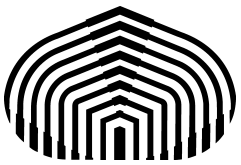

Universidad Simón Bolívar
Decanato de Estudios Profesionales
Coordinación de Ingeniería de la Computación

Reducciones automáticas de problemas de decisión a problemas de planificación

Por:
Aldo Fabrizio Porco Rametta
Alejandro Machado González

Realizado con la asesoría de:
Prof. Blai Bonet

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de Computación

Sartenejas, febrero de 2013

Resumen

Este trabajo tiene como objetivo el desarrollo de una herramienta que permita establecer una correspondencia entre problemas de decisión expresados en un lenguaje descriptivo y problemas de planificación automática. Notando que existen problemas que son modelables más naturalmente utilizando la lógica, la motivación del trabajo es proporcionar una forma de solucionar estos problemas especificados de una manera declarativa.

Para lograr esta meta se diseñaron dos reducciones orientadas específicamente a las clases de complejidad NP y PH, utilizando los formalismos de la Teoría de Complejidad Descriptiva que establece equivalencias entre lógicas de distintos poderes expresivos con clases de complejidad computacionales. Como los problemas de planificación automática son en general PSPACE-completos, es de interés observar que la primera traducción produce problemas de planificación restringidos que pertenezcan también a la clase NP, de modo que la resolución de problemas no sea más compleja de lo necesaria a causa de la reducción.

La herramienta desarrollada consta de un analizador sintáctico de un lenguaje basado en la lógica de segundo orden y de una serie de recorridos del árbol sintáctico para generar un problema de planificación STRIPS, expresado en lenguaje PDDL. Un aspecto importante de la herramienta es que cumple con las propiedades formales postuladas en el Teorema 2.1, que acotan su complejidad.

Se modelaron y solucionaron problemas de varios dominios y tamaños a modo de probar la efectividad de un planificador basado en SAT para resolver los problemas traducidos. Los resultados fueron satisfactorios para problemas pequeños: el planificador logró resolver la mayoría de ellos en menos de cinco minutos.

Parte de este trabajo fue presentado en la *21st International Conference on Automated Planning and Scheduling* (ICAPS 2011), y los investigadores del área han reconocido que los dominios generados por la herramienta son de interés práctico, pues se conocen sus propiedades formales y es posible estudiarlos a fondo para descubrir mejores heurísticas y optimizaciones para los planificadores.

Palabras clave: Planificación Automática, Teoría de Complejidad, Lógica, Ingeniería del Conocimiento, Inteligencia Artificial.

Índice general

Resumen i

Índice de Figuras v

Lista de Tablas vi

Acrónimos vii

Introducción 1

1. Marco teórico 4
1.1. Planificación automática 4
1.1.1. Complejidad en planificación 8
1.2. Complejidad Descriptiva 9
1.2.1. Lenguajes 9
1.2.1.1. Lógica de segundo orden 10
1.2.2. Estructuras de primer orden 11
1.2.3. Semántica formal 12
1.2.4. Abreviaciones sintácticas 14
1.2.5. Clases de complejidad 15
1.3. Modelación de problemas 15

2. Traducción de problemas NP 17
2.1. Perspectiva general 17
2.2. Reducción de LSO a STRIPS 18
2.2.1. Reducción del dominio 19
2.2.1.1. Acciones para las variables 19
2.2.1.2. Acciones para las subfórmulas 20
2.2.1.3. Otras acciones 21
2.2.1.4. Abreviaciones 21
2.2.2. Problema 22
2.3. Propiedades formales 22
2.3.1. Demostración de reducción 23
2.3.2. Tiempo de corrida de la reducción 27

2.3.3. Complejidad de los problemas generados 28
2.4. Diseño del lenguaje 29
2.4.1. Especificación de la fórmula Φ 29
2.4.2. Especificación de la firma σ 30
2.4.3. Especificación de la estructura $\mathcal{A}$ 31
2.5. Implementación de la herramienta 31
2.6. Simplificación de la traducción 33

3. Traducción de problemas PH **34**
3.1. Complejidad 34
3.2. Traducción a STRIPS 35
3.2.1. Sistema de tipos 35
3.2.2. Traducción del operador `so-forall` 37
3.2.3. Traducción del operador `so-exists` 39

4. Experimentos y resultados **41**
4.1. Escogencia del planificador 41
4.1.1. Ventanas de horizonte para la traducción de NP 42
4.1.2. Ventanas de horizonte para la traducción de PH 43
4.2. Diseño de los experimentos 44
4.2.1. Procedimiento 44
4.3. Experimentos y resultados de problemas NP 44
4.3.1. SAT 45
4.3.2. Problemas de grafos y 3DM 45
4.3.3. Número cromático 48
4.4. Experimentos y resultados de problemas PH 49
4.4.1. QBF 50
4.4.2. $\overline{\text{3Col}}$ 52
4.5. La herramienta en la web 53

5. Conclusiones y Recomendaciones **55**

A. Problemas NP y PH modelados en lógica **60**
A.1. Problemas NP 60
A.1.1. SAT, Satisfacibilidad proposicional 60
A.1.2. CLIQUE, Clique de tamaño k 61
A.1.3. CHD, Camino Hamiltoniano Dirigido 61
A.1.4. 3DM, *3-Dimensional Matching* 62
A.1.5. 3Col, 3-colorabilidad 63
A.1.6. kCol, k-colorabilidad 64
A.2. Problemas PH 65
A.2.1. UNSAT, No-Satisfacibilidad proposicional 65
A.2.2. $\exists\forall$-QBF, Fórmula *booleana* cuantificada Σ_p^2 65

A.2.3. $\overline{\text{3Col}}$, No-3-Colorabilidad 66

B. Caso de estudio: SAT **68**
B.1. Dominio 68
B.1.1. Entrada 68
B.1.2. Salida 69
B.2. Problema 73
B.2.1. Entrada 73
B.2.2. Salida 73
B.3. Solución 74

Índice de figuras

1.1. Ejemplo de estado inicial en *mundo de bloques* 6
1.2. Estado final en *mundo de bloques* 7
1.3. Grafo dirigido correspondiente a una estructura $\mathcal{A}$ 11
1.4. Grafo dirigido coloreado por una relación 14

2.1. Esquema de operación de la herramienta. 18
2.2. Gramática en BNF del lenguaje lógico 30

3.1. Acciones para realizar la iteración del cuantificador universal de segundo orden 38
3.2. Acciones que implementan la nueva traducción para el cuantificador existencial en SAT. 40

A.1. Grafo con una *clique* de tamaño $k = 6$ 61
A.2. Grafo con *camino hamiltoniano* 62
A.3. Ejemplo de *3-Dimensional Matching*, (Wikipedia, 2004) 63
A.4. Grafo con *3 Colorabilidad* 64
A.5. Ejemplo de grafo *No-3-Coloreable* 67

B.1. Arbol sintáctico de Φ_{SAT} 72

Lista de Tablas

4.1. Resultados de M para Sat 45
4.2. Resultados de M para Clique 46
4.3. Resultados de M para CHD 46
4.4. Resultados de M para 3DM 47
4.5. Resultados de M para 3Col 47
4.6. Resultados de M para kCol 48
4.7. Resultados de M sobre números cromáticos 49
4.8. Resultados de LAMA'11 para $\overline{\text{3Col}}$ 53

Acrónimos

3Col	**3-Col**orabilidad
3DM	**3-D**imensional **M**atching
CHD	**C**amino **H**amiltoniano **D**irigido
BNF	**B**ackus-**N**aur **F**orm
CNF	**C**onjunctive **N**ormal **F**orm
kCol	k-**Col**orabilidad
LPO	**L**ógica de **P**rimer **O**rden
LSO	**L**ógica de **S**egundo **O**rden
NP	**N**ondeterministic **P**olynomial time (clase de complejidad)
PDDL	**P**lanning **D**omain **D**efinition **L**anguage
PH	**P**olynomial **H**ierarchy (clase de complejidad)
QBF	**Q**uantified **B**oolean **F**ormula
SAT	boolean **SAT**isfiability
STRIPS	**S**tanford **R**esearch **I**nstitute **P**roblem **S**olver
TCD	**T**eoría de **C**omplejidad **D**escriptiva

$\Longleftrightarrow$	doble implicación, si y sólo si
$\Rightarrow$	implicación lógica
$[u := v]$	sustitución textual de u por v

Introducción

En Ciencias de la Computación se entiende por **problemas de decisión** los problemas en los que se plantea una pregunta en un sistema formal sobre una entrada arbitraria, cuya respuesta puede ser afirmativa o negativa. Los problemas de decisión son interesantes porque permiten modelar y razonar sobre aplicaciones reales. Por ejemplo, el problema de decisión de k-colorabilidad pregunta si los vértices de un grafo pueden ser coloreados con k colores distintos, de modo de que a ningún par de vértices conectados por una arista se les asigne el mismo color . En un compilador, la tarea de asignar los valores frecuentemente utilizados a registros para mejorar la eficiencia del código generado puede modelarse con un problema de k-colorabilidad: los vértices del grafo son las variables, y si dos variables se necesitan al mismo tiempo se coloca un arco entre ellas. Al encontrar una forma de colorear este grafo con k colores se ha hallado una forma de utilizar k registros distintos para almacenar las variables.

Los problemas de decisión pueden agruparse en distintas clases según su complejidad. Una importante clase de complejidad es NP (siglas de *Nondeterministic Polynomial time*), a la cual pertenecen problemas interesantes como k-colorabilidad, satisfacibilidad proposicional (**SAT**) y hallar un camino *hamiltoniano* en un grafo. Otra clase más general de problemas de interés para este trabajo es PH (*Polynomial Hierarchy*), que incluye problemas computacionalmente más complejos como formas restringidas de *Quantified Boolean Formula* (QBF).

El área de investigación de planificación automática estudia una forma general de resolución de problemas utilizando **planes**. Un problema de planificación consiste de un estado inicial, un conjunto de estados finales y un conjunto de acciones posibles. Luego, la tarea del **planificador** consiste en encontrar una secuencia de acciones que conduzca el estado inicial hasta un estado final, o determinar que tal

secuencia de acciones no existe.

Este trabajo se enfoca en el desarrollo de una herramienta que permite **reducir** un problema de decisión perteneciente a las clases NP o PH codificado en lógica de segundo orden, a un problema de planificación automática. La motivación para realizar esta herramienta es que existen planificadores muy eficientes, pero no es trivial expresar un problema de decisión cualquiera en términos de estado inicial, estados finales y acciones, como lo requieren los planificadores. A veces, la utilización de un lenguaje declarativo es una forma más concreta y directa para la descripción de un problema de decisión en particular. En estos casos, es preferible modelarlo en un lenguaje de más alto nivel y que la herramienta lo reduzca a un formato de entrada aceptado por los planificadores, los cuales pueden proceder entonces a computar su solución.

Otros autores ya han utilizado fragmentos de la lógica de segundo orden como un lenguaje de programación declarativo. En la obra de Reiter (1978) se describe un lenguaje de reglas y consultas a bases de datos que es un subconjunto de PROLOG, llamado DATALOG. A su vez, Cadoli et al. (1999) presentan una extensión de este lenguaje que permite la codificación de problemas NP y se desarrolla un método para compilar los problemas a código de PROLOG, el cual fue probado con pequeñas instancias del **problema de la mochila** (*knapsack problem*) y el **problema del viajante** (*traveling salesman problem*).

Mitchell y Ternovska (2005) proponen, por su parte, el lenguaje de Extensión de Modelos (MX), que es conceptualmente muy cercano a la lógica de segundo orden. Este trabajo se enfoca en la descripción formal del lenguaje y en discutir la posibilidad de diseñar un solucionador. En esta sección, los autores aconsejan la realización de una traducción a otro lenguaje para el cual existan solucionadores: es precisamente esto lo que se ha hecho en el presente trabajo con la función que traduce descripciones declarativas a problemas de planificación, área para la cual existe una amplia variedad de solucionadores.

Otro trabajo relacionado a este proyecto es el de Cadoli y Mancini (2006). Los autores presentan una manera de razonar sobre los problemas de decisión, intentando reformularlos para hacerlos más fáciles de resolver. Se utilizan sistemas especializados para la modelación de restricciones y solucionadores provenientes de muchos sub-campos de la Inteligencia Artificial. Sin embargo, dependiendo del

solucionador, se podría requerir que la fórmula declarativa a solucionar esté en un formato particular, lo cual no es tan conveniente como el enfoque "independiente del dominio" que provee la planificación automática.

El objetivo general del proyecto es permitir la resolución de problemas NP y de la jerarquía polinomial (PH) expresados a través de un lenguaje declarativo basado en la lógica de segundo orden. Así, problemas cuya aplicación es ampliamente conocida en Ciencias de la Computación podrán ser expresados en un lenguaje conciso y ser resueltos sin la intervención del usuario luego de la fase de modelación.

A fin de lograr el objetivo general, se plantean las siguientes metas específicas:

- El desarrollo de una herramienta que procese la descripción lógica de un problema en NP o PH y genere automáticamente un problema de planificación equivalente formulado en el lenguaje PDDL (*Planning Domain Definition Language*), de uso estándar en la comunidad de planificación automática.
- Una interfaz *web* para este sistema que permita un acceso flexible y rápido al traductor.

Este trabajo está dividido en cinco capítulos. El primer capítulo aporta las bases teórica sobre la que está fundamentada la traducción, incluyendo una introducción a las áreas de planificación automática y complejidad descriptiva. En el segundo capítulo se describe el lenguaje lógico diseñado para la formulación de problemas a alto nivel, se presenta una reducción base del fragmento de la lógica de segundo orden que captura a la clase NP al lenguaje PDDL y se demuestran las propiedades formales de esta traducción. El capítulo 3 presenta una extensión de la herramienta que permite traducir problemas más generales, pertenecientes a la jerarquía polinomial. El capítulo 4 trata sobre cómo fueron diseñados y llevados a cabo experimentos sobre diferentes problemas en NP y PH traducidos con la herramienta, además de ofrecer una discusión de los resultados obtenidos. Finalmente, el capítulo 5 contiene conclusiones e ideas para trabajos futuros.

Parte de este trabajo ha sido publicado en la *21st International Conference on Automated Planning and Scheduling* (ICAPS 2011; Porco, Machado, y Bonet), y enviado como artículo de investigación a la *23rd International Conference on Automated Planning and Scheduling* (ICAPS 2013).

Capítulo 1

Marco teórico

Este capítulo presenta brevemente los conceptos teóricos básicos para el desarrollo de este trabajo. Primero se explican los fundamentos de la planificación automática, área que provee un formalismo al cual se pueden traducir problemas de decisión. Luego, se define la disciplina de complejidad descriptiva, que determina la clase de complejidad computacional de distintos fragmentos de lógica de segundo orden según su expresividad. Finalmente, se explican dos ejemplos de problemas NP fácilmente modelables con el formalismo lógico propuesto.

1.1. Planificación automática

La planificación automática es un campo de investigación dentro de la Inteligencia Artificial. Su enfoque se basa en descubrir una secuencia de acciones a seguir para lograr un objetivo concreto (Russell y Norvig, 2010). En este trabajo se manejará la concepción **clásica** de planificación, en la cual todas las acciones tienen una consecuencia determinística y observable en ambientes estáticos (es decir, que sólo cambian con el resultado de las acciones aplicadas).

Fikes y Nilsson (1971) fueron pioneros en el área de planificación con su sistema STRIPS (*Stanford Research Institute Problem Solver*). El acrónimo STRIPS se ha empleado desde entonces para referirse al lenguaje *de facto* que se utiliza para expresar problemas de planificación.

Los investigadores de esta área han diseñado una representación compacta para expresar problemas de planificación de cualquier dominio, llamado PDDL (*Planning Domain Definition Language*). En este trabajo se considera el fragmento más simple de PDDL, que es equivalente a STRIPS. Este fragmento describe todo lo necesario para especificar un problema: el estado inicial, las acciones posibles con sus precondiciones y efectos, y los estados meta.

La Definición 1.1 presenta de manera formal qué es un problema de planificación. Los párrafos siguientes contienen una explicación intuitiva del problema.

Definición 1.1. Matemáticamente, se define un problema de planificación STRIPS como $P = \langle C, A, I, F \rangle$, donde:

- C es un conjunto de condiciones o **proposiciones**.
- A es un conjunto de **acciones**. Cada acción a es una tupla $a = \langle pre, adds, deletes \rangle$ que contiene el conjunto de precondiciones (condiciones que deben ser ciertas para poder aplicar la acción) y el conjunto de efectos positivos (*adds*) y negativos (*deletes*) que causa la aplicación de la acción.
- $I \subseteq C$ es el **estado inicial**. Los estados son especificados por el conjunto de condiciones que son verdaderas en ellos, las demás condiciones se suponen falsas. Esto se llama la suposición del **mundo cerrado** (Russell y Norvig, 2010).
- $F \subseteq C$ describe los **estados finales** o meta.

Cada estado se representa con un conjunto de proposiciones, representadas internamente como variables proposicionales pero expresadas frecuentemente como símbolos relacionales instanciados. Por ejemplo, en un problema de planificación canónico llamado *mundo de bloques*, una mano robótica intenta apilar bloques enumerados de manera de que todos los bloques estén en una sola pila ordenados de forma ascendente. Cada bloque puede tener a lo sumo un bloque encima suyo, y la mano sólo puede cargar un bloque a la vez. En este caso, proposiciones válidas que pueden ser parte de un estado son, por ejemplo, `manoVacía`, `sobreMesa(bloque0)`, `sobreMesa(bloque1)`, `libre(bloque0)`, `libre(bloque1)`. La Figura 1.1 ilustra un estado cuyas únicas proposiciones ciertas son estas mencionadas anteriormente.

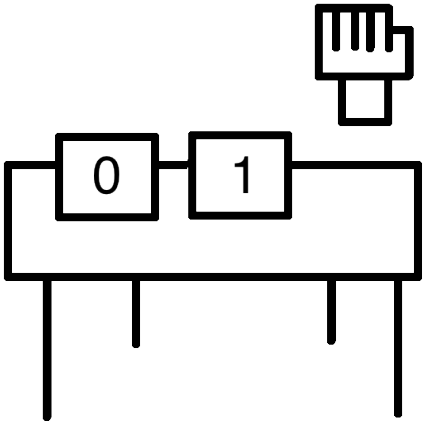

FIGURA 1.1: Posible estado de un problema en *mundo de bloques*.

Una acción en PDDL especifica qué debe ser cierto sobre el estado actual para que la acción sea aplicable. Este conjunto comprende las **precondiciones**. Asimismo, la acción declara qué proposiciones en el mundo cambian cuando es aplicada, lo cual constituye sus **efectos**. Para una notación simplificada, PDDL permite la utilización de **parámetros** en los símbolos relacionales que describen sus precondiciones y efectos. Estos parámetros pueden ser instanciados con cualquier objeto. Volviendo al ejemplo anterior, un conjunto de acciones podría ser:

```
(:action recoger_bloque
    :parameters (?b)
    :precondition (and (manoVacía) (libre ?b))
    :effect (and (enMano ?b) (not (manoVacía)) (not (sobreMesa ?b)))
)

(:action colocar_sobre
    :parameters (?b1 ?b2)
    :precondition (and (enMano ?b1) (libre ?b2))
    :effect (and (sobre ?b1 ?b2) (manoVacía) (not (enMano ?b1))
            (not (libre ?b2)))
)
```

Teniendo en cuenta esta especificación, el lector puede hacer el ejercicio de generar una secuencia ordenada de acciones que resuelva este ejemplo concreto, es decir, que conduzca desde el estado ilustrado en la Figura 1.1 hasta un estado en el cual los bloques estén "apilados en orden". La solución se presenta a continuación.

Como es costumbre, sólo se mencionan las proposiciones verdaderas en cada estado. Se ha coloreado de azul las nuevas proposiciones que han resultado de aplicar una acción (*adds*), y de rojo a las proposiciones que dejaron de ser ciertas en el siguiente estado producto de la acción aplicada (*deletes*).

```
estado: {sobreMesa(bloque0),  sobreMesa(bloque1),
         libre(bloque0), libre(bloque1),  manoVacía}
acción: recoger_bloque(bloque1)
estado: {enMano(bloque1), sobreMesa(bloque0),
         libre(bloque0), libre(bloque1)}
acción: colocar_sobre(bloque1, bloque0)
estado: {sobre(bloque1, bloque0), sobreMesa(bloque0),
          manoVacía, libre(bloque1)}
```

La Figura 1.2 es una representación de este estado final.

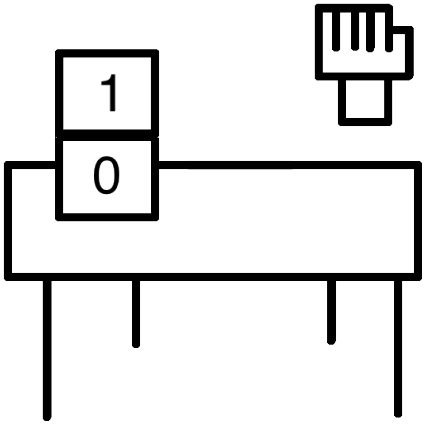

FIGURA 1.2: Estado final para un problema en *mundo de bloques*, luego de aplicar una secuencia de acciones al estado en la Figura 1.1

En resumen, un **problema de planificación** es expresado utilizando el lenguaje PDDL, que especifica su **estado inicial**, **acciones** (precondiciones y efectos de cada una) y los **estados finales**. Un **plan** es una secuencia ordenada de acciones que resuelve el problema de planificación.

Cuando se busca resolver varias instancias de un problema general se suelen agrupar los problemas de planificación en dominios. Dos problemas del mismo dominio comparten el mismo conjunto de acciones, mas no necesariamente el estado inicial o los estados meta. Por esto se dice que *mundo de bloques* es un **dominio**

de planificación, y el término **problema** se refiere a una instancia particular con estados inicial y final definidos, en el cual pueden utilizarse las acciones del dominio al que pertenece.

Un problema de planificación en PDDL es un par $\langle \mathsf{dom}, \mathsf{ins} \rangle$ con una descripción de un dominio y una instancia en el lenguaje PDDL (McDermott et al., 1998; Fox y Long, 2001). Como PDDL permite expresar las acciones parametrizadas en el dominio para ajustarlas a cualquier instancia, los planificadores deben implementar una función de instanciación $\mathfrak{G}$ que transforme la especificación en PDDL $\langle \mathsf{dom}, \mathsf{ins} \rangle$ en un problema STRIPS, en el que las acciones **no tienen parámetros**. Por ejemplo, en el problema STRIPS $\mathfrak{G}(\mathsf{dom}, \mathsf{ins})$ que corresponde a la instancia de *mundo de bloques* estudiada anteriormente, existen acciones `recoger_bloque_0` y `recoger_bloque_1`, así como operadores `colocar_sobre` para todos los pares ordenados de bloques.

1.1.1. Complejidad en planificación

Sea PLANSAT el problema de decisión que pregunta si existe un plan que resuelve un problema de planificación dado. Según el formalismo que se ha definido en la sección anterior, PLANSAT es **decidible** porque el número de estados es finito: un algoritmo basado en búsqueda desde el estado inicial que aplique todas las acciones posibles en cada estado acabará examinando todo el espacio y determinando si existe tal plan.

PLANSAT pertenece a la clase de complejidad PSPACE (Bylander, 1994). PSPACE es la clase de problemas que pueden ser resueltos por una máquina de Turing determinística con una cantidad polinomial (en n, donde n es el tamaño de su entrada) de espacio en memoria. Los problemas PSPACE son, en general, más difíciles de resolver que los problemas NP, los cuales a su vez son problemas combinatorios complejos para los cuales se cree que no existen soluciones generales eficientes. Como en muchos otros dominios de inteligencia artificial, los planificadores dependen de heurísticas de búsqueda para dar con las soluciones a problemas pequeños en un período de tiempo razonable.

Una restricción interesante a PLANSAT es no permitir efectos negativos en las

acciones. Si este es el caso, cualquier predicado instanciado que se haya agregado permanece cierto durante todo el plan, de modo que no es necesario que el operador instanciado sea aplicado más de una vez. Esto tiene como consecuencia que PLANSAT sin efectos negativos pertenezca a la clase NP (Ghallab, Nau, y Traverso, 2004).

1.2. Complejidad Descriptiva

La Teoría de Complejidad Descriptiva (TCD) es un área de investigación en la intersección entre matemáticas y las ciencias de la computación que estudia la teoría de complejidad desde un punto de vista matemático, sin utilizar modelos de computación como máquinas de Turing. La TCD surgió cuando se demostró que la clase de complejidad NP corresponde exactamente con la clase de problemas expresables en lógica de segundo orden existencial (Fagin, 1974). En los últimos años se ha establecido una correspondencia entre las clases de complejidad más importantes y lenguajes lógicos con distintos niveles de expresividad (Immerman, 1998). En esta sección se describen los elementos sintácticos de estos lenguajes, repasando conceptos relevantes de la lógica, y se explica la interpretación formal de las fórmulas construidas con estos lenguajes.

1.2.1. Lenguajes

Cada lenguaje lógico está construido a partir de un conjunto de símbolos, o **vocabulario**. El vocabulario se suele dividir en símbolos lógicos puros como '$\wedge$', '$\exists$', etc., símbolos de puntuación como '(' y ')', símbolos relacionales, símbolos funcionales y constantes. Los símbolos lógicos y de puntuación pertenecen a todos los lenguajes, mientras que los símbolos relacionales, funcionales y constantes varían de un lenguaje a otro. De modo que es conveniente definir la firma de un lenguaje como el conjunto finito de relaciones, funciones y constantes permitidas en las fórmulas. Las firmas son denotadas por tuplas como $\sigma = \langle P^1, Q^2, f^1, A, B\rangle$, que contiene dos símbolos relacionales P y Q de aridades 1 y 2, respectivamente, un símbolo funcional f de aridad 1, y dos constantes A y B.

Los símbolos funcionales de la forma $f(x) = y$ pueden representarse con una tupla de la relación $F(x, y)$, en la que se expresa que y es la imagen de x, siempre y cuando se agreguen restricciones sobre x y y que expresen que f es una función. Por esta razón se omiten los símbolos funcionales en lo sucesivo.

1.2.1.1. Lógica de segundo orden

Se denota con LPO(σ) (lógica de primer orden) y LSO(σ) (lógica de segundo orden) a los conjuntos de todas las fórmulas de lógica de primer y segundo orden basadas en la firma σ. La única diferencia entre la LPO y la LSO es que en esta última se permite la cuantificación sobre los símbolos relacionales. La fórmulas que forman parte de LSO son descritas en la Definición 1.2.

Definición 1.2. Una fórmula pertenece a lógica de segundo orden (LSO) si y sólo si puede construirse mediante las siguientes reglas.

1. Cualquier fórmula LPO(τ), dado $\tau \supseteq \sigma$, es una fórmula LSO(σ).
2. Si Φ and Ψ son fórmulas LSO(σ), entonces también lo son:
   - (Φ)
   - $\neg\Phi$
   - $\Phi \wedge \Psi$
   - $\Phi \vee \Psi$
   - $\Phi \Rightarrow \Psi$
3. Si la relación $R^a \notin \sigma$ y Φ es una fórmula LSO(σ), entonces '$(\exists R)\Phi$' y '$(\forall R)\Phi$' son fórmulas LSO(σ).

Las fórmulas sin variables o predicados libres se llaman **sentencias**. Entre las fórmulas de segundo orden, en este trabajo sólo se tratarán aquellas que sean de la forma

$$\Phi = (\mathcal{Q}_1 R_1^{a_1})(\mathcal{Q}_2 R_2^{a_2}) \cdots (\mathcal{Q}_n R_n^{a_n})\psi$$

donde cada $\mathcal{Q}_i \in \{\exists, \forall\}$, y ψ es una sentencia de primer orden sobre $\sigma \cup \{R_1^{a_1}, \ldots, R_n^{a_n}\}$. Esta forma es universal: todas las sentencias de segundo orden

pueden llevarse a una equivalente que cumpla con esta restricción. Si todos los $\mathcal{Q}_i$ son cuantificadores existenciales, se dice que Φ es una sentencia **existencial de segundo orden**. La clase de todas las sentencias existenciales de segundo orden, también llamada el **fragmento existencial de la LSO**, es denotado por SO$\exists$. Se define SO$\forall$ análogamente para el cuantificador universal, y si existe un k, $1 < k < n$, tal que $\mathcal{Q}_i = \exists$ para todo $i < k$ y $\mathcal{Q}_i = \forall$ para todo $i \geq k$, entonces la sentencia pertenece al segmento SO$\exists\forall$, y así sucesivamente.

1.2.2. Estructuras de primer orden

Las fórmulas lógicas son interpretadas con respecto a **estructuras de primer orden**. Una estructura de primer orden se define sobre la firma $\sigma = \langle R_1^{a_1}, \ldots, R_s^{a_s}, c_1, \ldots, c_t \rangle$, donde cada R_i es un símbolo relacional de aridad a_i, y cada c_j es una constante. Una estructura de primer orden es una tupla $\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_s^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_t^{\mathcal{A}} \rangle$ con un universo no vacío $|\mathcal{A}|$, donde cada $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$ es un subconjunto de a_i-tuplas de $|\mathcal{A}|$, y cada $c_j^{\mathcal{A}} \in |\mathcal{A}|$ es un elemento de $|\mathcal{A}|$. Sin pérdida de generalidad, puede asumirse que el universo siempre es de la forma $|\mathcal{A}| = \{0, 1, \ldots, n-1\}$. La TCD sólo está interesada en estructuras de universo finito, la clase de todas las estructuras finitas de la firma σ se denota por STRUC$[\sigma]$.

Como ejemplo, sea $\sigma = \langle E^2, s, t \rangle$ y $\mathcal{A} = \langle |\mathcal{A}|, E^{\mathcal{A}}, s^{\mathcal{A}}, t^{\mathcal{A}} \rangle$, donde $|\mathcal{A}| = \{0, 1, 2\}$, $E^{\mathcal{A}} = \{(0, 1), (1, 2)\}$, $s^{\mathcal{A}} = 0$ y $t^{\mathcal{A}} = 2$. Nótese que la firma σ puede ser utilizada para describir grafos dirigidos con vértices s y t, y la relación $E(x, y)$ indica que hay un arco entre los vértices x y y. $\mathcal{A}$ corresponde al grafo mostrado en la Figura 1.3.

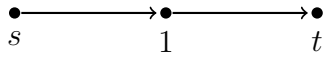

FIGURA 1.3: El grafo dirigido que corresponde a la estructura $\mathcal{A} = \langle |\mathcal{A}|, E^{\mathcal{A}}, s^{\mathcal{A}}, t^{\mathcal{A}} \rangle$ donde $|\mathcal{A}| = \{0, 1, 2\}$, $E^{\mathcal{A}} = \{(0, 1), (1, 2)\}$, $s^{\mathcal{A}} = 0$ y $t^{\mathcal{A}} = 2$.

La siguiente definición será de utilidad para trabajar sobre la semántica formal.

Definición 1.3. Sea $\mathcal{A} \in$ STRUC$[\sigma]$ y sea i una función i : VAR $\longrightarrow |\mathcal{A}|$, donde VAR es el conjunto de todas las variables. Se dice que el par $(\mathcal{A}, i)$ es una **interpretación** de una fórmula de segundo orden $\Phi \in$ LSO(σ).

Como ejemplo, una posible interpretación de la fórmula $\Phi = (\exists xy)(T(x, y))$ es $(\mathcal{A}, i)$, para toda variable x, donde $i(x) = 1$ y $\mathcal{A}$ es la estructura ejemplo mencionada anteriormente.

Definición 1.4. Se extiende i sobre todos los términos del lenguaje. Como no hay símbolos funcionales, sólo es necesario extender i sobre las constantes. Se define $i(a) = a^{\mathcal{A}}$ para toda constante $a \in \sigma$.

1.2.3. Semántica formal

Las definiciones 1.5 y 1.6 describen bajo qué circunstancias una fórmula Φ en LSO(σ) satisface una estructura $\mathcal{A}$, tal como es presentado por Immerman (1998).

Definición 1.5. Definición de Verdad. Sea $\mathcal{A} \in \text{STRUC}[\sigma]$. Se definen inductivamente las condiciones necesarias y suficientes para que una fórmula $\Phi \in \text{LSO}(\sigma)$ sea verdadera bajo la interpretación $(\mathcal{A}, i)$:

(a) $(\mathcal{A}, i) \vDash t_1 = t_2 \iff i(t_1) = i(t_2)$

(b) $(\mathcal{A}, i) \vDash R(t_1, \ldots, t_k) \iff \langle i(t_1), \ldots, i(t_k)\rangle \in R^{\mathcal{A}}$

(c) $(\mathcal{A}, i) \vDash \neg\Phi \iff (A, i) \nvDash \Phi$

(d) $(\mathcal{A}, i) \vDash \Phi \wedge \Psi \iff (\mathcal{A}, i) \vDash \Phi$ y $(\mathcal{A}, i) \vDash \Psi$

(e) $(\mathcal{A}, i) \vDash \Phi \vee \Psi \iff (\mathcal{A}, i) \vDash \Phi$ o $(\mathcal{A}, i) \vDash \Psi$

(f) $(\mathcal{A}, i) \vDash \Phi \Rightarrow \Psi \iff (\mathcal{A}, i) \vDash \neg\Phi \vee \Psi$

(g) $(\mathcal{A}, i) \vDash (\exists x)\Phi \iff$ existe $u \in |\mathcal{A}|$ tal que $(\mathcal{A}, i[x := u]) \vDash \Phi$,
donde $i[x := u](y) = \begin{cases} u & \text{si } y = x \\ i(y) & \text{si } y \neq x \end{cases}$

(h) $(\mathcal{A}, i) \vDash (\forall x)\Phi \iff (\mathcal{A}, i) \nvDash (\exists x)\neg\Phi$

(i) $(\mathcal{A}, i) \vDash (\exists R)\Phi \iff$ existe $R' \subseteq |\mathcal{A}|^k$ tal que $(\langle \mathcal{A}, R'\rangle, i) \vDash \Phi$,
donde $\langle \mathcal{A}, R'\rangle$ es la estructura que extiende a $\mathcal{A}$ donde R se interpreta como R'.

(j) $(\mathcal{A}, i) \vDash (\forall R)\Phi \iff (\mathcal{A}, i) \nvDash (\exists R)\neg\Phi$

Definición 1.6. Se dice que $\mathcal{A} \vDash \Phi$ si y sólo si para toda función i se cumple $(A, i) \vDash \Phi$. Se dice que $\vDash \Phi$, o que Φ es verdadera, si y sólo si para toda estructura $\mathcal{A} \in \text{STRUC}(\sigma)$ se cumple $\mathcal{A} \vDash \Phi$.

Volviendo al grafo de ejemplo $\mathcal{A}$, puede verificarse mediante semántica formal si $\mathcal{A} \vDash \Phi$, donde Φ es una fórmula arbitraria de LPO(σ). A continuación se comprueba si dos fórmulas particulares satisfacen la estructura $\mathcal{A}$.

1. $\mathcal{A} \vDash^{?} (\exists x)(E(s,x) \wedge E(x,t))$

   Sea $i : \text{VAR} \to |\mathcal{A}|$ arbitraria.
   Considere $j = i[x := 1]$.
   Se tiene por definición 1.5(b) que $(\mathcal{A}, j) \vDash E(s,1)$ y que $(\mathcal{A}, j) \vDash E(1,t)$.
   $\langle$por definición 1.5(d)$\rangle$
   $\quad (\mathcal{A}, j) \vDash E(s,1) \ \wedge E(1,t)$
   $\iff$ $\langle$definición 1.5(g)$\rangle$
   $\quad (\mathcal{A}, j) \vDash (\exists x)(E(s,x) \wedge E(x,t))$
   $\therefore$ $\mathcal{A} \vDash \Phi$, porque i es arbitraria.

2. La sentencia

$$\Phi_{2\text{CoL}} = (\exists R^1)(\forall xy)(E(x,y) \Rightarrow \neg(R(x) \iff R(y)))$$

   es cierta para un grafo si y sólo si es 2-coloreable. Se comprobará formalmente que $\mathcal{A} \vDash \Phi_{2\text{CoL}}$ y el grafo, por tanto, es 2-coloreable.

   Sea $i : \text{VAR} \to |\mathcal{A}|$ arbitraria.
   Considere $R' = \{1\}$, y sea $\langle A, R' \rangle$ la estructura que extiende a $\mathcal{A}$ donde R se interpreta como R'.
   Se tiene por definición 1.5(b) que $(\langle \mathcal{A}, R' \rangle, i) \vDash E(0,1)$ y que $(\langle \mathcal{A}, R' \rangle, i) \vDash E(1,2)$.
   También se tiene que $(\langle \mathcal{A}, R' \rangle, i) \vDash R'(1)$, $(\langle \mathcal{A}, R' \rangle, i) \nvDash R'(0)$, y $(\langle \mathcal{A}, R' \rangle, i) \nvDash R'(2)$.
   $\langle$por definición 1.5(d) y (c)$\rangle$
   $\quad (\langle \mathcal{A}, R' \rangle, i) \vDash E(0,1) \wedge E(1,2) \wedge \neg R'(0) \wedge R'(1) \wedge \neg R'(2)$
   $\iff$ $\langle$operadores de la lógica proposicional$\rangle$
   $\quad (\langle \mathcal{A}, R' \rangle, i) \vDash (E(0,1) \Rightarrow \neg R(0) \iff R(1)) \wedge$
   $\qquad\qquad (E(1,2) \Rightarrow R(1) \iff \neg R(2))$
   $\iff$ $\langle$definición 1.5(h)$\rangle$
   $\quad (\langle \mathcal{A}, R' \rangle, i) \vDash (\forall xy)(E(x,y) \Rightarrow \neg(R'(x) \iff R'(y)))$
   $\iff$ $\langle$definición 1.5(i)$\rangle$
   $\quad (\mathcal{A}, i) \vDash (\exists R^1)(\forall xy)(E(x,y) \Rightarrow \neg(R(x) \iff R(y)))$
   $\therefore$ $\mathcal{A} \vDash \Phi$, porque i es arbitraria.

La Figura 1.4 muestra una 2-coloración del grafo correspondiente a $\mathcal{A}$ inducida por R.

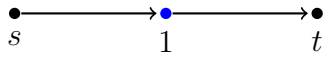

FIGURA 1.4: El grafo dirigido de la Figura 1.3, coloreado utilizando la relación R. Los nodos para los cuales R es verdadera son azules, los otros son negros.

1.2.4. Abreviaciones sintácticas

Con frecuencia es necesario cuantificar sobre una función de aridad k (f^k) en lugar de una relación. Esto puede hacerse cuantificando sobre una relación de aridad $k+1$ (F^{k+1}) y agregando fórmulas de primer orden que garanticen que F represente a f. Por ejemplo, una función parcial unaria f puede ser representada por la relación binaria F y la fórmula

$$\psi_{fun} = (\forall xyy')(F(x,y) \wedge F(x,y') \Rightarrow y = y').$$

De igual forma, se puede representar una función inyectiva añadiendo a las proposiciones la fórmula anterior más

$$\psi_{iny} = (\forall xx'y)(F(x,y) \wedge F(x',y) \Rightarrow x = x').$$

Finalmente, si es necesario que la función sea total debe agregarse

$$\psi_{tot} = (\forall x)(\exists y)(F(x,y)).$$

Se utilizarán las abreviaciones siguientes para denotar distintos tipos de funciones:

$(\exists F \in \text{Fun})$	función total
$(\exists F \in \text{Inj})$	función total inyectiva
$(\exists F \in \text{PFun})$	función parcial
$(\exists F \in \text{PInj})$	función parcial inyectiva

1.2.5. Clases de complejidad

El ejemplo anterior evidencia que una sentencia puede describir una colección de estructuras discretas finitas (como grafos dirigidos) que satisfacen una cierta propiedad (como 2-colorabilidad).

En la TCD, un problema de decisión corresponde a una clase de estructuras de primer orden que satisface a una sentencia, y por lo tanto los problemas de decisión pueden ser **modelados con fórmulas lógicas de segundo orden**. El trabajo de Fagin (1974) estableció que todos los problemas de decisión en NP pueden ser caracterizados por la clase de estructuras que satisfacen una sentencia de segundo orden existencial, es decir, NP = SO∃.

Esta lista muestra los resultados más importantes de la TCD sobre la caracterización de clases de complejidad (Immerman, 1998):

- Espacio logarítmico no determinista (NL) es igual a la LPO extendida con un operador de clausura transitiva.
- Espacio polinomial (P) es igual a las sentencias Horn de segundo orden (SO-Horn).
- Tiempo polinomial no determinista (NP) es igual a SO∃.
- Co-NP es igual a SO∀.
- La jerarquía de tiempo polinomial (PH) es igual a la lógica de segundo orden (LSO) completa.
- Espacio polinomial (PSPACE) es igual a la lógica de segundo orden extendida con un operador de clausura transitiva.

1.3. Modelación de problemas

Consideremos, por ejemplo, el problema de satisfacibilidad proposicional (SAT). Una instancia de SAT es una fórmula en forma normal conjuntiva con m cláusulas sobre n variables proposicionales, donde una cláusula es un subconjunto de literales

positivos y negativos. Sean P^2, N^2 dos símbolos relacionales que describen las ocurrencias positivas y negativas de los literales en las cláusulas: $P(x, y)$ expresa que la variable x aparece positiva en la cláusula y. Respectivamente, $N(x, y)$ expresa su ocurrencia negativa. Por ejemplo, $(p \vee \neg q \vee r) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee q)$ es codificado como $\mathcal{A} = \langle |\mathcal{A}|, N^{\mathcal{A}}, P^{\mathcal{A}} \rangle$, donde $|\mathcal{A}| = \{0, 1, 2\}$, $N^{\mathcal{A}} = \{(1, 0), (0, 1), (2, 1), (0, 2)\}$ y $P^{\mathcal{A}} = \{(0, 0), (2, 0), (1, 2)\}$.

La existencia de una asignación de valores de verdad a las variables que satisfaga una fórmula en forma normal conjuntiva (CNF) puede ser expresada con la sentencia $\mathrm{SO}\exists\Phi_{\mathrm{SAT}}$:[1]

$$(\exists T^1)(\forall y)(\exists x)[(P(x, y) \wedge T(x)) \vee (N(x, y) \wedge \neg T(x))]$$

La fórmula puede leerse como: existe una asignación de valores de verdad T tal que en toda cláusula hay al menos una variable que cumple una de las siguientes condiciones:

– la variable aparece positiva en la cláusula y está asignada a verdadero, i.e. $T(x)$

– la variable aparece negativa en la cláusula y está asignada a falso, i.e. $\neg T(x)$

Nótese que T es un *certificado*, pues muestra cuál es la asignación de variables necesaria para que la fórmula en CNF sea cierta.

En el Apéndice A se encuentran las fórmulas correspondientes a varios problemas NP, como camino hamiltoniano dirigido y colorabilidad de grafos, entre otros.

[1] Esta sentencia asume que el número de cláusulas es mayor o igual que el número de variables. En caso contrario, se debe añadir cláusulas tautológicas a la fórmula en CNF.

Capítulo 2

Traducción de problemas NP

Este capítulo describe el diseño de una herramienta capaz de transformar una descripción de un problema en lógica de segundo orden existencial en un problema de planificación perteneciente a la clase de complejidad equivalente (NP). Primero se explica el funcionamiento a alto nivel de la herramienta, en función de sus entradas y salidas. Luego, se define formalmente una función de traducción de lógica a planificación, demostrando las propiedades pertinentes. Finalmente, se presenta la gramática de un lenguaje de especificación basado en lógica y se describe brevemente la implementación de la herramienta, que consiste en un analizador sintáctico para el lenguaje lógico que luego realiza operaciones sobre el árbol sintáctico para traducir la entrada.

2.1. Perspectiva general

Desde el punto de vista del usuario, la contribución de esta herramienta es proveer una forma alternativa y descriptiva de especificar problemas de forma que sea posible obtener e interpretar su solución.

Como puede verse en la Figura 2.1, la herramienta tiene como entradas una firma σ, una sentencia $\Phi \in \mathrm{SO}\exists(\sigma)$, y una estructura finita de primer orden $\mathcal{A} \in \mathrm{STRUC}[\sigma]$ sobre la cual se intentará probar la propiedad definida por Φ. Las salidas son un dominio y una instancia PDDL que tienen un plan válido si y sólo

si $\mathcal{A}$ satisface Φ. Además, un **certificado** de la validez de la propiedad en Φ puede obtenerse del plan en tiempo lineal.

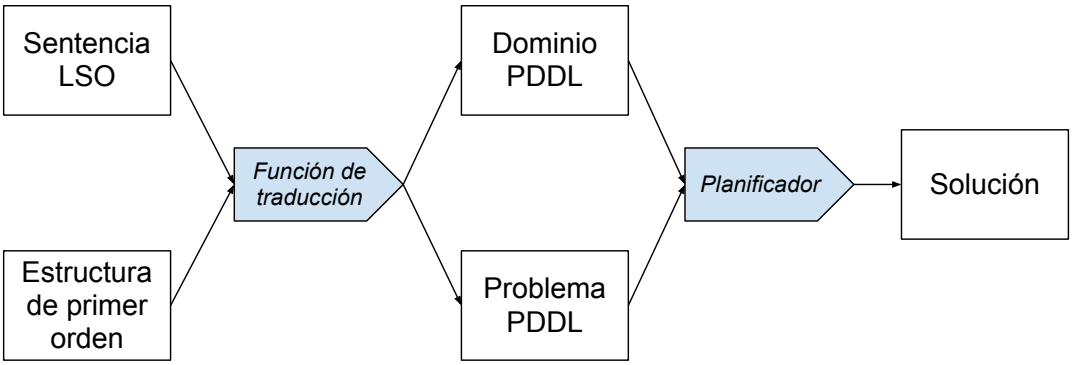

FIGURA 2.1: Esquema de operación de la herramienta.

La traducción se basa en convertir la tarea de demostrar la validez de la fórmula lógica en una tarea de planificación. Por lo tanto, las acciones del dominio de planificación hacen las veces de las definiciones de la semántica formal, probando cada tipo de subfórmula según un esquema similar a la definición 1.5. La información específica al problema, como el estado inicial, está codificada por la estructura de primer orden que recibe la herramienta. Se ha visto que estas estructuras pueden representar grafos, por lo que el *software* es especialmente útil para la modelación y resolución de problemas de grafos.

2.2. Reducción de LSO a STRIPS

Una **reducción** de un problema A a un problema B es una función computable f tal que para cada instancia ω, $\omega \in A$ si y sólo si $f(\omega) \in B$. La herramienta puede ser considerada un generador de reducciones entre problemas.

En este caso, la reducción se descompone en dos funciones

$$\mathfrak{D} : \text{Firmas} \times \text{SO}\exists \to \text{Dominios PDDL}\,,$$
$$\mathfrak{I} : \text{Firmas} \times \text{SO}\exists \times \text{STRUC} \to \text{Instancias PDDL}$$

tal que $\mathsf{dom} = \mathfrak{D}(\sigma, \Phi)$ es un dominio PDDL y $\mathsf{ins} = \mathfrak{I}(\sigma, \Phi, \mathcal{A})$ es una instancia PDDL.

Para obtener algo de interés teórico y práctico, la reducción debe correr en tiempo polinomial y su salida debe ser resoluble en NP para que la complejidad de resolver su salida no sea mayor que la complejidad de resolver su entrada.

2.2.1. Reducción del dominio

$\mathfrak{D}(\sigma, \Phi)$ produce un dominio para la firma σ y la sentencia $\Phi \in \mathrm{LSO}(\sigma)$ de la forma $(\exists R_1^{a_1}) \cdots (\exists R_n^{a_n})\psi$. Las acciones en el dominio se dividen en tres grupos: **acciones para colocar el valor de verdad de las variables de segundo orden** (relaciones cuantificadas), **acciones para probar la sentencia** ψ y **otras acciones**.

2.2.1.1. Acciones para las variables

Para cada variable de segundo orden R_i de aridad a_i, existe una acción `colocar-verdadera-Ri` con a_i parámetros que coloca la condición `Ri` y quita `libre-Ri`, el cual debe ser verdadero en el estado inicial de cualquier problema; estas condiciones se usan para denotar el valor de verdad de R_i. Por ejemplo, la acción para la relación T^1 en SAT es

```
(:action colocar_verdadera_T
  :parameters (?x)
  :precondition (and (conjetura) (no-T ?x))
  :effect (and (T ?x) (no (no-T ?x))))
```

Esta acción realiza una conjetura sobre el valor de verdad de una variable proposicional de SAT. Al ser aplicada, la variable x se considera **asignada a verdadero**. La proposición `conjetura` indica que el planificador está en la fase de suponer valores para las relaciones de segundo orden, previa a la fase de intentar demostrar que la suposición escogida es válida.

2.2.1.2. Acciones para las subfórmulas

Los operadores del segundo tipo están diseñados para construir una prueba de ψ (si existe) por inducción sobre la estructura de ψ. Para cada subfórmula θ de ψ, hay una proposición $\mathfrak{F}[\theta]$ que denota su validez, y hay una acción que la añade. Los parámetros de la proposición $\mathfrak{F}[\theta]$ son las variables libres de θ; la función $\mathfrak{F}[\cdot]$ se llama la *traducción de las proposiciones.*

El primer paso en la traducción es quitar todas las implicaciones y mover todas las negaciones hacia los literales mediante aplicaciones repetidas de las leyes de De Morgan. Luego, se generan las acciones recorriendo recursivamente todas las subfórmulas θ de ψ utilizando búsqueda en profundidad de la siguiente forma: [1]

- si $\theta(\bar{x}) = \bigwedge_{i=1}^{n} \theta_i(\bar{x}_i)$ con $\bar{x} = \cup_{i=1}^{n} \bar{x}_i$, entonces generar la acción `probar_conjuncion`$[\theta]$ con parámetros $\bar{x}$, precondición $\bigwedge_{i=1}^{n} \mathfrak{F}[\theta_i](\bar{x}_i)$ y efecto *add* $\mathfrak{F}[\theta](\bar{x})$,

- si $\theta(\bar{x}) = \bigvee_{i=1}^{n} \theta_i(\bar{x}_i)$ con $\bar{x} = \cup_{i=1}^{n} \bar{x}_i$, entonces generar n acciones de la forma `probar_disyuncion`$[\theta]_i(\bar{x}_i)$ con precondición $\mathfrak{F}[\theta_i](\bar{x}_i)$ y efecto *add* $\mathfrak{F}[\theta](\bar{x})$,

- si $\theta(\bar{x}) = (\exists y)\theta'(\bar{x}, y)$, entonces generar `probar_existencial`$[\theta](\bar{x}, y)$ con precondición $\mathfrak{F}[\theta'](\bar{x}, y)$ y efecto *add* $\mathfrak{F}[\theta](\bar{x})$,

- si $\theta(\bar{x}) = (\forall y)\theta'(\bar{x}, y)$ entonces generar dos acciones. La idea es probar $\theta(\bar{x})$, variando y sobre todos los objetos.

  Sea SUC una relación de aridad 2 que define un orden total sobre el universo de todos los objetos $|\mathcal{A}|$. SUC(x, y) indica que y es el sucesor de x.

  La primera acción `probar_universal_base`$[\theta](\bar{x})$ prueba $\theta'(\bar{x}, 0)$. La acción tiene parámetros $\bar{x}$, precondición $\mathfrak{F}[\theta'](\bar{x}, 0)$ y efecto *add* $\mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, 0)$. Observe que la traducción de la condición es aplicada a diferentes fórmulas, en las cuales la cuantificación está acotada por z.

  La segunda acción `probar_universal_inductivo`$[\theta]_1(\bar{x}, z', z'')$ prueba inductivamente $(\forall y \leq z)\theta'(x, y)$ una vez que $(\forall y < z)\theta'(x, y)$ es verdad. La acción tiene parámetros $\bar{x}, z', z''$, precondición $\mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, z') \wedge \mathfrak{F}[\theta'](\bar{x}, z'') \wedge$ SUC(z', z'') y efecto *add* $\mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, z'')$.

[1] $\theta(\bar{x})$ significa que $\bar{x}$ son las variables libres de θ.

Todas estas acciones tienen como precondición adicional la condición `prueba`. Además, nótese que no hay acciones para las fórmulas literales. Estas son manejadas por la traducción de condiciones, cuya función es asignar fórmulas a condiciones de esta manera:

- $\mathfrak{F}[Q(\bar{x})](\bar{x}) = Q(\bar{x})$,
- $\mathfrak{F}[\neg Q(\bar{x})](\bar{x}) = \texttt{no-}Q(\bar{x})$,
- $\mathfrak{F}[(\forall y)\theta'(\bar{x}, y)](\bar{x}) = \mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, \text{máx})$,
- en cualquier otro caso, $\mathfrak{F}[\theta](\bar{x}) = \texttt{es\_cierto\_<operador>\_<id>}(\bar{x})$ donde `<id>` es un identificador único para θ.

2.2.1.3. Otras acciones

La traducción requiere de dos otras acciones. La primera, llamada `empezar-prueba` sirve para cambiar la fase de 'conjetura' a 'prueba', tiene precondición `conjetura`, agrega `prueba` y quita `conjetura`. La otra acción se llama `probar-meta`, tiene precondición $\mathfrak{F}[\psi]$ y agrega `es_cierto_meta`.

2.2.1.4. Abreviaciones

Al haber abreviaciones, las acciones para las variables de segundo orden son extendidos para hacer la traducción más eficiente. Para $(\exists F \in \text{Fun})$, la precondición y el *delete* de `colocar-verdadera-F` son extendidos con la condición `(libre-F_dom ?x)` de modo de que pueda haber a lo sumo una condición $F(x, y)$ verdadera para cada x. Así, no hay necesidad de incluir la subfórmula ψ_{fun}.

De forma similar, para $(\exists F \in \text{Inj})$ la precondición y el *delete* son extendidos adicionalmente con la condición `(libre-F_ran ?y)`.

2.2.2. Problema

El problema PDDL es generado por $\mathfrak{I}(\sigma, \Phi, \mathcal{A})$, donde σ es una firma, $\Phi \in \mathrm{LSO}(\sigma)$ y $\mathcal{A} \in \mathrm{STRUC}[\sigma]$. Los objetos en el problema corresponden a los elementos en el universo $|\mathcal{A}| = \{0, \ldots, n-1\}$: 0 se asigna al objeto `cero`, $n-1$ al objeto `max`, y los otros elementos $0 < i < n-1$ a los objetos `obj_i`. El objetivo es alcanzar la condición `es_cierto_meta`, y la situación inicial consiste en condiciones describiendo el valor de verdad de todas las relaciones en $\mathcal{A}$ y las relaciones predefinidas como $<$, SUC y otras. Sin embargo, no es necesario incluir condiciones para relaciones que no se mencionan en Φ. Además, para cada variable de segundo orden R, la situación inicial tiene condiciones que denotan que los valores de verdad de R no han sido asignados (`libre-f`), y en casos en los que un símbolo represente una función o función inyectiva, la situación inicial incluye condiciones del tipo `libre-R_dom` y `libre-R_ran`.

Un ejemplo completo de la traducción de una instancia particular de SAT se encuentra en el Apéndice B.

2.3. Propiedades formales

Las propiedades más importantes a considerar en la herramienta son solidez, completitud y garantía de complejidad. Que la función de traducción sea sólida y completa significa que ella realmente implementa una reducción entre problemas de decisión, mientras que la garantía de complejidad se refiere al tiempo requerido para computar la reducción y a la complejidad de decidir la existencia de un plan en el problema generado.

Recuerde que la reducción se descompone en dos funciones $\mathfrak{D} : \text{Firmas} \times \text{SO}\exists \to$ Dominios PDDL, y $\mathfrak{I} :$ Firmas $\times$ SO$\exists \times$ STRUC $\to$ Instancias PDDL, tal que $\mathsf{dom} = \mathfrak{D}(\sigma, \Phi)$ es un dominio PDDL y $\mathsf{ins} = \mathfrak{I}(\sigma, \Phi, \mathcal{A})$ es una instancia PDDL.

Considere la función de *instanciación* $\mathfrak{G}$ que transforma el par $\langle \mathsf{dom}, \mathsf{ins} \rangle$ de un dominio y problema PDDL en un problema STRIPS $P = \mathfrak{G}(\mathsf{dom}, \mathsf{ins})$.

Formalmente, $f_{\sigma,\Phi} : \text{STRUC}[\sigma] \to \text{STRIPS}$ definida por

$$f_{\sigma,\Phi}(\mathcal{A}) = \mathfrak{G}(\mathfrak{D}(\sigma,\Phi), \mathfrak{I}(\sigma,\Phi,\mathcal{A}))$$

es una función que asigna a las σ-estructuras un problema instanciado STRIPS.

A continuación se demuestran las tres propiedades formales importantes sobre $f_{\sigma,\Phi}$.

Teorema 2.1. ***Propiedades formales de*** $f_{\sigma,\Phi}$

1. $f_{\sigma,\Phi}$ *es una reducción, esto es:*
   a) f *es* ***sólida****: Si el problema STRIPS* $f_{\sigma,\Phi}$ *tiene un plan, entonces* $\mathcal{A} \vDash \Phi$.
   b) f *es* ***completa****: Si* $\mathcal{A} \vDash \Phi$*, entonces existe un plan para el problema STRIPS* $f_{\sigma,\Phi}$.
2. $f_{\sigma,\Phi}$ *corre en tiempo polinomial.*
3. $f_{\sigma,\Phi}$ *no genera problemas más complejos que SO*$\exists$.

2.3.1. Demostración de reducción

Se demostrará que $\mathcal{A} \vDash \Phi$ si y sólo si $f_{\sigma,\Phi}(\mathcal{A})$ tiene solución, es decir, si y sólo si existe un plan que alcanza $\mathfrak{F}[\Phi]$ desde el estado inicial (que depende de $\mathcal{A}$).

Proposición 2.2. *Sean* $t_1, \ldots, t_k$ *términos instanciados, es decir, que no contienen variables. Sea* $\mathcal{A}$ *una estructura,* u *un elemento en* $|\mathcal{A}|$ *y 'a' una constante tal que* $a^{\mathcal{A}} = u$*. Entonces,* $(\mathcal{A}, i[x := u]) \vDash \varphi(t_1, \ldots, t_k, x)$ *si y sólo si* $(\mathcal{A}, i) \vDash \varphi(t_1, \ldots, t_k, a)$.

Demostración. Directa de la definición de verdad. □

Corolario 2.3. *Sean* $t_1, \ldots, t_k$ *términos instanciados. Sea* $\mathcal{A}$ *una estructura en donde todos los objetos tienen nombre (i.e., para todo* $u \in |\mathcal{A}|$ *existe una constante*

a tal que $a^{\mathcal{A}} = u$). Entonces, $(\mathcal{A}, i) \vDash (\exists x)\varphi(t_1, \ldots, t_k, x)$ si y sólo si existe una constante 'a' tal que $(\mathcal{A}, i) \vDash \varphi(t_1, \ldots, t_k, a)$.

Teorema 2.4. *Sea $\varphi \in LPO(\sigma)$ y $t_1, \ldots, t_k$ σ-términos instanciados. Sea $\mathcal{A} \in STRUC[\sigma]$ tal que todo objeto tiene nombre y sea $i : VAR \to |\mathcal{A}|$. Entonces, $(\mathcal{A}, i) \vDash \varphi(t_1, \ldots, t_k)$ si y sólo si existe un plan $\pi[t_1, \ldots, t_k]$ que alcanza $\mathfrak{F}[\varphi](t_1, \ldots, t_k)$ desde el **estado intermedio** $I[\mathcal{A}]$, el estado STRIPS en el cual los operadores que construyen la prueba empiezan a ser aplicables, definido por las siguientes condiciones:*

$$\begin{aligned} I[\mathcal{A}] = \{&\texttt{prueba}\} \cup \{\texttt{R}(a_1, \ldots, a_k) : \langle a_1^{\mathcal{A}}, \ldots, a_k^{\mathcal{A}} \rangle \in R^{\mathcal{A}}\} \\ &\cup \{\texttt{no-R}(a_1, \ldots, a_k) : \langle a_1^{\mathcal{A}}, \ldots, a_k^{\mathcal{A}} \rangle \notin R^{\mathcal{A}}\} \\ &\cup \{\texttt{SUC}(a, b) : a < b \wedge \neg(\exists c)(a < c < b)\} \end{aligned}$$

Demostración. Hacemos la demostración por inducción sobre la estructura de φ. El caso base corresponde a las fórmulas atómicas de la forma $\varphi = R(t_1, \ldots, t_k)$:

$$\begin{aligned} & (\mathcal{A}, i) \vDash \varphi(t_1, \ldots, t_k) \\ \iff & \langle \text{definición 1.5(b)} \rangle \\ & \langle t_1, \ldots, t_k \rangle \in R^{\mathcal{A}} \\ \iff & \langle \text{definición de I}[\mathcal{A}] \rangle \\ & \texttt{R}(t_1, \ldots, t_k) \in I[\mathcal{A}] \\ \iff & \langle \text{definición de } \mathfrak{F} \rangle \\ & \mathfrak{F}[\varphi](t_1, \ldots, t_k) \in I[\mathcal{A}] \\ \iff & \langle \text{el plan vacío alcanza la condición } \mathfrak{F}[\varphi](t_1, \ldots, t_k) \rangle \\ & \pi[t_1, \ldots, t_k] = \langle\rangle \textbf{ es el plan.} \end{aligned}$$

El caso $\varphi = \neg R(t_1, \ldots, t_k)$ es análogo al anterior, con $\texttt{not-R}(t_1, \ldots, t_k) \in I[\mathcal{A}]$ en lugar de $\texttt{R}(t_1, \ldots, t_k)$. Ahora consideramos las fórmulas más complejas:

Caso $\varphi = \varphi_1(t'_1, \ldots, t'_{k'}) \wedge \varphi_2(t''_1, \ldots, t''_{k''})$:

$(\mathcal{A}, i) \vDash \varphi(t_1, \ldots, t_k)$

$\iff$ ⟨definición 1.5(d)⟩

$(\mathcal{A}, i) \vDash \varphi_1(t'_1, \ldots, t'_{k'})$ y $(\mathcal{A}, i) \vDash \varphi_2(t''_1, \ldots, t''_{k''})$

$\iff$ ⟨hipótesis inductiva⟩

$\pi_1[t'_1, \ldots, t'_{k'}]$ y $\pi_2[t''_1, \ldots, t''_{k''}]$ son planes que alcanzan

$\mathfrak{F}[\varphi_1](t'_1, \ldots, t'_{k'})$ y $\mathfrak{F}[\varphi_2](t'_1, \ldots, t'_{k''})$ desde $I[\mathcal{A}]$

$\iff$ ⟨como no hay *deletes*, aplicar ambos planes en secuencia

garantiza alcanzar $\mathfrak{F}[\varphi](t_1, \ldots, t_k)$ desde $I[\mathcal{A}]$⟩

el plan $\langle \pi_1(t'_1, \ldots, t'_{k'});\ \pi_2(t''_1, \ldots, t''_{k_2});\ \texttt{probar\_y\_}\varphi(t_1, \ldots, t_k)\rangle$

alcanza la condición $\mathfrak{F}[\varphi](t_1, \ldots, t_k)$ **desde** $I[\mathcal{A}]$

Caso $\varphi = \varphi_1(t'_1, \ldots, t'_{k_1}) \vee \varphi_2(t''_1, \ldots, t''_{k_2})$:

$(\mathcal{A}, i) \vDash \varphi(t_1, \ldots, t_k)$

$\iff$ ⟨definición 1.5(e)⟩

$(\mathcal{A}, i) \vDash \varphi_1(t'_1, \ldots, t'_{k'})$ o $(\mathcal{A}, i) \vDash \varphi_2(t''_1, \ldots, t''_{k''})$

$\Rightarrow$ ⟨hipótesis inductiva, suponiendo que se utiliza la prueba de φ_1⟩

$\pi_1[t_1, \ldots, t_{k_1}]$ es un plan que alcanza $\mathfrak{F}[\varphi_1](t_1, \ldots, t_{k_1})$ desde $[\mathcal{A}]$

$\iff$ ⟨no hay *deletes*⟩

el plan $\langle \pi_1(t_1, \ldots, t_{k_1});\ \texttt{probar\_o\_}\varphi(t_1, \ldots, t_k)\rangle$

alcanza la condición $\mathfrak{F}[\varphi](t_1, \ldots, t_k)$ **desde** $I[\mathcal{A}]$

Puede llegarse a la misma conclusión utilizando la hipótesis inductiva con φ_2. Falta probar la dirección inversa ($\Leftarrow$):

el plan π_1 alcanza la condición $\mathfrak{F}[\varphi](t_1, \ldots, t_k)$ **desde** $I[\mathcal{A}]$

$\iff$ ⟨π_1 tiene la acción $\texttt{probar\_o\_}\varphi(t'_1, \ldots, t'_k)$⟩

φ_1 tiene un subplan para $\mathfrak{F}[\varphi](t'_1, \ldots, t'_k)$

$\Leftarrow$ ⟨hipótesis inductiva⟩

$\mathcal{A} \vDash \varphi_1$

$\Leftarrow$ ⟨fortalecimiento⟩

$\mathcal{A} \vDash \varphi$. El caso con φ_2 es análogo.

Caso $\varphi = (\exists x)\ \theta(t_1, \ldots, t_k, x)$:

$(\mathcal{A}, i) \vDash \varphi$

$\iff$ ⟨por corolario 2.3, ya que todo objeto en $|\mathcal{A}|$ tiene nombre, existe una constante 'a'⟩

$(\mathcal{A}, i) \vDash \theta(t_1, \ldots, t_k, a)$

$\iff$ ⟨hipótesis inductiva⟩

$\pi[t_1, \ldots, t_k, a]$ es un plan que alcanza $\mathfrak{F}[\theta](t_1, \ldots, t_k, a)$ desde $I[\mathcal{A}]$

$\iff$

el plan $\langle \pi[t_1, \ldots, t_k, a];$ `probar_existencial_`$\varphi(t_1, \ldots, t_k, a)\rangle$

alcanza la condición $\mathfrak{F}[\varphi](t_1, \ldots, t_k)$ **desde** $I[\mathcal{A}]$

Caso $\varphi = (\forall x)\ \theta(t_1, \ldots, t_k, x)$:

$(\mathcal{A}, i) \vDash \varphi$

$\iff$ ⟨definición 1.5(h), y todo objeto tiene nombre⟩

$(\mathcal{A}, i) \vDash \theta(t_1, \ldots, t_k, a)$, para toda constante a

$\iff$ ⟨hipótesis inductiva⟩

para toda constante a existe un plan $\pi[t_1, \ldots, t_k, a]$ que

alcanza $\mathfrak{F}[\theta](t_1, \ldots, t_k, a)$ desde $I[\mathcal{A}]$

$\iff$ ⟨existe un orden $a_i < a_{i+1}$ para todas las constantes, determinado por `SUC`, y no hay *deletes*⟩

el plan $\langle \pi_0[t_1, \ldots, t_k, a_0];$ `probar_universal_base_`$\varphi(t_1, \ldots, t_k, a_0);$

$\pi_1[t_1, \ldots, t_k, a_1];$ `probar_universal_inductivo_`$\varphi(t_1, \ldots, t_k, a_0, a_1);$

$\ldots$

$\pi_{n-1}[t_1, \ldots, t_k, a_{n-2}, a_{n-1}];$ `probar_universal_inductivo_`$\varphi(t_1, \ldots, t_k, a_{n-2}, a_{n-1})\rangle$

alcanza la condición $\mathfrak{F}[\varphi](t_1, \ldots, t_k)$ **desde** $I[\mathcal{A}]$

□

Definición 2.5. Sea $\Phi = (\exists R_1^{k_1}) \cdots (\exists R_m^{k_m})\varphi$. El **estado inicial** $\text{INIT}[\mathcal{A}]$ de un problema STRIPS generado por $f_{\sigma,\Phi}(\mathcal{A})$ es:

$$\begin{aligned}\text{INIT}(\mathcal{A}) = \{&\{\texttt{P}(a_1,\ldots,a_k) : \langle a_1^{\mathcal{A}},\ldots,a_k^{\mathcal{A}}\rangle \in P^{\mathcal{A}}\}\\ &\cup \{\texttt{no-P}(a_1,\ldots,a_k) : \langle a_1^{\mathcal{A}},\ldots,a_k^{\mathcal{A}}\rangle \notin P^{\mathcal{A}}\}\\ &\cup \{\texttt{SUC}(a,b) : a < b \wedge \neg(\exists c)(a < c < b)\}\\ &\cup \{\texttt{libre-R\_i}(t)) : \text{ para todo } t \in |\mathcal{A}|^{k_i}, R_i \in \{R_1,\ldots,R_m\}\}\}\end{aligned}$$

Finalmente, puede probarse que la función es una reducción:

Teorema 2.6. *Sea* $\Phi = (\exists R_1)\cdots(\exists R_n)\varphi$. $\mathcal{A} \vDash \Phi$ *si y sólo si existe un plan* π *que alcanza* $\mathfrak{F}[\Phi]$ *desde el estado inicial* $\text{INIT}(f_{\sigma,\Phi}(\mathcal{A}))$.

Demostración. Por definición, $\mathcal{A} \vDash \Phi$ si y sólo si $\langle \mathcal{A}, R_1', \ldots, R_n'\rangle \vDash \varphi$. Por Teorema 2.4, esto es verdad si y sólo si existe un plan π' que alcanza $\mathfrak{F}[\varphi]$ a partir de $I[\mathcal{A}, R_1', \ldots, R_n']$. Basta mostrar que es posible alcanzar el estado $I[\mathcal{A}, R_1', \ldots, R_n']$ desde $\text{INIT}(f_{\sigma,\Phi}(\mathcal{A}))$. Sea $u_j = a_j^{\mathcal{A}}$ para $1 \le j \le k$.

El plan

$\langle$ $\langle$`colocar-verdadera-Ri`$(a_1,\ldots,a_{k_i})$ para cada $\langle u_1,\ldots,u_k\rangle \in R_i\rangle$ para cada R_i ;
$\langle$`colocar-falsa_Ri`$(a_1,\ldots,a_{k_i})$ para cada $\langle u_1,\ldots,u_k\rangle \notin R_i\rangle$ para cada R_i ;
`empezar-prueba`() $\rangle$

alcanza el estado $I[\mathcal{A}, R_1', \ldots, R_n']$ **desde** $\text{INIT}(f_{\sigma,\Phi}(\mathcal{A}))$.

□

2.3.2. Tiempo de corrida de la reducción

En esta sección se demuestra que $f_{\sigma,\Phi}$ es una reducción del problema NP expresado por Φ a un problema STRIPS que corre en tiempo polinomial.

Recuerde que $f_{\sigma,\Phi}(\mathcal{A}) = \mathfrak{G}(\mathfrak{D}(\sigma,\Phi), \mathfrak{I}(\sigma,\Phi,\mathcal{A}))$. Para un dom fijo, la función $\mathsf{ins} \rightsquigarrow \mathfrak{G}(\mathsf{dom},\mathsf{ins})$ corre en un tiempo polinomial $\mathcal{O}(\|\mathsf{ins}\|^k)$ para algún k que sólo depende de dom; de hecho, k es la máxima aridad de un predicado o acción

en el dominio. Nótese que en este caso el dominio puede generarse de manera independiente del problema utilizando sólo una fórmula de segundo orden. Al no depender de $\mathcal{A}$, el dominio es fijo y $\mathfrak{D}$ corre en tiempo polinomial.

La función de traducción $\mathfrak{I}$ corre en tiempo polinomial en el tamaño de la estructura $\mathcal{A}$, puesto que sólo se agregan condiciones que equivalen al estado de las relaciones $P \in \sigma$ al estado inicial.

2.3.3. Complejidad de los problemas generados

Se debe demostrar que los problemas de planificación producidos por $f_{\sigma,\Phi}(\mathcal{A})$ tienen a lo sumo la misma complejidad que SO$\exists$.

Decidir la existencia de un plan general para un problema STRIPS no está en NP porque incluso los planes más cortos pueden ser de tamaño exponencial en el número de condiciones y acciones instanciadas.

Como se ha mencionado en la sección 1.1.1, se sabe que verificar la existencia de un plan para problemas de planificación sin efectos negativos está en NP (Bylander, 1994). La prueba depende del hecho de que un plan óptimo no necesita repetir acciones, y por lo tanto es de tamaño lineal. Puede extenderse esta noción: si todas las acciones que agregan efectos negativos pueden ser aplicadas a lo sumo una vez, verificar la existencia de un plan está en NP, pues el tamaño del plan debe ser de tamaño polinomial.

Definición 2.7. Un problema de planificación $P = \langle C, A, I, F\rangle$ es de tipo *máximo-1* si y sólo si las acciones pueden ser particionadas en $A = A_0 \cup A_1$, donde:

- Ninguna de las acciones de A_0 tiene efectos negativos, es decir, $(\forall a \in A_0)(del(a) = \emptyset)$.
- Todas las acciones de A_1 tienen una precondición que no es añadida por ninguna acción, y que es borrada apenas a es ejecutada, es decir, $(\forall aa' \in A_1)\ (\exists p \in pre(a) \cap del(a))\ (p \notin add(a'))$

El conjunto de todos los problemas *máximo-1* se denota como STRIPS-1.

Teorema 2.8. *Para toda estructura $\mathcal{A}$, $f_{\sigma,\Phi}(\mathcal{A})$ es un problema STRIPS-1.*

Demostración. Ninguna de las acciones de $f_{\sigma,\Phi}(\mathcal{A})$ tienen *deletes* excepto `empezar-prueba` y las acciones `colocar-verdadera` y `colocar-falsa`, pero todas estas borran una precondición que no es añadida por ninguna otra acción (i.e., pertenecen a A_1 según la definición de STRIPS-1). □

Teorema 2.9. *El problema de decisión de existencia de un plan STRIPS-1 es NP-completo.*

Demostración.
Inclusión: Todas las acciones de A_1 pueden ser aplicadas a lo sumo una vez. Como estas acciones pueden borrar proposiciones (tienen *deletes*), en el peor caso puede requerirse aplicar **todas** las acciones de A_0 antes de aplicar otra acción de A_1. Por lo tanto, el tamaño de un plan es a lo sumo cuadrático en el número total de acciones.
Dificultad: $f_{\sigma_{\text{SAT}},\Phi_{\text{SAT}}}$ reduce SAT a STRIPS-1 en tiempo polinomial. □

Por lo tanto, la herramienta traduce cualquier problema NP, codificado con una sentencia SO∃, a un problema STRIPS-1 que se ubica en la clase de complejidad NP.

2.4. Diseño del lenguaje

2.4.1. Especificación de la fórmula Φ

El lenguaje de entrada de la herramienta debe ser capaz de expresar sintácticamente cualquier fórmula LSO existencial. En el capítulo 4 se presenta una extensión de la herramienta que le permite traducir problemas en la clase más compleja PH, por lo que es necesario agregar al lenguaje, además de los operadores estudiados hasta el momento, el operador ∀ de segundo orden, `so-forall`. La Figura 2.2 muestra la gramática del lenguaje propuesto en forma Backus-Naur, que permite especificar fórmulas lógicas generales de segundo orden.

⟨fbf-so⟩ ⟶ `(so-exists (`⟨lista-rel⟩`)` ⟨fbf-so⟩`)`
⟶ `(so-forall (`⟨lista-rel⟩`)` ⟨fbf-so⟩`)`
⟶ ⟨fbf-po⟩

⟨lista-rel⟩ ⟶ `<REL> <int>` ⟨lista-rel⟩ | `<REL> <int>`
⟶ `<REL>` ⟨tipo-fun⟩ ⟨lista-rel⟩ | `<REL>` ⟨tipo-fun⟩

⟨tipo-fun⟩ ⟶ Fun | PFun | Inj | PFun

⟨fbf-po⟩ ⟶ `(<REL>` ⟨lista-var⟩`)`
⟶ `(not` ⟨fbf-po⟩`)`
⟶ `(and` ⟨lista-fbf-po⟩`)`
⟶ `(or` ⟨lista-fbf-po⟩`)`
⟶ `(implies` ⟨fbf-po⟩ ⟨fbf-po⟩`)`
⟶ `(exists (`⟨lista-var⟩`)` ⟨fbf-po⟩`)`
⟶ `(forall (`⟨lista-var⟩`)` ⟨fbf-po⟩`)`

⟨lista-var⟩ ⟶ `<VAR>` ⟨lista-var⟩ | `<VAR>`

`<VAR>` $\in$ `\?[a-z][a-z0-9_]*`,
`<REL>` $\in$ `\?[A-Z][A-Z0-9_]*`,
`<int>` $\in \mathbb{Z}^+$

FIGURA 2.2: Gramática en BNF del lenguaje que se utiliza para expresar Φ en la herramienta.

2.4.2. Especificación de la firma σ

Para especificar una firma $\sigma = \langle R_1^{k_1}, \ldots, R_n^{k_n} \rangle$ se utiliza un archivo de texto que sigue la notación:

```
?R_1 k_1
...
?R_n k_n
```

Las relaciones `?R_i` pueden tener cualquier nombre de acuerdo a la expresión regular `\?[A-Z][A-Z0-9]*`.

Para especificar constantes en la firma se utilizan relaciones unarias que son ciertas sólo para el elemento de $|\mathcal{A}|$ que interpreta la constante. Esto simplifica el procesamiento. Por ejemplo, si se quiere agregar una constante K, simplemente se agrega una relación unaria a la firma `?K 1`.

2.4.3. Especificación de la estructura $\mathcal{A}$

La estructura de primer orden $\mathcal{A}$ será transformada en una instancia PDDL mediante $\mathfrak{I}(\sigma, \Phi, \mathcal{A})$.

Una estructura $\mathcal{A}$ se especifica en un archivo de texto que contiene una k-tupla por cada línea, precedida por la relación que le corresponde. Para expresar que $(a_1, \ldots, a_k) \in R_i$ en $\mathcal{A}$, se agrega la línea `(?R_i a_1 ... a_k)`.

Como se ha dicho anteriormente, las constantes se denotan utilizando relaciones unarias. Por lo tanto, si por ejemplo quiere expresarse $K = máx$, basta con añadir la línea `?K max`.

2.5. Implementación de la herramienta

La herramienta fue implementada en el lenguaje de programación *Python*.

La primera parte del *software* consiste en un analizador sintáctico LL(1) implementado de forma descendente recursiva. Según este esquema, se construye un árbol sintáctico de acuerdo a la gramática de la Figura 2.2. Puede verse un ejemplo del árbol sintáctico resultante al analizar la fórmula de SAT en el Apéndice B.

Cuando el árbol sintáctico está construido, el siguiente paso es transformar las implicaciones a disyunciones. Esto puede hacerse económicamente con un recorrido cualquiera por el árbol en el que se reemplace el operador `implies` por el operador `or` y se coloque una negación en el primer argumento del operador.

Posteriormente, debe realizarse una propagación de las negaciones hacia los literales, pues la traducción que se ha definido sólo considera negaciones en este

nivel. Afortunadamente, esto se logra con un sencillo recorrido en profundidad del árbol guiado por el siguiente algoritmo:

```
proc recorrido(nodo, modo):
    para cada hijo de nodo:
        si hijo es NOT:
            hijo = sucesor(hijo) // se elimina la negación
            recorrido(hijo, !modo)
        si no:
            si modo es PROPAGAR_NEGACIONES:
                si hijo es LITERAL:
                    colocar NOT antes de LITERAL
                si no:
                    hijo = complemento(hijo)
                    recorrido(hijo, modo)
            si no:
                recorrido(hijo, modo)
```

Donde:

- `(!modo)` es una operación que convierte el modo `RECORRER` en `PROPAGAR_NEGACIONES` y viceversa.
- `complemento(AND) = OR`, `complemento(OR) = AND`, `complemento(EXISTS) = FORALL`, `complemento(FORALL) = EXISTS`.

El recorrido se inicia con la llamada `recorrido(arbol, RECORRER)`, donde `arbol` es el nodo superior (la subfórmula más externa). El algoritmo descrito no es más que la aplicación recursiva de las leyes de De Morgan hasta lograr que todas las negaciones aparezcan junto a literales, en lugar de en subfórmulas generales.

Luego, se hace un último recorrido del árbol para generar las condiciones y acciones de PDDL que configurarán la traducción del dominio PDDL. Para obtener $\mathsf{dom} = \mathfrak{D}(\sigma, \Phi)$ a partir del árbol sintáctico se realiza una búsqueda en profundidad de la siguiente manera:

```
traducir(nodo):
    para cada hijo de nodo:
        traducir(hijo)
    agregar_acciones(nodo)
```

`agregar_acciones(nodo)` genera las acciones que especifica la traducción de la subfórmula particular que se encuentra en `nodo`, tal como es descrito en la sección 2.2.

Finalmente, para obtener $\mathsf{ins} = \mathfrak{I}(\sigma, \Phi, \mathcal{A})$ se interpreta la estructura $\mathcal{A}$ en el archivo de texto como la situación inicial del problema de planificación, y se especifica que la instancia ins pertenece al dominio dom, con lo cual las acciones aplicables sobre el problema ins serán exactamente las especificadas en dom, ya traducidas con el último recorrido del árbol sintáctico.

2.6. Simplificación de la traducción

Una sencilla modificación a la traducción es reemplazar las condiciones `libre-Ri`, presentes en el estado inicial, con condiciones `no-Ri`. Con esta simplificación, sólo es necesaria la acción `colocar-verdadera`, pues inicialmente se supone que ninguna k-tupla pertenece a R_i^k. De este modo, se establece una *conjetura inicial* y se reduce el espacio de búsqueda al eliminar acciones.

Las propiedades formales de la reducción se mantienen incorporando la simplificación. Se utilizará esta versión simplificada de la reducción para realizar los experimentos en el Capítulo 4.

Capítulo 3

Traducción de problemas PH

Este capítulo describe la extensión de la herramienta desarrollada en este trabajo para la traducción de problemas de decisión pertenecientes a la clase de complejidad de la jerarquía polinomial, PH, los cuales son capturados por la lógica de segundo orden. En primer lugar se habla brevemente sobre la complejidad de esta clase de problemas. Luego se presenta, mediante ejemplos, una traducción alternativa a la expuesta en el capítulo anterior que permite traducir cualquier problema expresado en lógica general de segundo orden a un problema de planificación automática.

3.1. Complejidad

Al contrario que en el caso de NP, esta vez no se ofrecen garantías de complejidad sobre la extensión de la reducción: los problemas STRIPS generados por ella en general no serán solucionables en tiempo polinomial no determinístico. Esto se debe a que los problemas de esta clase sólo admiten planes de longitud exponencial en el peor caso ya que PH contiene, además de a NP, las clases coNP y Σ_k^p para todo $k \geq 1$. La clase Σ_p^2, por ejemplo, es igual a NP^{NP}, la clase de problemas resolubles en tiempo polinomial no determinístico contando un oráculo para resolver problemas NP.

Una característica de NP es la disponibilidad de un *certificado* que valida en tiempo polinomial una respuesta al problema de decisión: tal garantía no es ofrecida en estas clases superiores. De este modo, los problemas de planificación que produce la reducción son cortos, pues pueden ser codificados por las traducciones de estructuras finitas $\mathcal{A}$, pero sus soluciones son, potencialmente, muy largas.

coNP es la clase complementaria a NP, y es capturada por la lógica SO$\forall$ que contiene fórmulas que se apegan a la Ecuación 3.1:

$$\Phi = (\forall R_1^{a_1}) \cdots (\forall R_n^{a_n})\psi \tag{3.1}$$

En general, Σ_k^p es capturada por fórmulas cuya estructura contiene k bloques alternantes de cuantificadores, empezando con los existenciales. Por ejemplo, $\Sigma_2^p =$ SO$\exists\forall$ corresponde a fórmulas de la forma:

$$\Phi = (\exists R_1^{a_1}) \cdots (\exists R_n^{a_n})(\forall S_1^{a'_1}) \cdots (\forall S_m^{a'_m})\psi \tag{3.2}$$

Considere ahora la fórmula $\Phi = (\forall R^1)\psi$ y una estructura $\mathcal{A}$ con universo $|\mathcal{A}|$. Se dice que Φ es válida en $\mathcal{A}$ si y sólo si para toda interpretación $R^{\mathcal{A}}$ de R, donde $R^{\mathcal{A}} \subseteq |\mathcal{A}|$, se tiene $\langle \mathcal{A}, R^{\mathcal{A}} \rangle \vDash \psi$. Como existen $2^{\|\mathcal{A}\|}$ interpretaciones distintas de R, una demostración de $\mathcal{A} \vDash \Phi$ puede ser de **tamaño exponencial**. Por ejemplo, si Φ_{UNSAT} denota el problema complementario a satisfacibilidad proposicional, entonces una prueba de que la estructura $\mathcal{A}$ es no satisfacible es de tamaño exponencial, dado que debe considerar todos los valores de verdad para las variables proposicionales de $\mathcal{A}$. Por ello, las soluciones para el problema de planificación P generado por el par $\langle \Phi_{\text{UNSAT}}, \mathcal{A} \rangle$ codifican tales pruebas y son de tamaño exponencial.

3.2. Traducción a STRIPS

3.2.1. Sistema de tipos

Antes de extender la traducción a PH es conveniente agregarle un sistema sencillo de tipos.

Sea τ^* un sistema, creado a partir de un conjunto finito $\tau = \{\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_N\}$ de **tipos atómicos**, donde τ^* es el conjunto más pequeño que contiene t y todos los tipos de la forma $t = t' \times t''$ para $t' \in \tau^*$ y $t'' \in \tau$.

La idea es que a cada objeto en una estructura de primer orden le es asignado un tipo atómico, siendo $\mathbf{t}_0$ el tipo que contiene a todos los objetos. Los tipos se utilizan para restringir las iteraciones sobre las cuantificaciones de primer y segundo orden en las fórmulas. Por ejemplo, $(\forall x \in t)$ se refiere a una cuantificación de primer orden sobre todos los objetos de tipo $t \in \tau$, mientras que $(\forall R^t)$ se refiere a un predicado de segundo orden universalmente cuantificado R de tipo $t \in \tau^*$. La única restriccion que se exige es que los tipos complejos en $\tau^* \setminus \tau$ sólo aparezcan en cuantificaciones de segundo orden.

Considere una fórmula general LSO de la forma

$$\Phi = (\mathcal{Q}_1 R_1^{t_1})(\mathcal{Q}_2 R_2^{t_2}) \cdots (\mathcal{Q}_n R_n^{t_n})\psi \tag{3.3}$$

donde cada $\mathcal{Q}_i \in \{\exists, \forall\}$ es un cuantificador de segundo orden, R_i es un símbolo relacional de tipo $t_i \in \tau^*$, y ψ es una sentencia de primer orden.

Sea $\mathcal{A}$ una estructura de primer orden en STRUC$[\sigma]$. Se muestra cómo generar un problema de planificación P que tiene solución si y sólo si $\mathcal{A} \vDash \Phi$.

Según la traducción descrita en el capítulo 2, P tiene condiciones de la forma $\mathfrak{F}[\theta]$ para cada subfórmula θ de ψ y con parámetros que corresponden a las variables libres de θ. Por ejemplo, si Φ es la fórmula para SAT:

$$\begin{aligned}
\Phi_{\text{SAT}} &= (\exists T^{\text{Var}})\psi_{\text{SAT}}\,, \\
\psi_{\text{SAT}} &= (\forall y \in \text{Cls})(\exists x \in \text{Var}) \\
&\qquad [(P(x,y) \wedge T(x)) \vee (N(x,y) \vee \neg T(x))]
\end{aligned}$$

donde 'Var' y 'Cls' son los tipos para variables y cláusulas, entonces existe una condición $\mathfrak{F}[\theta]$ con parámetros $\langle x, y\rangle$ para la subfórmula $\theta(x,y) = N(x,y) \vee \neg T(x)$. En ψ_{SAT}, las relaciones $P(x,y)$ y $N(x,y)$ denotan que la variable x aparece positiva o negativa, respectivamente, en la cláusula y.

3.2.2. Traducción del operador `so-forall`

Para la extensión de la herramienta, debe tenerse en cuenta cómo las relaciones cuantificadas se construyen y se intercalan con las pruebas de las subfórmulas. Este problema no surge con los problemas SO∃, debido a que hay una sola interpretación que construir para cada fórmula existencialmente cuantificada, pero para fórmulas generales es necesario construir e intentar probar varias interpretaciones diferentes. Para aclarar este punto, considere la fórmula para UNSAT, que contiene una relación unaria T cuantificada universalmente:

$$\begin{aligned}\Phi_{\text{UNSAT}} &= (\forall T^{\text{Var}})\psi_{\text{UNSAT}} \\ \psi_{\text{UNSAT}} &= (\exists y \in \text{Cls})(\forall x \in \text{Var}) \\ &\qquad [(P(x,y) \wedge \neg T(x)) \vee (N(x,y) \wedge T(x)) \wedge NotIn(x,y)]\end{aligned}$$

Donde la relación $NotIn(x, y)$ indica que la variable x no está presente en la cláusula y.

Esta fórmula expresa que para toda relación T sobre variables proposicionales, existe una cláusula y tal que para cada variable x, o x aparece positiva en y y x es falsa, o x aparece negativa en y y x es verdadera, o x no aparece en y. UNSAT se puede traducir automáticamente a STRIPS considerando acciones que **iteran** sobre todas las posibles relaciones T, y acciones para obtener la condición $\mathfrak{F}[\psi_{\text{UNSAT}}]$ para cada una de estas T.

Como T es una relación unaria, hay 2^n relaciones distintas en n variables. Cada relación puede ser codificada con una cadena binaria de tamaño n (un bit por variable) tal que el i-ésimo bit es 1 si y sólo si la i-ésima variable x_i pertenece a T. Luego, iterar sobre todas las relaciones es equivalente a iterar sobre todas las cadenas binarias de tamaño n: puede hacerse empezando con la relación vacía, que corresponde a '$0\ldots00$', y sucesivamente "sumando 1 a la cadena" hasta llegar a '$1\ldots11$'.

La Figura 3.1 muestra 5 acciones que realizan la iteración en el caso de Φ_{UNSAT}: estas acciones utilizan como condiciones adicionales $\mathfrak{F}[\Phi_{\text{UNSAT}}]$, `necesita-`$\mathfrak{F}[\Phi_{\text{UNSAT}}]$,

[A1] ComenzarPrueba:
Pre: necesita-$\mathfrak{F}[\Phi_{\text{UNSAT}}]$
Efe: para cada $x \in$ Var: no-T(x) ,
$\neg$necesita-$\mathfrak{F}[\Phi_{\text{UNSAT}}]$, necesita-$\mathfrak{F}[\psi_{\text{UNSAT}}]$

[A2] PrimeraIteración(x):
Pre: $\mathfrak{F}[\psi_{\text{UNSAT}}]$, VarPrimera-T(x)
Efe: $\neg\mathfrak{F}[\psi_{\text{UNSAT}}]$, Marcador-T(x)

[A3] SiguienteSobreExcluidos(x):
Pre: Marcador-T(x) , no-T(x)
Efe: $\neg$no-T(x) , T(x) , $\neg$Marcador-T(x) , necesita-$\mathfrak{F}[\psi_{\text{UNSAT}}]$

[A4] SiguienteSobreIncluidos(x, y):
Pre: Marcador-T(x) , T(x) , VarSuc-T(x, y)
Efe: $\neg$T(x) , no-T(x) , $\neg$Marcador-T(x) , Marcador-T(y)

[A5] TerminarPrueba(x):
Pre: Marcador-T(x) , T(x) , VarÚlt-T(x)
Efe: $\neg$T(x) , no-T(x) , $\neg$Marcador-T(x) , $\mathfrak{F}[\Phi_{\text{UNSAT}}]$

Figura 3.1: Acciones para iterar sobre las 2^n relaciones unarias T que codifican las 2^n asignaciones de verdad para las n variables proposicionales de UNSAT.

'Marcador-T(x)', 'VarPrimera-T(x)', 'VarSuc-T(x, y)' y 'VarÚlt-T(x)'. Los últimos tres tipos de condición son estáticos: se agregan al estado inicial e implementan un orden estático de los objetos que se refieren a variables proposicionales.

Se describirá brevemente cómo funciona la iteración. El estado inicial contiene las condiciones estáticas que definen a los tipos Var y Cls, y las relaciones binarias $N(x, y)$ y $P(x, y)$, más la condición necesita-$\mathfrak{F}[\Phi_{\text{UNSAT}}]$. El estado final contiene solamente la condición $\mathfrak{F}[\Phi_{\text{UNSAT}}]$. Inicialmente, la única acción aplicable es A1, que interpreta a T como la relación vacía y agrega la condición necesita-$\mathfrak{F}[\psi_{\text{UNSAT}}]$.

Esta condición activa a las acciones para construir una prueba de ψ_{UNSAT} para el T actual. Una vez que ψ_{UNSAT} ha sido probada, se agrega la condición $\mathfrak{F}[\psi_{\text{UNSAT}}]$.

Cuando ψ_{UNSAT} es demostrado, A2 se convierte en la única acción aplicable y **borra la condición** $\mathfrak{F}[\psi_{\text{UNSAT}}]$, agregando Marcador$(x)$ para la primera variable en el orden estático. Luego de A2, A3 debe ser aplicado, y cambia T del modelo que corresponde a $0\ldots00$ al modelo que corresponde a $0\ldots01$. También agrega necesita-$\mathfrak{F}[\psi_{\text{UNSAT}}]$, lo que exige una demostración de ψ_{UNSAT} para la nueva T.

La iteración procede de manera similar hasta demostrar ψ_{UNSAT} para el útlimo modelo, que corresponde a $1\ldots11$. Entonces, A5 es la única acción aplicable y agrega $\mathfrak{F}[\Phi_{\text{UNSAT}}]$, completando el plan.

Nótese que este método de iteración es extensible a relaciones de aridad $k > 1$. El único cambio que se necesita es reemplazar x y y por k-tuplas de variables. De igual forma, las condiciones que implementan el orden estático deben reemplazarse por condiciones que implementen un orden estático sobre k-tuplas.

3.2.3. Traducción del operador `so-exists`

Para garantizar la interoperabilidad con la traducción que se ha presentado arriba para cuantificadores universales de segundo orden, y asegurar la posibilidad de anidar cuantificadores existenciales y universales a fin de capturar PH, debe modificarse ligeramente la traducción del operador existencial de segundo orden.

Concretamente, se mostrará la nueva traducción para el caso de SAT. La Figura 3.2 muestra las acciones que llevan a cabo tal traducción. Las traducciones de SAT y UNSAT comparten un mismo protocolo que controla cuáles operadores se activan o desactivan dependiendo de qué parte de la fórmula está siendo demostrada.

Por ejemplo, una fórmula en Σ_2^p de la forma $\Phi = (\exists T^{t_1})(\forall R^{t_2})\psi$ puede ser analizada como

$$\Phi = (\exists T^{t_1})\Psi \quad \text{con} \quad \Psi = (\forall R^{t_2})\psi \tag{3.4}$$

de modo que las acciones de cuantificadores existenciales (parecidas a las de SAT) puedan ser combinadas acciones de cuantificadores universales (parecidas a las de UNSAT). Para realizar la combinación, el operador E5 (ver Figura 3.2) del existencial de segundo orden interno debe borrar las condiciones de todas las subfórmulas para tener un "estado limpio" para la siguiente prueba.

[E1] ComenzarPrueba:
Pre: necesita-$\mathfrak{F}[\Phi_{\text{SAT}}]$
Efe: para cada $x \in$ Var: no-T(x) , $\neg$necesita-$\mathfrak{F}[\Phi_{\text{SAT}}]$, conjetura-T

[E2] ColocarVerdadera(x):
Pre: conjetura-T
Efe: T(x) , $\neg$no-T(x)

[E3] ColocarFalsa(x):
Pre: conjetura-T
Efe: no-T(x) , $\neg$T(x)

[E4] ProbarSubfórmula:
Pre: conjetura-T
Efe: $\neg$conjetura-T , necesita-$\mathfrak{F}[\psi_{\text{SAT}}]$

[E5] TerminarPrueba:
Pre: $\mathfrak{F}[\psi_{\text{SAT}}]$
Efe: $\neg\mathfrak{F}[\psi_{\text{SAT}}]$, $\mathfrak{F}[\Phi_{\text{SAT}}]$

Figura 3.2: Acciones que implementan la nueva traducción para el cuantificador existencial en SAT.

Capítulo 4

Experimentos y resultados

Este capítulo presenta una serie de experimentos realizados para evaluar hasta qué punto es práctico resolver problemas en NP y PH modelados en lógica y traducidos por la herramienta. En primer lugar, se justifica la escogencia de un planificador basado en SAT para la realización de los experimentos y se presenta una forma de calcular cotas inferiores y superiores para acotar la búsqueda de una solución. Luego, se explica el protocolo de realización de los experimentos y se describen brevemente los problemas en NP y PH que se utilizarán. Finalmente, se discuten los resultados obtenidos en estos experimentos.

4.1. Escogencia del planificador

De acuerdo con Russell y Norvig (2010), las estrategias más comunes para resolver un problema de planificación son la utilización de planificadores basados en SAT (SAT-*planners*), la búsqueda heurística y la búsqueda basada en un grafo de planificación.

Rintanen (2005) expone que para ciertos dominios es conveniente considerar la noción de planes paralelos: planes que permiten la aplicación de varias acciones "a la vez": si existen n acciones que afectan y dependen de condiciones disjuntas, hay $n!$ planes que son equivalentes (i.e., llevan al mismo estado) que serían explorados por un planificador serial. Si n es grande, esto puede ser combinatoriamente

complejo. Por tanto, un planificador que tome en cuenta planes paralelos tendría ventajas sobre un planificador serial.

Recuerde que en los dominios traducidos por la herramienta propuesta en este trabajo, las acciones `colocar-verdadera` aplicadas a diferentes variables o tuplas son completamente independientes, y por tanto paralelizables.

Existen planificadores basados en SAT que resuelven problemas de planificación paralela de manera muy eficiente. Se escogió el SAT-*planner* del estado del arte `M` (Rintanen, 2010) para realizar los experimentos, debido a su superior desempeño.

4.1.1. Ventanas de horizonte para la traducción de NP

En esta sección se derivan cotas estrictas sobre la longitud de los planes paralelos para los problemas resultantes. Estas cotas se utilizan con un planificador basado en SAT para demostrar que un problema STRIPS no tiene solución o para mejorar el desempeño de los planificadores.

Una ventana de horizonte para un problema STRIPS P es un intervalo de la forma $[i, f]$, tal que P tiene solución si y sólo si tiene un plan paralelo de tamaño $\ell \in [i, f]$. Las ventanas se pueden utilizar para podar el espacio de búsqueda.

La estructura recursiva del problema generado permite el cálculo de ventanas de horizonte no triviales. Como todas las acciones `colocar-verdadera` pueden ser aplicadas de manera concurrente, un plan paralelo necesita a lo sumo un paso para ejecutarlas. El plan también requiere de las acciones `empezar-prueba` y `probar-meta`. De este modo, la ventana de horizonte es $[2, 3]$ más la ventana de horizonte $long(\psi)$ de la sentencia ψ. Las ventanas de horizonte son definidas inductivamente por

- $long(\theta) \doteq [0, 0]$ si θ es un literal,
- $long(\wedge_{i=1}^{n}\theta_i) \doteq 1 + \bigvee_{i=1}^{n} long(\theta_i)$,
- $long(\vee_{i=1}^{n}\theta_i) \doteq 1 + \bigwedge_{i=1}^{n} long(\theta_i)$,
- $long((\exists y)\theta(\bar{x}, y)) \doteq 1 + long(\theta)$, y

– $long((\forall y)\theta(\bar{x}, y)) \doteq \|A\| + long(\theta)$,

donde $\mathcal{A}$ es la estructura asociada al problema, y las operaciones entre ventanas y escalares son $[a, b] \vee [a', b'] \doteq [\text{máx}(a, a'), \text{máx}(b, b')]$, $[a, b] \wedge [a', b'] \doteq [\text{mín}(a, a'), \text{máx}(b, b')]$ y $c + [a, b] \doteq [c + a, c + b]$.

SAT, por ejemplo, tiene la ventana $[\|\mathcal{A}\| + 5, \|\mathcal{A}\| + 6]$, lo que significa que el CNF codificado por la estructura $\mathcal{A}$ es satisfacible si y sólo si existe un plan paralelo de longitud $\|\mathcal{A}\| + 5 \leq \ell \leq \|\mathcal{A}\| + 6$.

4.1.2. Ventanas de horizonte para la traducción de PH

En esta sección se muestra cómo calcular cotas superiores e inferiores estrictas sobre la longitud de planes paralelos para fórmulas $\Phi \in$ LSO. Como en la sección anterior, se procede inductivamente sobre la estructura de Φ:

1. si $\Phi = \psi$ (fórmula de primer orden), entonces $long(\Phi) = long(\psi)$.
2. si $\Phi = (\exists R^t)\Psi$, entonces $long(\Phi) = 3 + [\![long = u]\!] + long(\Psi)$.
3. si $\Phi = (\forall R^t)\Psi$, entonces $long(\Phi) = 2^{\#t} \cdot long(\Psi) + 3 - 1$, donde
   $\#t$ es el número de tuplas de tipo t,
   $2^{\#t}$ es el número de relaciones R,
   y $2^{\#t+1} - 2$ es el número de bits cambiados cuando se incrementa un contador de $\#t$ bits desde cero hasta su máximo (Cormen, Leiserson, y Rivest, 1990).

En 2, $[\![long = u]\!]$ es 1 o 0 dependiendo de si *long* se refiere a la cota superior u o no.

4.2. Diseño de los experimentos

4.2.1. Procedimiento

Los experimentos consistieron en la modelación, traducción y resolución de problemas NP y PH. Para cada tipo de problema se ejecutó el planificador `M` sobre instancias de distintos tamaños y características, para luego analizar cuántos y cuáles instancias fueron resueltas satisfactoriamente en un tiempo limitado, y cuáles fueron los tiempos de corrida. Los detalles sobre la modelación de cada problema pueden encontrarse en el Apéndice A.

Es importante recalcar que, como se ha demostrado la correctitud de la herramienta, **siempre se puede hallar una solución (si la instancia es positiva) o demostrar que no existe tal solución (si la instancia es negativa)**. Como no se dispone de máquinas con cantidad arbitraria de memoria ni de tiempo ilimitado, en las tablas de resultados se refleja la cantidad de problemas a los que el computador pudo responder dentro de sus límites. Esto no quiere decir que la herramienta falla en algunos casos, sino que la complejidad de un problema puede exceder la cantidad de tiempo y espacio prefijado para su resolución.

4.3. Experimentos y resultados de problemas NP

Se realizaron experimentos en los problemas NP-completos: Sat, Clique, CHD (Camino *Hamiltoniano* Dirigido), 3DM (*3-Dimensional Matching*), 3Col (3-colorabilidad) y kCol (k-colorabilidad). También se computó el número cromático de grafos aleatorios utilizando la herramienta como un oráculo.

Los experimentos se realizaron en un procesador Intel Xeon corriendo a 1.86 GHz, con 2 GB de memoria RAM. Cada instancia se intentó resolver utilizando el planificador `M` durante 30 minutos, con un límite de 1GB de memoria.

A continuación se presentan los resultados desglosados por dominio. Para cada tipo de problema, la tabla muestra el número de instancias resueltas (N^*), el número total de instancias (N), el número de instancias resueltas que satisfacen la

propiedad (#pos), el número de instancias resueltas que **no** satisfacen la propiedad (#neg), y el tiempo promedio de resolución en segundos. Sobre cada tabla se incluye el tamaño de la ventana de horizonte particular del problema (*long*).

4.3.1. SAT

Las instancias de SAT fueron tomadas del repositorio SATLIB[1], un sitio web hecho por Hoos y Stützle (2000) que aloja una librería de problemas SAT que pertenecen a la región de la *fase de transición* (Gent y Walsh, 1994), es decir, tienen ciertas propiedades que los hacen difíciles de resolver. Los problemas de tipo uf20, uf50 y uf75 son instancias aleatorias con 20, 50 y 75 variables proposicionales, respectivamente, mientras que las instancias de tipo uuf50 y uuf75 son no satisfacibles de tamaño 50 y 75.

SAT: $long = [n + 5, n + 6]$

	N^*/N	#pos.	#neg.	tiempo prom. (seg)
uf20	40/40	40	0	1.7
uf50	40/40	40	0	146.7
uf75	15/40	15	0	362.1
uuf50	40/40	0	40	548.5
uuf75	1/40	0	1	1,746.4

TABLA 4.1: Resultados de M para SAT

M fue capaz de resolver todos los problemas pequeños de SAT, tardándose más en las instancias no satisfacibles (para probar que no había solución). El límite de lo resoluble por el planificador con las restricciones fijadas estuvo entre las 50 y 75 variables.

4.3.2. Problemas de grafos y 3DM

Las instancias de problemas de grafos se generaron de acuerdo con el modelo $G(n, p)$ de Bollobás (2001); es decir, grafos de n nodos con p probabilidad de

[1] `http://www.satlib.org`

ocurrencia de cada arista. Las instancias para 3DM se generaron escogiendo aleatoriamente tripletas de $\{0, \ldots, n-1\}^3$ con probabilidad p, siendo p variable.

CLIQUE: $long = [2n + 4, 3n + 7]$

	N^*/N	#pos.	#neg.	tiempo prom. (seg)
10-3	40/40	22	18	1.2
10-4	40/40	12	28	2.2
10-5	40/40	1	39	32.3
15-3	40/40	22	18	10.5
15-4	40/40	11	29	36.6
15-5	39/40	4	35	74.3
15-6	37/40	1	36	79.4
20-3	40/40	25	15	40.2
20-4	40/40	17	23	72.6
20-5	39/40	10	29	159.6
20-6	34/40	4	30	185.2
25-3	40/40	30	10	111.9
25-4	40/40	18	22	231.0
25-5	39/40	10	29	387.5
25-6	36/40	8	28	394.1

TABLA 4.2: Resultados de M para CLIQUE. Se utiliza la notación $x - y$ para denotar que el problema fue hallar una clique de tamaño y en un grafo con x nodos.

Es interesante que la herramienta sea capaz de producir problemas de *clique* resolubles en menos de cinco minutos para grafos de hasta 25 nodos, sin la utilización de ninguna heurística ni optimización específica al problema.

CHD: $long = [n + 3, n + 10]$

	N^*/N	#pos.	#neg.	tiempo prom. (seg)
10	40/40	15	25	1.1
15	39/40	18	21	63.7
20	31/40	20	11	70.0
25	29/40	20	9	202.1
30	22/40	20	2	629.1

TABLA 4.3: Resultados de M para CHD. La primera columna indica el número de nodos del grafo.

El planificador tuvo mayores dificultades en este dominio que en el anterior, a juzgar por la cantidad de problemas resueltos con el mismo número de nodos.

3DM: $long = [3n + 4, 3n + 6]$

	N^*/N	#pos.	#neg.	tiempo prom. (seg)
10	40/40	36	4	9.6
15	40/40	40	0	251.5
20	13/40	13	0	1,191.0
25	0/40	0	0	–

Tabla 4.4: Resultados de M para 3DM. La primera columna indica la cardinalidad del conjunto.

3DM es un dominio particularmente difícil: como la cota inferior es $3n + 4$, el planificador debe empezar a buscar planes que son más largos que en los dominios anteriores. De cualquier modo, es conveniente poder resolver este problema de hipergrafos para instancias pequeñas utilizando sólo planificación automática en lugar de un *software especializado*, gracias a la herramienta.

3Col: $long = [2n + 4, 2n + 7]$

	N^*/N	#pos.	#neg.	tiempo prom. (seg)
10	40/40	18	22	0.1
15	40/40	24	16	0.9
20	40/40	12	28	3.0
25	40/40	30	10	8.9
30	40/40	9	31	20.9
40	40/40	4	36	75.1
50	40/40	1	39	196.7

Tabla 4.5: Resultados de M para 3Col. La primera columna indica el número de nodos del grafo.

3Col fue el dominio que arrojó las mejores resultados, comparándolo con otros problemas de grafos de tamaño similar. Nótese que en la mayoría de los casos de los grafos más grandes no existía una 3-coloración válida, y la ventana de horizontes le permitió al planificador dar una respuesta en un tiempo relativamente corto.

Los problemas de kCol arrojaron buenos resultados hasta considerar los grafos de 25 nodos, para los cuales no se halló ninguna solución dentro de los 30 minutos por problema que se le dio al computador. Esto muestra que la dificultad de los problemas NP escala muy rápidamente, incluso cuando hay aumentos pequeños en el tamaño del problema.

kCOL: $long = [2n + 4, 3n + 6]$

	N^*/N	#pos.	#neg.	tiempo prom. (seg)
10-2	40/40	9	31	1.9
10-3	40/40	18	22	2.8
10-4	40/40	27	13	11.0
15-2	40/40	7	33	33.5
15-3	40/40	16	24	46.5
15-4	40/40	24	16	91.7
20-2	40/40	3	37	254.9
20-3	40/40	12	28	395.9
20-4	40/40	20	20	497.3
25-2	0/40	0	0	–
25-3	0/40	0	0	–
25-4	0/40	0	0	–

TABLA 4.6: Resultados de M para kCOL. La primera columna indica el número de nodos del grafo y el número de colores de los que se intenta colorear.

En total, de todos los 1920 problemas que se intentó resolver, fueron solucionadas 1614 instancias (el 84 %), 706 de las cuales eran positivas (es decir, se consiguió un plan que solucionaba el problema.

4.3.3. Número cromático

El número cromático χ de un grafo $G = (V, E)$ es el menor k tal que G es k-coloreable. Determinar este número es un problema NP-hard, pero puede hacerse probando su k-colorabilidad por valores incrementales[2] de $k = 1, \ldots, |V|$.

La Tabla 4.7 muestra los detalles de corrida de ocho instancias de grafos a los cuales se les determinó su número cromático utilizando la herramienta y el planificador M como un oráculo para decidir la colorabilidad para distintos k.

Nótese que los mayores tiempos de corrida corresponden normalmente a la prueba de k-colorabilidad para $k = \chi - 1$. Esto se debe a que para un grafo con número cromático χ no es trivial descartar que sea $(\chi-1)$-coloreable, y el programa debe agotar todas las posibles formas de colorearlo antes de rendirse e intentar con

[2]Esto se puede mejorar realizando una búsqueda binaria en k.

		k-colorabilidad						
instancia	χ	1	2	3	4	5	6	7
10-0.75 (1)	5	2	2	6	101	**3**		
10-0.75 (2)	5	1	2	2	6	**4**		
10-0.85	7	2	2	3	6	4	1,265	**4**
15-0.25	2	27	**62**					
15-0.60	5	27	29	54	118	**72**		
15-0.70	6	28	28	33	47	329	**67**	
20-0.10	3	214	350	**705**				
20-0.25	4	211	272	1,261	**837**			

TABLA 4.7: Resultados de M sobre el cómputo de números cromáticos en grafos aleatorios. La primera columna muestra el número de nodos del grafo y su probabilidad según el modelo $G(n, p)$. Para cada instancia, la tabla muestra el número cromático χ, el tiempo (en segundos) para demostrar o descartar la k-colorabilidad para valores incrementales de k. El último valor de k para cada instancia es el número cromático.

un color adicional. No se logró determinar qué propiedad del grafo causó mayores tiempos de corrida en el último valor de k para las instancias 15-0.25 y 20-0.10.

4.4. Experimentos y resultados de problemas PH

Se llevaron a cabo diferentes experimentos para probar la dificultad de los problemas generados. Las dos categorías de problemas estudiados fueron QBF, un problema que se ubica en PH, y el complemento al problema de 3-colorabilidad de un grafo, $\overline{\text{3Col}}$, perteneciente a la clase co-NP. Para QBF se utilizó M, el mismo planificador basado en SAT con el que se resolvieron los problemas NP. Para $\overline{\text{3Col}}$ se utilizó otro planificador del estado del arte, llamado LAMA'11 (Richter y Westphal, 2010).

Los experimentos se ejecutaron en un CPU Xeon 5140 corriendo a 2.33 GHz, con 2 GB de RAM y un tiempo límite de una hora y media.

4.4.1. QBF

Se realizaron experimentos sobre varios tipos de problemas QBF (*quantified Boolean formula*). QBF es una generalización de SAT en la cual se tiene una fórmula CNF con variables cuantificadas existencial o universalmente. Por ejemplo, la fórmula

$$(\exists y_1 y_2)\ (\forall x_1 x_2)\ ((x_1 \vee \neg y) \wedge (\neg x_2 \vee y) \wedge (\neg x_1 \vee \neg y_2 \vee y_1))$$

expresa el problema de hallar una asignación de verdad para y_1 y y_2 tal que la fórmula interna sea satisfecha por todas las posibles asignaciones de verdad de x_1 y x_2. Los QBF estudiados, que se ubican en los niveles iniciales de la jerarquía polinomial, son de la forma siguiente:

Forma del QBF	clase de complejidad
$\exists\forall$	Σ_p^2
$\forall\exists$	co-Σ_p^2
$\exists\forall\exists$	Σ_p^3
$\forall\exists\forall$	co-Σ_p^3

Por ejemplo, la fórmula Φ para $\forall\exists$-QBF es:

$$\begin{aligned}&(\forall T_1^{\mathrm{V}_1})(\exists T_2^{\mathrm{V}_2})(\forall y \in \mathrm{Cls})(\exists x \in \mathrm{Var})\\ &\quad\big[P(x,y) \wedge [(\mathrm{V}_1(x) \wedge T_1(x)) \vee (\mathrm{V}_2(x) \wedge T_2(x))]\big] \vee\\ &\quad\big[N(x,y) \wedge [(\mathrm{V}_1(x) \wedge \neg T_1(x)) \vee (\mathrm{V}_2(x) \wedge \neg T_2(x))]\big]\end{aligned}$$

donde V_1 y V_2 son tipos que particionan las variables en universales y existenciales.

Se presentan los resultados de resolver, utilizando el planificador M, los cuatro tipos de problemas QBF mencionados, traducidos por la herramienta. Las instancias son aleatorias, generadas con BlocksQBF[3], que está basado en el modelo aleatorio de Chen y Interian (2005). Cada problema tiene 150 cláusulas. En las columnas de las tablas se muestra el número de variables de cada tipo, el número de instancias que se intentaron resolver (n), el número de instancias resueltas con

[3] `http://fmv.jku.at/blocksqbf`

solución (+) y en las que no hay solución (−), y promedios de tiempo en segundos y longitud del plan paralelo (para las instancias resueltas donde hay solución).

∃∀-QBF

#∃	#∀	n	+	−	tiempo (seg)	long. plan
60	1	5	–	5	184.3	–
	2	5	–	1	4,382.5	–
100	1	5	4	1	176.6	316
	2	5	3	2	3,471.9	628

De los problemas QBF-∃∀ con 60 variables existenciales y una universal fueron resueltos en tiempos relativamente cortos, aunque no al nivel de un solucionador especializado de problemas QBF. Es probable que las instancias con 60 variables existenciales y 2 universales no tuvieran ninguna solución, pero el planificador no pudo determinarlo porque fue abortado al agotarse el tiempo estipulado para los experimentos. Con 100 variables existenciales, el planificador encuentra soluciones con mayor facilidad. El incremento de tiempo al agregar una sola variable universal adicional es enorme, pues debe construirse una prueba para todos los modelos posibles de dos variables independientes.

∃∀∃-QBF

#∃	#∀	#∃	n	+	−	tiempo (seg)	long. plan
10	2	30	5	–	5	4,199.2	–
	2	50	5	–	5	2,313.9	–
30	2	30	5	–	5	3,210.7	–
	2	50	5	–	5	3,166.3	–
50	2	30	5	–	1	3,313.4	–
	2	50	5	3	2	3,450.9	640

El planificador pudo determinar que la mayoría de los problemas ∃∀∃-QBF no tenían solución. Los tiempos fueron cercanos al límite de 1.5 horas, y los planes no fueron significativamente más largos que en los problemas ∃∀-QBF, debido a que se mantuvo fijo el número de variables universales (que son las responsables por un crecimiento exponencial en el tamaño del plan).

Los tiempos y longitudes en los problemas ∀∃-QBF fueron muy similares a los de problemas complementarios ∃∀-QBF. Como era de esperarse, las instancias con más de dos variables universales no pudieron ser resueltos, ni siquiera al disminuir

∀∃-QBF

#∀	#∃	n	+	−	tiempo (seg)	long. plan
1	100	5	5	–	147.9	319
2	30	5	–	5	2,060.4	–
	60	5	4	1	3,100.7	637
	80	5	5	–	2,893.2	637
	100	5	5	–	2,092.8	637
3	15	5	–	–	–	–

significativamente el número de variables existenciales buscando que el planificador pudiera demostrar que los problemas no tienen solución. Para solucionar estas instancias haría falta mucho más tiempo, y se prevé que la longitud de los planes sea mucho mayor.

∀∃∀-QBF

#∀	#∃	#∀	n	+	−	tiempo (seg)	long. plan
1	60	1	5	1	–	404.7	635
	60	2	5	–	–	–	–
	80	1	5	5	–	403.2	635
	100	1	5	5	–	390.8	635
2	60	1	5	2	–	456.7	1,269
	60	2	5	–	–	–	–
	80	1	5	5	–	499.5	1,269
	100	1	5	5	–	454.9	1,269

Puede resultar curioso que las pruebas de ∀∃∀-QBF para dos variables cuantificadas universalmente en el nivel más externo dieron como resultado mejores tiempos que las del tipo ∃∀-QBF que tenían dos variables dependientes de la cuantificación universal. Esto se debe a que en estas últimas pruebas había instancias negativas, por lo que era necesario evaluar todos los posibles valores de la cuantificación existencial, incurriendo en un mayores tiempos de corrida.

4.4.2. $\overline{\textbf{3Col}}$

El complemento de 3Col, $\overline{\text{3Col}}$, es el problema de decisión que pregunta si un grafo **no** es 3-coloreable. En el caso de que la respuesta para un grafo sea afirmativa, la prueba de ello debe mostrar que se realizó la asignación de los tres

colores posibles a todos los nodos del grafo, y en todos los casos fue imposible probar que la coloración era válida. Este problema está en co-NP, su formulación detallada se encuentra en el Apéndice A.

Para probar este dominio, se generaron nuevamente grafos aleatorios con el modelo $G(n, p)$. Los experimentos se realizaron en la misma máquina y con las mismas restricciones que los experimentos de problemas QBF.

En este caso no se obtuvo buenos resultados con el planificador M. Los resultados de LAMA'11 (Richter y Westphal, 2010) se muestran en la Tabla 4.8.

$\|V\|$	n	$+$	$-$	tiempo (seg)	long. plan
4	5	1	4	0.6	1,731
5	5	2	3	41.0	6,695
6	5	2	3	280.3	26,163
7	5	2	2	38.2	102,935
8	5	1	2	211.9	406,851
9	5	–	1	0.3	–

TABLA 4.8: $\overline{\text{3Col}}$ en grafos aleatorios. Las columnas muestran el número de vértices ($|V|$), el número de instancias (n), el número de instancias positivas ($+$) y negativas ($-$), los tiempos promedio en segundos y la longitud del plan de las instancias positivas resueltas.

Sorprendentemente, LAMA'11 es capaz de encontrar planes sumamente largos: en una instancia consiguió uno con más de 400.000 pasos. Se cree que la razón por la cual este planificador es capaz de lograr semejante hazaña es que, a diferencia de M, es de tipo secuencial y la resolución de problemas en co-NP no requiere de conjeturas (pues no hay cuantificadores existenciales de segundo orden), sino de la aplicación lineal de acciones. En este caso llama la atención que el planificador tardó una gran cantidad de tiempo en el grafo de 6 vértices a pesar de que la longitud del plan no era excesivamente grande. La razón exacta de esto es desconocida.

4.5. La herramienta en la web

Para la presentación de este trabajo en la Conferencia Internacional de Planificación Automática (ICAPS '11) se implementó una sencilla aplicación web

utilizando *Python* y tecnología *CGI*.

La aplicación web consta de una sección en donde el usuario puede introducir una fórmula Φ en LSO y una firma σ para generar un dominio de planificación en PDDL, utilizando el lenguaje lógico y la sintaxis de archivos de texto definidos en el Capítulo 2 sobre campos de texto libres en la ventana del navegador. El usuario puede escoger si desea utilizar la primera reducción, diseñada para problemas NP, o la segunda, que le permite traducir problemas de lógica de segundo orden general.

Al haber generado un dominio, se le da al usuario la posibilidad de descargar el dominio traducido en PDDL, y se le presenta otro campo de texto para permitirle especificar una estructura finita $\mathcal{A} \in \text{STRUC}[\sigma]$. Al completar este paso, el usuario tiene la posibilidad de descargar tanto el dominio como el problema PDDL, archivos que puede utilizar con el planificador de su preferencia para la resolución de problemas.

En el sitio web también se proveen fórmulas predefinidas para los problemas NP y PH estudiados en este trabajo, a modo de facilitar el uso de la herramienta con problemas frecuentes a los potenciales usuarios. Se prevé que la herramienta web esté disponible bajo el subdominio del Grupo de Inteligencia Artificial de la Universidad Simón Bolívar, en `http://www.gia.usb.ve`.

Capítulo 5

Conclusiones y Recomendaciones

Este trabajo ha presentado dos reducciones de la lógica de segundo orden a problemas de planificación automática que se adhieren a la Teoría de Complejidad Descriptiva. La herramienta recibe como entrada una firma σ, una sentencia en lógica de segundo orden (segundo orden existencial, en el caso de la traducción de NP) y una estructura $\mathcal{A} \in \text{STRUC}[\sigma]$, y producen como salida un par de archivos en PDDL $\langle \mathsf{dom}, \mathsf{ins} \rangle$, que representan un dominio y una instancia de planificación en STRIPS. La conversión corre en tiempo polinomial en el tamaño de la estructura $\|\mathcal{A}\|$, y por lo tanto puede usarse como un método eficiente para generar reducciones de problemas NP a STRIPS. Esta salida puede utilizarse con un planificador para obtener una solución al problema original.

Los resultados de los experimentos muestran que la herramienta puede utilizarse para resolver problemas específicos en NP o PH, observando que tienen un buen desempeño sobre problemas pequeños. Así, por ejemplo, un profesional del área de computación que tenga la necesidad de resolver problemas pequeños de grafos, o requiera realizar pruebas de concepto para estudiar problemas académicos o de la vida real, puede seguir el siguiente proceso para conseguir una solución rápida que cumpla con sus objetivos:

1. Comparar el problema de interés con problemas de decisión conocidos en el área de teoría de grafos, optimización, u otras.

2. Modelar el problema de interés utilizando lógica de segundo orden, y escribirlo en la sintaxis propuesta por este trabajo.

3. Utilizar la herramienta para producir el dominio y distintas instancias en PDDL. Usar un planificador para conseguir soluciones a estas instancias y analizar las soluciones arrojadas para constatar cómo se interpretan en el contexto del problema de interés.

Debe notarse que no todos los problemas son fácilmente modelables utilizando un lenguaje descriptivo como el que hemos desarrollado en este trabajo. Así como no existe un lenguaje de programación **ideal** que permita la expresión de cualquier problema de una forma sencilla, sino que existen distintos **paradigmas** y el programador debe tener la experticia de reconocer cuál es el más adecuado para cada problema determinado, no hay una única forma "óptima" de tratar con todos los problemas de decisión. Por ejemplo, sería particularmente engorroso intentar modelar el problema de *subset sum* (**suma de subconjuntos**), utilizando el lenguaje lógico descrito, a pesar de que es un problema NP-completo, pues se debe simular un sistema formal aritmético que permita la suma de números en tal lógica. La teoría establece que es posible, pero no se recomienda el uso de la herramienta para ello ya que el ejercicio puede tomar mucho tiempo.

Además, la herramienta es de interés para los investigadores del área de planificación automática, pues se conocen ciertas propiedades acerca de los problemas de planificación que genera, incluyendo los horizontes de búsqueda en los que se encuentra una solución si ésta existe. Los problemas pueden utilizarse como una medida de comparación entre diferentes enfoques de cómo construir planificadores generales y eficientes, particularmente en el caso de *SAT-planners* y en la búsqueda de planes paralelos.

La reducción de NP a planificación expuesta en este trabajo fue presentada en el mes de junio de 2011 en la Conferencia Internacional de Planificación Automática (ICAPS '11) en la ciudad de Freiburg, Alemania (Porco, Machado, y Bonet, 2011). La extensión a la herramienta que le permite procesar problemas en PH ha sido propuesta como artículo de investigación corto en ICAPS '13, conferencia que tendrá lugar en Roma, Italia. Se recibirán noticias sobre su aceptación en enero de 2013.

Para seguir adelante con esta línea de investigación se proponen las siguientes recomendaciones:

– Diseñar una segunda extensión a la herramienta, modificando el lenguaje para incorporar el formalismo de **clausura transitiva**, necesario para resolver problemas generales de la clase de complejidad PSPACE. Con tal extensión, el poder expresivo del lenguaje presentado aquí se equipararía a PDDL.

– Optimizar las reducciones para distintos tipos de planificadores, ya que el trabajo se concentró en planfificadores basados en SAT. Como punto de partida podría investigarse a fondo por qué el planificador LAMA '11 tuvo mejores resultados que M en el dominio $\overline{\text{3Col}}$.

Bibliografía

Bollobás, B. 2001. *Random Graphs*. Cambridge University Press, second edition.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. 69:165–204.

Cadoli, M.; Palopoli, L.; Schaerf, A.; y Vasile, D. 1999. NP-SPEC: An executable specification language for solving all problems in NP. In *Proc. 1st. Int. Workshop on Practical Aspects of Declarative Languages*.

Cadoli, M., y Mancini, T. 2006. Automated reformulation of specifications by safe delay of constraints. 170.

Chen, H., y Interian, Y. 2005. A model for generating random quantified boolean formulas. In *In Proc. of 9th International Joint Conference on Artificial Intelligence*, 66–71.

Cormen, T.; Leiserson, C.; y Rivest, R. 1990. *Introduction to Algorithms*. MIT Press.

Fagin, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. *American Mathematical Society* 7:27–41.

Fikes, R., y Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. 1:27–120.

Fox, M., y Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. See `http://www.dur.ac.uk/d.p.long/competition.html`.

Gent, I. P., y Walsh, T. 1994. The sat phase transition. 105–109. John Wiley And Sons.

Ghallab, M.; Nau, D. S.; y Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Hoos, H. H., y Stützle, T. 2000. Satlib: An online resource for research on sat. 283–292. IOS Press.

Immerman, N. 1998. *Descriptive Complexity*. Springer.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; y Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

Mitchell, D., y Ternovska, E. 2005. A framework for representing and solving np search problems. In *Proc. 20th. Nat. Conf. on Artificial Intelligence*.

Porco, A.; Machado, A.; y Bonet, B. 2011. Automatic polytime reductions of NP problems into a fragment of STRIPS. In *Proc. 21st Int. Conf. on Automated Planning and Scheduling*. Freiburg, Germany: AAAI Press.

Reiter, R. 1978. On closed-world data bases. In Gallaire, H., y Minker, J., eds., *Logic and data bases*. Plenum Press.

Richter, S., y Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. In *Journal of Artificial Intelligence Research 39*, 127–177.

Rintanen, J. 2005. Introduction to automated planning, course notes.

Rintanen, J. 2010. Heuristics for planning with SAT. In Cohen, D., ed., *Proc. 16th Int. Conf. on Principles and Practice of Contraint Programming*, 414–428. St. Andrews, Scotland: Springer: LNCS 6308.

Russell, S., y Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Pearson Education, Third edition.

Wikipedia. 2004. 3-dimensional matchings — Wikipedia, the free encyclopedia. [en línea; accedido el 5 de diciembre de 2012].

Apéndice A

Problemas NP y PH modelados en lógica

A.1. Problemas NP

A.1.1. SAT, Satisfacibilidad proposicional

Existe una asignación de verdad T, tal que para toda cláusula y, existe una variable x, tal que: x está positiva en y y $T(x)$, o x está negativa en y y $not - T(x)$,

```
(so-exists (?T 1)
  (forall (?y)
    (exists (?x)
      (or
        (and (?P ?x ?y)
             (?T ?x))
        (and (?N ?x ?y)
             (not (?T ?x)))))))
```

A.1.2. CLIQUE, Clique de tamaño k

En esta fórmula se utiliza la relación unaria K para modelar la constante k del problema. Si se quiere que k sea igual a 3, por ejemplo, se agrega a la instancia `?K zero`, `?K obj1` y `?K obj2`.

```
(so-exists (?F Inj)
    (forall (?x) (forall (?y)
      (implies (and (< ?x ?y)
                    (exists (?z) (and (?F ?x ?z) (?K ?z)))
                    (exists (?z) (and (?F ?y ?z) (?K ?z))))
               (?E ?x ?y)))))
```

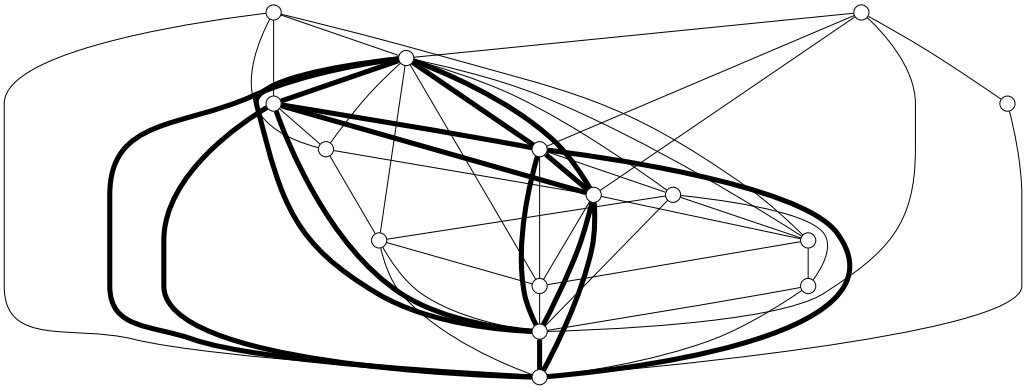

FIGURA A.1: Grafo con una clique de tamaño $k = 6$

A.1.3. CHD, Camino Hamiltoniano Dirigido

Esta fórmula expresa que existe una manera ordenada $F(a, b)$ de visitar los nodos de grafo (la posición a la tiene el nodo b), en la que para toda posición x, si no es la última, entonces existe un nodo y_1 que le corresponde esta posición, y además a la siguiente posición x_2 le corresponde otro nodo y_2, y hay un arco entre y_1 y y_2.

```
(so-exists (?F Inj)
```

```
(forall (?x)
  (implies (< ?x MAX)
           (exists (?y1)
             (and (?F ?x ?y1)
               (exists (?y2)
                 (and
                   (exists (?x2)
                     (and (SUC ?x ?x2) (?f ?x2 ?y2)))
                   (?E ?y1 ?y2)))))))
```

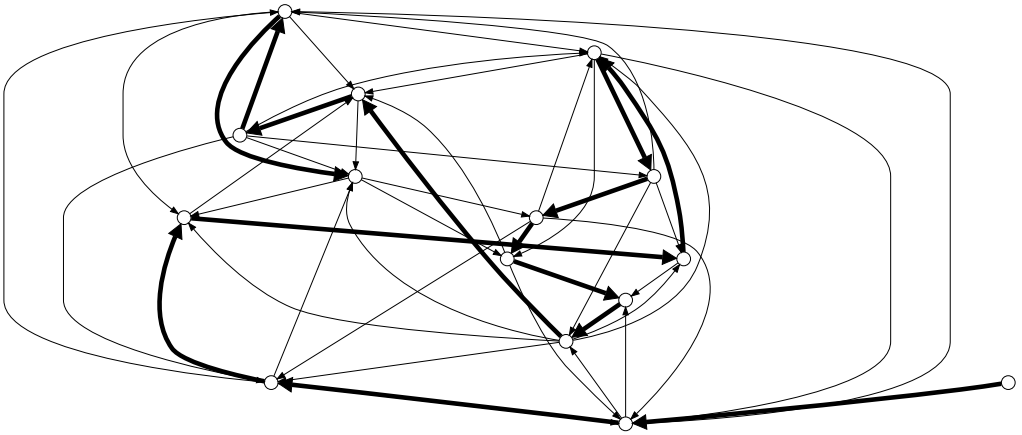

FIGURA A.2: Grafo con *camino hamiltoniano*

A.1.4. 3DM, *3-Dimensional Matching*

Esta fórmula expresa que existen dos funciones biyectivas $F(x, y)$ y $G(y, z)$, que por transitividad entre ellas simulan un subconjunto de tripletas x-y-z de $T(x, y, z)$, que no comparten ninguno de sus elementos (propiedad lograda la biyectividad de dichas funciones).

```
(so-exists (?F Inj)
  (so-exists (?G Inj)
    (and
      (forall (?x) (exists (?y) (?F ?x ?y))) ; F is total
      (forall (?x) (exists (?y) (?G ?x ?y))) ; G is total
```

```
(forall (?x) (forall (?y) (forall (?z)
  (implies (and (?F ?x ?y) (?G ?x ?z))
           (?T ?x ?y ?z))))))))
```

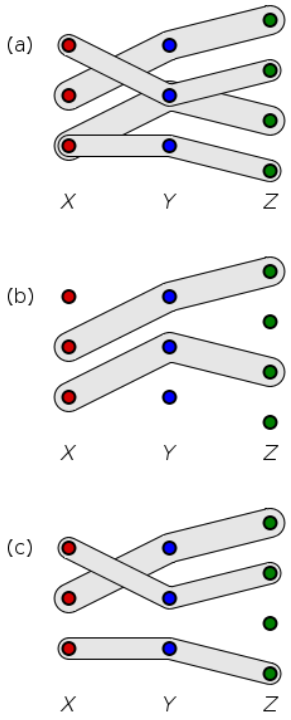

FIGURA A.3: Ejemplo de *3-Dimensional Matching*, (Wikipedia, 2004)

A.1.5. 3Col, 3-colorabilidad

Esta fórmula expresa que existe una asignación de colores R, G y B tal que para todos los nodos de un grafo, no hay dos vertices adyacentes del mismo color, los vertices tienen almenos y a lo sumo un color.

```
(so-exists (?R 1) (so-exists (?G 1) (so-exists (?B 1)
  (forall (?x)
    (and
      ; no hay dos vértices adyacentes del mismo color
      (forall (?y)
        (implies (?E ?x ?y) (not (or (and (?R ?x) (?R ?y))
                                     (and (?G ?x) (?G ?y))
                                     (and (?B ?x) (?B ?y))))))
```

```
; los vértices tienen al menos un color
(or (?R ?x) (?G ?x) (?B ?x))

; los vértices tienen a lo sumo un color
(implies (?R ?x) (and (not (?G ?x)) (not (?B ?x))))
(implies (?G ?x) (and (not (?R ?x)) (not (?B ?x))))
(implies (?B ?x) (and (not (?R ?x)) (not (?G ?x)))))))))
```

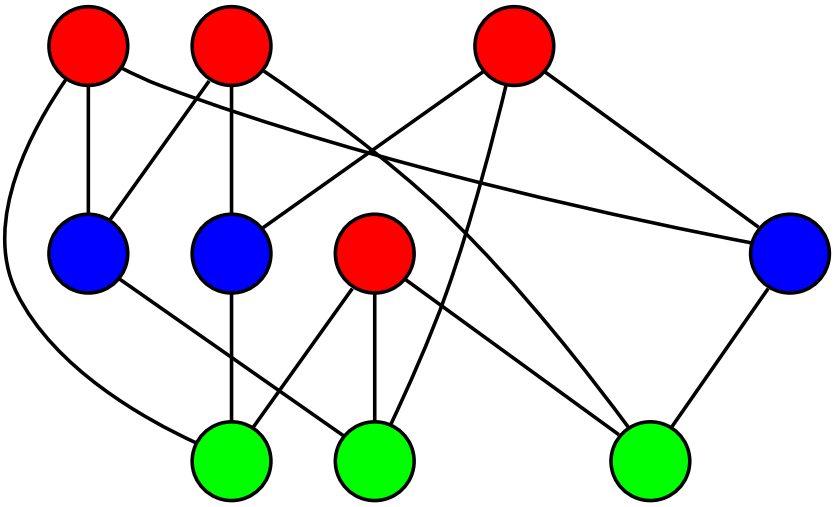

FIGURA A.4: Grafo 3-coloreable

A.1.6. kCol, k-colorabilidad

Esta fórmula expresa que existe una función de asignación de colores F, tal que a todo nodo se le es asignado un color y, donde $K(y)$ quiere decir que el color y es menor al máximo color k; ademá, para todos los otros nodos, si este esta conectado a ellos, no tienen el mismo color.

```
(so-exists (?F Func)
```

```
(forall (?x)
  (and (exists (?y) (and (?F ?x ?y) (?K ?y)))
    (forall (?y)
      (implies (?E ?x ?y)
               (not (exists (?z)
                 (and (?F ?x ?z) (?F ?y ?z)))))))))
```

A.2. Problemas PH

A.2.1. UNSAT, No-Satisfacibilidad proposicional

Esta fórmula expresa que para toda relación T sobre variables proposicionales, existe una cláusula y tal que para cada variable x, o x aparece positiva en y y x es falsa, o x aparece negativa en y y x es verdadera, o x no aparece en y.

```
(so-forall (?T 1 @ist)
    (exists (?y @cls)
        (forall (?x @var)
            (or (and (?N ?x ?y)
                     (?T ?x)
                )
                (and (?P ?x ?y)
                    (not (?T ?x))
                )
                (?NotIn ?x ?y)
             ))))
```

A.2.2. $\exists\forall$-QBF, Fórmula *booleana* cuantificada Σ_p^2

Esta fórmula expresa que existe una relación `E0` sobre variables proposicionales, tal que para toda relación `A0` sobre variables proposicionales, para toda cláusula

c existe una variable x, tal que: o x aparece positiva en c y es verdadera existencial o universalmente, o x aparece negativa en c y es negativa existencial o universalmente.

```
(so-exists (?E0 1 @ise0)
    (so-forall (?A0 1 @isa0)
        (forall (?c @cls)
            (exists (?x @var)
                (or
                    (and (?P ?x ?c) (?E0 ?x))
                    (and (?P ?x ?c) (?A0 ?x))
                    (and (?N ?x ?c) (not (?E0 ?x)))
                    (and (?N ?x ?c) (not (?A0 ?x)))
                )))))
```

A.2.3. $\overline{\text{3Col}}$, No-3-Colorabilidad

Esta fórmula expresa que para toda posible coloración de nodos, donde los colores son expresados con las relaciones R y G, o dos nodos unidos por un arco tienen el mismo color, o un nodo tiene dos colores al mismo tiempo.

```
(so-forall (?R 1 @node)
    (so-forall (?G 1 @node)
        (exists (?x @node)
            (or
                (exists (?y @node)
                    (or (and (?E ?x ?y) (?R ?x) (?R ?y))
                        (and (?E ?x ?y) (?G ?x) (?G ?y))
                        (and (?E ?x ?y) (not (?R ?x))
                             (not (?R ?y)) (not (?G ?x))
                             (not (?G ?y)))
                    )
                )
                (and (?R ?x) (?G ?x))))))
```

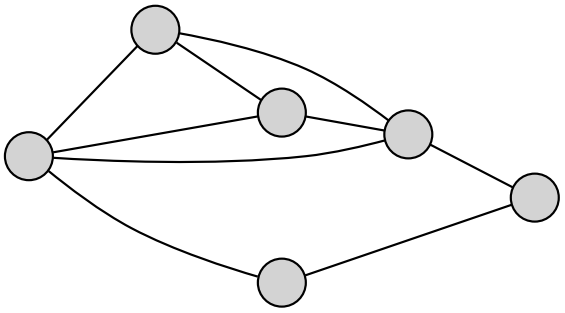

FIGURA A.5: Ejemplo de grafo No-3-Coloreable

Apéndice B

Caso de estudio: SAT

En este sección se considera un problema pequeño de SAT y se describe el procedimiento de traducción y obtención de un plan. La instancia de SAT que se resolverá es

$$(p \vee \neg q \vee r) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee q). \tag{B.1}$$

B.1. Dominio

B.1.1. Entrada

La modelación del dominio SAT fue descrita en la sección 1.3. Utilizando el lenguaje lógico que se ha definido en este capítulo, tenemos que Φ_{SAT} es:

```
(so-exists (?T 1) (forall (?y) (exists (?x)
      (or (and (?P ?x ?y) (?T ?x))
          (and (?N ?x ?y) (not (?T ?x)))))))
```

La firma σ_{SAT} consta de las relaciones N y P, las cuales varían de instancia a instancia y definen las cláusulas CNF del problema.

B.1.2. Salida

La herramienta crea el árbol sintáctico de Φ_{SAT}, el cual se muestra en detalle en la Figura B.1, y genera el archivo PDDL que se presenta a continuación.

```
(define (domain SAT)
  (:constants zero max)

  (:predicates
    (es_cierto_conjuncion_2 ?x ?y) (es_cierto_conjuncion_6 ?x0 ?x1)
    (es_cierto_existencial_8 ?x0) (es_cierto_universal_9 ?x0)
    (es_cierto_disyuncion_7 ?x0 ?x1) (es_cierto_meta)
    (N ?x ?y) (P ?x ?y) (T ?x) (no-T ?x)
    (SUC ?x ?y)
  )
```

El encabezado del archivo enumera las proposiciones del problema de planificación.

```
  (:action colocar_verdadera_T
    :parameters (?x)
    :precondition (and (conjetura) (no-T ?x))
    :effect (and (T ?x) (not (not-T ?x))))
```

La acción `colocar_verdadera_T` establece que su parámetro de entrada x, una variable proposicional, se considerará **verdadera** en la construcción de la prueba. Si esta acción no se aplica sobre una variable proposicional, se asume el caso contrario (la variable es falsa), como es descrito en la sección 2.6.

```
  (:action empezar-prueba
    :precondition (conjetura)
    :effect (and (prueba) (not (conjetura)))) )
```

La acción `empezar-prueba` es necesaria para que el planificador pase de la fase de conjetura a la fase de construcción de la prueba utilizando los valores de T previamente escogidos.

```
(:action probar_conjuncion_6
  :parameters (?x ?y)
  :precondition (and (prueba)
                     (N ?x ?y) (not-T ?x))
  :effect (es_cierto_conjuncion_6 ?x ?y))

(:action probar_conjuncion_2
  :parameters (?x ?y)
  :precondition (and (prueba)
                     (P ?x ?y) (T ?x))
  :effect (es_cierto_conjuncion_2 ?x ?y))
```

Arriba se muestra la traducción de las dos subfórmulas más internas. Establecer una conjunción de dos proposiciones se logra al comprobar que ambas son ciertas por separado. Nótese que se agrega la proposición `prueba` como precondición, por lo que la acción `empezar-prueba` debe ejecutarse antes que cualquiera de estas acciones.

```
(:action probar_disyuncion_7_0
  :parameters (?x ?y)
  :precondition (and (prueba)
                     (es_cierto_conjuncion_2 ?x ?y))
  :effect (es_cierto_disyuncion_7 ?x ?y))

(:action probar_disyuncion_7_1
  :parameters (?x ?y)
  :precondition (and (prueba)
                     (es_cierto_conjuncion_6 ?x ?y))
  :effect (es_cierto_disyuncion_7 ?x ?y))
```

Para probar la disyunción es suficiente probar cualquiera de sus operandos, en este caso las conjunciones internas. Por esta razón se incluyen dos acciones que agregan la proposición `es_cierto_disyuncion_7`.

```
(:action probar_existencial_8
  :parameters (?x ?y)
  :precondition (and (prueba)
                     (es_cierto_disyuncion_7 ?x ?y))
  :effect (es_cierto_existencial_8 ?x))

(:action probar_universal_base_9
  :precondition (and (prueba)
                     (es_cierto_existencial_8 zero))
  :effect (es_cierto_universal_9 zero))

(:action probar_universal_inductivo_9
  :parameters (?y1 ?y2)
  :precondition (and (prueba)
                     (SUC ?y1 ?y2)
                     (es_cierto_universal_9 ?y1)
                     (es_cierto_existencial_8 ?y2))
  :effect (es_cierto_holds_forall_9 ?y2))
```

Las acciones que hacen la demostración de los operadores *existencial* y *para todo* se muestran arriba. Nótese como el planificador debe iterar sobre todos los objetos para probar inductivamente el operador universal.

```
(:action probar-meta
  :precondition (es_cierto_universal_9 max)
  :effect (es_cierto_meta))
```

Finalmente, la última acción establece que el estado meta es alcanzable cuando se haya demostrado el operador universal para el último objeto (es decir, para todos).

FIGURA B.1: Árbol sintáctico que construye el *parser* al analizar la sentencia Φ_{SAT}. Los nodos terminales se marcan con un óvalo.

B.2. Problema

B.2.1. Entrada

La estructura finita que codifica la instancia descrita por la fórmula B.1. es $\mathcal{A} = \langle |\mathcal{A}| = \{\texttt{zero}, \texttt{obj1}, \texttt{max}\}, P, N\rangle$, donde P y N son descritos con un archivo de texto de entrada que contiene las líneas:

```
; primera cláusula
(P zero zero)
(N obj1 zero)
(P max zero)

; segunda cláusula
(N zero obj1)
(N max obj2)

; tercera cláusula
(N zero max)
(P obj1 max)
```

Esto codifica $(p \vee \neg q \vee r) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee q)$, con p renombrada a `zero`, q a `obj1`, y r a `max`.

B.2.2. Salida

La salida de la herramienta es el siguiente archivo PDDL que establece las condiciones iniciales del problema:

```
(define (problem p)
    (:domain sat)
    ; zero y max fueron definidas como constantes del dominio
    (:objects obj1)
```

```
    (:init
        (guess)
        (SUC zero obj1)
        (SUC obj1 max)
        (not_t zero)
        (not_t obj1)
        (not_t max)
        (P zero zero)
        (N obj1 zero)
        (P max zero)
        (N zero obj1)
        (N max obj1)
        (N zero max)
        (P obj1 max)
    )
    (:goal (es_cierto_meta))
)
```

B.3. Solución

Se utilizó M para hallar una solución al problema de planificación generado por la herramienta. El plan encontrado fue el siguiente:

```
0 : (colocar_verdadera_T max)
1 : (colocar_verdadera_T obj1)
2 : (empezar-prueba)
3 : (probar_conjuncion_2 zero max)
4 : (probar_conjuncion_6 max zero)
5 : (probar_conjuncion_6 obj1 zero)
6 : (probar_disyuncion_7_0 zero max)
7 : (probar_disyuncion_7_1 max zero)
8 : (probar_disyuncion_7_1 obj1 zero)
9 : (probar_existencial_8 max zero)
10 : (probar_existencial_8 obj1 zero)
```

```
11 : (probar_existencial_8 zero max)
12 : (probar_universal_base_9)
13 : (probar_universal_inductivo9 zero obj1)
14 : (probar_universal_inductivo9 obj1 max)
15 : (probar-meta)
```

La primera parte del plan, antes del operador `empezar-prueba`, es la respuesta al problema de hallar una asignación de verdad a las variables p, q y r que satisfaga la fórmula B.1. La asignación es $T = \{\texttt{obj1}, \texttt{max}\}$, que es equivalente a $p = false$, $q = true$, $r = true$. La segunda parte del plan muestra la demostración de que la asignación escogida hace de Φ_{SAT} una fórmula válida para la estructura de entrada.