \overline{b}|_{\neg}, x_{i+1}, i^*),\\ &\qquad \mathit{RelTriangle}(A|_{x_i}, D|_{x_i}, \overline{b}|_{x_i}, x_{i+1}, i^*))\end{aligned}$$

2. Si x_i es par (i.e., divide verticalmente a la matriz):

$$\begin{aligned}&\mathit{RelTriangle}(A, D, \overline{b}, x_i, i^*)\\ &\quad = \mathit{RelApply}(\\ &\qquad \mathit{RelTriangle}(A|_{\neg x_i}, D|_{\neg x_i}, \overline{b}|_{\neg x_i}, x_{i+1}, i^*),\\ &\qquad \mathit{RelTriangle}(A|_{x_i}, D|_{x_i}, \overline{b}|_{x_i}, x_{i+1}, i^*),\\ &\qquad \overline{b}, x_{i+1}, \mathit{min}^{i^*})\end{aligned}$$

3. Si A y D son nodos terminales:

$$\mathit{RelTriangle}(A, D, \overline{b}, x_i, i^*) = \mathit{Apply}(A, D, +)$$

donde los vectores de bits $\overline{b}|_{x_i}$ y $\overline{b}|_{\neg x_i}$ se obtienen a partir del vector de bits $\overline{b}$, estableciendo el i-ésimo bit a 1 y a 0, respectivamente.

Procedimiento RelApply Sean (i) A y B dos MTBDD, (ii) $\overline{b}$ un vector de bits de longitud suficiente para contener todas las variables utilizadas por los MTBDD anteriores, (iii) x_i la menor variable asociable con un nodo no terminal, y (iv) $\square$ una operación binaria, término-a-término y sensible a la posición. Definimos $\mathit{RelApply}(A, B, \overline{b}, x_i, \square)$ como sigue:

1. Si A y B no son ambos nodos terminales:

$$\begin{aligned}&\mathit{RelApply}(A, B, \overline{b}, x_i, \square)\\ &\quad = \mathit{newNode}(x_i,\\ &\qquad \mathit{RelApply}(A|_{\neg x_i}, B|_{\neg x_i}, \overline{b}|_{\neg x_i}, x_{i+1}, \square),\\ &\qquad \mathit{RelApply}(A|_{x_i}, B|_{x_i}, \overline{b}|_{x_i}, x_{i+1}, \square))\end{aligned}$$

2. Si A y B son ambos nodos terminales:

$$\mathit{RelApply}(A, B, \overline{b}, x_i, \square) = A \,\square^{\overline{b}}\, B$$

Además del orden alternado de las variables, *RelTriangle* requiere utilizar las mismas variables para codificar tanto las columnas de la matriz, como las filas de la matriz derecha. Podemos lograr esto mediante la transposición del vector de la derecha antes de aplicar el procedimiento. Para el cálculo del MTBDD que representa la transpuesta de un vector, podemos simplemente intercambiar variables pares por nones en el MTBDD del vector original.

El primer y segundo casos de *RelTriangle* dividen a la matriz y recursivamente aplican el algoritmo para las dos mitades. El tercer caso opera directamente en los valores terminales: los valores de las celdas en las matrices. Como el algoritmo desciende de forma recursiva a través del MTBDD, el vector de bits $\bar{b}$ acarrea los valores de las variables que conducen a los nodos terminales.

También, en el primer caso de *RelTriangle*, utilizamos *RelApply* para calcular mín^{i^*}. Esta operación calcula un MTBDD cuyos nodos terminales tienen el menor i^*-ésimo componente, de entre aquellos que ya tienen el menor $P(\bar{b})$-ésimo componente (ver Algoritmo 2).

En el primer caso de *RelApply*, descendemos recursivamente a través de las ramas *lo* y *hi*, registrando la trayectoria hecha con el vector de bits $\bar{b}$. Al llegar a los nodos terminales, en el segundo caso, basta con aplicar el operador sensible a la posición especificada a los valores.

Observe que en ambos procedimientos el parámetro x_i itera sobre todas las posibles variables de los MTBDD, incluso cuando dichas variables no aparecen en los MTBDD dados. Estos pasos adicionales están presentes en el algoritmo de Fujita et al. [FMY93] , pero no en el algoritmo de Bahar et al. [BFG$^+$97]. En nuestro caso, sin embargo, se necesitan estos pasos adicionales de cómputo para registrar la posición exacta en la matriz al llegar a los nodos terminales.

Computando la matriz de rutas más cortas Hasta ahora, mediante *RelTriangle* podemos calcular los costos de la ruta más corta, de cada vértice al destino seleccionado (es decir, calculamos el menor $dist(s)$). El último paso del proceso es poder calcular también las rutas más cortas en sí mismas. En nuestra representación, podemos realizar este último cálculo simplemente manipulando las matrices disponibles.

Nuestra meta es computar una matriz de adyacencia σ tal que si existe una transición del vértice v al vértice v' en σ, entonces la transición $v \to v'$ es parte de una ruta más corta en la gráfica de juego dada. También, en σ, el costo de la transición $v \to v'$ es el costo acumulado de ir desde v hacia el destino dado en la ruta más corta.

Sea A la matriz de adyacencia de alguna gráfica de juego G. Sea D el vector de costos óptimo calculado por *RelTriangle*. La operación $A + D^\top$ es similar a

la ejecución de una relajación parcial. Añadimos a cada arco (de A) la aproximación computada (en $D^\top$). Los vértices que incrementen su aproximación no pueden estar en un camino más corto, pues previamente la aproximación ya convergió al valor óptimo. Por lo tanto, en esta matriz resultante, podemos podar los excedentes, y establecemos a $\overline{\infty}$ los valores de las celdas recortadas. Como resultado, terminamos con una matriz de adyacencia que sólo tiene transiciones para las rutas más cortas: una matriz de rutas más cortas.

Para podar la matriz como describimos anteriormente, usamos la operación sensible a la posición y término-a-término $BelowThreshold_i$:

$$BelowThreshold_i(a, b) = \begin{cases} a & \text{if } a_i \leq b_i \\ \overline{\infty} & \text{otherwise} \end{cases}$$

donde i es el parámetro de posición, en nuestro caso el turno del jugador (dado por la fila de los operandos en la matriz).

Por último, enumeramos todos los pasos del algoritmo simbólico en el Algoritmo 3, donde G es una gráfica de juego, d es el estado destino, e i^* es un agente distinguido. También, $initialAproximation(d)$ es la primera aproximación de los costos acumulados para alcanzar d desde cualquier vértice de la gráfica: esto es, $initialAproximation(d)$ asigna $\overline{0}$ al i-ésimo componente si $v_i = d$, en cualquier otro caso asigna $\overline{\infty}$.

Algoritmo 3 Bellman–Ford simbólico con múltiples jugadores y turnos

1: BELLMAN–FORD(G, d, i^*):
2: $A \leftarrow adjacencyMatrix(G)$
3: $D \leftarrow initialAproximation(d)$
4: **for** $x \leftarrow 1$ **to** $|V| - 1$ **do**
5: $\quad D^\top \leftarrow transpose(D)$
6: $\quad D \leftarrow RelTriangle(A, D^\top, \overline{0}, i^*)$
7: $D^\top \leftarrow transpose(D)$
8: $tmp \leftarrow Apply(A, D^\top, +)$
9: $\sigma \leftarrow RelApply(tmp, D, \overline{0}, BelowThreshold)$

5.4.5. Ejemplo

En este apartado mostramos algunos pasos clave de la ejecución del algoritmo Bellman-Ford extendido.

Considere el gráfica de juego G de la Figura 5.3 (a). La matriz de adyacencia A y el vector de costos óptimos D para G son los siguientes (para facilitar la lectura y resaltar los valores importantes, en las siguientes matrices sustituimos cada valor $\overline{\infty}$ por un $-$):

$$A = \begin{pmatrix} 0,0 & 1,1 & 2,1 & - \\ - & 0,0 & 3,0 & 1,1 \\ - & - & 0,0 & 1,2 \\ - & - & - & 0,0 \end{pmatrix} \qquad D = \begin{pmatrix} 2,2 \\ 1,1 \\ 1,2 \\ 0,0 \end{pmatrix}$$

Para representar estas matrices como MTBDD debemos utilizar una codificación binaria alternada. Recordemos que usamos los bits en posiciones impares para codificar las filas de la matriz, y los bits en posición par para codificar las columnas.

En la Figura 5.3 (b) se muestra la codificación binaria directa de A, seguido de su codificación alternada. A continuación, en la Figura 5.3 (c) se muestra la representación de A como un MTBDD.

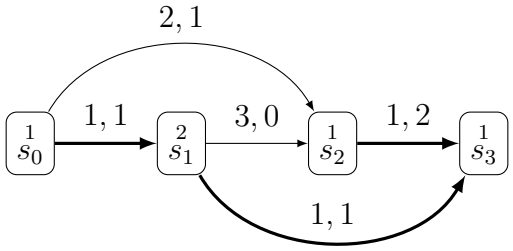
Añadimos los costos óptimos para cada transición de la siguiente manera:

$$A + D^\top = \begin{pmatrix} 2,2 & 2,2 & 3,3 & - \\ - & 1,1 & 4,2 & 1,1 \\ - & - & 1,2 & 1,2 \\ - & - & - & 0,0 \end{pmatrix}$$

Podamos la gráfica de juego manteniendo sólo aquellas transiciones que no excedan de los costos óptimos:

$$\mathit{BelowThreshold}(A + D^\top, D) = \begin{pmatrix} 2,2 & \boxed{2,2} & - & - \\ - & 1,1 & - & \boxed{1,1} \\ - & - & 1,2 & \boxed{1,2} \\ - & - & - & 0,0 \end{pmatrix}$$

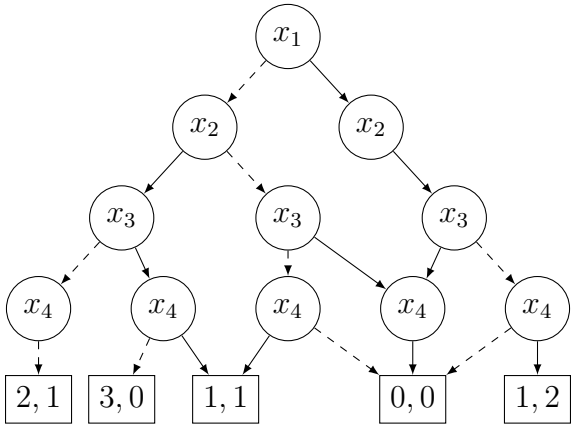
En este jemplo, marcamos la ruta corta $s_0 s_1 s_3$ con flechas gruesas en la Figura 5.3, y enmarcamos los costos acumulados de las transiciones de esta ruta en la matriz anterior. También, observe que, de manera análoga a un equilibrio de Nash perfecto por subjuegos, debemos encontrar una ruta más corta desde cualquier estado que pueda llegar al destino. Por este motivo, también marcamos la transición $s_2 \to s_3$.



(a)

s, s'	Codificación binaria	Alternada	Valor
s_0, s_1	0001	0001	$1, 1$
s_0, s_2	0010	0100	$2, 1$
s_1, s_2	0110	0110	$3, 0$
s_1, s_3	0111	0111	$1, 1$
s_2, s_3	1011	1101	$1, 2$
s_0, s_0	0000	0000	$0, 0$
s_1, s_1	0101	0011	$0, 0$
s_2, s_2	1010	1100	$0, 0$
s_3, s_3	1111	1111	$0, 0$
$\vdots$	$\vdots$	$\vdots$	∞, ∞

(b)



(c)

Figura 5.3: (a) Ejemplo de gráfica de juego G, (b) Codificación binaria de G, (c) Representación de G como un MTBDD

5.5. Verificación de modelos y rutas más cortas

Una de las principales aplicaciones de los algoritmos simbólicos para gráficas es en la *Verificación de modelos* (ver [CE82, CES86, BCM+92]). En pocas palabras, la verificación de modelos es una técnica para la verificación automática de especificaciones formales. En el enfoque de la verificación de modelos, utilizamos sistemas de transiciones de estados no deterministas, como representación de los modelos. En este enfoque, a veces describimos modelos como *estructuras de Kripke*. Al usa un modelo de Kripke, podemos usar alguna lógica modal para describir las especificaciones del modelo. De entre todas las lógicas modales, hay algunas opciones de uso común. Aquí nos centraremos en la lógica temporal, y más concretamente en CTL (*Computation Tree Logic*). Para una introducción detallada a la verificación de modelos, el lector puede consultar el libro de Baier y Katoen [BK08].

En esta sección, mostramos cómo utilizar el algoritmo de Bellman-Ford extendido en la verificación de modelos. En primer lugar, discutimos sobre la estrecha relación entre el algoritmo de etiquetamiento de estados y los algoritmos de ruta más corta. A continuación, mostramos una extensión de CTL para la descripción de modelos con transiciones con pesos. Finalmente, mostramos algunas aplicaciones potenciales de este enfoque.

5.5.1. CTL y etiquetamiento de estados

CTL es un lenguaje útil para describir las propiedades de sistemas discretos, no deterministas, y con tiempo ramificado. Sintácticamente, CTL extiende la lógica proposicional con las siguientes modalidades temporales de trayectorias individuales:

- $\mathbf{X}\varphi$: la fórmula φ se satisface en el siguiente estado;
- $\mathbf{F}\varphi$: la fórmula φ se satisface ahora o en algún estado futuro;
- $\mathbf{G}\varphi$: la fórmula φ se satisface globalmente (i.e., de ahora en adelante);
- ψ $\mathbf{U}$ φ: la fórmula φ se satisface en algún estado futuro alcanzable al pasar sólo por estados que satisfacen ψ.

Adicionalmente, en CTL estos operadores de trayectorias individuales van precedidos de un cuantificador **A** o **E** de trayectorias. Por ejemplo:

- $\mathbf{EF}\varphi$: *"hay un posible estado futuro donde se satisface la fórmula φ"*;

- $\mathbf{AG}\varphi$: *"la fórmula φ siempre se satisface, de ahora en adelante en cualquier posible estado futuro".*

Formalmente, dado un conjunto $\mathcal{P} \stackrel{\text{def}}{=} \{p, q, \ldots\}$ de proposiciones atómicas, construimos las fórmulas CTL, en forma normal existencial, de acuerdo con las siguiente gramática:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \sigma$$
$$\sigma ::= \mathbf{EX}\varphi \mid \mathbf{EG}\varphi \mid \mathbf{E}\,[\varphi\ \mathbf{U}\ \varphi]$$

donde $p \in \mathcal{P}$ y $\top \notin \mathcal{P}$.

También definimos los otros operadores de CTL de la siguiente manera:

$$\mathbf{EF}\varphi \stackrel{\text{def}}{=} \mathbf{E}\,[\top\ \mathbf{U}\ \varphi]$$
$$\mathbf{AX}\varphi \stackrel{\text{def}}{=} \neg\mathbf{EX}\neg\varphi$$
$$\mathbf{AG}\varphi \stackrel{\text{def}}{=} \neg\mathbf{EF}\neg\varphi$$
$$\mathbf{AF}\varphi \stackrel{\text{def}}{=} \neg\mathbf{EG}\neg\varphi$$
$$\mathbf{A}\,[\psi\ \mathbf{U}\ \varphi] \stackrel{\text{def}}{=} \neg\mathbf{E}\,[\neg\varphi\ \mathbf{U}\ (\neg\psi \wedge \neg\varphi)] \wedge \neg\mathbf{EG}\neg\varphi$$

y utilizamos las definiciones habituales de los otros conectores booleanos (por ejemplo, mediante el uso de las leyes de De Morgan).

Definimos un *modelo de Kripke* como la siguiente estructura relacional:

$$\mathfrak{M} \stackrel{\text{def}}{=} (S, R, \ell)$$

donde:

- $S \stackrel{\text{def}}{=} \{s_0, \ldots, s_m\}$ es un conjunto finito de estados;
- $R \subseteq S \times S$ es una relación binaria serial (i.e., todo estado tiene un sucesor) llamada relación de accesibilidad;
- $\ell : S \to 2^{\mathcal{P}}$ es una función total que etiqueta a los estados en S con proposiciones atómicas de $\mathcal{P}$.

La interpretación de CTL es local en cada estado. La relación de satisfacción $\models$ relaciona a pares $(\mathfrak{M}, s)$, $s \in S$, con fórmulas de acuerdo con las siguientes reglas:

- $(\mathfrak{M}, s) \models \top$;

- $(\mathfrak{M}, s) \models p$ sii $p \in \ell(s)$;
- $(\mathfrak{M}, s) \models \neg\varphi$ sii $(\mathfrak{M}, s) \not\models \varphi$;
- $(\mathfrak{M}, s) \models \varphi \wedge \psi$ sii $(\mathfrak{M}, s) \models \varphi$ and $(\mathfrak{M}, s) \models \psi$;
- $(\mathfrak{M}, s) \models \mathbf{EX}\varphi$ sii existe un $s' \in S$ tal que $(s, s') \in R$ y $(\mathfrak{M}, s') \models \varphi$;
- $(\mathfrak{M}, s) \models \mathbf{EG}\varphi$ sii existe una trayectoria infinita π tal que $\pi[0] = s$ y $(\mathfrak{M}, \pi[i]) \models \varphi$ para todo $i \geq 0$;
- $(\mathfrak{M}, s) \models \mathbf{E}[\psi\ \mathbf{U}\ \varphi]$ sii existe una trayectoria π y un $i \geq 0$ tales que $\pi[0] = s$, $(\mathfrak{M}, \pi[i]) \models \varphi$, y $(\mathfrak{M}, \pi[j]) \models \psi$ para todo $0 \leq j < i$;

donde una trayectoria π es una secuencia infinita de estados de S tal que $\pi[i]$ denota el i-ésimo estado en dicha secuencia, y para todo $i \geq 0$, $(\pi[i], \pi[i+1]) \in R$.

El algoritmo de *etiquetamiento de estados* calcula el conjunto $Sat(\varphi)$ de todos los estados que satisfacen la fórmula φ. Empezando por las proposiciones atómicas y la función de etiquetamiento ℓ, inducimos el conjunto $Sat(\varphi)$ de forma ascendente en la estructura de la fórmula de acuerdo con las siguientes reglas (ver el capítulo 6 de [BK08] para mayor detalle de esta caracterización):

$$
\begin{aligned}
Sat(p) &\stackrel{\text{def}}{=} \{s \mid p \in \ell(s)\} \\
Sat(\top) &\stackrel{\text{def}}{=} S \\
Sat(\neg\varphi) &\stackrel{\text{def}}{=} S \setminus Sat(\varphi) \\
Sat(\varphi \wedge \psi) &\stackrel{\text{def}}{=} Sat(\varphi) \cap Sat(\psi) \\
Sat(\mathbf{EX}\varphi) &\stackrel{\text{def}}{=} Pre(Sat(\varphi)) \\
Sat(\mathbf{EG}\varphi) &\stackrel{\text{def}}{=} \text{el mayor subconjunto } S' \subseteq S \text{ tal que:} \\
&\qquad \text{(i) } S' \subseteq Sat(\varphi) \\
&\qquad \text{(ii) } s \in Pre(S') \text{ implica } s \in S' \\
Sat(\mathbf{E}[\psi\ \mathbf{U}\ \varphi]) &\stackrel{\text{def}}{=} \text{el menor subconjunto } S' \subseteq S \text{ tal que:} \\
&\qquad \text{(i) } Sat(\varphi) \subseteq S' \\
&\qquad \text{(ii) } s \in Sat(\psi) \text{ y } s \in Pre(S') \\
&\qquad\qquad \text{implica } s \in S'
\end{aligned}
$$

donde $Pre(X) = \{s \in S \mid (s, s') \in R \text{ para algún } s' \in X\}$ es la preimagen bajo R de algún conjunto de estados X.

En este capítulo nos interesa principalmente en el caso $Sat(\mathbf{E}[\psi\ \mathbf{U}\ \varphi])$. Para este caso, comenzamos computando $Sat(\varphi)$. Luego, continuamos acumulando iterativamente la preimagen hasta llegar a un punto fijo. En cada iteración, también podemos registrar la transición realizada por cada predecesor acumulado, y por lo tanto definimos una ruta de todos los estados en S a un estado en $Sat(\varphi)$. Es fácil probar que esos caminos tienen un número mínimo de transiciones.

Debemos tomar en cuenta que es posible definir la preimagen $Pre(X)$ en términos de la composición relacional de R con $\{(s, s) \mid s \in X\}$. De hecho, esta definición conduce a una eficiente implementación simbólica de Sat (ver [CGP99]).

En base a las observaciones anteriores, proponemos en primer lugar usar transiciones con pesos en los modelos, y después extender CTL para describir este tipo de modelos. Nuestra propuesta de extensión CTL emplea fórmulas como $\mathbf{min}[\alpha\ \mathbf{U}\ \varphi] < \mathbf{min}[\beta\ \mathbf{U}\ \varphi]$. Podemos utilizar estas fórmulas para, por ejemplo, comparar los costos de las rutas α contra los costos de las rutas β, para alcanzar un estado φ.

En las siguientes subsecciones, perseguimos aún más estas ideas. En primer lugar, consideramos modelos con transiciones pesadas. A continuación, consideramos modelos pesados con múltiples agentes y turnos, es decir, gráficas de juego.

5.5.2. CTL-con-Costos

En este apartado extendemos CTL con fórmulas de comparación de costos. La finalidad es capturar el comportamiento de los modelos con transiciones pesadas. También, para la verificación de modelos, mostramos que es posible utilizar un algoritmo de ruta más corta como una extensión del algoritmo de etiquetamiento de estados.

Dado un conjunto $\mathcal{P} \stackrel{\text{def}}{=} \{p, q, \ldots\}$ de proposiciones atómicas, construimos las fórmulas de CTL-con-costos de acuerdo con las siguiente gramática:

$$\begin{aligned}
\varphi &::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \sigma \\
\sigma &::= \mathbf{EX}\varphi \mid \mathbf{EG}\varphi \mid \mathbf{E}\,[\varphi\ \mathbf{U}\ \varphi] \mid \zeta \bowtie \zeta \\
\zeta &::= c \mid \mathbf{min}\,[\varphi\ \mathbf{U}\ \varphi] \mid \mathbf{max}\,[\varphi\ \mathbf{U}\ \varphi] \\
\bowtie &::= < \mid > \mid \leq \mid \geq \mid =
\end{aligned}$$

donde $p \in \mathcal{P}$, $\top \notin \mathcal{P}$, and $c \in \mathbb{R}^+$.

También definimos las siguientes fórmulas de costos:

$$\mathbf{min}\,[\mathbf{F}\varphi] \stackrel{\text{def}}{=} \mathbf{min}\,[\top\ \mathbf{U}\ \varphi]$$
$$\mathbf{max}\,[\mathbf{F}\varphi] \stackrel{\text{def}}{=} \mathbf{max}\,[\top\ \mathbf{U}\ \varphi]$$

Los operadores de costos **min** y **max** se refieren al costo acumulado de la ruta más corta y la ruta más larga, respectivamente, para alcanzar a un estado que satisface el lado derecho del operador. Por ejemplo:

- $\mathbf{max}[\mathbf{F}\varphi] < 10$: la ruta más larga (más costosa) que alcanza un estado φ tiene costo acumulado menor que 10;
- $\mathbf{min}[\alpha\ \mathbf{U}\ \varphi] < \mathbf{min}[\beta\ \mathbf{U}\ \varphi]$: una ruta más corta de estados α mejor costos sobre una ruta más corta de estados β para alcanzar un estado φ.

Las fórmulas de CTL-con-costos se interpretan en las estructuras relacionales:

$$\mathfrak{M} \stackrel{\text{def}}{=} (S, R, \ell, C)$$

donde:

- definimos S, R, y ℓ como en la subsección anterior;
- la función total $C : S^2 \to \mathbb{R}^+ \cup \{\infty\}$ asigna costos a transiciones, y para todo $(s, s') \in S^2$, C se satisface (i) si $s = s'$ entonces $C(s, s') = 0$, (ii) si $s \neq s'$ y $(s, s') \in R$ entonces $C(s, s') \in \mathbb{R}^+$, y (iii) si $s \neq s'$ y $(s, s') \notin R$ entonces $C(s, s') = \infty$.

Definimos el *costo acumulado de una trayectoria finita* $\pi = s_0, \ldots, s_m$ como la suma:

$$\mathit{Cost}(\pi) \stackrel{\text{def}}{=} \sum_{k=0}^{m-1} C(s_k, s_{k+1})$$

La relación de satisfacción para el fragmento de CTL se define como en la subsección anterior. Para los nuevos operadores de comparación de costos se utiliza la siguiente semántica:

- $(\mathfrak{M}, s) \models \zeta \bowtie \xi$ sii $\mathit{Value}(s, \zeta) \bowtie \mathit{Value}(s, \xi)$;

donde definimos la función *Value* como sigue:

- $\mathit{Value}(s, c) \stackrel{\text{def}}{=} c$ si $c \in \mathbb{R}^+$;

- $Value(s, \mathbf{min}[\psi\ \mathbf{U}\ \varphi]) \stackrel{\text{def}}{=} \text{mín}_\pi\{Cost(\pi)\}$ tal que $\pi[0] = s$ y para algún $k \geq 0$, $\pi[k] \in Sat(\varphi)$ y $\pi[i] \in Sat(\psi)$ para todo $0 \leq i < k$;

- $Value(s, \mathbf{max}[\psi\ \mathbf{U}\ \varphi]) \stackrel{\text{def}}{=} \text{máx}_\pi\{Cost(\pi)\}$ tal que $\pi[0] = s$ y para algún $k \geq 0$, $\pi[k] \in Sat(\varphi)$ y $\pi[i] \in Sat(\psi)$ para todo $0 \leq i < k$.

Tengamos en cuenta que cuando se maximiza un costo acumulado, de haber una trayectoria que contenga ciclos, entonces $Value(s, \mathbf{max}[\psi\ \mathbf{U}\ \varphi])$ estaría mal definida. En tales casos suponemos que $Value(s, \mathbf{max}[\psi\ \mathbf{U}\ \varphi]) = \infty$.

De acuerdo con las definiciones anteriores, para extender el algoritmo de etiquetamiento del estados requerimos calcular el siguiente conjunto:

$$Sat(\zeta \bowtie \xi) \stackrel{\text{def}}{=} \{s \in S \mid Value(s, \zeta) \bowtie Value(s, \xi)\}$$

Para computar $Sat(\zeta \bowtie \xi)$ debemos calcular $Value(s, \zeta)$ para cada s del modelo. Por lo tanto, debemos calcular o bien la menor o la mayor ruta, desde todos los estados a algún destino dado.

Podemos calcular las rutas más cortas usando el algoritmo Bellman–Ford o el algoritmo de Dijkstra. Como mencionamos en la Sección 3.3, para el cálculo de rutas más largas podemos simplemente invertir los valores de las funciones de costos. Sin embargo, debemos tener un cuidado especial cuando se maximiza, pues el algoritmo de Dijkstra no maneja ciclos con pesos negativos. En estos casos, podemos usar el algoritmo de Bellman–Ford como ya describimos en la Sección 3.

En el resto del capítulo, supondremos el uso del algoritmo de Bellman–Ford. También damos por hecho la solución discutida antes para problemas de maximización, y centramos la presentación en el problema de minimización.

El primer paso para utilizar el algoritmo Bellman–Ford para computar *Value* es condensar el conjunto destino en un solo estado nuevo. Dado un modelo $\mathfrak{M} = (S, R, \ell, C)$ y dos fórmulas de CTL-con-costos ψ y φ, definimos el siguiente modelo:

$$\mathfrak{M}|_{\psi \mathbf{U} \varphi} \stackrel{\text{def}}{=} (S', R', \ell', C')$$

donde:

- $S' \stackrel{\text{def}}{=} \{s_\varphi\} \cup ((S \setminus Sat(\neg\psi)) \setminus Sat(\varphi))$, con s_φ un estado nuevo que no pertenece a S;

- $R' \stackrel{\text{def}}{=} \{(s,t) \mid (s,t) \in R \text{ y } s,t \in S'\} \cup \{(s, s_\varphi) \mid s \in S', t \in Sat(\varphi), \text{ y } (s,t) \in R\} \cup \{(s_\varphi, s_\varphi)\}$;

- $\ell'(s) \stackrel{\text{def}}{=} \ell(s)$ para todo $s \in S'$, y $\ell'(s_\varphi) \stackrel{\text{def}}{=} \bigcup_{s \in Sat(\varphi)} \ell(s)$;

- $C'(s,t) \stackrel{\text{def}}{=} C(s,t)$ para todo $s,t \in S'$, $C'(s,s_\varphi) \stackrel{\text{def}}{=} \text{mín}_{t \in Sat(\varphi)}\{C(s,t)\}$, y $C'(s_\varphi,s_\varphi) \stackrel{\text{def}}{=} 0$.

Observemos que no tomamos en cuenta las transiciones de salida de $Sat(\varphi)$, y que siempre agregamos el lazo (s_φ, s_φ). La razón es que sólo verificamos propiedades de alcanzabilidad para estos conjuntos.

El algoritmo Bellman–Ford calcula un mapeo $dist(s)$ que asigna a s el costo acumulado de la ruta más corta para llegar al destino dado.

Usando $dist(s)$ podemos calcular $Value(s,\zeta)$:

- si $\zeta = c$ entonces $Value(s,\zeta) = c$;
- si $\zeta = \mathbf{min}[\psi\ \mathbf{U}\ \varphi]$ entonces $Value(s,\zeta) = dist(s)$, usando el modelo $\mathfrak{M}|_{\psi \mathbf{U} \varphi}$ y el estado destino s_φ;
- para todo $s \in Sat(\varphi)$ establecemos $Value(s,\zeta) = 0$;
- para todo s en $\mathfrak{M}$ pero no en $\mathfrak{M}|_{\psi \mathbf{U} \varphi}$, establecemos $Value(s,\zeta) = \infty$.

5.5.3. Caso multi-agente

Extendemos el enfoque de la subsección anterior a escenarios multi-agente. Consideramos modelos que tienen un conjunto finito de agentes (o jugadores), donde cada agente se asocia con una función de costos y toma turnos en cada estado. Estos modelos son básicamente una subclase de las gráficas de juego, aumentadas con una función de etiquetamiento de nodos.

Para el caso con múltiples agentes, enriquecemos la sintaxis de los operadores de comparación de costos de la subsección anterior. En estos nuevos operadores debemos especificar el agente cuyo costo deseamos comparar.

Sean $\mathcal{P} = \{p, q, \ldots\}$ un conjunto de proposiciones atómicas, y $\mathcal{A} = \{1, \ldots, n\}$ un conjunto finito de agentes. Definimos la sintaxis de CTL-con-costos multi-agente de acuerdo con la siguiente gramática BNF:

$$\begin{aligned}
\varphi &::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \sigma \\
\sigma &::= \mathbf{EX}\varphi \mid \mathbf{EG}\varphi \mid \mathbf{E}\,[\varphi\ \mathbf{U}\ \varphi] \mid \zeta \bowtie \zeta \\
\zeta &::= c \mid \mathbf{min}_i\,[\varphi\ \mathbf{U}\ \varphi] \mid \mathbf{max}_i\,[\varphi\ \mathbf{U}\ \varphi] \\
\bowtie &::= < \mid > \mid \leq \mid \geq \mid =
\end{aligned}$$

donde $p \in \mathcal{P}$, $i \in \mathcal{A}$ y $c \in \mathbb{R}^+$.

Las fórmulas de este nuevo lenguaje se interpretan en las estructuras relacionales:

$$\mathfrak{M} \stackrel{\text{def}}{=} (S, R, \ell, \{C_i\}_{i \in \mathcal{A}}, \mathcal{A}, P)$$

donde:

- definimos S, R y ℓ como en las secciones anteriores;
- para todo $i \in \mathcal{A}$, la función total $C_i : S \times S \to \mathbb{R}^+ \cup \{\infty\}$ asigna costos a las transiciones para el agente i, y también, C_i satisface (i) $C_i(s,s) = 0$ para todo $s \in S$, (ii) $C_i(s,t) \in \mathbb{R}^+$ si $(s,t) \in R$, y (iii) $C_i(s,t) = \infty$ si $(s,t) \notin R$;
- la función total $P : S \to \mathcal{A}$ asigna los turnos de los jugadores a los estados.

Dado una ruta $\pi = s_0, \ldots, s_m$ y un agente $i \in \mathcal{A}$, definimos el costo acumulado:

$$Cost_i(\pi) \stackrel{\text{def}}{=} \sum_{k=0}^{m-1} C_i(s_k, s_{k+1})$$

Observemos que el modelo $\mathfrak{M} = (S, R, \ell, \{C_i\}_{i\in\mathcal{A}}, \mathcal{A}, P)$ conforma una gráfica de juego. En consecuencia, podemos utilizar las mismas definiciones de ruta más corta de la Sección 2.

Definimos la relación de satisfacción para este lenguaje de manera similar al lenguaje sin agentes. El fragmento de CTL tiene la semántica usual, y para los operadores de comparación de costos definimos la relación de satisfacción de la siguiente manera:

- $(\mathfrak{M}, s) \models \zeta \bowtie \xi$ sii $Value(s,\zeta) \bowtie Value(s,\xi)$

donde *Value* se define como:

- $Value(s,c) \stackrel{\text{def}}{=} c$ si $c \in \mathbb{R}$;
- $Value(s, \mathbf{min}_i[\psi\ \mathbf{U}\ \varphi]) \stackrel{\text{def}}{=} \text{mín}_\pi\{Cost_i(\pi)\}$ tal que para algún $k \geq 0$, π es una ruta más corta de $\pi[0] = s$ a $\pi[k]$, $\pi[k] \in Sat(\varphi)$, y $\pi[j] \in Sat(\psi)$ para todo $0 \leq j < k$;
- $Value(s, \mathbf{max}_i[\psi\ \mathbf{U}\ \varphi]) \stackrel{\text{def}}{=} \text{máx}_\pi\{Cost_i(\pi)\}$ tal que para algún $k \geq 0$, π es una ruta más corta de $\pi[0] = s$ a $\pi[k]$, $\pi[k] \in Sat(\varphi)$, y $\pi[j] \in Sat(\psi)$ para todo $0 \leq j < k$.

Para computar *Value* podemos proceder como antes, primero condensando el conjunto de destino en un sólo estado. Dados un modelo

$$\mathfrak{M} = (S, R, \ell, \{C_i\}_{i\in\mathcal{A}}, \mathcal{A}, P)$$

y una fórmula CTL-con-costos y agentes φ, definimos el siguiente modelo:

$$\mathfrak{M}|_{\psi \mathbf{U} \varphi} \stackrel{\text{def}}{=} (S', R', \ell', \{C'_i\}_{i \in \mathcal{A}}, \mathcal{A}, P')$$

donde:

- $S' \stackrel{\text{def}}{=} \{s_\varphi\} \cup ((S \setminus Sat(\neg\psi)) \setminus Sat(\varphi))$ con $s_\varphi \notin S$;
- $R' \stackrel{\text{def}}{=} \{(s,t) \mid (s,t) \in R \text{ y } s,t \in S'\} \cup \{(s, s_\varphi) \mid s \in S', t \in Sat(\varphi), \text{ y } (s,t) \in R\} \cup \{(s_\varphi, s_\varphi)\}$;
- $\ell'(s) \stackrel{\text{def}}{=} \ell(s)$ para todo $s \in S'$ y $\ell'(s_\varphi) \stackrel{\text{def}}{=} \bigcup_{s \in Sat(\varphi)} \ell(s)$;
- las funciones de costos son como sigue:
  - $C'_i(s,t) \stackrel{\text{def}}{=} C_i(s,t)$ para todo $s,t \in S'$
  - $C'_i(s_\varphi, s_\varphi) \stackrel{\text{def}}{=} 0$
  - $C'_i(s, s_\varphi) \stackrel{\text{def}}{=} \text{mín}_{t \in Sat(\varphi)}\{C_i(s,t)\}$ si $i = P(s)$
  - $C'_j(s, s_\varphi) \stackrel{\text{def}}{=} \text{mín}_{t \in Sat(\varphi)}\{C_j(s,t)\}$ para $i = P(s)$ y $j \neq i$
- $P'(s) = P(s)$ si $s \in S'$, y $P(s_\varphi) = 1$.

Para calcular las rutas más cortas utilizamos el algoritmo Bellman–Ford extendido que se muestra el Algoritmo 2. En este algoritmo, $dist_i(s)$ es el costo de la ruta más corta desde el estado s hasta de destino d para el agente i, e i^* es un agente distinguido sujeto a la optimización.

Al igual que en el apartado anterior, definimos el cálculo $Value(s, \zeta)$ de la siguiente manera:

- si $\zeta = c$ entonces $Value(s, \zeta) = c$;
- si $\zeta = \mathbf{min}_i[\psi \ \mathbf{U} \ \varphi]$ entonces $Value(s, \zeta) = dist_i(s)$, usando el modelo $\mathfrak{M}|_{\psi \mathbf{U} \varphi}$ y el estado destino s_φ;
- para todo $s \in Sat(\varphi)$ establecemos $Value(s, \zeta) = 0$;
- para todo s in $\mathfrak{M}$ pero no en $\mathfrak{M}|_{\psi \mathbf{U} \varphi}$, establecemos $Value(s, \zeta) = \infty$.

5.5.4. Ejemplos

Planificación y programación

En este apartado, mostramos cómo utilizar nuestra extensión de CTL con un único agente para el análisis de problemas de planificación y programación. Nos centramos en el análisis de proyectos utilizando el método de Ruta crítica (Kelley y Walker [KW59]) en escenarios no probabilísticos (es decir, sin incertidumbre en la duración de las tareas, véase más adelante).

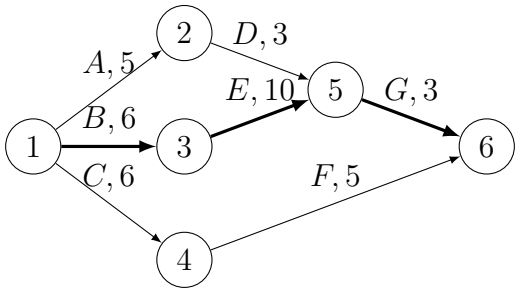
Un proyecto consta de tareas o trabajos ordenados secuencialmente, con algunos trabajos ejecutándose posiblemente en paralelo. Cada trabajo tiene una duración, y algunos trabajos pueden depender de la terminación de otros para comenzar. El método de la ruta crítica analiza un proyecto buscando secuencias de trabajos tales que si no se completan a tiempo, pueden retrasar todo el proyecto. Además, este método busca también los trabajos que pueden retrasarse sin afectar a la finalización total del proyecto.

Un proyecto puede describirse gráficamente utilizando redes de actividades. Estas redes son gráficas acíclicas con pesos, también llamadas *gráficas PERT*. En una gráfica PERT, los nodos representan la finalización de algunos trabajos, y se llaman *hitos*. Los arcos de la gráfica representan los trabajos del proyecto, y asocian a cada trabajo con una duración. La secuencia y la dirección de los arcos representan las dependencias entre los trabajos. Las rutas que no pueden acortarse sin retrasar todo el proyecto se llaman *rutas críticas*. El tiempo extra que algunos trabajos puede tomar sin retrasar el proyecto se llama *tiempo de holgura*.

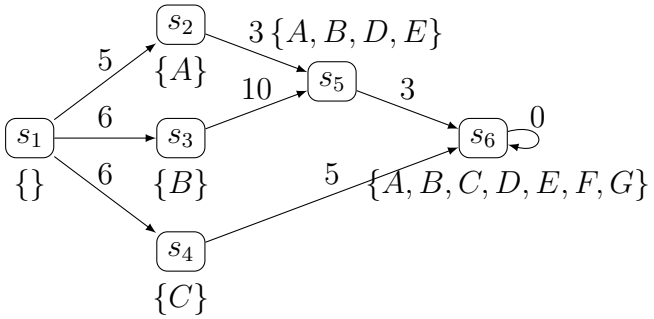
Podemos transformar fácilmente una gráfica PERT en un modelo de CTL-con-costos. La Figura 5.4 muestra la gráfica PERT de un proyecto de ejemplo. El proyecto consta de siete trabajos, de A a G, y seis hitos, de 1 a 6. Cada arco de la gráfica PERT está etiquetada con el nombre del trabajo su duración. La ruta crítica, $1 \to 3 \to 5 \to 6$, tiene una duración de 19 unidades de tiempo, y está enfatizada con flechas gruesas.

Para convertir la gráfica PERT de la Figura 5.4 (a), al modelo de CTL-con-costos de la Figura 5.4 (b), podemos seguir los siguientes pasos:

- creamos un estado s_i para cada hito i;
- añadimos una transición (s_i, s_j) a R sii hay un trabajo X que lleve al proyecto del hito i al hito j (es decir, hay una transición $i \xrightarrow{X} j$ en la gráfica PERT), y establecemos $C(s_i, s_j) = c$ sii c es la duración del trabajo X;



(a)



(b)

Figura 5.4: Un proyecto de ejemplo (a) su gráfica PERT y (b) su modelo de Kripke

- para cada trabajo X, si hay una transición $i \xrightarrow{X} j$, entonces añadimos X a las etiquetas de s_j, y también a las etiquetas de todos los descendientes de s_j;
- añadimos la etiqueta *ini* al primer hito (el estado inicial), y la etiqueta *end* al último hito (el estado final);
- Finalmente, establecemos $C(s, s) = 0$ para todo s, y añadimos (s_m, s_m) a R si m es el último hito.

Por ejemplo, podemos verificar lo siguiente:

$$s_1 \models \bigwedge_{X \in Jobs} \neg X$$

$$s_6 \models \bigwedge_{X \in Jobs} X$$

donde *Jobs* es el conjunto de todos los trabajos del proyecto.

Podemos verificar la duración de la ruta crítica, además de la duración de los tiempos de holgura:

$$s_1 \models \mathbf{max}\,[\mathbf{F}\,end] = 19$$
$$s_1 \models \mathbf{min}\,[\mathbf{F}\,end] = 11$$

Podemos también comparar rutas (posiblemente agregando información al modelo con proposiciones atómicas:

$$s_1 \models \mathbf{max}\,[(ini \vee \neg C)\ \mathbf{U}\ end] > \mathbf{max}\,[(ini \vee C)\ \mathbf{U}\ end]$$

Finalmente, observemos que al verificar una fórmula como:

$$s_1 \models \mathbf{max}\,[\mathbf{F}\,end] > 0$$

podemos usar los métodos descritos en la sección 4.4 para computar la ruta crítica del proyecto, y podemos usar fórmulas similares para computar también otras rutas.

Para verificar la fórmula anterior procedemos como se especifica en la semántica de la Sección 5.2, esto es, computamos $Value(s_1, \mathbf{max}\,[\mathbf{F}\,end]) = Value(s_1, \mathbf{max}\,[\top\ \mathbf{U}\ end])$.

Como se trata de un problema de maximización, invertimos el signo de las funciones de costos. Después, usamos el algoritmo Bellman–Ford como producto de matrices. Los operandos iniciales son la siguiente matriz de adyacencia A y la aproximación inicial D_0:

$$A = \begin{pmatrix} 0 & -5 & -6 & -6 & \infty & \infty \\ \infty & 0 & \infty & \infty & -3 & \infty \\ \infty & \infty & 0 & \infty & -10 & \infty \\ \infty & \infty & \infty & 0 & \infty & -5 \\ \infty & \infty & \infty & \infty & 0 & -3 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix} \quad D_0 = \begin{pmatrix} \infty \\ \infty \\ \infty \\ \infty \\ \infty \\ 0 \end{pmatrix}$$

Aplicamos el producto iterativamente, y alcanzamos el resultado deseado $D = D_3$ después de tres iteraciones:

$$D_1 = \begin{pmatrix} \infty \\ \infty \\ \infty \\ -5 \\ -3 \\ 0 \end{pmatrix} \quad D_2 = \begin{pmatrix} -11 \\ -6 \\ -13 \\ -5 \\ -3 \\ 0 \end{pmatrix} \quad D_3 = \begin{pmatrix} -19 \\ -6 \\ -13 \\ -5 \\ -3 \\ 0 \end{pmatrix}$$

Antes de restaurar los signos de las transiciones, computamos la matriz de rutas más cortas como se describe en la Sección 4.4:

$$A + D^\top = \begin{pmatrix} -19 & -11 & -19 & -11 & \infty & \infty \\ \infty & -6 & \infty & \infty & -6 & \infty \\ \infty & \infty & -13 & \infty & -13 & \infty \\ \infty & \infty & \infty & -5 & \infty & -5 \\ \infty & \infty & \infty & \infty & -3 & -3 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

y entonces aplicamos la operación *BelowThreshold* para obtener la matriz de rutas más cortas σ:

$$\sigma = \mathit{BelowThreshold}(A + D^\top, D)$$
$$= \begin{pmatrix} -19 & \infty & \boxed{-19} & \infty & \infty & \infty \\ \infty & -6 & \infty & \infty & \boxed{-6} & \infty \\ \infty & \infty & -13 & \infty & -13 & \infty \\ \infty & \infty & \infty & -5 & \infty & \boxed{-5} \\ \infty & \infty & \infty & \infty & -3 & \boxed{-3} \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

Las transiciones $1 \to 3 \to 5 \to 6$ de la ruta crítica se marcan con flechas gruesas en la Figura 5.4 (a), y enmarcamos el tiempo restante de cada hito en la matriz anterior (omitimos el paso de restauración de los signos). Al computar esta matriz, también computamos tanto los tiempos de la ruta crítica, como la ruta crítica misma.

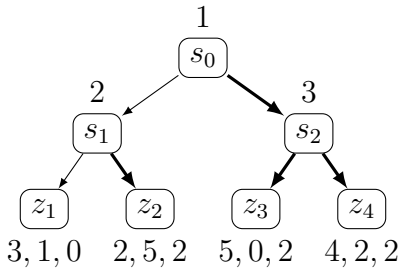
Si verificamos otras fórmulas como las descritas en esta sección, podemos computar los tiempos de otras rutas con tiempos de holgura, y usar la resta aritmética para computar el tiempo de holgura de alguna ruta en particular.

Juegos con información perfecta

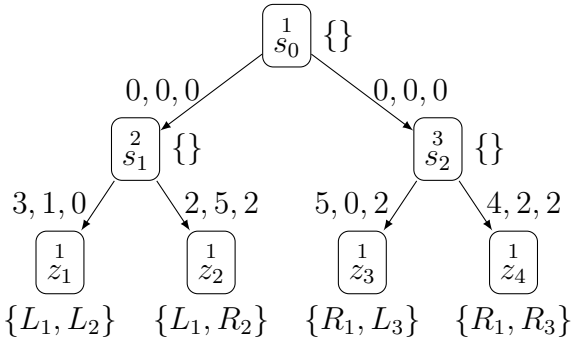
Nuestras gráficas de juego están principalmente motivadas por los juegos en forma extensiva con información perfecta. En esta sección, presentamos un ejemplo sencillo que muestra cómo se puede utilizar nuestro lenguaje para computar y el razonar sobre las soluciones de inducción hacia atrás de estos juegos –véase, por ejemplo, [OR94] para una discusión más amplia acerca de los juegos.

Considere el juego en forma extensiva de la Figura 5.5 (a). Para definir un modelo que represente a este juego –Figura 5.5(b)–, procedemos de la siguiente manera:

- Establecemos una correspondencia uno a uno entre los nodos del árbol y los estados del modelo;
- añadimos una lazo para cada hoja z –omitimos estos lazos en la Figura 5.5(b) por claridad–;
- definimos $C_i(s, s') = 0$ para cada transición (s, s') tal que s no es una hoja; y
- definimos $C_i(s, z) = c$ si el jugador i obtiene c unidades de utilidad al alcanzar z.



(a)



(b)

Figura 5.5: (a) Un juego de tres jugadores y (b) su modelo de Kripke

Podemos extender aún más este modelo codificando información acerca de sus estrategias con proposiciones atómicas. Para este juego, codificamos las estrategias del jugador i con las proposiciones atómicas L_i y R_i. Usando estas proposiciones, caracterizamos las estrategias y perfiles de estrategias. Por

ejemplo:

$$z_1 \models L_1 \wedge \neg R_1$$
$$z_2 \models L_1 \wedge \neg R_1$$
$$z_4 \models R_1 \wedge R_3$$

Podemos verificar la utilidad que obtienen los agentes al seguir dichas estrategias:

$$s_0 \models \mathbf{max}_1[\mathbf{F}(R_1 \wedge R_3)] = 5$$
$$s_0 \models \mathbf{max}_1[\mathbf{F}(L_1)] = 2$$

Incluso, como en el ejemplo anterior, al verificar una fórmula como:

$$a \bowtie \mathbf{max}_1 \left[\mathbf{F}(L_1 \vee L_2 \vee L_3 \vee R_1 \vee R_2 \vee R_3)\right]$$

podemos usar la matriz de rutas más cortas definida en la sección 4.4 para computar la solución del juego, en lugar de sólo dar una caracterización de dicha solución (a y el comparador se pueden elegir arbitrariamente).

Observemos que el juego tiene dos soluciones perfectas por sub-juegos (marcadas con flechas gruesas). Una de estas soluciones es mejor para el agente 1, y la otra para el agente 2. Con los operadores $\mathbf{max}_1$ y $\mathbf{max}_2$ podemos computar la mejor solución para el agente 1 y 2, respectivamente.

Para el cómputo de equilibrios perfectos por subjuegos, normalmente se utiliza el algoritmo de inducción hacia atrás (cf. [OR94]). Este cómputo se puede hacer en tiempo lineal en el tamaño del árbol de juego, pues es equivalente a un recorrido por profundidad. Sin embargo, muchos juegos en la vida real tienen un espacio de estados muy grande. El Ajedrez y Go son ejemplos comunes de este tipo de juegos: el espacio de estados en estos juegos crece exponencialmente en función del número de posibles jugadas (i.e., tienen complejidad espacial $O(2^m)$). Para juegos de suma-cero con dos jugadores es posible optimizar el cómputo usando, por ejemplo, poda Alfa-Beta, y así reducir el espacio de búsqueda hasta $O(2^{m/2})$ (ver [RN03]). Para juegos con información imperfecta, el cómputo de soluciones es computacionalmente más difícil. Gambit [MMT14] es una herramienta que goza de cierta popularidad para el análisis de estos juegos. McKelvey y McLennan [MM96] reseñan los algoritmos implementados en Gambit. Existen algoritmos eficientes para resolver juegos en forma normal o estratégica, y es posible resolver juegos en forma extensiva transformándola a su equivalente en forma estratégica. Sin embargo, esta transformación agrega una penalización exponencial en el tiempo. Es posible también resolver juegos extensivos directamente, pero es un problema

computacionalmente demandante, incluso hasta clases sencillas de juegos son NP-difíciles (ver [MM96]).

Comparado con los métodos mencionados en el párrafo anterior, el método de éste capítulo propone una representación espacial eficiente de los juegos, y puede resultar útil para ciertas clases de juegos.

5.6. Resultados experimentales

Para ilustrar nuestros métodos se implementamos un prototipo en C++ del procedimiento *RelTriangle* y del cómputo de la matriz de rutas más cortas. Nuestro prototipo implementa una representación estándar de MTBDD usando una tabla de nodos (para algunos detalles de implementación de BDD puede consultarse [MT98]). Ejecutamos todas las pruebas en una máquina con 4GB de RAM.

Los casos de prueba consistieron en gráficas de juego con 2^n estados (es decir, tienen matrices de adyacencia de tamaño $2^n \times 2^n$) para $n = 2, \ldots, 17$. Para cada valor de n nos ejecutamos 20 pruebas diferentes generadas semi-aleatoriamente.

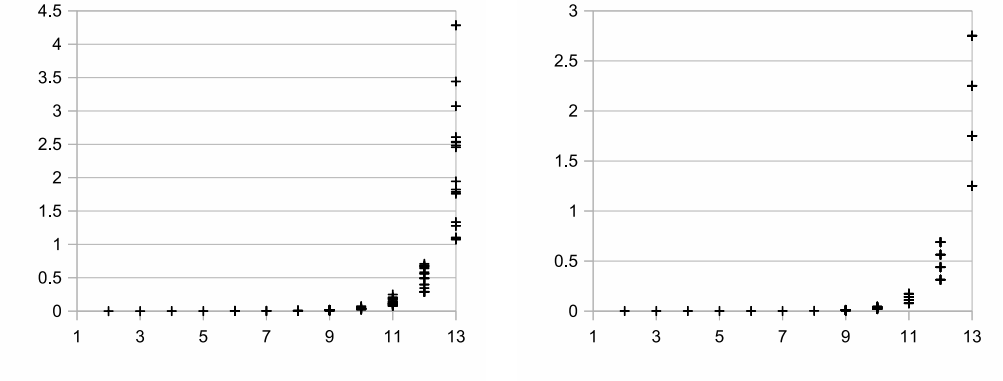
Para hacer una comparación, ejecutamos las mismas pruebas para el algoritmo simbólico y su versión no simbólica, ambas usando el producto de matrices. En la Figura 5.6 se muestran los resultados del rendimiento del algoritmo no simbólico, y en la Figura 5.7 se muestran los resultados del rendimiento del algoritmo simbólico.

El algoritmo no simbólico agota la memoria de la computadora para todas las pruebas con $n > 13$, ya que la mayoría de dichas pruebas requeriría más de 4GB de memoria para almacenar sus matrices de adyacencia. El mayor ejemplo que ejecutamos con el algoritmo simbólico tenía una matriz de adyacencia de tamaño $2^{17} \times 2^{17}$ con cuatro agentes. Teniendo en cuenta que, en este ejemplo, cada celda de la matriz tiene cuatro valores de punto flotante, una representación explícita de la matriz necesitaría 512GB de memoria (este es el límite actual para un equipo que ejecuta Microsoft Windows 8, ver [MSD13]).

La eficiencia espacial del algoritmo simbólico tiene, sin embargo, una penalización de tiempo. Si bien las pruebas del algoritmo no simbólico requieren unos pocos segundos de cálculo, las pruebas más grandes tomaron varias horas utilizando el algoritmo simbólico.

Estas pruebas no son concluyentes sobre la eficiencia de ningún algoritmo. Podemos, sin embargo, utilizar estos resultados para ilustrar el beneficio potencial de los algoritmos simbólicos. Aunque la representación explícita de las matrices es más rápida, podemos ver cómo el uso de BDD permite tratar con

problemas más grandes, que de otra forma tendrían representaciones explícitas costosas.



(a) (b)

Figura 5.6: Resultados del algoritmo no simbólico: (a) Tiempo en segundos de la ejecución de Bellman–Ford extendido (b) Tamaño en GB del proceso residente en memoria (medido por el comando `time` de Unix)

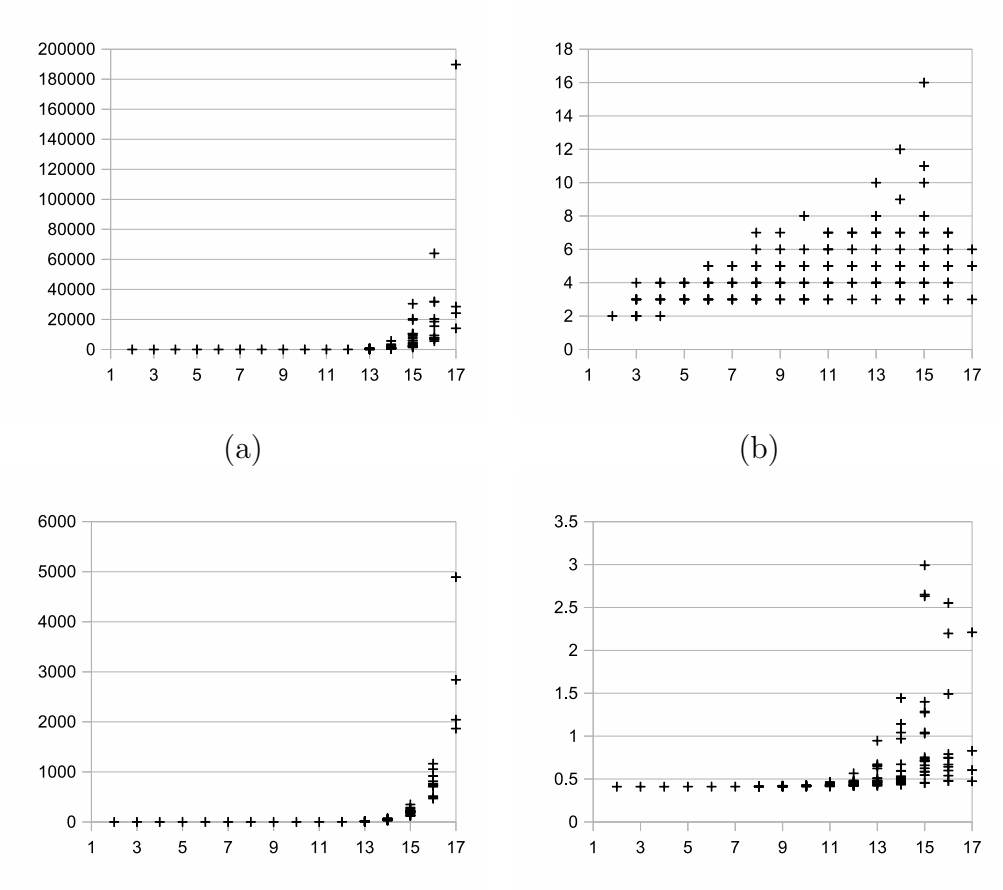


Figura 5.7: Resultados del algoritmo simbólico: (a) Tiempo en segundos de la ejecución de Bellman–Ford extendido (b) Número máximo de iteraciones de *RelTriangle* antes de converger (c) Tiempo en segundos del cálculo de las rutas más cortas a partir del resultado de Bellman–Ford (d) Tamaño en GB del proceso residente (medido por el comando `time` de Unix)

5.7. Conclusiones

Es posible utilizar algoritmos de ruta más corta para la solución de problemas multi-objetivo discretos, y para la búsqueda de soluciones en estructuras de redes similares a juegos –ver el trabajo de Lozovanu y Pickl [LP09], la reseña de Garroppo et al. [GGT10], y las referencias ahí contenidas. Una ventaja de este enfoque es la posibilidad de utilizar versiones simbólicas de estos algoritmos de ruta más corta. Un algoritmo simbólico es capaz de tratar con problemas grandes mediante el uso de estructuras de datos eficientes como los BDD. Algunos algoritmos de ruta más corta ya tienen implementaciones simbólicas basados en BDD, por ejemplo [BFG+97, FMY93, Saw04].

En este capítulo, ampliamos los procedimientos existentes de BDD de [FMY93] usadas en una implementación simbólica de Bellman–Ford. Nuestras extensiones permiten el cálculo de operaciones término-a-término y sensibles a la posición. Tales operaciones son sensibles a la posición de los operandos en una matriz, o, en otras palabras, sensibles a la ruta que conduce a un nodo terminal en un BDD. Utilizamos estos procedimientos extendidos para calcular las rutas más cortas en gráficas de juego. Nuestras gráficas de juego son generalizaciones de juegos finitos, no cooperativos, y con información perfecta. Por lo tanto, nuestro algoritmo simbólico es también aplicable a este tipo de juegos, y equivale a encontrar todos los equilibrios de Nash perfectos por subjuegos. Por otra parte, también proponemos el uso de estos algoritmos –tanto el original como las versiones extendidas, y tanto el simbólico como sus variantes no simbólicas– en la verificación de modelos con CTL. Para este objetivo, presentamos progresivamente dos extensiones CTL enfocadas a expresar especificaciones de gráficas de juego. Por último, reportamos algunos experimentos con una implementación prototipo de nuestros algoritmos extendidos.

Los experimentos que reportamos comparan la versión simbólica con la no simbólica de nuestro algoritmo. No comparamos con otros métodos, como, por ejemplo, los algoritmos de matrices dispersas. La razón es que, en la medida que sabemos, estos algoritmos no se han desarrollado.

Mediante el uso de un algoritmo simbólico para computar soluciones de juegos, esperamos beneficios en ambas áreas. Por ejemplo, podemos aplicar la verificación formal y algoritmos simbólicos a problemas de optimización y de teoría de juegos, o viceversa.

A partir de estos objetivos, podemos articular algunas direcciones para futuras investigaciones. Sería interesante averiguar si es posible aplicar algoritmos similares a otros problemas multi-objetivo o con múltiples restricciones. También resulta interesante la aplicación de algoritmos simbólicos y otras técnicas de verificación formal a otras clases de juegos. Algunos ejemplos de

otras clases de juegos son los juegos con información imperfecta, juegos con iteraciones infinitas, y juegos cooperativos. Por último, también resulta interesante continuar investigando los posibles beneficios del uso de algoritmos y técnicas de optimización aplicados a la verificación de modelos.

6

Conclusiones generales

En esta tesis tomo como punto de partida una línea de investigación de la tradición lógica que estudia la relación entre lógica y juegos. En esta línea de investigación se busca la caracterización lógica de las soluciones de juegos, usando lógicas modales o extensiones de éstas. La mayor parte del trabajo previo en esta tradición basa sus caracterizaciones en la validez formal o deductiva. Desde un punto de vista computacional, esto implica que un método automatizado deberá usar un demostrador automático de teoremas. Desde otro enfoque, se puede usar también un método semántico, esto es, basar las caracterizaciones en satisfacción. De nuevo, desde el punto de vista computacional, el método semántico permite el uso de un verificador de modelos. La ventaja de un verificador de modelos es la posibilidad de trabajar eficientemente con problemas muy grandes, los cuales son comunes en la práctica.

El objetivo principal de la tesis es utilizar el enfoque de verificación de modelos para caracterizar y calcular soluciones de juegos. Para lograr este objetivo propongo tres aproximaciones sucesivas, las cualquier presento en los capítulos 3, 4, y 5.

En el capítulo 3 propongo usar una extensión de la lógica temporal probabilística PCTL para caracterizar equilibrios de Nash en estrategias mixtas. Con estas extensiones, después de describir el juego y su solución propuesta, podemos utilizar un verificador de modelos para determinar si la solución es correcta. Aquí, la contribución consiste, por una parte, en la caracterización en sí, pues hasta donde conozco no existían caracterizaciones lógicas de equilibrios con estrategias probabilísticas. Por otra parte, es también una contribución, como primer acercamiento, el posible uso de un verificador de modelos para determinar la corrección de la solución de un juego.

La segunda aproximación del capítulo 4 se acerca más a la otra parte del objetivo principal de la tesis: usar un verificador de modelos para calcular

soluciones de juegos. En esta aproximación sigo la metodología más común. Parto de un lenguaje lógico, en este caso la lógica dinámica proposicional (PDL), y lo extiendo lo necesario para lograr el objetivo. Exploro la conexión entre una familia de axiomas y la estructura de los equilibrios de Nash. Estos axiomas, llamados de confluencia modal, caracterizan de manera general el razonamiento contrafáctico de los agentes en un juego. Al extender PDL con operadores de la lógica híbrida es posible trasladar el problema de la validez a la satisfacción, y por lo tanto de la demostración automática de teoremas a la verificación de modelos. Más aún, en esta segunda aproximación se vuelve innecesario proponer *a priori* una solución, pues el propio verificador de modelos encontrará todas las posibles soluciones al verificar las fórmulas que propongo.

Una desventaja de extender PDL con operadores de la lógica híbrida es la degradación del desempeño de los algoritmos de verificación. Aunque es posible calcular las soluciones del juego, el procedimiento se vuelve poco viable en la práctica. No obstante, en las conclusiones del capítulo propongo algunas líneas de investigación para enfrentar el problema del desempeño.

Tanto en la lógica como en las ciencias computacionales, es ampliamente conocido que el desempeño computacional se degrada al extender la expresividad de un lenguaje lógico. Por esta razón, en la tercera aproximación del capítulo 5 utilizo una metodología diferente. En este capítulo no parto de la lógica, sino de sus algoritmos de verificación. Una vez obtenido un algoritmo eficiente, extiendo la lógica para aprovechar las capacidades del algoritmo. Las contribuciones de este capítulo son varias. Por una parte, hago patente y exploto la relación existente entre tres problemas que se estudian por separado: el cómputo de rutas más cortas, el cálculo de equilibrios de Nash, y el algoritmo de etiquetamiento de estados. Los tres problemas comparten la misma base estructural.

A partir de esta observación, extiendo el algoritmo Bellman–Ford para calcular rutas más cortas en gráficas con pesos a rutas en gráficas con múltiples pesos, agentes, y turnos, a las que denomino *gráficas de juegos*. Las gráficas de juegos generalizan a una familia de juegos en forma extensiva. Una ruta más corta en una gráfica de juegos es entonces una generalización de un equilibrio de Nash en un juego en forma extensiva. Además, el algoritmo de etiquetamiento de estados que se usa en la verificación de modelos, es también un algoritmo de ruta más corta en gráficas sin pesos. De esta forma, es posible adecuar el algoritmo de etiquetamiento de estados para usar el algoritmo Bellman–Ford, y así construir un verificador para modelos basados en gráficas de juegos. En el último paso de esta metodología extiendo también la lógica temporal CTL. Las extensiones a CTL que propongo describen la estructura de las gráficas de

juegos. Como resultado, una vez descrito el juego como una gráfica podemos utilizar el verificador de modelos para calcular equilibrios de Nash de forma eficiente, además de calcular y decidir otras propiedades.

De manera más general, vale la pena destacar algunas observaciones derivadas de la evolución del proceso metodológico que seguí en la tesis. Como ya lo mencioné, es bien sabido la degradación del desempeño computacional que surge al extender los lenguajes lógicos. Sin embargo, la metodología tradicional para resolver problemas con lógica es normalmente sintáctica en primera instancia, semántica en segunda instancia, y algorítmica al final. Esto implica mayor complejidad computacional, y también, en muchas ocasiones, mayor complejidad matemática. Dado el desarrollo actual de la lógica y de sus algoritmos, es posible que en algunos casos, como en el de esta tesis, la metodología conversa resulte más favorable, sobre todo si se tiene en mente la aplicación computacional. Esto es, trabajar inicialmente desde la semántica, y sobre todo, desde los algoritmos, para pasar finalmente a la parte sintáctica.

Para lograr un buen desarrollo de esta metodología conversa, puedo rescatar un aspecto que encontré en el caso particular de este trabajo de investigación. En la tesis trabajo con lenguajes y algoritmos que cuentan actualmente con un desarrollo maduro. Esto permite tener una visión más clara de la cadena algoritmo-semática-sintaxis. Con este panorama, resulta más sencillo encontrar analogías y conexiones entre otros problemas y algún eslabón particular, además de sus implicaciones al el resto de la cadena. De esta manera, la solución de un problema puede tener cierta libertad metodológica, resolviendo primero los aspectos de mayor interés y extendiendo su alcance a los demás ámbitos que le circundan.

Bibliografía

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[AtC07] Carlos Areces and Balder ten Cate. Hybrid logics. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*. Elsevier, 2007.

[BB12] R. Berghammer and S. Bolus. On the use of binary decision diagrams for solving problems on simple games. *European Journal of Operational Research*, 2012.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142 – 170, 1992.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*. Cambridge University Press, New York, NY, USA, 2001.

[BFG+97] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10:171–206, 1997.

[BFW06] Paolo Ballarini, Michael Fisher, and Michael Wooldridge. Automated game analysis via probabilistic model checking: a case study. In *Proceedings of the Third Workshop on Model Checking and Artificial Intelligence*, pages 125–137, 2006.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[Bol11] S. Bolus. Power indices of simple games and vector-weighted majority games by means of binary decision diagrams. *European Journal of Operational Research*, 210(2):258–272, 2011.

[Bon01] Giacomo Bonanno. Branching time, perfect information games, and backward induction. *Games and Economic Behavior*, 36:57–73, 2001.

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

[CE82] Edmund Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Proc Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer Berlin / Heidelberg, 1982. 10.1007/BFb0025774.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd. Ed.* MIT Press, 2009.

[CMZ+93] E.M. Clarke, K.L. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. pages 54 – 60, june 1993.

[DB95] Ashvin Dsouza and Bard Bloom. Generating BDD models for process algebra terms. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 16–30, London, UK, UK, 1995. Springer-Verlag.

[DCDS01] Pallab Dasgupta, P. P. Chakrabarti, Jatindra Kumar Deka, and Sriram Sankaranarayanan. Min-max computation tree logic. *Artificial Intelligence*, 127(1):137 – 162, 2001.

[EFT92] Reinhard Enders, Thomas Filkorn, and Dirk Taubner. Generating BDDs for symbolic model checking in CCS. In *Proceedings of the 3rd International Workshop on Computer Aided Verification*, CAV '91, pages 203–213, London, UK, UK, 1992. Springer-Verlag.

[FdR06] Massimo Franceschet and Maarten de Rijke. Model checking hybrid logics (with an application to semistructured data). *Journal of Applied Logic*, 4(3):279–304, 2006.

[FMY93] Masahiro Fujita, Patrick C. McGeer, and Jerry Chih-Yuan Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *ICCAD-93*, 1993.

[GGT10] Rosario G. Garroppo, Stefano Giordano, and Luca Tavanti. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Comput. Netw.*, 54(17):3081–3107, 2010.

[GR09] Pedro A. Góngora and David A. Rosenblueth. A characterization of mixed-strategy Nash equilibria in PCTL augmented with a cost quantifier. In *Proceedings of the 10th International Workshop on Computational Logic in Multi-Agent Systems*, pages 139–155, Germany, 2009. Institut für Informatik, Technische Universität Clausthal.

[GR14] Pedro A. Góngora and David A. Rosenblueth. A symbolic shortest path algorithm for computing subgame-perfect nash equilibria. *Int. Journal of Applied Mathematics and Computer Science (por publicarse)*, 2014.

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:102–111, 1994.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000.

[HMS99] H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In B. Plateau, W. J. Stewart, and M. Silva, editors, *3rd Int. Workshop on the Numerical Solution of Markov Chains, Zaragoza, Spain*, pages 188–207. Prensas Universitarias de Zaragoza, 1999.

[HvdHMW03] Paul Harrenstein, Wiebe van der Hoek, John-Jules Ch. Meyer, and Cees Witteveen. A modal characterization of Nash equilibrium. *Fundamenta Informaticae*, 57(2-4):281–321, 2003.

[Jam08] Wojciech Jamroga. A temporal logic for multi-agent mdp's. In *Proceedings of the AAMAS Workshop on Formal Models for Multi-Robot Systems*, pages 29–34, 2008.

[Kri63] Saul A. Kripke. Semantical analysis of modal logic I. *Zeitschr. Math. Logik Grund. Math*, 9:67–96, 1963.

[Kuh53] Harold W Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games*, 2(28):193–216, 1953.

[KW59] James E. Kelley, Jr and Morgan R. Walker. Critical-path planning and scheduling. In *Papers presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '59 (Eastern), pages 160–173, New York, NY, USA, 1959. ACM.

[LP09] Dmitrii Lozovanu and Stefan Pickl. *Optimization and Multiobjective Control of Time-Discrete Systems*. Springer, 2009.

[MM96] Richard D McKelvey and Andrew McLennan. Computation of equilibria in finite games. In H. M. Amman, D. A. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, volume 1 of *Handbook of Computational Economics*, chapter 2, pages 87–142. Elsevier, 00 1996.

[MMT14] Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory, version 13.1.2. *http://www.gambit-project.org*, 2014.

[MSD13] MSDN. Memory limits for windows releases. Microsoft Developer Network `http://msdn.microsoft.com/en-us/library/windows/desktop/aa366778(v=vs.85).aspx`, 2013.

[MT98] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VSLI Design: OBDD - Foundations and Applications*. Springer-Verlag, 1998.

[Nas50] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, 1950.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, Cambridge, Massachusetts, 1994.

[Pop94] Sally Popkorn. *First Steps in Modal Logic*. Cambridge University Press, Cambridge, England, 1994.

[Pri57] Arthur N. Prior. *Time and modality*. Oxford University Press, Oxford, UK, 1957.

[PT91] Solomon Passy and Tinko Tinchev. An essay in combinatory dynamic logic. *Information and Computation*, 93:263–332, 1991.

[PW03] Marc Pauly and Michael Wooldridge. Logic for mechanism design — a manifesto. In *In Proceedings of the 2003 Workshop on Game Theory and Decision Theory in Agent Systems (GTDT-2003)*, 2003.

[RL07] Franco Raimondi and Alessio Lomuscio. Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *Journal of Applied Logic*, 5(2):235 – 251, 2007.

[RN03] Suart Russell and Peter Norvig. *Artificial Intelligence. "A modern approach"*. Prentince Hall, 2003.

[Saw04] Daniel Sawitzki. Experimental studies of symbolic shortest-path algorithms. In Celso C. Ribeiro and Simone L. Martins, editors, *Experimental and Efficient Algorithms*, volume 3059 of *Lecture Notes in Computer Science*, pages 482–497. Springer, 2004.

[TvdHW08] Nicolas Troquard, Wiebe van der Hoek, and Michael Wooldridge. Model checking strategic equilibria. In Doron Peled and Michael Wooldridge, editors, *MoChArt*, volume 5348 of *Lecture Notes in Computer Science*, pages 166–188. Springer, 2008.

[vB05] Johan van Benthem. Open problems in logic and games. In S. Artemov, H. Barringer, A. d'Avila Garcez, L. Lamb, and

J. Woods, editors, *Essays in Honour of Dov Gabbay*, pages 229–264, London, 2005. King's College Publications.

[vB12] Johan van Benthem. In praise of strategies. In Jan van Eijck and Rineke Verbrugge, editors, *Games, Actions and Social Software*, volume 7010 of *Lecture Notes in Computer Science*, pages 96–116. Springer Berlin Heidelberg, 2012.

[vBGR08] Johan van Benthem, Patrick Girard, and Olivier Roy. Everything else being equal: A modal logic for ceteris paribus preferences. *Journal of Philosophical Logic*, 38(1):83–125, 2008.

[vBar] Johan van Benthem. *Logic in Games*. Amsterdam University Press, to appear.

[vdHJW05] Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. A logic for strategic reasoning. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 157–164, New York, NY, USA, 2005. ACM.

[vdHRW07] Wiebe van der Hoek, Mark Roberts, and Michael Wooldridge. Social laws in alternating time: Effectiveness, feasibility, and synthesis. *Synthese*, 156(1):1–19, 2007.

[Ven05] G. Venkatesh. Reasoning about game equilibria using temporal logic. *Lecture Notes in Computer Science*, (3328), 2005.

[vNM44] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.