UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Lógicas para razonar sobre grafos con datos

Tesis presentada para optar al título de Doctor de la Universidad de Buenos Aires
en el área de Ciencias Matemáticas

Edwin Pin Baque

Director:	Santiago Figueira
Director adjunto:	Diego Figueira
Consejera de estudios:	Teresa Krick

Buenos Aires, 2025

Lógicas para razonar sobre grafos con datos

Resumen

En esta tesis analizamos varios problemas relevantes en el área de Teoría de Bases de Datos y nos enfocamos particularmente en casos donde la información se almacena en una estructura con forma de grafo. En la actualidad resulta indispensable realizar estudios formales sobre este tipo de estructuración de los datos debido a la creciente popularidad de sistemas para el manejo de información como la World Wide Web, o su extensión la Web Semántica, que basan su operabilidad en tecnologías que generan y analizan la información almacenada en documentos con formatos como XML o grafos RDF. En general, el mantenimiento de una base de datos suele darse a través de lenguajes ontológicos y para estructuras con forma de grafos han surgido varias propuestas en las últimas décadas, como el Web Ontology Language (OWL), que es una familia de lógicas de descripción usadas para gestionar la forma de un grafo a través de meta-información, o el Shapes Constraint Language (SHACL) que valida la información de un grafo RDF a través de formas. Similarmente, se han creado lenguajes para realizar consultas sobre grafos, la mayoría de los cuales toman como base implementaciones navegacionales relativas a propiedades básicas de grafos, como accesibilidad, conectividad, verificación de caminos, etc.

Bajo esta noción, muchos planteamientos teóricos surgen naturalmente, por ejemplo, la inconsistencia de una base de datos respecto a un conjunto de restricciones es un problema básico que se puede analizar bajo un rigor lógico. Por un lado, hay varias maneras de formalizar esta situación, que depende tanto de las clases de estructuras a considerar como del lenguaje ontológico utilizado para razonar sobre tales estructuras. Y por otro lado, hay varias maneras de lidiar con bases inconsistentes en la práctica, ya sea tratando de eliminar la inconsistencia manipulando la base de datos, o manteniendo el estado actual de la base pero identificando la información *certera*, es decir, los datos que prevalecerán independientemente del tipo de modificación que se haga sobre la base original. Concretamente, los problemas que se estudiaron en este trabajo son

- Reparación de un data-grafo inconsistente respecto a un conjunto de restricciones definido sobre el lenguaje GXPath y varios de sus fragmentos;
- Controlabilidad finita del problema de decisión Ontology Mediated Query Answering (OMQA), cuyos resultados fueron caracterizados por ciertos patrones de grafos.

En el primer punto, se obtuvieron múltiples resultados de complejidad según distintos criterios de reparación y se analizaron algunos casos donde el método de reparación es determinístico. En el segundo punto se definió un tipo de grafo subyacente especial asociado a consultas regulares, que llamamos *esqueleto*, con el que se dio una descripción completa de la propiedad de controlabilidad finita para OMQA en base a características topológicas específicas de los esqueletos.

En otra línea de trabajo, también se planteó un nuevo lenguaje denominado CPDL$^+$, que extiende al conocido Propositional Dynamic Logic (PDL), con un operador de programas recursivo que llamamos *programa conjuntivo* y que es compatible con los demás operadores sintácticos de PDL. La semántica de este lenguaje se da a través de Estructuras de Kripke, que es una variación de sistemas de transición y que puede entenderse como un grafo con múltiples etiquetas en ejes

y nodos. En relación a los puntos ya discutidos, CPDL$^+$ puede entenderse como un lenguaje de consulta sobre data-grafos. Para este lenguaje estudiamos varios problemas lógicos y computacionales que avalan su importancia. Demostramos que extiende estrictamente a distintas familias de lenguajes; que el problema de satisfacibilidad de CPDL$^+$ es decidible; que tiene la propiedad de modelo con ancho de árbol acotado (es decir, si una fórmula φ de CPDL$^+$ es satisfacible, es satisfacible en un modelo con ancho de árbol acotado); que tiene un juego de bisimulación que lo caracteriza; y analizamos el problema model checking asociado. Además, muchos de estos resultados se obtuvieron analizando distintos fragmentos de CPDL$^+$ definidos a través de las clases de grafos con ancho de árbol acotado, con lo cual obtuvimos una jerarquización estrictamente creciente de CPDL$^+$.

Palabras clave: bases de datos, grafos, ontología, respuestas consistentes, satisfacibilidad, decibilidad, lógica modal, lógicas de descripción, lógica de primer orden, segundo orden monádico, lenguajes regulares

Logics to reason over graphs with data

Abstract

In this thesis we analize several problems that are relevant in the field of Database Theory and we focus particularly on those cases where the information is stored in a graph-like structure. Nowadays it turns out essential to do formal research about this kind of structures dued to the increasing popularity of data management systems such as the Wold Wide Web, or its extension the Semantic Web, whose operability is based on technologies that generate and analize the data stored in documents like XML files or RDF graphs. Generally, the management of a database is usually specified by an ontology language and several of such languages have been proposed for graph-like structures in the last decades, like Web Ontology Language (OWL), which is a family of description logics used to manage shapes of graphs through metadata information, or Shapes Constrints Language (SHACL) which validates information in a RDF graph through shapes. In a similar way, many more query languages have been created, most of which are based on the implementation of basic navigational properties related to graphs, like reachability, connectedness, path verification, etc.

Under this notion, multiple theoretical approaches emerge naturally, for instances, the inconsistency of a database with respect to a set of constraints is a basic problem that can be studied with logic rigor. On one hand, there are many ways to formalize this situation, according to the class of structures to consider and the ontology language used to reason about them. On the other hand, there are also many ways to deal with inconsistency in pactrice, whether through elimination of the inconsistency by changing the database, or by keeping the inconsistency but at the same time identifying the *certain* information, that is, the data that prevails no matter how the inconsistency in the database is fixed. Concretely, we study the following problems:

- Repairing of an inconsistent data-graph with respect to a set of constraints over the language GXPath and fragments of GXPath;
- Finite controlability of the decision problem Ontology Mediated Query Answering (OMQA), whose results are characterized by graph patterns.

For the first item we obtained several complexity results according to different criteria of repairing and studied some cases where the repairing method turned out to be deterministic. For the second item we defined a special underlying graph structure for regular queries that we called *skeleton*, which allow us to give a complete description for the finite controllability property of OMQA according to specific topological characteristics of the skeletons.

In another venue, we developed a new language called CPDL$^+$, that extends the well-known Propositional Dynamic Logic (PDL) with a recursive program operator called *conjuctive program*, which is compatible with PDL's syntactic operators. The semantic of this language is given by Kripke structures, which is a variation of transition systems, similar to a graph with multiple labels on edges and nodes. In conection with the previous discussion, CPDL$^+$ can be thought as a query language over graph databases. For this language we study several logic and computational problems that support its importance. We prove that CPDL$^+$ is far more expressive than many

families of languages; that CPDL$^+$ satisfiability problem is decidable; that it has the bounded-treewidth-model property (that is, if a formula φ of CPDL$^+$ is satisfiable, it is satifiable in a model with bounded treewidth); that CPDL$^+$ is characterized by a bisimulation game; and we analize the CPDL$^+$ model checking problem. Moreover, many of these results were obtained analizing distinct fragments of CPDL$^+$ given by classes of graphs with bounded treewidth, for which we interpreted CPDL$^+$ as a strictly increasing hierarchy.

Key words: database, graphs, ontology, consistent answers, satisfiability, decidability, modal logic, description logics, first order logic, monadic second order logic, regular languages

Agradecimientos

En primer lugar quiero agradecer a mis directores, Santiago Figueira y Diego Figueira, no solo por ser excelentes guías en este largo trayecto sino por la paciencia que me tuvieron cuando trastabillaba al seguirles el paso.

A Vanina Martínez, Sergio Abriola, Santiago Cifuentes y Nina Pardal, con quienes siempre es divertido trabajar.

Y a mis compañeros de oficina, Nico, Nacho, Tomi, Sergio (nuevamente), Gabo, Guido y Martín, por hacer de este espacio el mejor lugar de trabajo.

Dedicatoria

A mis padres María Baque y Homero Pin, a mi sobrina Clarissa Van der Dijs Pin y especialmente a la memoria de mi hermana,

Angie Pin Baque.

Que Dios los acompañe siempre.

Contenido

Introducción ii

Contribuciones ix

Nociones básicas xx

1 Reparación de data-grafos 1
- 1.1 Preliminares . . . 4
- 1.2 Computando reparaciones . . . 8
  - 1.2.1 Subreparaciones . . . 9
  - 1.2.2 Superreparaciones . . . 18

2 Controlabilidad finita de consultas regulares mediadas por ontologías 35
- 2.1 Preliminares . . . 37
- 2.2 Resultados Principales . . . 42
- 2.3 Reducciones a GNFO . . . 45
- 2.4 Los casos que no son finitamente controlables . . . 53

3 Extensiones de PDL mediante propiedades de grafos 57
- 3.1 Preliminares . . . 59
- 3.2 PDL y sus extensiones . . . 61
- 3.3 Relación de CPDL$^+$ con loop-CPDL e ICPDL . . . 66
- 3.4 Criterios de simulación para CPDL$^+$(TW$_k$) . . . 73
  - 3.4.1 Relación de simulación . . . 73
  - 3.4.2 Relación de bisimulación . . . 80
  - 3.4.3 Conexión con la lógica modal clásica . . . 84
- 3.5 Separabilidad . . . 87
- 3.6 Satisfacibilidad de CPDL$^+$ . . . 89
  - 3.6.1 Satisfacibilidad sobre árboles ω-regulares . . . 93
  - 3.6.2 De CPDL$^+$ a autómatas . . . 97
- 3.7 Model Checking de CPDL$^+$ . . . 106

4 Conclusiones 109

Introducción

La tendencia actual en el área de Bases de Datos tradicionales es aceptar que la inconsistencia es inherente a cualquier aplicación real, especialmente en lo que concierne a Big Data management, y que como tal debe ser formalizada y usada en los procesos de manejo de datos, idea explorada en muchos artículos relevantes y ampliamente citados como [19, 57, 96]. Más recientemente, este enfoque se ha trasladado al contexto de bases de datos semi-estructuradas, como se ve en varios trabajos que involucran distintos formatos de almacenamiento de información no tabulares tales como: documentos XML [90, 161], NoSQL databases con semántica BASE (Basically-Available, Soft-state, Eventual consistency) [27], y el caso que nos compete en este trabajo, bases de datos orientadas a grafos [32, 44, 49]. En muchos de estos trabajos podemos observar diferentes enfoques para lidiar con problemas de inconsistencia, ya sea eliminando los datos conflictivos, agregando datos para complementar información requerida, o manteniendo la inconsistencia pero identificando la información que estará siempre presente en un posible o eventual arreglo de la base de datos. Una idea común en todas estas propuestas es el uso de herramientas provenientes de la lógica y de la teoría de modelos, puesto que muchos de los lenguajes comúnmente utilizados en la práctica para el acceso y el manejo de la información, como SQL, ProLog, Datalog, entre otros, se fundamentan en fragmentos de la lógica de primer orden, y por tanto, un análisis riguroso en base a aspectos teóricos resulta indispensable así como interesante.

En el marco general que engloba a este trabajo analizamos un par de problemas relacionados a (1) el manejo de la inconsistencia presentada en bases de datos orientadas a grafos con respecto a cierto conjunto de restricciones, los cuales se forman según los requisitos de una ontología preestablecida, y (2) el estudio y desarrollo de nuevos formalismos lógicos que permitan explorar aspectos estructurales y sintácticos distintos a los ya conocidos y aplicados sobre modelos con forma de grafos.

Desde un punto de vista formal resulta necesario, y a veces ineludible, explorar distintas alternativas descriptivas y notacionales para representar a las bases de datos concisamente como objetos matemáticos y como se ve en la Figura 1, esto podría realizarse de múltiples maneras. A pesar del uso masivo de las bases de datos orientadas a grafos en las últimas décadas, no se ha logrado establecer un concenso de representabilidad en el ámbito operativo, teniendo en la actualidad dos grandes alternativas de modelización: los RDF graphs vinculados a los estándares de la Web Semántica, y los Property graphs vinculados a estándares ISO (International Organization for Standardization). Por esta razón, no hay un lenguaje estándar único para razonar sobre bases de datos orientadas a grafos (tal como lo es SQL para el modelo relacional de bases de datos), por lo que es usual encontrar, entre los argumentos que abogan en favor del uso de las bases de

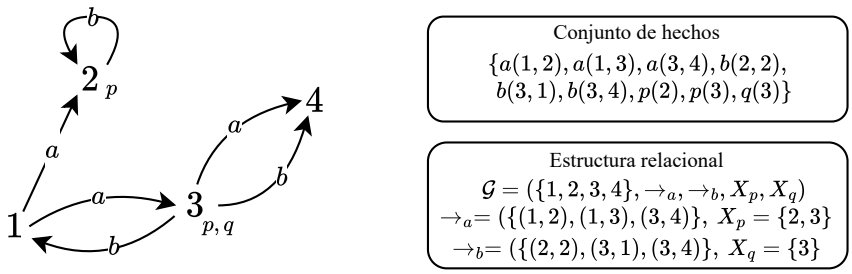

Figura 1: Por lo general, trabajamos con estructuras con forma de grafos con etiquetas en nodos y ejes, como el gráfico de la izquierda. A la derecha, dos maneras conjuntivas de representar a esta estructura.

datos orientadas a grafos, un compilado de distintas opciones que basan su desarrollo en técnicas o herramientas muy variadas entre sí. Es el caso de lenguajes declarativos de consulta como Cypher y GQL (Graph Query Language), que fueron diseñados con la misma perspectiva operacional de SQL, y que son usados en sistemas como Neo4j, que está fundamentalmente desarrollado en Java [170]. Si bien en términos generales una base de datos orientada a grafos es básicamente un grafo con (múltiples) etiquetas en ejes y nodos, esta noción también puede diferir conceptualmente, dada la amplia variedad de comunidades científicas que han abordado este tipo de estructuras. A lo largo de este trabajo usaremos distintos paradigmas para representar a este tipo de estructuras, como detallamos a continuación con una breve explicación:

- **Data-grafos**: en el Capítulo 1 utilizamos el mismo esquema teórico de [127, 169] y definimos un *data-grafo* como una terna $G = (V, E, D)$ donde (V, E) es un digrafo con conjunto de nodos V, conjunto de ejes etiquetados E, y una (*única*) función D que asigna a cada nodo en V un valor de dato. En los trabajos citados se argumenta por qué este tipo de asignación de datos sobre los nodos no implica una limitación con respecto a la alternativa de asignar múltiples valores, como suele hacerse en la práctica. Más a favor de este argumento, en el Teorema 27 del Capítulo 1 mostramos mediante una reducción cómo relacionar una base de conocimientos a un data-grafo (con restricciones) de manera eficiente y preservando la semántica de la ontología subyacente, con el propósito de hallar la complejidad de un problema de decisión intrínsecamente relacionado a propiedades de satisfacibilidad. A pesar de que el mencionado teorema muestra un resultado negativo de indecibilidad, queda claro que en un sentido de representabilidad el concepto de data-grafo engloba muchas nociones estructurales vistas en otro tipo de modelos (particularmente en lo que se refiere a estructuras finitas), varios de los cuales usaremos en los siguientes capítulos. Otra razón para haber utilizado este esquema tiene que ver con el tipo de lenguajes altamente expresivos que fueron considerados para razonar sobre data-grafos, de lo que hablaremos más adelante.
- **Grafos de conocimiento**: una exposición similar sobre lo alejado que estamos de plantear una definición definitiva y conciliatoria de lo que es un data-grafo en general y de cómo estos deberían ser representados, se encuentra en [117]. En el caso específico de las áreas de Representación del Conocimiento e Inteligencia Artificial, una larga discusión se ha generado a raíz del término *knowledge graph* usado por Google para referirse a determinadas herramientas de mejoras semánticas aplicadas en sus motores de búsqueda [159]. Como se expone en [38],

"aunque no haya un acuerdo preciso, es de entendimiento común que un grafo de conocimiento es una larga red de entidades, e instancias". Sin embargo, Bellomarini et al. indican más adelante que esto aún no es suficiente, más aún, considerando que a nivel tecnológico, un sistema de gestión y manejo de bases de conocimiento (KGMS) apropiado debe ser capaz de realizar tareas como (i) incorporar grandes cantidades de datos; (ii) incorporar conocimiento basado en reglas; (iii) realizar tareas complejas de razonamiento dadas instancias de los puntos anteriores. Bajo esta perspectiva, se establece que como mínimo un grafo de conocimiento debe describirse mediante (a) *hechos* que forman parte de una verdad observable y (b) *reglas* que determinan verdades derivables de nuestro conocimiento observado, lo cual podría ser en un sentido sintáctico, bajo sistemas axiomáticos y derivaciones, o a nivel semántico, como nos compete en este trabajo. En términos más técnicos, un grafo de conocimiento se puede entender como un par $\mathcal{K} = (\mathcal{A}, \mathcal{T})$, donde $\mathcal{T}$ es un conjunto denominado TBox que consta de reglas pertenecientes a un lenguaje de ontologías (de lo que hablaremos más adelante), y $\mathcal{A}$ es un conjunto (posiblemente infinito) denominado ABox de cierto tipo de verdades atómicas o *hechos*, es decir, expresiones de la forma $P(\bar{t})$, donde P es un *concepto* (i.e. un predicado unario) o un *rol* (i.e. un predicado binario) y $\bar{t}$ es una tupla de términos constantes o *nulls*. Este enfoque relativo al esquema relacional es ampliamente utilizado en Representación del Conocimiento, muy particularmente dentro de la comunidad que trabaja con *lógicas de descripción* (DLs) [56, 131]. Bajo este tipo de estructuras, presentamos en el Capítulo 2 algunos resultados que se basan en determinar propiedades de derivabilidad (semantic entailment) de consultas navegacionales de grafos, a partir de una base de conocimiento con ontologías restringidas según determinados fragmentos de la lógica de primer orden. Dado que el tipo de ontologías consideradas en esa publicación lindan y se relacionan íntimamente con varias DLs conocidas, optamos por usar implícitamente el concepto de grafo de conocimiento a lo largo de ese capítulo.

- **Estructuras de Kripke**: originalmente concebido como un modelo de transición para analizar semánticamente varios problemas de sistemas axiomáticos de la lógica modal [123], las estructuras de Kripke se convirtieron en el modelo estándar sobre los cuales razonan varias familias de lenguajes denominadas *lógicas dinámicas*, entre las que se encuentran la lógica modal, Propositional Dynamic Logic (PDL) [88], e incluso toda una serie de lenguajes nuevos definidos en [85] y que presentaremos en el Capítulo 3. Las lógicas dinámicas se proponen como herramientas descriptivas de propiedades matemáticas y proposicionales de programas. Tradicionalmente, las estructuras de Kripke se entienden como una variante de los sistemas de transición propuestos por Kripke, y se definen como un grafo cuyos nodos representan estados o mundos alcanzables de un sistema y cuyos ejes representan transiciones entre estados, junto con un etiquetamiento de nodos que representan propiedades satisfechas en el estado. Siguiendo con nuestra discusión previa, una estructura de Kripke es básicamente una base de datos orientada a grafos. Si bien esta asociación de estructuras de Kripke vistas como bases de datos no es usualmente considerada entre los investigadores de las lógicas dinámicas (en principio porque las lógicas dinámicas se estudian con un propósito distinto al de los lenguajes ontológicos y de consulta), se vuelve relevante para la cohesión de este trabajo, en particular porque uno de los objetivos conseguidos en [85] fue definir un lenguaje lo suficientemente expresivo para

capturar a varias lógicas dinámicas ya conocidas, así como a varias extensiones de RPQ que es la base de todo sistema de consulta con propiedades navegacionales o de caminos, esencial para el análisis de bases de datos orientadas a grafos.

Como se estableció en cada uno de los puntos previos, la razón para trabajar con distintos esquemas teóricos se debe en parte a la gran cantidad de lenguajes existentes en la literatura para razonar sobre estructuras orientadas a grafos, incluidas las lógicas dinámicas. A lo largo de esta investigación decidimos estudiar varios de estos enfoques, lo que conllevó a considerar la posibilidad de unificar varios aspectos terminológicos y de notación, lo cual se ha realizado hasta cierta medida. Sin embargo, como ya hemos argumentado, muchos de los lenguajes explorados requieren de ciertas particularidades semánticas e interpretativas que hemos decidido mantener en esta presentación. Describiremos a continuación cuáles lenguajes fueron estudiados en este trabajo y su influencia o uso que tendrán en los siguientes capítulos.

- **Regular Path Query** (RPQ): la particularidad de trabajar con estructuras con forma de grafos supone que propiedades básicas de grafos vinculadas a aspectos topológicos (dígase, accesibilidad, conectividad, camino más corto, etc.), deberían ser idealmente implementables en algún lenguaje lógico que razone sobre este tipo de estructuras, de ahí el origen de los denominados *lenguajes navegacionales*. Entre los lenguajes de este estilo, y de los que representan un núcleo dentro del área, está RPQ [1, 35, 64], que se compone principalmente de expresiones de la forma $x \xrightarrow{L} y$, donde x, y son variables y L es una expresión regular definida sobre un alfabeto preestablecido, y que se interpreta sobre una estructura como el conjunto de pares de nodos (u, v) conectados por un camino etiquetado con una palabra de L. RPQ y varias de sus conocidas extensiones como 2RPQ, C2RPQ y UC2RPQ (definidos en la sección de Nociones Básicas), estarán presentes a lo largo de este trabajo cumpliendo distintos roles. Particularmente serán bastante relevantes en el Capítulo 2 para el planteamiento del problema central (finite controllability of Ontology Mediated Query Answering) donde se utiliza como lenguaje de consulta, y en el Capítulo 3 definimos una lógica altamente expresiva que denominamos CPDL$^+$, que también contiene a UC2RPQ. Notemos que los lenguajes basados en RPQ resultan insuficientes para plantear consultas que involucren información de datos de nodos, así que el contexto en el que este lenguaje aparecerá en los problemas expuestos por lo general involucrará alguna extensión o relación con otros lenguajes que enlazan propiedades de accesibilidad y de datos o metadatos, en simultáneo. RPQ induce un lenguaje ontológico llamado *Regular Path Constraint* (RPC) que se compone de expresiones de tipo $L_1 \subseteq L_2$, donde L_1 y L_2 son un par de RPQs. Este lenguaje será relevante en algunos resultados como el Teorema 26, bajo la semántica usada en [32]. Un trabajo previo donde se utilizó esta ontología es [1], pero los RPC se evalúan en grafos con un elemento distinguido, por lo que los RPC en este caso se usan para imponer condiciones locales, a diferencia de la semántica que adoptamos en este trabajo.

- **Modal logic y guarded fragments**: la consideración de técnicas y resultados propios de la lógica modal en el marco de la lógica de predicados generó varios aportes teóricos importantes a finales de los años 90, relevantes hasta la actualidad tanto en las áreas de lógica matemática como en teoría de bases de datos. Con esto en mente, entre los primeros trabajos que trascendieron ampliamente se encuentra el de [15], donde se presenta al denominado *guarded-fragment* de la lógica de primer orden (GFO). Este lenguaje se define como la colección

de fórmulas de primer orden con ciertos patrones de cuantificación relativizada que emulan el comportamiento de los operadores modales. Así por ejemplo, si $\bar{x}, \bar{y}$ son tuplas de variables de primer orden, $\psi(\bar{x}, \bar{y})$ una fórmula en GFO con variables libres contenidas en $\bar{x}, \bar{y}$ y $\alpha(\bar{x}, \bar{y})$ una expresión atómica, entonces también están en GFO las expresiones

$$(\forall \bar{x}.\alpha)\psi := \forall \bar{x}(\alpha \to \psi) \quad \text{y} \quad (\exists \bar{x}.\alpha)\psi := \exists \bar{x}(\alpha \wedge \psi).$$

En cualquiera de estas expresiones a α se le denomina *guarda*. Estas formas de cuantificación restringida engloban la interpretación usual de los operadores modales $\Box$ y $\Diamond$ bajo la semántica de estructuras de Kripke por medio de la conocida traducción estándar [18, 46], que permite traducir las fórmulas de la lógica modal a fórmulas equivalentes de la lógica de primer orden. La importancia de esta correspondencia se ve reforzada por propiedades descriptivas de modelos como el Teorema de Caracterización de van Benthem [165], que establece que la lógica modal es el fragmento *invariante por bisimulación* de la lógica de primer orden. Como se demuestra en [15] y en varios trabajos posteriores igual de relevantes como [53, 107, 110], esta amalgama resultó sumamente provechosa en la búsqueda de fragmentos decidibles de la lógica de primer orden que gozaran de buenas propiedades computacionales y matemáticas, ya provistas en la lógica modal básica, tal como sucede con GFO. Por ejemplo, es bien sabido que el problema de satisfacibilidad de GFO es decidible y 2ExpTime-completo, que posee la *propiedad de modelo finito* (i.e. si una fórmula de GFO es satisfacible, es satisfacible en un modelo finito) y la *propiedad de modelo tipo-árbol* (i.e. si una fórmula de GFO es satisfacible, es satisfacible en un modelo de *treewidth* acotado). Varios de estos resultados se obtuvieron adaptando la caracterización por *bisimulación* de la lógica modal, algo que discutiremos con mayor profundidad en el Capítulo 3, donde abordamos una serie de problemas similares en la Sección 3.4.2. En lo concerniente a bases de datos orientadas a grafos y razonamiento ontológico, la incorporación de aspectos modales para gestionar este tipo de estructuras resulta natural al tener en cuenta por un lado que varias lógicas de descripción pueden sumergirse como fragmentos de GFO [67, 106], y por otro lado, que muchos lenguajes navegacionales son adaptables con operadores modales, como veremos más adelante.

- **Guarded negation first order**: con un enfoque similar al de los trabajos citados en el punto anterior, en [30] definen el fragmento *guarded negation* de la lógica de primer orden (GNFO), que generaliza a GFO preservando varias de sus buenas propiedades: GNFO es decidible y 2ExpTime-completo, tiene la propiedad de modelo tipo-árbol y la de modelo finito, decidir si una sentencia de GNFO tiene un modelo finito es 2ExpTime-completo, y el poder expresivo de GNFO está caracterizado por un juego de bisimulación. Con GNFO también es posible absorber otros fragmentos de la lógica de primer orden que son incomparables con GFO, por ejemplo, el fragmento *unary negation* (UNFO) [30, 164], por lo que GNFO es más expresivo que ambos lenguajes. Análogo a la definición de GFO, en GNFO solo se puede introducir el símbolo de negación en una expresión si las variables libres se encuentran al alcance de una *guarda*. Esto lo presentaremos formalmente en la Sección 2.1 del Capítulo 2, donde además veremos la relación de GNFO con otros fragmentos importantes de la lógica de primer orden que abarcan distintos tipos de restricciones usados en teoría de bases de datos y bases de conocimiento asociadas a lógicas de descripción.

- **Description Logics** (DL): la incorporación del Web Ontology Language en los estándares del World Wide Web Consortium (W3C) significó en los primeros años de la década de los 2000 una gran oportunidad para explorar el alcance pragmático de varios lenguajes de representación, entre ellos las lógicas de descripción. Uno de los focos centrales del W3C, abordado en los protocolos de la Web Semántica[1], es la búsqueda de un sistema integrador capaz de realizar una lectura exitosa y desambiguada de la información proveniente de distintas fuentes, particularmente a través de una correcta representación de los metadatos. En este sentido, se requiere el uso de un lenguaje apropiado para describir el contenido de una base de datos mediante conceptos, categorizaciones y relaciones entre entidades (tal cual como en un grafo de conocimiento), con el que se pueda razonar e inferir de manera automatizada la información que pueda resultar significativa para algún usuario. Varias lógicas de descripción bastante estudiadas durante la década de los 90 en el ámbito de Representación del Conocimiento cumplían con estos requerimientos, además de contar con una serie de buenas propiedades computacionales que las posicionaron entre las opciones más destacadas para su implementación en la Web Ontology Language. De las lógicas de descripción más primigenias y representativas se encuentra $\mathcal{ALC}$ [22, 155], que se distingue por proporcionar una amplia variedad de constructores de concepto y de fórmulas a través de una sintaxis simple y que resulta ser equivalente semánticamente a la lógica (multi)modal básica [154]. Debido a esto último, podríamos pensar en $\mathcal{ALC}$ como un fragmento de la lógica de primer orden que se puede sumergir en las lógicas GFO y GNFO ya discutidas, por lo que varios de los resultados obtenidos en el Capítulo 2 valdrán particularmente en algunas clases de fórmulas correspondientes a ciertas lógicas de descripción. Por otra parte, en la Sección 1.2.2 del Capítulo 1 trabajaremos específicamente con $\mathcal{ALCOIF}$, una lógica de descripción altamente expresiva que extiende a $\mathcal{ALC}$ con conceptos nominales ($\mathcal{O}$) y de inversión ($\mathcal{I}$) y que permite establecer restricciones de funcionalidad sobre los roles ($\mathcal{F}$). El uso de $\mathcal{ALCOIF}$ (Definición 10) resulta bastante significativo para establecer una relación no trivial entre una DL bastante expresiva como $\mathcal{ALCOIF}$ y un lenguaje de carácter navegacional del que hablaremos a continuación.
- **Reg-GXPath**: En [127] se investigan varios lenguajes de consulta que fueron diseñados para combinar topología de grafos y datos. Algunas extensiones conocidas de XPath cumplen con estas características, pero generalmente los lenguajes basados en XPath se usan muy particularmente para procesar la información contenida en documentos XML [62], los cuales suelen abstraerse como estructuras con forma de árbol. En la mayoría de los casos, al trabajar con fragmentos o extensiones de XPath se mantiene el mismo criterio semántico, por ejemplo, para fragmentos de XPath sobre árboles no ordenados [135] y árboles ordenados [91, 104]; y para versiones *data-aware* de XPath[2] sobre árboles no ordenados [82] y árboles ordenados [5, 48]. En muy contados casos se ha tratado a XPath como lenguaje de consulta para estructuras de grafos más generales [58, 134]. Precisamente en [127] se definió una familia de lenguajes tipo XPath, denominada GXPath, para razonar sobre grafos con datos. Varios de estos lenguajes se construyen considerando ciertos operadores implementados en extensiones de XPath como

[1] https://www.w3.org/TR/owl-features/

[2] Estas versiones involucran el uso de operadores modales que testean comparación de valores datos, los cuales también serán considerados en Reg-GXPath.

en otros lenguajes de carácter modal como PDL. Por ejemplo, uno de tales lenguajes es $\#\mathsf{GXPath}_{\mathrm{reg}}(\mathsf{c},\mathsf{eq})$, que aquí hemos renombrado Reg-GXPath (Definición 12) por comodidad lectora, el cual se define por recursión mutua entre expresiones de nodo y camino, y que cuenta con varios tests que analizan el valor de dato almacenado en los nodos de un data-grafo. Otra característica que distingue a Reg-GXPath es el uso del operador de *complementación* para expresiones de camino, con lo cual podemos expresar otras propiedades navegacionales de interés como *intersección* o incluso *contención* de caminos, siendo esto último estudiado en muchos casos vía RPCs. Motivados por esta relación, en el Capítulo 1 proponemos una noción de consistencia (Definición 15), que puede entenderse como un criterio de satisfacibilidad global de un conjunto de expresiones de Reg-GXPath sobre una estructura de data-grafo. Bajo esta semántica, es posible relacionar a Reg-GXPath con otros lenguajes, particularmente en el ámbito de modelos finitos. Por ejemplo, como ya mencionamos en el punto anterior, en la demostración del Teorema 26 traducimos $\mathcal{ALCOIF}$ a Reg-GXPath. En [127, Section 4.2] se plantea otra posible relación de la familia GXPath con el conocido PDL, que es también relevante en este trabajo.

- **Propositional Dynamic Logic** (PDL): Originalmente propuesto como un lenguaje para razonar sobre el comportamiento de programas [88], PDL ha sido adaptado y considerado en otros ámbitos, por ejemplo, como lenguaje de consulta para bases de datos orientadas a grafos [8, 162]. En un principio, el propósito de estudiar un lenguaje como PDL era encontrar un conjunto de axiomas y reglas de inferencia para probar propiedades de programas como *terminación* o *equivalencia*. En [156] se propone un esquema axiomático y reglas de inferencia para PDL el cual se demuestra completo. Este análisis fue pulido posteriormente en [122] donde proporcionan una demostración corta y elemental de la completitud del sistema de Segerberg. Trabajos similares han surgido a lo largo de los años para extensiones de PDL [72, 147]. Por otra parte, PDL también ha recibido mucha atención por el lado de la Teoría de Modelos. En su trabajo original Fischer y Ladner dieron varios resultados de esta índole, como los siguientes: PDL tiene la *propiedad de modelo pequeño*, es decir, si una fórmula de PDL es satisfacible entonces es satisfacible en un modelo de tamaño a lo sumo exponencial en el tamaño de la fórmula; la *propiedad de modelo de árbol*, es decir, si una fórmula de PDL es satisfacible entonces es satisfacible en un modelo con forma de árbol; el problema de satifacibilidad de PDL es ExpTime-completo; etc. Existen muchas extensiones de PDL cuyos problemas de satisfacibilidad asociados comprenden una complejidad que va desde ser ExpTime hasta ser indecidibles, siendo [114] uno de los primeros trabajos que explora los límites de la decibilidad de lenguajes basados en PDL. Algunos trabajos similares y de bastante relevancia para esta tesis (específicamente para el Capítulo 3) son [66, 103] que abordan el problema de satisfacibilidad de PDL extendido con operadores *loop, intersection* y *converse*, y demostraremos varios resultados que subsumen ambos papers bajo la propuesta de una nueva extensión de PDL.

En la próxima sección hablaremos más específicamente de los resultados obtenidos en este trabajo y de cómo fueron utilizados los esquemas estructurales y sintácticos presentados en esta introducción para el planteamiento de los principales problemas a tratar.

Contribuciones

A lo largo de esta investigación nos dedicamos a estudiar distintos problemas que son centrales en Teoría de Bases de Datos, Teoría del Conocimiento y Lógica Dinámica, que se pueden abordar con técnicas computacionales y matemáticas relacionadas a Lógica Clásica y Teoría de Modelos. Concretamente, nuestros resultados se basan en el estudio de los siguientes problemas:

- Reparación de un data-grafo bajo distintos criterios semánticos y ontológicos.
- Análisis de respuestas a consultas mediadas por ontologías (denominado CQA u OMQA según distintos contextos) y la propiedad de controlabilidad finita.
- Extensiones expresivas de PDL que preservan buenas propiedades lógicas y computacionales y sus relaciones con otros lenguajes lógicos.

Detallaremos a continuación cómo se abordaron estos planteamientos con una breve descripción de los mismos, y hablaremos de los casos o versiones que serán analizados en cada capítulo y cuáles resultados originales se obtuvieron, los cuales pueden ser de interés para las comunidades en las áreas de Bases de Datos, Teoría del Conocimiento, Lógica Matemática, Teoría de Modelos, Lógicas Dinámicas, Lógicas Modales, y semejantes. En esta sección mencionaremos las publicaciones [7, 85, 86] cuyos contenidos forman parte de esta tesis doctoral.

Capítulo 1:
Reparación de data-grafos

Cuando lidiamos con bases de datos inconsistentes, una serie de cuestionamientos surgen naturalmente: ¿es identificable el origen de la inconsistencia?, ¿es posible modificar la base para restablecer la consistencia?, ¿es costoso realizar estos cambios? Desde la aparición del trascendental artículo [19], que aborda estas preguntas en el contexto de bases de datos relacionales con restricciones expresables en *standard format*,[3] fueron adaptadas las ideas de ese trabajo a otros esquemas, incluyendo a las bases de datos orientadas a grafos sujetas a distintos tipos de restricciones. En este trabajo, Arenas, Bertossi y Chomicki formalizaron la idea de *reparación* de una base de datos inconsistente como una base de datos consistente que difiere mínimamente de la base original, donde esto último se establece mediante una noción de distancia entre estructuras dada por propiedades conjuntistas. Muy ligado al concepto de reparación se encuentra el de *Consistent Query Answering* (CQA), que se basa en identificar las respuestas a consultas que siempre

[3] Este formato de fórmulas incluye familias básicas de restricciones como *denial constraints, functional dependencies* e *inclusion dependencies*.

aparecerán en posibles reparaciones de una base inconsistente, lo cual es relevante en muchos contextos, por ejemplo, cuando existe más de una manera de restaurar la consistencia. Esta noción semántica ha sido clasificada posteriormente como *certain semantics* o *AR semantics* [43], para contrastarla con otras opciones de tolerancia de inconsistencia. Para nuestro propósito usaremos implícitamente la semántica de *brave repairs* de CQA, que bajo cierta perspectiva puede entenderse como la propiedad contraria de *certain semantics*.

En el Capítulo 1 exponemos los resultados obtenidos en [7], donde fueron analizadas varias versiones del problema de reparación de un data-grafo inconsistente con respecto a restricciones provenientes del lenguaje Reg-GXPath y varios fragmentos de este. La noción de consistencia utilizada en este trabajo (Definición 15) se presenta bajo un criterio de satisfacción global que permite subsumir mediante traducciones computables varios tipos de lenguajes, ya sea de restricción como los RPCs (visto en la demostración del Teorema 26) u ontológicos como $\mathcal{ALCOIF}$ (visto en la demostración del Teorema 27). Varios resultados de evaluación y *model checking* para expresiones en Reg-GXPath fueron demostrados en [127] y son aplicables a nuestra semántica (Teorema 20), lo cual aprovechamos para acotar polinomialmente determinados procesos a lo largo del capítulo. Los criterios de reparación considerados se basan en propiedades de subconjuntos (cuando es válido eliminar información, introducido en la Sección 1.2.1 como *subreparación*) y superconjuntos (cuando es válido agregar información, introducido en la Sección 1.2.2 como *superreparación*), que suelen considerarse como planteamientos razonables en la práctica, particularmente bajo suposiciones de incorrectitud e incompletitud de la información [61].

Nuestros planteamientos se hacen en general con el formato *input*/*output*, donde en este caso el *input* es un par $(G, \mathcal{R})$ con G un data-grafo (Definición 11) y $\mathcal{R}$ un conjunto finito de restricciones, y el *output* es una reparación de G con respecto a $\mathcal{R}$ (para la versión funcional del problema de reparación) o una respuesta afirmativa o negativa (para la versión de decisión) según si existe o no una reparación de G con respecto a $\mathcal{R}$. Tanto en el contexto de subreparaciones como en el de superrepaciones, abordamos en un principio el problema de reparación de data-grafos considerando la totalidad de los recursos sintácticos de Reg-GXPath, lo cual derivó en varios resultados de intratabilidad e indecibilidad. En específico, obtenemos que dado un *input* $(G, \mathcal{R})$,

- es NP-completo decidir si existe una subreparación no vacía de G con respecto a $\mathcal{R}$ (Teoremas 21 y 22);
- es indecidible decidir si existe una superreparación de G con respecto a $\mathcal{R}$ (Teorema 26 y 27).

Debido al aspecto negativo de estos resultados, buscamos casos tratables limitando el poder expresivo de las restricciones. De entre todos los fragmentos de Reg-GXPath definidos en el Capítulo 1 el más destacado para este propósito es el denominado Reg-GXPath^{pos}, que se obtiene a partir de Reg-GXPath prohibiendo el uso de la negación $\neg$ en las expresiones de nodo, y del operador de complementación en las expresiones de camino. Para *inputs* $(G, \mathcal{R})$ con $\mathcal{R}$ un conjunto finito de restricciones en Reg-GXPath^{pos}, obtuvimos lo siguiente:

- en caso de existir, se puede computar en tiempo polinomial una subreparación no vacía de G con respecto a $\mathcal{R}$ cuando este solo consta de expresiones de nodo (Teoremas 25);
- es polinomial decidir si existe una superreparación de G con respecto a $\mathcal{R}$ cuando $\mathcal{R}$ solo consta de expresiones de nodo (Teorema 33);

- es polinomial decidir si existe una superreparación de G con respecto a $\mathcal{R}$ cuando solo se utiliza una cantidad finita y fija de valores de datos (Teorema 35).

Un resumen detallado de todos estos resultados se encuentra en la Tabla 1.1, donde además se especifican otros aspectos técnicos. Para los resultados tratables también se demostraron algunas propiedades semánticas que se plantearon específicamente para obtener la cota de complejidad polinomial deseada, pero cuya aplicabilidad podrían independizarse de los fines de esta tesis. Entre −24,29−31 y el Teorema 32, que además permiten concluir la correctitud de los Algoritmos 1 y 2, los cuales están asociados al problema de computar una subreparación y decidir si existe una superreparación, respectivamente.

Si bien el artículo [7] está expresamente dedicado al problema de computar reparaciones de un data-grafo inconsistente, en algunos casos relacionaremos nuestros planteamientos a casos concretos de CQA estudiados en [32, 151], específicamente para las demostraciones de los Teoremas 26 y 27. Sin embargo, el siguiente capítulo del que hablaremos a continuación estará enteramente dedicado a analizar este problema en el contexto de bases de conocimiento.

Capítulo 2:

Controlabilidad finita de consultas regulares mediadas por ontologías

El mencionado artículo [19] inspiró un planteamiento análogo de CQA bajo el esquema de bases de conocimiento que suele denominarse *Ontology-Mediated Query Answering* (OMQA) [42, 45, 126] y que surge de la necesidad de adoptar estrategias de consulta respecto a *semánticas que toleran inconsistencias*, ante la imposibilidad de identificar y corregir errores de los datos almacenados en una base de conocimientos. Más específicamente, OMQA se basa en el cómputo de las respuestas comunes de una consulta bajo una semántica restringida por factores ontológicos y estructurales. Por ejemplo, en [19, 126] las consultas se evalúan en subreparaciones (es decir, bases consistentes que difieren mínimamente de la base original, como en el Capítulo 1, Sección 1.2.1), mientras que en [42, 45] se evalúan en subestructuras consistentes (es decir, la semántica queda determinada explícitamente por la ontología bajo una noción de incorrectitud de los datos). Otros tipos de semántica se discutirán en el Capítulo 1 Sección 1.2.2, tomados de los artículos [32, 151] mencionados previamente y que vinculamos a la noción de superreparación.

En [86], cuyo contenido se desarrollará a lo largo del Capítulo 2, estudiamos versiones de OMQA con las siguientes características:

a) las estructuras sobre las que se razonan son conjuntos de hechos, o ABoxes en la terminología de bases de conocimiento;

b) la ontología $\mathcal{O}$, está determinada por fragmentos de GNFO (Sección 2.1A), y a un subconjunto finito Γ de $\mathcal{O}$ se le denomina TBox en la terminología de bases de conocimiento;

c) el lenguaje de consulta $\mathcal{Q}$ está dado por fragmentos de UC2RPQ definidos a partir de ciertas clases de grafos subyacentes especiales que hemos llamado *esqueletos* (Sección 2.2);

d) la semántica depende del conjunto de *interpretaciones* de un par (G, Γ), donde G es un ABox finito y Γ un TBox (notemos que la semántica no es de reparaciones ni subestructuras como

en los artículos citados anteriormente, dado que en [86] trabajamos con otra noción semántica que será explicada en breve.).

Para hacer notoria la dependencia respecto a $\mathcal{O}$ y $\mathcal{Q}$ como los lenguajes particulares a tratar, denotaremos por $\text{OMQA}_{\mathcal{Q},\mathcal{O}}$ (leído como OMQA de $\mathcal{Q}$ bajo $\mathcal{O}$) al problema de decisión que toma como *input* una tupla $(G, \Gamma, q, \bar{a})$, donde G es una base de conocimiento (o modelo), Γ es un conjunto finito de fórmulas en $\mathcal{O}$, q es una consulta en $\mathcal{Q}$ de aridad k y $\bar{a}$ es una k-tupla de elementos de G. El *output* es *sí*, cuando $\bar{a}$ pertenece a la evaluación de q en *toda interpretación* de (G, Γ), donde esto último alude de manera muy abstracta a la semántica referida en *d)*. Los detalles formales de esta propiedad se presentarán en la Sección 2.1 en términos de *entailment*. Cabe destacar que en [86] distinguimos dos nociones distintas de OMQA de acuerdo a la definición concreta de *interpretación* a considerar: cuando aceptamos que estas pueden ser de cardinal *infinito* trataremos con el problema OMQA a secas, pero si las interpretaciones solo pueden ser *finitas* trataremos con *finite* OMQA (o FOMQA para recortar). Si estas dos nociones coinciden bajo un esquema particular de los puntos *a)*−*c)*, decimos que OMQA de $\mathcal{Q}$ bajo $\mathcal{O}$ es *finitamente controlable*. El estudio de esta propiedad es justamente el eje central del Capítulo 2, donde analizaremos varias versiones de OMQA que cumplen esta propiedad y otras que no. Algunos trabajos previos que abordan planteamientos similares al nuestro son [149, 150], donde demuestran la controlabilidad finita de OMQA para bases de datos relacionales con *semántica de mundo abierto*[4], y consultas UCQ bajo distintos tipos de restricciones, como *key dependencies* e *inclusion dependencies*, que son básicas y fundamentales en Teoría de Bases de datos; [13], donde usan un esquema similar a los trabajos de Rosati pero bajo otro tipo de restricciones que generalizan a las *inclusion dependencies*; y [101], respecto a bases de conocimiento y lógicas de descripción que involucran roles transitivos. Ninguno de estos trabajos aborda el caso de consultas regulares o navegacionales. Por otra parte, algunos trabajos que sí consideran lenguajes de consulta navegacionales y demuestran la decibilidad del problema FOMQA sin utilizar propiedades de controlabilidad finita son [113], respecto a consultas UCRPQ y restricciones en $\mathcal{ALC}$ y [112], respecto a consultas P2RPQ (*positive existential two-way regular path queries*) y restricciones en la lógica de descripción $\mathcal{SQ}_u$.

La primera versión de OMQA que analizamos fue OMQA de UC2RPQ bajo GNFO, que resulta ser decidible (Proposición 36), pero no queda claro si la versión FOMQA también lo es. El resultado principal de este capítulo (Teorema 38) expone una caracterización completa de las versiones finitamente controlables de OMQA basadas en fragmentos de UC2RPQ y GNFO y como consecuencia directa de este teorema obtenemos que OMQA de UC2RPQ bajo GNFO no es finitamente controlable, pero la decidibilidad de FOMQA sigue siendo un problema abierto. El Teorema 38 se puede considerar como una respuesta parcial a esta pregunta, pues proporciona condiciones suficientes sobre $\mathcal{Q}$ y $\mathcal{O}$ que aseguran la decidibilidad de $\text{FOMQA}_{\mathcal{Q},\mathcal{O}}$.

El uso de GNFO como el lenguaje ontológico base se debe tanto a razones técnicas como integrales. Técnicas, porque haremos uso de muchas de las propiedades de GNFO mencionadas en la Introducción, además de un procedimiento conocido como *chase* (introducido en la Sección 2.1 con referencias a trabajos previos como [70,140,150]) que resulta fundamental para nuestros resultados; e integrales, pues como se expuso en la Introducción, GNFO abarca muchos tipos de lenguajes de

[4]El postulado de *semántica de mundo abierto* estipula que los hechos que no aparecen en una base no son necesariamente falsos. Esto es similar a nuestra exposición informal de interpretación del punto *d)*.

restricciones usados en Teoría de Bases de Datos, así como varias lógicas de descripción usadas en Representación del Conocimiento, por lo que los resultados de este capítulo pueden ser relevantes en ambas áreas. Los fragmentos precisos de GNFO que utilizaremos en este capítulo (ver Figura 2) son familias de fórmulas de primer orden que se pueden describir como *reglas existenciales* (también llamadas reglas Datalog$^{\pm}$ o *tgds*, Sección 2.1B).

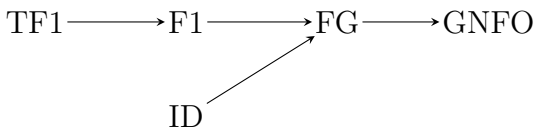

Figura 2: Mapa de los lenguajes ontológicos considerados en el Capítulo 2, donde cada flecha indica el orden de expresividad entre estos, siendo el lenguaje a la izquierda de la flecha el menos expresivo. Las siglas se refieren a los siguientes fragmentos de la lógica de primer orden: FG es por *frontier-guarded tgds*, F1 es por *frontier-1 tgds*, TF1 es por *term-frontier tgds*, e ID es por *inclusion dependencies*.

Es bien conocida la contención de todos estos fragmentos en GNFO, pero damos un pequeño argumento de su validez en la Sección 2.1C donde mostramos que cualquier fórmula de FG es traducible a una expresión equivalente en GNFO. En [23,25,26] se hallan relaciones más detalladas de expresabilidad, decidibilidad y complejidad de los lenguajes F1, ID y FG, entre muchos otros. El uso de constantes en expresiones de FG puede ser motivo de discusión y su inclusión u omisión en la sintaxis se suele decidir por razones técnicas. Dado que nosotros sí permitimos el uso de constantes en las fórmulas de F1, surgió la necesidad de distinguir algunas expresiones en base a un criterio más detallado de guarda y de términos frontera, dando lugar al fragmento TF1 que caracteriza uno de los casos de controlabilidad finita estudiados en este capítulo, debido a una propiedad estructural relacionada al *chase* que analizamos en la Sección 2.1(c).

Los fragmentos de UC2RPQ que utilizaremos se definen mediante propiedades estructurales referentes al *grafo subyacente* de una consulta. En muchos trabajos se ha utilizado la noción de *grafo de Gaifman* para asociar consultas a grafos, en particular en los casos de *conjunctive queries* [21, 105] y consultas regulares [31]. Esto permite estudiar clases de consultas basadas en propiedades de grafos como aciclicidad o *bounded treewidth*, que suelen estar ligadas a procesos polinomiales. Sin embargo, en muchos casos esta representación puede ser insuficiente o inadecuada para la obtención de resultados más rigurosos. Otras nociones de grafos utilizadas para representar propiedades estructurales de consultas han sido *grafos de incidencia* [60], *graph patterns* [33], o los ya mencionados *esqueletos* que fueron introducidos en [86] y que veremos formalmente al comienzo de la Sección 2.2. Sin mayores detalles, los esqueletos son multigrafos dirigidos con nodos y ejes etiquetados, donde los nodos están asociados a las variables de una consulta y los ejes a los átomos de la consulta. Las etiquetas de nodo indican cuáles variables aparecen libres o ligadas en la consulta y las etiquetas de ejes indican cuáles átomos se corresponden a un lenguaje finito o infinito. Respecto a este tipo de etiquetado establecimos ciertas propiedades topológicas de grafos (por ejemplo, un *ciclo infinito* se entiende como un ciclo no dirigido donde algún eje está etiquetado con un lenguaje infinito), dando paso al planteamiento de las siguientes clases de esqueletos:

$\mathcal{S}_0$: esqueletos que no contienen ciclos infinitos
$\mathcal{S}_1$: esqueletos cuyos ciclos infinitos contienen un nodo libre
$\mathcal{S}_2$: esqueletos cuyos ciclos infinitos y ligados contienen un nodo a distancia finita de uno libre

Dada una clase de esqueletos $\mathcal{S}$, el fragmento UC2RPQ($\mathcal{S}$) consta de todas las consultas en UC2RPQ cuyos esqueletos pertenecen a $\mathcal{S}$. Puesto que $\mathcal{S}_0 \subsetneq \mathcal{S}_1 \subsetneq \mathcal{S}_2$, obtenemos una cadena creciente de lenguajes de consulta:

$$\text{UC2RPQ}(\mathcal{S}_0) \subsetneq \text{UC2RPQ}(\mathcal{S}_1) \subsetneq \text{UC2RPQ}(\mathcal{S}_2) \subsetneq \text{UC2RPQ}$$

Estos fragmentos establecen un patrón de expresividad que muestra un contraste respecto a los lenguajes ontológicos, pues como se infiere del Teorema 38, para mantener la propiedad de controlabilidad finita, entre más expresiva sea la ontología menos expresivo debe ser el lenguaje de consulta, y viceversa. El gráfico de la Figura 3 ilustra esta situación junto con la referencia exacta de los resultados más importantes de este capítulo.

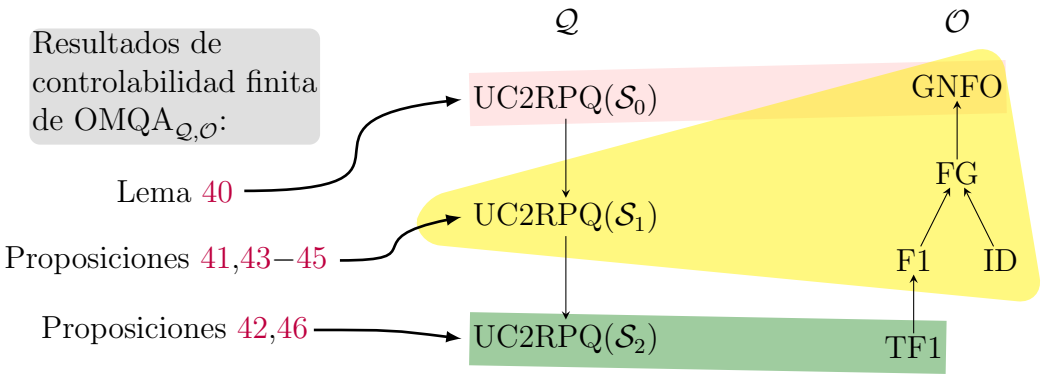

Figura 3: Diagrama de los resultados más importantes del Capítulo 2. Las regiones coloreadas indican el rango en el que varían las consultas y los conjuntos de restricciones en el problema OMQA$_{\mathcal{Q},\mathcal{O}}$ donde se cumple la respectiva propiedad de controlabilidad finita. Note que el orden de expresividad entre los lenguajes de consulta es inverso al orden de expresividad de las ontologías. El compendio de todos estos resultados conforman el planteamiento del Teorema 38.

Capítulo 3:
Extensiones de PDL *mediante propiedades de grafos*

En este capítulo presentamos un lenguaje nuevo llamado CPDL$^+$ (Sección 3.2), el cual fue definido originalmente en [85], y que extiende a PDL con el operador de programas *converse* y un operador recursivo que hemos denominado *programa conjuntivo*, que asemeja la noción de clausura de un lenguaje bajo cuantificación existencial y conjunción lógica, de manera similar a como se define el fragmento CQ de la lógica de primer orden, o C2RPQ como extensión de 2RPQ, o CQPDL como extensión de PDL [40]. Cuando se compone con los demás operadores de PDL también podemos asemejar otras nociones como *nesting*, explorado en [50] para expresiones y consultas regulares.

La particularidad del operador de *programa conjuntivo* es que se introduce junto a la noción de *grafo subyacente*, que servirá para establecer criterios de expresabilidad basados en propiedades topológicas de grafos. Como es usual en lenguajes basados en PDL, la semántica adoptada es el de estructuras de Kripke.

De manera similar a los artículos [66, 88, 103] donde estudian PDL y extensiones de este a nivel semántico, haremos un análisis análogo de CPDL^+, en el que gran parte de los resultados se basan en estudiar varios fragmentos específicos que restringen el tipo de programas conjuntivos posibles bajo condiciones de índole estructural. Más específicamente, dada una clase $\mathcal{G}$ de grafos finitos y conexos, definimos $\text{CPDL}^+(\mathcal{G})$ como el fragmento de CPDL^+ tal que los grafos subyacentes asociados a los programas conjuntivos están en $\mathcal{G}$ (Sección 3.2). Varios lenguajes de este estilo serán utilizados para obtener propiedades de CPDL^+ por medio de una representación jerárquica, es decir, analizaremos CPDL^+ a través de cadenas crecientes de fragmentos.

Como primeros resultados, procedemos a verificar que los conocidos lenguajes loop-CPDL e ICPDL son equivalentes a $\text{CPDL}^+(\mathcal{G}_{\circlearrowleft})$ y $\text{CPDL}^+(\mathcal{G}_{\cap})$ respectivamente, para ciertas clases finitas $\mathcal{G}_{\circlearrowleft}$ y $\mathcal{G}_{\cap}$ (Proposiciones 51 y 52), quedando establecido así que el operador de programa conjuntivo engloba distintos tipos de operadores que aumentan las capacidades expresivas de PDL y además que cualquier expresión de loop-CPDL e ICPDL es representable sintácticamente por ciertos patrones de grafos bastante simples. El esquema de estos resultados se da vía *traducciones*, por ejemplo, para

$$(\mathcal{L}_1, \mathcal{L}_2) \in \{(\text{loop-CPDL}, \text{CPDL}^+(\mathcal{G}_{\circlearrowleft})), (\text{CPDL}^+(\mathcal{G}_{\circlearrowleft}), \text{loop-CPDL})\}$$

obtenemos que para toda expresión e de $\mathcal{L}_1$ existe una expresión equivalente $\text{Tr}(e)$ de $\mathcal{L}_2$, donde Tr es una función que realiza sustituciones sintácticas de manera inductiva con respecto a la estructura de e. Para ICPDL y $\text{CPDL}^+(\mathcal{G}_{\cap})$ se puede plantear una solución análoga, pero para la demostración de la Proposición 52 optamos por una vía más larga aunque más provechosa para lo que abordaremos en las siguientes secciones. En este punto haremos uso de ciertas clases $\mathcal{G}$ definidas con respecto a un parámetro ampliamente utilizado en Teoría de Grafos denominado *treewidth*. Intuitivamente, el treewidth es un parámetro natural que indica qué tan similar es un grafo a un árbol, y dado un grafo G, su treewidth (denotado como $\text{tw}(G)$) se calcula en base a las *descomposiciones de árbol* de este. Estos conceptos los repasaremos en la Sección 3.1, donde definiremos las clases TW_k con $k \geq 1$, que consta de todos los grafos finitos G tales que $\text{tw}(G) \leq k$. Retomando la relación con ICPDL, demostramos por una serie de resultados de reescritura (Lemas 53−58) que los siguientes lenguajes son equivalentes:

$$\text{ICPDL}, \quad \text{CPDL}^+(\mathcal{G}_{\cap}), \quad \text{CPDL}^+(\text{TW}_1), \quad \text{CPDL}^+(\text{TW}_2)$$

En términos de expresabilidad, la equivalencia entre $\text{CPDL}^+(\text{TW}_1)$ y $\text{CPDL}^+(\text{TW}_2)$ resulta intrigante, pues a pesar de la robustez y variedad de grafos en TW_2 con respecto a TW_1,[5] semánticamente no hay un aumento real del poder expresivo de $\text{CPDL}^+(\text{TW}_2)$ con respecto al de $\text{CPDL}^+(\text{TW}_1)$, simplemente $\text{CPDL}^+(\text{TW}_2)$ tiene más expresiones que $\text{CPDL}^+(\text{TW}_1)$.

[5] TW_1 solo consta de árboles. TW_2 se puede caracterizar de muchas maneras, pero sin entrar en detalles contiene a TW_1 y además grafos que no tienen a K_4 como menor, por ejemplo los ciclos [73].

Es natural preguntarse en este punto si todos los fragmentos $\mathrm{CPDL}^+(\mathrm{TW}_k)$ son equivalentes entre sí, o en caso contrario, si existe una regla que determina cuáles fragmentos $\mathrm{CPDL}^+(\mathrm{TW}_{k+1})$ son más expresables que $\mathrm{CPDL}^+(\mathrm{TW}_k)$. Esto quedará resuelto en las siguientes secciones del capítulo en cuestión, donde daremos criterios de *bisimulación* que caracterizan a cada $\mathrm{CPDL}^+(\mathrm{TW}_k)$ con $k \geq 2$. La propiedad de bisimulación suele describirse como un *juego Spoiler-Duplicador*, donde Spoiler y Duplicador son dos contrincantes que bajo ciertas reglas intentan demostrar que dos estructuras son distintas (misión de Spoiler) o que son iguales (misión de Duplicador). En general, los juegos Spoiler-Duplicador se describen por *rondas*, que constan de una jugada realizada por Spoiler y de una respuesta dada por Duplicador. Los juegos Spoiler-Duplicador son particularmente valiosos en Teoría de Modelos, ya que funcionan como técnicas para verificar *equivalencia lógica* entre estructuras. Las reglas de este tipo de juegos y las condiciones de victoria se suelen correlacionar con lenguajes lógicos, lo que permite en muchos casos entender con bastante rigurosidad el alcance del poder expresivo de tales lenguajes, en especial en lo que concierne a modelos finitos. Algunos ejemplos clásicos de juegos Spoiler-Duplicador[6] son:

- *Juegos de Ehrenfeucht-Fraïssé*: definido en [94] y adaptado al formato de juego en [78], este juego permite obtener un criterio de equivalencia lógica entre estructuras respecto al fragmento $\mathcal{L}_m$ de la lógica de primer de orden que consta de todas las sentencias con *rango de cuantificación* a lo sumo m. En este caso, el número constante m se corresponde con la máxima cantidad de rondas de las que consta una partida del juego.

- *Juegos de piedras*: en [118, 121] definen juegos que constan de una cantidad finita de piedras, digamos k, que Spoiler y Duplicador deben disponer sobre los elementos del universo de un par de estructuras. Después de jugar un máximo de m rondas se verifica si Duplicador fue siempre capaz de generar homomorfismos parciales. En el caso de [118] el juego propuesto se analiza con *alternancia*,[7] que otorga a Spoiler la posibilidad de escoger la estructura donde realizará su jugada. Con este criterio, se demuestra que el juego se corresponde con el fragmento $\mathcal{L}_m^k$ de la lógica de primer orden que consta de todas las fórmulas de rango de cuantificación a lo sumo m y que se expresan con a lo sumo k variables. En el caso de [121] analizan un juego similar pero sin la condición de alternancia, lo que denominan como *existential k-pebble game*, y la condición de victoria de Duplicador se verifica *ad aeternum*, es decir, para toda ronda m Duplicador debe ser capaz de generar un homomorfismo parcial. El lenguaje que se corresponde con este juego es *la lógica infinitaria* $\mathcal{L}^k$ que consta de expresiones que no tienen más de k variables, y que se obtienen de fórmulas atómicas, igualdades, negación de igualdades, disyunción infinitaria, conjunción infinitaria y cuantificación existencial.

- *Juegos de bisimulación*: este tipo de juegos se pueden entender como un juego con una sola piedra. En el caso básico, asumimos que las estructuras sobre las que se desarrolla el juego son grafos con etiquetas en ejes y nodos. A diferencia de los juegos previos, en la bisimulación se establece una condición de movimiento relativo a la posición de la ronda previa, esto es, en cada ronda Spoiler escoge el grafo en el que va a jugar y mueve la correspondiente piedra

[6] Más recientemente este tipo de juegos se han analizado con un rigor algebraico [2, 137], habiendo sido caracterizados por ciertas familias de categorías de comónadas.

[7] Los juegos Spoiler-Duplicador también se suelen llamar *back-and-forth games*, donde el adjetivo *back-and-forth* alude a la noción de alternancia aquí mencionada.

a un nodo a distancia a lo sumo 1 de donde se encontraba la piedra al comienzo de la ronda (en otras palabras, Spoiler puede dejar la piedra donde está o moverla a un nodo adyacente). Esta restricción permite establecer conexiones con operadores modales y de hecho, el lenguaje capturado por esta noción de juego es justamente la lógica modal básica. La noción de bisimulación fue introducida independientemente por van Benthem [165] en el contexto de teoría de correspondencia modal, por Milner [136] y Park [142] en teoría de concurrencia, y por Forti y Honsell [92] en teoría de conjuntos sin el axioma de buena fundación. Ver [152] para una perspectiva histórica.

Nuestro formato de juego será definido en la Sección 3.4, descrito como una mezcla de juego de bisimulación y juego de piedras. Por conveniencia, hemos dividido nuestro análisis en varias subsecciones para lidiar por separado con distintas características del juego. En específico, en la Sección 3.4.1 definimos la relación de *simulación* $\mathbf{G}[\rightharpoonup_k]$ como un juego no alternante con $k \geq 3$ piedras, donde en cada ronda Spoiler mueve cualquiera de las piedras de su modelo a un nodo que se encuentre a distancia a lo sumo 1 de la posición relativa de las $k-1$ piedras restantes, y Duplicador debe responder generando algún homomorfismo parcial. La condición de victoria de Duplicador es como la considerada en [121], es decir, Duplicador tiene una estrategia ganadora en cualquier número de rondas. Bajo estas descripciones y considerando algunas condiciones semánticas extras,[8] demostramos el Teorema 60 que caracteriza el juego de simulación $\mathbf{G}[\rightharpoonup_k]$ con el fragmento positivo de $\mathrm{CPDL}^+(\mathrm{TW}_{k-1})$ (expresiones de $\mathrm{CPDL}^+(\mathrm{TW}_{k-1})$ que no contienen el símbolo $\neg$). El desfase en el subíndice se debe a una correlación con un desfase también presente en el concepto de treewidth.

En la Sección 3.4.2 definimos la relación de *bisimulación* $\mathbf{G}[\rightleftharpoons_k]$ que es la versión alternante de $\mathbf{G}[\rightharpoonup_k]$. Tal como se describió antes para los juegos de piedras, alternante quiere decir que Spoiler puede escoger el modelo donde jugar al comienzo de cada ronda, pero en nuestro caso solo puede cambiar de modelo con la condición de que las piedras estén *apiladas* (definido en **J3**). Aunque esto parezca una imposición bastante restrictiva de alternancia, en realidad esto se correlaciona con algunas características puntuales de CPDL^+: este lenguaje es de naturaleza modal, para expresiones unarias contamos con el operador de negación, y además sus expresiones deben armarse respetando cierta noción de conexidad. La caracterización de esta versión de juego de bisimulación con el lenguaje $\mathrm{CPDL}^+(\mathrm{TW}_{k-1})$ queda demostrado con los Teoremas 62 y 63.

Antes de mostrar la aplicabilidad de estos resultados en el análisis del poder expresivo de los lenguajes $\mathrm{CPDL}^+(\mathrm{TW}_k)$, veremos en la Sección 3.4.3 una reducción de nuestro esquema de juego de bisimulación al juego de bisimulación básico de la lógica modal, que por comodidad, lo presentamos en dos partes: el Teorema 64 para la relación $\mathbf{G}[\rightharpoonup_k]$ y el Teorema 65 para $\mathbf{G}[\rightleftharpoons_k]$. De estos resultados se desprende que el problema de decisión asociado al juego de bisimulación de $\mathrm{CPDL}^+(\mathrm{TW}_k)$ es decidible en tiempo polinomial (Corolario 67).

En la Sección 3.5 usaremos los juegos de bisimulación para demostrar que $\mathrm{CPDL}^+(\mathrm{TW}_k)$ es más expresable que $\mathrm{CPDL}^+(\mathrm{TW}_{k-1})$, para todo $k \geq 3$ (ver Figura 4). Esto se deducirá de la

[8] Por lo general, los juegos Spoiler-Duplicador se plantean de modo que una estrategia ganadora de Duplicador implique una propiedad de equivalencia lógica entre modelos, pero el recíproco de esto puede no valer a menos que se consideren ciertas restricciones. Cuando estas se imponen, se suele denominar a esa caracterización del juego como *propiedad de Hennessy-Milner*, en analogía al Teorema de Hennessy-Milner que da condiciones suficientes para que el concepto de bisimulación básico coincida con el de equivalencia de modelos respecto a la lógica modal.

propiedad de modelo de treewidth k planteada en el Corolario 69 y que establece que toda fórmula satisfacible de $\text{CPDL}^+(\text{TW}_k)$ también se satisface en un modelo numerable de treewidth a lo sumo k. En la Introducción ya hemos hablado de propiedades similares, como

- la *propiedad de modelo finito* satisfecha en lenguajes como GNFO, PDL y la lógica modal,
- la *propiedad de modelo tipo-árbol* satisfecha en algunos guarded fragments de la lógica de primer orden e incluso de primer orden extendido con operadores de punto fijo [111],
- la *propiedad de modelo de árbol* satisfecha por la lógica modal, PDL y las lógicas de descripción análogas [22].

Más parecido a nuestro resultado, en [66] reducen el problema de satisfacibilidad de loop-CPDL al de hallar un *cactus graph*[9] que cumple con ciertas condiciones de camino. Por otra parte, en [103] demuestran que ICPDL tiene la *propiedad de modelo casi-árbol*, o más específicamente, que toda fórmula de ICPDL satisfacible, también se satisface en un modelo con treewidth a lo sumo 2. Considerando que en este punto ya fue demostrada la equivalencia entre ICPDL y $\text{CPDL}^+(\text{TW}_2)$, nuestra propiedad de modelo de treewidth k es una generalización compatible con la propiedad de modelo casi-árbol de ICPDL.

$$\text{CPDL}^+(\text{TW}_1) = \text{CPDL}^+(\text{TW}_2) \longrightarrow \text{CPDL}^+(\text{TW}_3) \longrightarrow \text{CPDL}^+(\text{TW}_4) \longrightarrow \cdots = \text{CPDL}^+$$

Figura 4: Relaciones entre los lenguajes $\text{CPDL}^+(\text{TW}_k)$, donde cada flecha indica el orden de expresividad entre estos, siendo el lenguaje a la izquierda de la flecha el menos expresivo. La primera igualdad es parte del Corolario 59 y las flechas son casos particulares del Teorema 70. La unión de todos estos fragmentos conforman el lenguaje CPDL^+.

Del Corolario 69 y de la relación que podemos establecer con lógicas de punto fijo (Proposición 50) también se deriva una noción simplificada de satisfacibilidad de CPDL^+: para verificar que una fórmula de CPDL^+ es satisfacible basta con ver que es satisfacible sobre modelos numerables con treewidth acotado. Naturalmente, procederemos a estudiar la complejidad de este problema de decisión en la Sección 3.6. Para esta parte basamos nuestras demostraciones en las realizadas en [103] para satisfacibilidad de ICPDL, adaptándolas al nuevo operador de programa conjuntivo. La estrategia se basa en una serie de reducciones vinculadas a problemas de decisión relacionadas a propiedades de autómatas. En la Sección 3.6.1 definiremos un TWAPTA $\mathcal{T}$ como un tipo especial de autómata alternante que reconoce árboles infinitos ω-regulares y en base a estos conceptos demostraremos que

- para toda fórmula φ de CPDL^+ existe una fórmula φ' de CPDL^+ y un TWAPTA $\mathcal{T}$ tales que φ es satisfacible si y solo si φ' es satisfacible en un árbol $T \in L(\mathcal{T})$, o en otras palabras, el problema de satisfacibilidad de fórmulas en CPDL^+ es reducible al problema de satisfacibilidad de CPDL^+ respecto a árboles ω-regulares;
- para toda fórmula φ de CPDL^+, existe un TWAPTA $\mathcal{T}(\varphi)$ tal que si $T \in L(\mathcal{T}(\varphi))$, entonces T codifica una estructura de Kripke K que satisface a φ, con lo cual podemos plantear una

[9] Un *cactus graph* es un grafo conexo tal que todo par de ciclos distintos no se conectan en más de un nodo. En particular, todo cactus graph tiene treewidth a lo sumo 2.

reducción del problema de satisfacibilidad de CPDL$^+$ respecto a árboles ω-regulares al problema *emptiness* de lenguajes aceptados por TWAPTAs (ver Teorema 71 tomado de [167]).

Con esto obtenemos que satisfacibilidad de CPDL$^+$ es decidible y aprovechando varios resultados de [103], algunos de los cuales los adaptamos para expresiones en CPDL$^+$, se deducen cotas de complejidad para varias versiones del problema de satisfacibilidad (Teorema 83). En esta parte se vuelven relevantes varias relaciones que deduciremos entre determinados parámetros denominados *conjunctive width* e *intersection width* (Proposiciones 80−82). Los resultados más importantes de este capítulo junto con un mapa de las relaciones de expresividad de CPDL$^+$ con otros lenguajes se encuentran resumidos en la Figura 3.1.

Para finalizar este capítulo, en la Sección 3.7 damos cotas de complejidad del problema Model Checking para expresiones de CPDL$^+$.

Esquemas de razonamiento

Datos vs. sin datos / ontología vs. sin ontología

Por último, destacamos que a lo largo del trabajo se estudiaron tres esquemas de razonamiento semántico que dependen de factores tanto estructurales como descriptivos para el planteamiento formal de los distintos problemas a tratar:

(1) *grafos con datos y con ontologías*: en el Capítulo 1 usamos un lenguaje *data-aware* (es decir, cuya sintaxis involucra operadores de *comparación de datos*), utilizado para analizar problemas con semántica especificada por restricciones, como reparación y CQA;

(2) *grafos sin datos y con ontologías*: en el Capítulo 2 analizamos OMQA y FOMQA (problemas análogos a CQA) sobre bases de conocimiento, cuyos hechos interpretamos como información estructural y por lo tanto, se trata de estructuras sin datos;

(3) *grafos sin datos y sin ontologías*: en el Capítulo 3 usamos estructuras de Kripke (nuevamente, modelos sin datos) de manera irrestricta, es decir, sin conjuntos de restricciones que limiten la semántica.

El cuarto esquema, *grafos con datos y sin ontologías*, fue estudiado por los directores de esta tesis en varios proyectos pasados, como en [5, 6, 82–84] para estructuras de árboles con datos, y también en [3, 87] para grafos con datos.

Nociones básicas

Grafos y Lenguajes Navegacionales

El término *grafo* será utilizado a lo largo de este trabajo para referirse a muchos tipos de modelos, en particular, en varias partes hablaremos del grafo subyacente de una expresión o fórmula, y este será representado como un grafo no dirigido (con posibles etiquetas en ejes y nodos), pero los detalles puntuales serán presentados en cada capítulo, dado que el uso de estos grafos se restringe a propiedades muy específicas. En otro contexto, también hablaremos del grafo asociado a distintos tipos de estructuras, que en general los tratamos como grafos dirigidos con etiquetas en ejes y nodos, lo cual desarrollaremos a continuación.

Por $\mathbb{A}$ denotamos a un conjunto no vacío numerable de símbolos binarios, cuyos elementos serán denominados de distintas formas a lo largo del trabajo: *etiquetas de ejes* (Capítulo 1), *roles* (Capítulo 2), *programas básicos* o *atómicos* (Capítulo 3). La razón de esta indeterminación terminológica fue discutida en la Introducción, pero en la generalidad de este trabajo asumimos que los símbolos de $\mathbb{A}$ se interpretan como es usual, es decir, como relaciones binarias respecto a un dominio, lo que podemos visualizar como un grafo con ejes etiquetados. Más formalmente, una interpretación de $\mathbb{A}$, o un grafo (etiquetado) sobre $\mathbb{A}$ es una tupla $G = (V, \{\rightarrow_r | \ r \in \mathbb{A}\})$, donde V es el conjunto de nodos (o dominio de G) y $\rightarrow_r \subseteq V \times V$ es el conjunto de ejes o aristas con etiqueta r. Escribimos $u \xrightarrow{r} v$ para indicar que $(u, v) \in \rightarrow_r$. y usamos el término *arista* para referirnos también a $u \xrightarrow{r} v$.

Independientemente del formalismo teórico que usemos en cada capítulo, conviene ver al conjunto $\mathbb{A}$ como un alfabeto cuyos elementos se usan para describir patrones de caminos. Una forma básica y ampliamente utilizada en distintas áreas para consultar sobre estructuras con forma de grafos es el de *expresión regular*.

Definición 1 (Sintaxis de expresiones regulares)**.** Una expresión regular L sobre $\mathbb{A}$ está dada por la gramática:

$$L ::= \varepsilon \mid r \mid L \circ L \mid L \cup L \mid L^*,$$

donde ε se denomina la palabra vacía y $r \in \mathbb{A}$. Una expresión regular es 2-way cuando agregamos a la gramática las expresiones r^- con $r \in \mathbb{A}$, o en otras palabras, L es una expresión regular 2-way si es una expresión regular sobre $\mathbb{A}^{\pm} = \mathbb{A} \cup \{r^- \mid r \in \mathbb{A}\}$.

Si en una expresión L no se utiliza la estrella de Kleene decimos que es no recursiva, y si además no se utiliza $\cup$ decimos que es una palabra. El símbolo $\circ$ se corresponde con la *concatenación* que

por lo general omitiremos de la notación. Para una expresión regular L, usamos L^+ para denotar a LL^*, y L^{k+1} para denotar a LL^k con $k \in \mathbb{N}$ y $L^0 = \varepsilon$.

Definición 2 (Semántica de expresiones regulares)**.** Dado un grafo G sobre $\mathbb{A}$, cualquier expresión regular sobre $\mathbb{A}$ queda también interpretada con respecto a G bajo la siguiente semántica:

$$\begin{aligned}
[\![\varepsilon]\!]_G &:= \{(u,u) \mid u \in V\},\\
[\![r]\!]_G &:= \ \rightarrow_r,\\
[\![r^-]\!]_G &:= \{(u,v) \in V^2 \mid (v,u) \in [\![r]\!]_G\},\\
[\![L_1 \star L_2]\!]_G &:= [\![L_1]\!]_G \star [\![L_2]\!]_G \text{ para } \star \in \{\circ, \cup\},\\
[\![L^*]\!]_G &:= \text{la clausura reflexiva y transitiva de } [\![L]\!]_G.
\end{aligned}$$

Notemos que para la definición de $[\![L_1 L_2]\!]_G$ estamos usando la composición estándar de relaciones: dadas dos relaciones binarias R, S sobre un conjunto V,

$$RS := \{(x,z) \mid \exists y \in V.(x,y) \in R, (y,z) \in S\}.$$

Además, para todo entero $k \geq 0$, $R^{k+1} := RR^k$, donde $R^0 = \{(x,x) \mid x \in V\}$. De esta manera, tenemos que para toda expresión regular L y $k \geq 0$, ocurre que $[\![L^k]\!]_G = [\![L]\!]^k_G$.

La importancia de la Definición 2 radica en que ahora podemos hacer referencia a caminos en un grafo G inducidos por expresiones regulares. En general, un camino en G es una sucesión de la forma

$$u_0 \xrightarrow{r_1} u_1 \xrightarrow{r_2} u_2 \xrightarrow{r_3} \cdots u_{n-1} \xrightarrow{r_n} u_n, \tag{A}$$

donde los u_i son nodos de G y $u_{i-1} \xrightarrow{r_i} u_i$ son aristas de G. Decimos que (A) es un camino simple si todos los u_i son distintos entre sí; y que (A) es un ciclo si $u_0 = u_n$.

La etiqueta de (A) es la palabra $L = r_1 \cdots r_n$. Bajo estas condiciones decimos que hay un camino de u_0 a u_n con etiqueta L, denotado como $u_0 \xrightarrow{L} u_n$. Esta idea es fácilmente generalizable a expresiones regulares.

Definición 3. Sea L una expresión regular (2-way) sobre $\mathbb{A}$, G un grafo sobre $\mathbb{A}$ y u, v nodos de G. Decimos que hay un camino de u a v con etiqueta en L, denotado $u \xrightarrow{L} v$, si $(u,v) \in [\![L]\!]_G$.

Dado que toda expresión regular está asociada a un lenguaje regular, la definición anterior equivale a decir que existe una palabra en el lenguaje regular asociado a L que es etiqueta de un camino de u a v. En muchos casos a lo largo del trabajo haremos uso de esta noción, en especial cuando requiramos el uso de autómatas para los resultados más técnicos.

Por otra parte, todos los problemas que se plantearán en cada capítulo dependerán en gran medida del razonamiento semántico asociado a lenguajes de interés en el ámbito de estructuras con forma de grafos, ya sea a modo de *consultas* o en un sentido *ontológico*. En cualquier caso, el uso de lenguajes navegacionales basados en expresiones regulares serán fundamentales. En adelante, Var denota a un conjunto infinito numerable de variables.

Definición 4. Una regular path query sobre el alfabeto $\mathbb{A}$, o RPQ, es una expresión de la forma $x \xrightarrow{L} y$ donde $x, y \in \mathsf{Var}$ y L es una expresión regular sobre $\mathbb{A}$. Una 2-way RPQ, o 2RPQ, se define

análogamente, pero con L una expresión regular 2-way sobre $\mathbb{A}$. Cuando se sobreentiendan las variables, reescribimos la (2)RPQ $x \xrightarrow{L} y$ simplemente como L.

La semántica para este tipo de expresiones es la esperada. Dada una (2)RPQ $q = x \xrightarrow{L} y$ y un grafo G con dominio V, un homomorfismo de q sobre G es una interpretación de variables $h : \{x, y\} \to V$ tal que $(h(x), h(y)) \in [\![L]\!]_G$. Tomando $[\![q]\!]_G$ como el conjunto de todos los pares $(h(x), h(y))$ donde h es un homomorfismo de q sobre G, es fácil ver que si $x \neq y$, entonces $[\![q]\!]_G = [\![L]\!]_G$, lo que justifica el abuso de notación de la Definición 4. También diremos que $q = x \xrightarrow{L} y$ es no recursiva o que es una palabra cuando en correspondencia L sea no recursiva o una palabra.

Otro lenguaje bastante importante surge de aplicar un proceso similar al utilizado para definir el conjunto CQ de *consultas conjuntivas* de la lógica de primer orden,[10] pero aplicado a (2)RPQ. En lo que sigue extenderemos directamente a 2RPQ, ya que las extensiones análogas de RPQ surgen como fragmentos que no utilizan los símbolos r^-.

Definición 5. Una fórmula de C2RPQ sobre $\mathbb{A}$ es una expresión de la forma

$$\exists \bar{z}\big(x_1 \xrightarrow{L_1} y_1 \wedge \cdots \wedge x_m \xrightarrow{L_m} y_m\big), \tag{B}$$

donde para todo $i = 1, \ldots, m$, x_i, y_i son variables (no necesariamente distintas entre sí), $\bar{z}$ es una tupla de variables tomadas de entre los x_i e y_i, y cada L_i es una 2RPQ sobre $\mathbb{A}$. A este tipo de expresiones se les denota como $q(\bar{x})$, donde $\bar{x}$ es una tupla de las variables que no aparecen en $\bar{z}$. A las variables que aparecen en $\bar{x}$ se les denomina variables libres de q y a las que aparecen en $\bar{z}$ variables ligadas. Si no hay variables libres se denota simplemente q y se dice que es una expresión Booleana. La aridad de $q(\bar{x})$ es la dimensión de $\bar{x}$.

La semántica se da nuevamente por interpretaciones: para una C2RPQ q igual que (B) y un grafo G con dominio V, un homomorfismo de q sobre G es una interpretación h que asigna a cada variable x_i e y_i un elemento de V y tal que $(h(x_i), h(y_i)) \in [\![L_i]\!]_G$ para todo $i = 1, \ldots, m$. Si k es la dimensión de $\bar{x}$, entonces $[\![q]\!]_G$ se corresponde con el conjunto de k-tuplas $h(\bar{x})$[11], donde h es un homomorfismo de q en G. Escribimos $()$ para la tupla vacía. Si q es Booleana, escribimos $[\![q]\!]_G = \{()\}$ si existe un homomorfismo de q sobre G, o $[\![q]\!]_G = \{\}$ en caso contrario. Escribimos $G, h \models q$ para indicar que h es un homomorfismo de q sobre G.

Notemos que el uso del término *homomorfismo* para referirnos a interpretaciones que hacen verdaderas a las fórmulas q tiene sentido cuando vemos a este tipo de expresiones como grafos (ver Figura 5).

Definición 6. Una unión de C2RPQ, o UC2RPQ, es una disyunción finita $q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$ de C2RPQs q_i que comparten la misma tupla $\bar{x}$ de variables libres. La aridad de una tal UC2RPQ es la dimensión de $\bar{x}$ y si esta es 0, decimos que la fórmula es Booleana.

[10] Un CQ sobre $\mathbb{A}$ es básicamente un CRPQ en el que no se usa la estrella de Kleene ni la unión, ver [71, Section 2.1.4] para una definición más general.

[11] Abuso de notación: dada una tupla $\bar{x} = (x_1, \ldots, x_k)$, $h(\bar{x})$ denota la k-tupla $(h(\bar{x}_1), \ldots, h(\bar{x}_k))$.

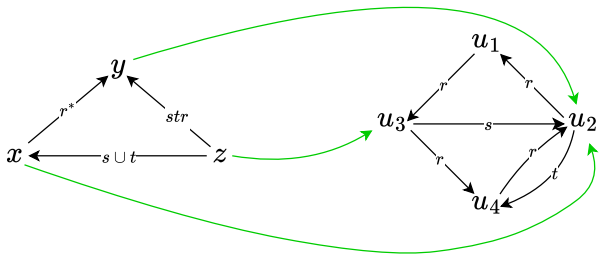

Figura 5: A la izquierda el grafo correspondiente al CRPQ $q(x, y, z) = x \xrightarrow{r^*} y \wedge z \xrightarrow{str} y \wedge z \xrightarrow{s \cup t} x$ y a la derecha un grafo G, ambos definidos sobre el alfabeto $\mathbb{A} = \{r, s, t\}$. Las flechas verdes indican cómo es una interpretación h tal que $G, h \models q$, o en otras palabras, h certifica que $(u_2, u_2, u_3) \in [\![q]\!]_G$.

La semántica para fórmulas UC2RPQ se obtiene extendiendo naturalmente la de C2RPQ vista en la Definición 5, e interpretando $\vee$ de la manera usual. Podemos definir un lenguaje ontológico basado en expresiones regulares.

Definición 7. Una regular path constraint, o RPC, es una expresión de la forma $L_1 \subseteq L_2$, donde L_i es un RPQ. Dado un grafo G, decimos que G satisface $L_1 \subseteq L_2$, denotado $G \models L_1 \subseteq L_2$, si $[\![L_1]\!]_G \subseteq [\![L_2]\!]_G$. Una 2RPC se define análogamente tomando cada L_i como un 2RPC.

Valiéndonos del uso de variables podemos definir C2RPC y UC2RPC de manera análoga (ver [32]), pero no se dará el caso en este trabajo en el que recurramos a tales expresiones, por lo que omitimos su planteamiento formal. Una (2)RPC $L_1 \subseteq L_2$ se dice que es una restricción de palabra si L_1, L_2 son palabras.

En muchos casos haremos uso de estas familias de expresiones como han sido presentadas, pero en general también estamos interesados en extensiones de RPQ que involucren valores de datos o etiquetas de nodos en interacción con expresiones regulares. Una forma usual de realizar este tipo de adaptaciones es a través de operadores con características modales, por lo que hablaremos un poco al respecto.

Lógica Modal

Además de $\mathbb{A}$ consideremos otro conjunto infinito numerable $\mathbb{P}$ disjunto de $\mathbb{A}$. A los elementos de $\mathbb{P}$ los llamamos etiquetas de nodos.[12] En este caso, extendemos la noción previa de grafo a la de grafo *con nodos etiquetados* dado como una tupla $G = (V, \{\rightarrow_r | \ r \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\})$ donde $X_p \subseteq V$ para todo $p \in \mathbb{P}$.

Definición 8 (Sintaxis)**.** La lógica (multi)modal básica, o ML, sobre $\mathbb{A}$ y $\mathbb{P}$ está dada por la gramática

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle r \rangle \varphi$$

donde $p \in \mathbb{P}$ y $r \in \mathbb{A}$. A $\langle r \rangle$ se le denomina operador modal con índice r o diamante r.

[12] Al igual que sucede con las etiquetas de ejes, los elementos de $\mathbb{P}$ tendrán distintos nombres a lo largo del trabajo. Por otra parte, en el Capítulo 1 no usaremos etiquetas de nodos sino valores de datos, cuya definición e interpretación será dada en ese capítulo.

La idea de los operadores modales es establecer desde un punto de vista local la noción de accesibilidad a través de un eje con la etiqueta correspondiente al índice del operador modal. Más formalmente,

Definición 9 (Semántica de ML)**.** Dado un grafo G sobre $\mathbb{A}$ y $\mathbb{P}$, las expresiones de ML también quedan interpretadas con respecto a G bajo la siguiente semántica:

$$\begin{aligned}
[\![p]\!]_G &:= X_p,\\
[\![\neg\varphi]\!]_G &:= V \setminus [\![\varphi]\!]_G,\\
[\![\varphi_1 \wedge \varphi_2]\!]_G &:= [\![\varphi_1]\!]_G \cap [\![\varphi_2]\!]_G,\\
[\![\langle r\rangle\varphi]\!]_G &:= \{u \in V \mid \exists v \in V. u \xrightarrow{r} v,\ v \in [\![\varphi]\!]_G\}.
\end{aligned}$$

Algunas generalizaciones de ML surgen de considerar operadores modales indexados por expresiones complejas, que pueden ser generadas ya sea con lenguajes regulares, o por recursión mutua como veremos en los lenguajes usados en los próximos capítulos (Reg-GXPath y PDL son ejemplos de esto, pero estos serán presentados más adelante). Otras generalizaciones surgen de la interpretación de ML como un fragmento de la lógica de primer orden, que a su vez se puede sumergir en otras clases de fórmulas de primer orden, como por ejemplo, GNFO que será presentado en el Capítulo 2.

La relación de ML con lenguajes ontológicos ha sido estudiado ampliamente en distintas áreas y su uso resulta particularmente importante en Teoría de Representación del Conocimiento.

Bases de Conocimiento y Lógicas de Descripción

Se dará el caso en este trabajo en el que debamos saltar a planteamientos propios y usuales de la Teoría de Representación del Conocimiento. Por ejemplo, en el Capítulo 1 requerimos el uso de un problema computacional indecidible con respecto a un tipo de restricciones pertenecientes a una lógica de descripción altamente expresiva; y en el Capítulo 2 el problema de estudio central (*Ontology Mediated Query Answering*) lo presentamos con la nomenclatura y terminología de bases de conocimiento.

Consideremos un vocabulario compuesto de un conjunto finito de predicados Pred y un conjunto infinito de constantes $\mathsf{N_I}$ (también denominados nombres de individuos). Los predicados *unarios* usualmente se denominan conceptos y al conjunto de conceptos lo denotamos por $\mathsf{N_C}$; los predicados *binarios* se denominan roles, y al conjunto de roles lo denotamos por $\mathsf{N_R}$.[13] Usamos $\mathsf{N_R^{\pm}}$ para denotar el conjunto de todos los roles junto con sus inversas, es decir, roles de la forma r^- para $r \in \mathsf{N_R}$.

Un término es un elemento de $\mathsf{Var} \dot\cup \mathsf{N_I}$. Un átomo α es de la forma $P(\bar t)$ donde P es un predicado de aridad k y $\bar t$ una k-tupla de términos. Un ground atom o hecho es un átomo cuyos términos son solo constantes. Denotamos por $terms(\alpha)$ al conjunto de términos en α y extendemos la notación a cualquier conjunción de átomos. Para $r \in \mathsf{N_R}$, a veces escribiremos $t \xrightarrow{r} t'$ en lugar de $r(t, t')$. También usaremos las letras $x, y, z, \ldots$ [resp. $\bar x, \bar y, \bar z, \ldots$] para denotar variables [resp.

[13] Una identificación posible con lo mostrado en las secciones previas es que los conceptos son etiquetas de nodos y los roles son etiquetas de ejes, pero notemos que en Pred tenemos relaciones de cualquier aridad.

tuplas de variables]; $a, b, c, \ldots$ [resp. $\bar{a}, \bar{b}, \bar{c}, \ldots$] para denotar constantes [resp. tuplas de constantes]; y $t, t', \ldots$ [resp. $\bar{t}, \bar{t}', \ldots$] para denotar términos [resp. tuplas de términos].

Un modelo o ABox es un conjunto (posiblemente infinito) de hechos. El dominio activo de un modelo $\mathcal{G}$, denotado $adom(\mathcal{G})$, es el conjunto de todas las constantes que aparecen en sus hechos. Para un modelo $\mathcal{G}$, denotamos por $\mathcal{G}^{\pm}$ a la extensión de $\mathcal{G}$ donde se agregaron los hechos $r^{-}(b, a)$ para cada $r(a, b) \in \mathcal{G}$ con $r \in \mathsf{N_R}$.

Cualquier modelo $\mathcal{G}$ se puede visualizar como un hipergrafo, pero dado que estamos principalmente interesados en estudiar propiedades relacionadas a grafos, por lo general simplificamos la noción de grafo (subyacente) de $\mathcal{G}$ a lo siguiente: dado un modelo $\mathcal{G}$, definimos $graph(\mathcal{G})$ como el conjunto $\{P(\bar{a}) \in \mathcal{G} \mid P \in \mathsf{N_C} \cup \mathsf{N_R}\}$, que es un grafo (posiblemente infinito) cuyo conjunto de nodos es (ímplicitamente) $adom(\mathcal{G})$, y cuyos ejes, etiquetas de ejes y etiquetas de nodos están dados por los hechos de $\mathcal{G}$ que provienen de predicados binarios y unarios. Casi siempre cuando hablemos de $\mathcal{G}$, estaremos refiriéndonos específicamente a $graph(\mathcal{G})$, a menos que se especifique lo contrario o quede claro del contexto. Todas las propiedades básicas de grafos (accesibilidad, camino entre nodos, etiqueta de caminos, camino simple, ciclos, etc) son aplicables a ABoxes $\mathcal{G}$ entendiéndose que se analizan respecto a $graph(\mathcal{G})$. De igual forma, cualquier expresión de UC2RPQ definida sobre el conjunto de roles que aparecen en un modelo $\mathcal{G}$ también es interpretable en $\mathcal{G}$ por medio de $graph(\mathcal{G})$.

Como se mencionó previamente, usaremos varios lenguajes lógicos para razonar sobre modelos, entre los cuales están GNFO y varias lógicas de descripción que presentaremos a continuación.

Definición 10 (Sintaxis)**.** La lógica de descripción $\mathcal{ALC}$ está dada por la gramática

$$A, B ::= \top \mid C \mid \neg A \mid A \sqcap B \mid A \sqcup B \mid \forall r.A \mid \exists r.A \tag{C}$$

donde $C \in \mathsf{N_C}$, $r \in \mathsf{N_R}$. A cualquier expresión de (C) se le denomina concepto (complejo). Una inclusión de conceptos o axioma $\mathcal{ALC}$ es una expresión de la forma $A \sqsubseteq B$ donde A, B son conceptos.

En la definición anterior hemos presentado el lenguaje básico $\mathcal{ALC}$ que fue introducido en [154, 155]. Con el tiempo varias extensiones de $\mathcal{ALC}$ se han considerado extendiendo la sintaxis presentada en (C) con nuevos conceptos, por ejemplo, la lógica de descripción $\mathcal{ALCOI}$ se obtiene añadiendo a la sintaxis de $\mathcal{ALC}$ operadores nominales (n$\mathcal{O}$minals), es decir, los conceptos $\{a\}$ con $a \in \mathsf{N_I}$; e inversas de roles ($\mathcal{I}$nverses) para expresiones con cuantificadores, es decir, los conceptos $\forall r^{-}.A$ y $\exists r^{-}.A$. El conjunto de los nominales lo denotamos por $\mathsf{N}_{\{\mathsf{I}\}}$, que asumimos disjunto de $\mathsf{N_C} \cup \mathsf{N_R^{\pm}} \cup \mathsf{N_I}$.[14] Un axioma $\mathcal{ALCOI}$ $A \sqsubseteq B$ se define análogo a un axioma $\mathcal{ALC}$ salvo que A, B son conceptos de $\mathcal{ALCOI}$.[15]

La lógica de descripción $\mathcal{ALCOIF}$ se define como $\mathcal{ALCOI}$ salvo que añadimos un nuevo tipo de axioma de la forma $Fun(r)$ con $r \in \mathsf{N_R}$, que se interpreta como una condición de funcionalidad sobre el rol r.

Una base de conocimientos, o KB de ahora en adelante, es un par $\mathcal{K} := (\mathcal{G}, \Gamma)$, donde $\mathcal{G}$ (la *ABox*) es un conjunto finito de hechos y Γ (la *TBox*) es un conjunto finito de axiomas. En lo

[14] También asumimos que hay una función computable de $\mathsf{N_I}$ a $\mathsf{N}_{\{\mathsf{I}\}}$ dada por $a \mapsto \{a\}$.

[15] Las lógicas de descripción $\mathcal{ALCI}$ y $\mathcal{ALCO}$ también suelen aparecer en la literatura. Ver [22] para una fuente más amplia del tema.

que sigue consideraremos axiomas de $\mathcal{ALCOIF}$ pero como ya hemos mencionado previamente, Γ podría considerarse en base a otro tipo de formalismos lógicos, como GNFO que será utilizado en el Capítulo 2. Denotamos $cn(\mathcal{K})$ al conjunto de nombres de concepto en $\mathcal{K}$, $roles(\mathcal{K})$ al conjunto de roles en $\mathcal{K}$, $ind(\mathcal{K})$ al conjunto de individuos en $\mathcal{K}$ (esto incluye a los individuos que determinan algún nominal en $\mathcal{K}$) y $nom(\mathcal{K}) \subseteq \mathsf{N}_{\{\mathrm{I}\}}$ al conjunto de nominales en $\mathcal{K}$ (es decir, los nominales que aparecen en alguna inclusión de conceptos en Γ).

Definimos una interpretación o asignación sobre una KB $\mathcal{K}$ como un par $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ donde $\Delta^{\mathcal{I}}$ es un conjunto no vacío llamado *dominio* de $\mathcal{I}$, y $\cdot^{\mathcal{I}}$ es una función llamada *función de interpretación*, cuyo dominio es $cn(\mathcal{K}) \cup roles(\mathcal{K}) \cup ind(\mathcal{K})$ y está definida como sigue:

- $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ para cada nombre de concepto C;
- $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ para cada nombre de rol r;
- $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ para cada nombre de individuo a.[16]

La semántica para los conceptos (C) se definen de la siguiente manera:

$$\top^{\mathcal{I}} := \Delta^{\mathcal{I}}, \quad (\neg A)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}, \quad \{a\}^{\mathcal{I}} := \{a^{\mathcal{I}}\},$$
$$(A \sqcap B)^{\mathcal{I}} := A^{\mathcal{I}} \cap B^{\mathcal{I}}, \quad (A \sqcup B)^{\mathcal{I}} := A^{\mathcal{I}} \cup B^{\mathcal{I}},$$
$$(\forall r.A)^{\mathcal{I}} := \{u \in \Delta^{\mathcal{I}} \mid \forall v.(u,v) \in r^{\mathcal{I}} \Rightarrow v \in A^{\mathcal{I}}\},$$
$$(\exists r.A)^{\mathcal{I}} := \{u \in \Delta^{\mathcal{I}} \mid \exists v.(u,v) \in r^{\mathcal{I}} \wedge\ v \in A^{\mathcal{I}}\}.$$

Una interpretación $\mathcal{I}$ es un modelo de $\mathcal{K} = (\mathcal{G}, \Gamma)$ si se cumplen las siguientes condiciones:

(I1) $a^{\mathcal{I}} \in C^{\mathcal{I}}$ para cada aserción $C(a)$ en $\mathcal{G}$;

(I2) $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ para cada aserción $r(a,b)$ en $\mathcal{G}$;

(I3) $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ para cada inclusión de conceptos $A \sqsubseteq B$ en Γ;

(I4) para cada $Fun(r)$ en Γ es cierto que $(u,v),(u,w) \in r^{\mathcal{I}}$ implica $v = w$, para todo $u, v, w \in \Delta^{\mathcal{I}}$.

El conjunto de nombres de concepto usados en cualquier KB se puede extender con nuevos símbolos para obtener un KB equivalente donde las inclusiones de conceptos admiten una *forma normal*, como se realiza en [101], por lo que asumimos sin pérdida de generalidad que cualquier TBox de $\mathcal{ALCOIF}$ solo contiene inclusiones de concepto de la forma

$$\textstyle\bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j, \quad A \sqsubseteq \forall r.B, \quad A \sqsubseteq \exists r.B, \quad A \equiv \{a\}, \tag{D}$$

donde A_i, B_j, A, B son nombres de concepto y $\{a\}$ es un nominal. Esto concluye todo lo relacionado a la ontología que se utilizará en la Sección 1.2.2.

Algunos conceptos computacionales

En varias instancias de este trabajo hablaremos de la complejidad de determinados problemas y también haremos uso de cierta terminología que suele usarse en teoría de bases de datos. En esta

[16] En el Capítulo 2 haremos uso de la *standard name assumption* que establece que los nombres de individuo se interpretan como sí mismos en cualquier interpretación.

parte no ahondaremos en detalles técnicos, solo aclararemos la terminología que se usará a lo largo del trabajo.

En general, decimos que un problema de decisión es indecidible si no existe un algoritmo (o máquina de Turing) que lo resuelva, en caso contrario, se dice que es decidible. Existen muchas formas para clasificar problemas de decisión pero las más tradicionales se basan en el uso de recursos temporales o espaciales. Algunas clases de complejidad que mencionaremos son:

- PTime, la clase de los problemas de decisión que se resuelven en tiempo polinomial por una máquina de Turing determinista.
- NP, la clase de los problemas de decisión que se resuelven en tiempo polinomial por una máquina de Turing no determinista.
- coNP, la clase de problemas de decisión cuyos complementos se encuentran en NP.
- ExpTime, la clase de los problemas de decisión que se resuelven en tiempo exponencial por una máquina de Turing determinista. 2ExpTime (3ExpTime) es análogo para tiempo doblemente (resp. triplemente) exponencial.
- PSpace, la clase de los problemas de decisión que se resuelven en espacio polinomial por una máquina de Turing determinista.

Dado un problema de decisión L y una clase de complejidad $\mathcal{C}$, decimos que L es $\mathcal{C}$-hard si todo problema de $\mathcal{C}$ se puede reducir a L mediante reducciones que no usen recursos superiores a los de $\mathcal{C}$. Por ejemplo, para las clases PTime y NP se suele pedir que las reducciones sean polinomiales, mientras que para clases como ExpTime se pueden usar reducciones exponenciales. Decimos que L es $\mathcal{C}$-completo si L es $\mathcal{C}$-hard y además pertenece a $\mathcal{C}$.

En muchos de los planteamientos que abordaremos en este trabajo se tratarán con problemas de decisión que requieren varios valores de entrada, que pueden ser un grafo G y un conjunto de expresiones $\mathcal{R}$. En algunos casos, uno de estos parámetros podrían fijarse, lo cual es una simplificación común y razonable para problemas computacionales que lidian con (in)consistencia de bases de datos y que da lugar a los términos de *data complejidad* (cuando se fija $\mathcal{R}$), y de *complejidad de expresión* cuando se fija G. En modo contrario, cuando ambos parámetros varían, el análisis computacional de los problemas se denomina *complejidad combinada*. Toda esta terminología la adoptamos de [168].

Capítulo 1

Reparación de data-grafos

Enfoque computacional El libre acceso a grandes volúmenes de datos induce el desarrollo de nuevas aplicaciones que lidien correctamente con los índices semánticos y búsquedas de respuestas consistentes, lo cual en muchos casos involucra tareas avanzadas de razonamiento por encima de la data existente. Modelos alternativos para el almacenamiento de datos, tales como las *bases de datos orientadas a grafos* se vuelven cada vez más populares a medida que permiten representar efectivamente una gran cantidad de datos voluminosos. Este tipo de estructuras son especialmente útiles cuando la topología subyacente que relaciona a los individuos de una base de datos es tan importante como los datos en sí, como sucede en análisis de redes sociales [80], procedencia de datos [14] y Web Semántica [20]. En este capítulo usaremos como estructuras a los data-grafos como fueron introducidos en [127]. Una base de datos orientada a grafos es comúnmente *consultada* a través de lenguajes navegacionales tales como los RPQ o generalizaciones de este [35], que capturan la noción de conectividad entre nodos unidos por un camino etiquetado con una palabra perteneciente a un lenguaje regular (Definición 4). En muchos casos, a estos lenguajes de consulta se les añaden nuevas operaciones para aumentar su expresabilidad, lo que en consecuencia podría aumentar la complejidad de evaluación entre otros aspectos computacionales.

RPQ y sus extensiones más comunes (por ejemplo C2RPQ o NRE [34]) solo pueden actuar sobre los ejes del data-grafo, dejando a un lado cualquier posible interacción con la data almacenada en los nodos. Esta limitación condujo al diseño de una familia de lenguajes denominada GXPath [127], cuyo propósito principal es el de consultar sobre estructuras como los data-grafos, como una generalización del uso del lenguaje XPath [157] sobre estructuras tipo XML[1]. Análogo al caso de bases de datos relacionales, para los data-grafos también se espera que la forma de estos esté condicionada por ciertas reglas denominadas *restricciones de integridad*. Cuando la base de datos no satisface estas restricciones, una manera de abordar tal problema es buscando una base de datos *similar* (es decir, una que difiera lo mínimo posible de la base original) que sí satisfaga las restricciones, o en otras palabras, se busca una *reparación* de la base de datos con respecto a las restricciones dadas. Algunos de los primeros trabajos que abordan formalmente el problema de hallar reparaciones de bases de datos tradicionales bajo cierto tipo de restricciones expresables en Cálculo Relacional son [19, 97], donde se proponen algunas técnicas basadas en propiedades con-

[1] https://www.w3.org/TR/xpath/

juntistas que también asumimos bajo el esquema de data-grafos. En [19, 41] presentan en paralelo las nociones de reparación y de *respuestas consistentes de consultas* (CQA), que será de relevancia más adelante en este capítulo y también será el foco de otros resultados que abordaremos en el Capítulo 2. Sin embargo, el problema de computar reparaciones ha sido tratado independientemente de su conexión con CQA [9, 61, 95, 172], e incluso varios sistemas de gestión de bases de datos cuentan con sus propias herramientas de reparación automatizada.

En [7] estudiamos el problema de reparación de un data-grafo inconsistente con respecto a ciertos tipos de restricciones. Nuestra propuesta se basó en considerar varios lenguajes de consulta definidos en [127] y establecer un criterio de satisfacibilidad global para que puedan ser utilizados como ontologías. Un trabajo previo que se basa en una consideración similar es [12], donde adaptan PDL con una semántica de *grafos conexos con elemento raíz* y establecen un formalismo lógico que captura varios tipos de restricciones como inclusiones de concepto o de camino, valiéndose del comportamiento modal de PDL. En nuestro caso, consideramos lenguajes que además de evaluar la existencia de determinados patrones de caminos, estos pueden verse influenciados por la data almacenada en los nodos. A pesar del reciente interés por establecer a GQL como el lenguaje estándar para data-grafos [93], hemos optado por trabajar con Reg-GXPath (Definición 12) y varios de sus fragmentos, por las siguientes razones: (1) Reg-GXPath cuenta con las características sintácticas necesarias para relacionar patrones de camino con lectura de datos, además de contar con una noción de comparación de valores de datos básica; (2) Reg-GXPath cubre varias de las funcionalidades requeridas en un lenguaje navegacional, incluidas los patrones de nodo y de camino implementadas en GQL; (3) varios de los resultados de complejidad obtenidos son válidos incluso para fragmentos menos expresivos que Reg-GXPath, como Core-GXPath o Reg-GXPath^{pos}; (4) Reg-GXPath es fácil de interpretar y de comparar con otros lenguajes tradicionales y cuenta con varias propiedades computacionales favorables, todo lo cual fue discutido en [127].

Enfoque lógico/matemático La terminología *reparación* que abordaremos en este capítulo alude desde un punto de vista pragmático a la idea de redefinir, cambiar, eliminar, agregar o actualizar aspectos puntuales de una estructura que generan algún tipo de inconsistencia con respecto a ciertos requerimientos ontológicos. Desde un punto de vista lógico, esto se puede entender como el proceso de analizar la satisfacibilidad de una fórmula perteneciente a algún lenguaje formal. Todas las formas listadas para reparar una estructura inconsistente pueden plantearse en términos de teoría de modelos, por ejemplo, el eliminar o agregar información a una estructura se puede entender en términos de subestructuras, mientras que un cambio o actualización de los datos, se puede entender como una reinterpretación de un símbolo relacional o funcional. Quizás el ingrediente extra que difiere en lo que respecta a consideraciones lógicas usuales sea el requerimiento de que una tal reparación debe ser *minimal*, entendiéndose por esto que si es posible manipular la estructura para obtener la consistencia deseada, una reparación es aquella que se consigue con la mínima cantidad de modificaciones hechas a la estructura original.

Las estructuras presentadas en este capítulo también pueden entenderse en el sentido usual de la teoría de modelos. Un *data-grafo* es esencialmente una estructura sobre un lenguaje relacional que solo consta de relaciones unarias y binarias, por lo que siempre recurriremos al apoyo visual cuando representemos un data-grafo simplemente como un grafo dirigido con etiquetas en ejes y nodos. Dado que el marco teórico utilizado en [7] es el mismo que el de [127], mantendremos gran

parte de la notación y terminología de ambos artículos. Otras razones para llamar a las estructuras *data-grafos* en lugar de simplemente *grafos* son: (1) Un data-grafo tiene también la particularidad de que la asignación de nodos a etiquetas es funcional, a diferencia del enfoque relacional donde los nodos pueden contar con múltiples etiquetas. Esto suele aplicarse en modelado de archivos XML cuando se describen estructuralmente como *data-árboles* con un solo atributo por nodo [157], pero también puede pensarse como una simplificación menor, como veremos en algunos resultados más adelante. Un data-grafo se entiende entonces como una generalización natural de un data-árbol. (2) Otro aspecto a considerar es que en teoría de bases de datos es normal pensar y asumir que las estructuras son entes finitos, lo que representa otra diferencia tácita con el enfoque relacional de la teoría de modelos donde, exceptuando consideraciones semánticas, las estructuras pueden ser de cualquier cardinal, pudiendo ser entonces estructuras infinitas. El prefijo *data* que aparece en los conceptos de este capítulo sugiere que nos limitamos a estructuras finitas.

En la Sección 1.1 damos todas las definiciones y notación necesarias para el desarrollo y explicación de los resultados principales y repasamos la noción de reparación, bajo el esquema ontológico propuesto para el lenguaje Reg-GXPath. En la Sección 1.2 damos los distintos criterios de reparación, *subset repair* (Sección 1.2.1) y *superset repair* (Sección 1.2.2), los correspondientes problemas de decisión que denominamos ∃Subset Repair y ∃Superset Repair, sus versiones funcionales, analizamos la complejidad de estos problemas bajo distintos criterios ontológicos, y analizamos algunos casos tratables. En la Tabla 1.1 se puede ver un resumen de los resultados obtenidos en este capítulo, destacando las particularidades que permiten dar con las cotas de complejidad respectivas.

<table>
<tr><th rowspan="2">lenguaje / criterio</th><th colspan="2">Reg-GXPath^{pos}</th><th colspan="2">Reg-GXPath</th></tr>
<tr><th>path expr.</th><th>node expr.</th><th>path expr.</th><th>node expr.</th></tr>
<tr><td>subset repair</td><td>NP-completo en d.c. incluso para restricciones en Core-GXPath (Teo. 21)</td><td>PTime en c.c. incluso para versión funcional (Teo. 25)</td><td>NP-completo en d.c. (Lower bound por Teo. 21 o Teo. 22)</td><td>NP-completo en d.c. (Teo. 22)</td></tr>
<tr><td rowspan="2">superset repair</td><td>NP-completo en e.c. incluso para restricciones en Core-GXPath (Teo. 34)</td><td>PTime en c.c. (Teo. 33)</td><td rowspan="2">Indecidible en d.c. (Teo. 26)</td><td rowspan="2">Indecidible en c.c. con posible adaptación a e.c. (Teo. 27)</td></tr>
<tr><td colspan="2">PTime con bounded labels* (Teo. 35)</td></tr>
</table>

d.c.: data complejidad; c.c.: complejidad combinada; e.c.: complejidad de expresión
*bounded labels: finitas etiquetas de ejes y nodos

Tabla 1.1: Resultados de la complejidad computacional de las distintas variaciones consideradas de los problemas ∃Subset Repair y ∃Superset Repair bajo restricciones de Reg-GXPath y varios de sus fragmentos. En la fila superior la separación de los lenguajes en *path expressions* y *node expressions* indica que el resultado se obtuvo específicamente para restricciones de ese tipo.

1.1 Preliminares

A lo largo de este capítulo el conjunto de etiquetas (de ejes) $\mathbb{A}$ es finito y $\mathbb{D}$ denota un conjunto infinito numerable disjunto de $\mathbb{A}$, cuyos elementos se denominan valores (de datos). Para los ejemplos y las demostraciones, supondremos que tanto $\mathbb{D}$ como $\mathbb{A}$ son conjuntos que constan de palabras, i.e. secuencias finitas del alfabeto latino.

Definición 11. Un data-grafo sobre $\mathbb{D} \dot{\cup} \mathbb{A}$ es una tupla $G = (V, D, \{\rightarrow_r | \ r \in \mathbb{A}\})$ donde V es un conjunto, el par $(V, \{\rightarrow_r | \ r \in \mathbb{A}\})$ es un grafo sobre $\mathbb{A}$ y $D : V \rightarrow \mathbb{D}$ es una función que asigna a cada nodo un valor de dato.[2]

Simplificamos la notación de data-grafo a (V, D, E), donde $E := \{(u, a, v) \mid u \xrightarrow{a} v\}$. En general, un data-grafo $G = (V, D, E)$ lo pensamos como un grafo dirigido y etiquetado cuyo conjunto de nodos es V, cada nodo $u \in V$ tiene asignado un valor en $D(u) \in \mathbb{D}$ y $(u, a, v) \in E$ es un eje o flecha etiquetada (ver Figura 1.1). Denotamos ∂E al conjunto de pares $(u, v) \in V^2$ tales que $(u, a, v) \in E$ para alguna etiqueta a. Cuando los valores de datos sean irrelevantes, (por ejemplo, al estudiar solo propiedades de camino), denotaremos un data-grafo simplemente como $G = (V, E)$. En general, asumimos que V y E son conjuntos finitos. Como en [127], definimos el tamaño de G como $|G| := |V| + |E|$.

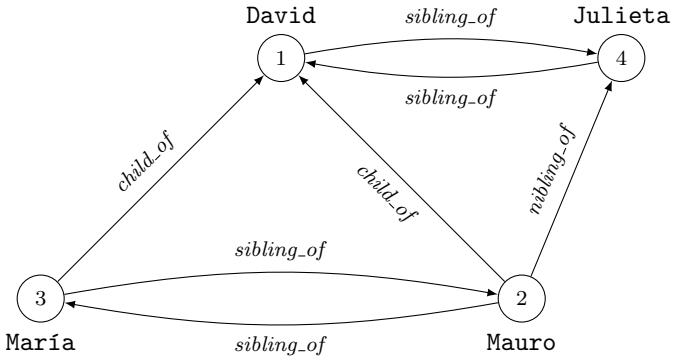

Figura 1.1: Ejemplo de un data-grafo G donde los nodos representan personas y los ejes relaciones de parentesco. En este caso tenemos cuatro individuos (o nodos) $1, 2, 3, 4$, una función D que les asigna nombres y el conjunto E que indica la relación familiar que hay entre cada par de individuos.

A continuación definimos el lenguaje con el cual realizaremos consultas sobre este tipo de estructuras.

Definición 12. Reg-GXPath[3] es el lenguaje que se compone de las siguientes expresiones, dado por un esquema de recursión mutuo:

$$\varphi, \psi ::= \mathsf{c}^{=} \mid \mathsf{c}^{\neq} \mid \varphi \wedge \psi \mid \neg\varphi \mid \langle\alpha\rangle \mid \langle\alpha = \beta\rangle \mid \langle\alpha \neq \beta\rangle$$
$$\alpha, \beta ::= \varepsilon \mid \sqcup \mid a \mid a^{-} \mid \alpha \circ \beta \mid \alpha \cup \beta \mid \overline{\alpha} \mid \alpha^{*} \mid \alpha^{n,m} \mid \varphi?,$$

[2] Los valores de datos difieren de las etiquetas de nodo, que no serán usadas en este capítulo.

[3] Originalmente en [127] denotaron a este lenguaje como $\#\mathsf{GXPath}_{\mathrm{reg}}(\mathsf{c},\mathsf{eq})$, pues surge como una combinación de varios extensiones conocidas de XPath. Nosotros optamos por una notación más simple, al igual que para los fragmentos presentados más adelante.

donde $\mathtt{c} \in \mathbb{D}$, $a \in \mathbb{A}$ y $n, m \in \mathbb{N}$. A las fórmulas del primer renglón se les llama expresiones de nodo y a las del segundo renglón se les llama expresiones de camino. Usamos el símbolo η para referirnos a una expresión cualquiera de Reg-GXPath. En la Figura 1.2 se muestra la interpretación semántica $[\![\eta]\!]_G$ de cualquier expresión η dado un data-grafo G.

Semántica para expresiones de nodo

$$
\begin{aligned}
[\![\mathtt{c}^{=}]\!]_G &:= \{u \in V \mid D(u) = \mathtt{c}\},\\
[\![\mathtt{c}^{\neq}]\!]_G &:= V \setminus [\![\mathtt{c}^{=}]\!]_G,\\
[\![\varphi \wedge \psi]\!]_G &:= [\![\varphi]\!]_G \cap [\![\psi]\!]_G,\\
[\![\varphi \vee \psi]\!]_G &:= [\![\varphi]\!]_G \cup [\![\psi]\!]_G,\\
[\![\neg\varphi]\!]_G &:= V \setminus [\![\varphi]\!]_G,\\
[\![\langle\alpha\rangle]\!]_G &:= \{u \in V \mid \exists v \in V.(u,v) \in [\![\alpha]\!]_G\},\\
[\![\langle\alpha = \beta\rangle]\!]_G &:= \{u \in V \mid \exists u_1, u_2 \in V.(u,u_1) \in [\![\alpha]\!]_G, (u,u_2) \in [\![\beta]\!]_G, D(u_1) = D(u_2)\},\\
[\![\langle\alpha \neq \beta\rangle]\!]_G &:= \{u \in V \mid \exists u_1, u_2 \in V.(u,u_1) \in [\![\alpha]\!]_G, (u,u_2) \in [\![\beta]\!]_G, D(u_1) \neq D(u_2)\},
\end{aligned}
$$

Semántica para expresiones de camino

$$
\begin{aligned}
[\![\varepsilon]\!]_G &:= \{(u,u) \mid u \in V\},\\
[\![\sqcup]\!]_G &:= \partial E,\\
[\![a]\!]_G &:= \{(u,v) \in V^2 \mid (u,a,v) \in E\},\\
[\![a^-]\!]_G &:= \{(u,v) \in V^2 \mid (v,a,u) \in E\},\\
[\![\alpha \star \beta]\!]_G &:= [\![\alpha]\!]_G \star [\![\beta]\!]_G \text{ para } \star \in \{\circ, \cup, \cap\},\\
[\![\overline{\alpha}]\!]_G &:= V^2 \setminus [\![\alpha]\!]_G,\\
[\![\varphi?]\!]_G &:= \{(u,u) \mid u \in V, u \in [\![\varphi]\!]_G\},\\
[\![\alpha^{n,m}]\!]_G &:= \textstyle\bigcup_{k=n}^{m} ([\![\alpha]\!]_G)^k,\\
[\![\alpha^*]\!]_G &:= \text{la clausura reflexiva y transitiva de } [\![\alpha]\!]_G.
\end{aligned}
$$

Figura 1.2: Semántica para el lenguaje Reg-GXPath (Definición 12) dado un data-grafo $G = (V, D, E)$.

Dado que el complemento de una expresión de camino funciona como una negación sobre relaciones binarias, usamos la notación $\alpha \Rrightarrow \beta$ para denotar $\overline{\alpha} \cup \beta$ y para expresiones de nodo usamos la notación usual $\varphi \Rightarrow \psi$ para denotar $\neg\varphi \vee \psi$. De igual manera consideramos los operadores $\alpha \cap \beta$ y $\varphi \vee \psi$ definidos como es usual y cuya semántica se incluyó en la Figura 1.2. Una relación interesante bajo este contexto es que $\varphi \vee \psi$ equivale a $\langle \varphi? \cup \psi? \rangle$, por lo que podemos seguir aludiendo al operador $\vee$ incluso si no contamos con $\neg$.

El tamaño de una expresión η de Reg-GXPath, denotado $|\eta|$, es el número de símbolos que la componen, salvo si $\eta = \alpha^{n,m}$ que en este caso se define como $|\alpha| + \log n + \log m$.

A pesar de que usamos tipografías distintas para representar a los elementos de $\mathbb{D}$ y $\mathbb{A}$, por lo general usaremos la notación $\downarrow_a$ y $\downarrow_a^-$ para distinguir con mayor facilidad a las etiquetas a y a^- que aparezcan en una expresión η de Reg-GXPath. Por ejemplo, escribimos $\downarrow_{child\_of} (\mathtt{Maria}^{=})? \downarrow_{sister\_of}$ en lugar de $child\_of(\mathtt{Maria}^{=})?sister\_of$. Algunos fragmentos de Reg-GXPath que se usarán más adelante son:

- Core-GXPath que se obtiene restringiendo el uso de la estrella de Kleene a etiquetas a y sus inversas a^-.
- Reg-GXPath^{pos} que se obtiene restringiendo el uso de la negación $\neg\varphi$ y del complemento $\overline{\alpha}$ en cualquier expresión. Aquí se mantienen el resto de operadores, incluyendo la expresión $\mathtt{c}^{\neq}$, que semánticamente es la negación de $\mathtt{c}^{=}$.
- Core-GXPath^{pos} que es la intersección de los dos anteriores.

Ejemplo 13. Note que toda expresión regular L (Definición 1) es también una expresión de camino de Core-GXPath^{pos}. Abusando un poco de la notación, en general diremos que 2RPQ (Definición 4) es un fragmento estricto de Core-GXPath^{pos}. □

Por otra parte, Reg-GXPath y sus fragmentos presentados están limitados a expresar relaciones unarias y binarias. Esto los hace incomparable con otras extensiones bastante usadas de 2RPQ, como C2RPQ y UC2RPQ. Este hecho es consecuencia de [127, Theorem 4.24].

Ejemplo 14. Con Core-GXPath podemos expresar la idea de *amistad transitiva* en una red social mediante $\alpha := \downarrow^+_{friend\_of}$. Este es un ejemplo común de expresión regular y no requiere mucho de las capacidades de Core-GXPath. Una propiedad más interesante que mezcla expresiones regulares con valores de datos es la siguiente:

$$\beta := \langle\downarrow_{follows} (\mathtt{Luis}^{=})?\rangle?\downarrow^+_{friend\_of}\langle\downarrow_{follows} (\mathtt{Luis}^{=})?\rangle?,$$

que pide identificar pares de nodos que son *amigos por transitividad y que siguen a alguien con el valor de dato* `Luis` (aunque no necesariamente a un mismo usuario de nombre `Luis`). En una base de datos social donde hay enlaces familiares, podemos identificar a las personas que *no tienen descendientes tocayos* con la siguiente expresión:

$$\varphi := \neg\langle\varepsilon = \downarrow^+_{father\_of}\rangle.$$

Note que las dos últimas propiedades no es posible expresarlas en muchos lenguajes navegacionales como RPQ o C2RPQ, dado que en estas no se permite comparar valores de datos. □

Consistencia Dado un data-grafo G específico y una expresión η de Reg-GXPath, podríamos preguntarnos si esta captura a todos los nodos de G. Cuando esto suceda pensaremos en η como una *restricción de integridad* de G y en caso contrario, asumiremos que ha surgido un tipo de *inconsistencia* entre G y η. Más rigurosamente, definimos la noción de consistencia de la siguiente manera:

Definición 15. Sea $G = (V, D, E)$ un data-grafo y $\mathcal{R} = \mathcal{R}_n \cup \mathcal{R}_p$ un conjunto finito de restricciones, donde $\mathcal{R}_n$ y $\mathcal{R}_p$ consisten de expresiones de nodo y camino, respectivamente. Decimos que $(G, \mathcal{R})$ es consistente, denotado $G \models \mathcal{R}$, si las siguientes condiciones se cumplen:

- para todo $\varphi \in \mathcal{R}_n$, tenemos que $[\![\varphi]\!]_G = V$, denotado $G \models \varphi$;
- para todo $\alpha \in \mathcal{R}_p$, tenemos que $[\![\alpha]\!]_G = V^2$, denotado $G \models \alpha$.

En otro caso, decimos que $(G, \mathcal{R})$ es inconsistente. Cuando $[\![\varphi]\!]_G \neq V$ [resp. $[\![\alpha]\!]_G \neq V^2$], decimos que G no satisface φ [resp. α]. Decimos que un nodo u [resp. un par de nodos (u, v)]

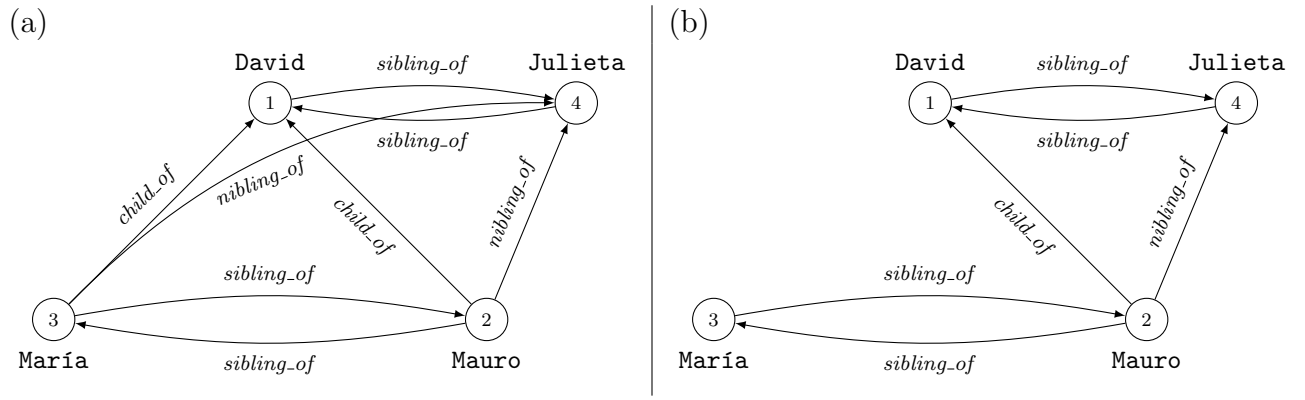

Figura 1.3: (a) Modificación de la base de datos de la Figura 1.1 aplicando el proceso (1). (b) Análogo aplicando el proceso (2). En ambos casos se toma el conjunto de restricciones $\mathcal{R}$ del Ejemplo 16.

de G incumple una expresión de nodo φ [resp. una expresión de camino α] si $u \notin [\![\varphi]\!]_G$ [resp. $(u, v) \notin [\![\alpha]\!]_G$]. Parafraseando, $(G, \mathcal{R})$ es inconsistente si alguna expresión de $\mathcal{R}$ se incumple.

A lo largo del trabajo decimos simplemente que G es (in)consistente, cuando $\mathcal{R}$ se sobreentienda. Todos los resultados de complejidad obtenidos consideran a $\mathcal{R}$ conteniendo solo expresiones de nodo o solo expresiones de camino, por lo que en muchos casos vamos a considerar simplemente $\mathcal{R} = \mathcal{R}_n$ o $\mathcal{R} = \mathcal{R}_p$. Los subíndices n y p siempre serán un indicativo del caso a tener en cuenta.

Ejemplo 16. Consideremos el data-grafo G de la Figura 1.1 y el siguiente conjunto de restricciones de integridad $\mathcal{R} = \{\alpha, \beta\}$ donde

$$\alpha := \downarrow_{sibling\_of} \Rrightarrow \downarrow^{-}_{sibling\_of},$$
$$\beta := \downarrow_{child\_of} \downarrow_{sibling\_of} \Rrightarrow \downarrow_{nibling\_of},$$

que expresan la *simetría de hermanos* y que *el hijo de un hermano es necesariamente un sobrino*, respectivamente. Notemos que G es inconsistente ya que $(3, 4) \notin [\![\beta]\!]_G$. Una manera para solucionar este problema sería quitar de nuestro conjunto de restricciones a la expresión β, lo que sería suficiente pues $[\![\alpha]\!]_G = V^2$. Dado que por lo general las restricciones representan reglas inamovibles de una teoría o realidad del mundo y que la consistencia suele ser un requisito epistemológico a garantizar, quisiéramos *arreglar* desde la data conocida el problema de inconsistencia sin modificar nuestro conjunto de restricciones preestablecido. Hay dos soluciones notorias que se obtienen al modificar el conjunto de flechas E de G (ver Figura 1.3): (1) agregando la relación $(3, \textit{nibling\_of}, 4)$; (2) eliminando la relación $(3, \textit{child\_of}, 1)$.

Notemos que si bien eliminar la flecha $(1, \textit{sibling\_of}, 4)$ arreglaría el problema con β, la inconsistencia persistiría con α, pero una solución plausible sería eliminar el nodo 4 (junto con todas sus flechas incidentes) del data-grafo, aunque esto podría considerarse como un cambio abrupto respecto a la base de datos original.

Reparaciones Dado un data-grafo G inconsistente con respecto a un conjunto de restricciones $\mathcal{R}$, nos gustaría computar un nuevo data-grafo G' consistente y que difiera mínimamente de G, el cual llamaremos *reparación*. Hay diferentes nociones de reparación en la literatura, algunas de las

cuales son: las conjuntistas [163], las basadas en atributos [171], y las basadas en cardinalidad [130]. En este trabajo nos limitamos a trabajar con las reparaciones conjuntistas para establecer una semántica de *mínima diferencia* entre data-grafos.

La noción de minimalidad que usaremos está basada en la diferencia entre los conjuntos de nodos y ejes del data-grafo original y su posible reparación. Por lo general, este concepto se suele presentar en términos de la diferencia simétrica de conjuntos, pero en varios trabajos se ha establecido que este criterio viene acompañado de una alta complejidad, incluso para ontologías simples como se ve en [9, 126]. Por ello, es común que solamente se consideren reparaciones conjuntistas que se obtienen solo añadiendo o solo eliminando información como se realiza en [32, 132, 163]. A estos procesos los llamamos *reparaciones subconjuntistas* y *superconjuntistas*, según el caso.

Definición 17. Sean $G = (V, D, E)$ y $G' = (V', D', E')$ un par de data-grafos. Decimos que G es subdata-grafo de G' [sim. G' es superdata-grafo de G], denotado $G \subseteq G'$ si y solo si $V \subseteq V'$, $E \subseteq E'$ y para todo $u \in V$, $D(u) = D'(u)$.

Como es usual, dado un data-grafo $G = (V, D, E)$ y un conjunto $X \subseteq V$, podemos definir el subdata-grafo generado por X como la tupla $G_X = (X, D_X, E_X)$ donde $D' := D|_X$ y

$$E_X := \{(u, a, v) \in E \mid u, v \in X\}.$$

G_X se puede entender como *el subdata-grafo de G más grande con dominio X*. Dados dos subdata-grafos $G_1 = (V_1, D_1, E_1)$, $G_2 = (V_2, D_2, E_2)$ de G definimos la unión de G_1 y G_2 como la tupla $G_1 \cup G_2 = (V_1 \cup V_2, D', E_1 \cup E_2)$ donde

$$D'(u) := \begin{cases} D_1(u) & \text{si } u \in V_1, \\ D_2(u) & \text{si } u \in V_2. \end{cases}$$

Note que D' está bien definida pues D_1 y D_2 coinciden en $V_1 \cap V_2$. De manera análoga podemos definir la intersección de G_1 y G_2, denotado $G_1 \cap G_2$.

Definición 18. Sean G y G' un par de data-grafos y $\mathcal{R}$ un conjunto de restricciones. Decimos que G' es una reparación subconjuntista o subreparación de G [resp. superconjuntista o superreparación] con respecto a $\mathcal{R}$ si:

- $(G', \mathcal{R})$ es consistente;
- $G' \subseteq G$ [resp. $G \subseteq G'$]; y
- no existe un data-grafo H tal que $(H, \mathcal{R})$ sea consistente y $G' \subsetneq H \subseteq G$ [resp. $G \subseteq H \subsetneq G'$].

Denotamos al conjunto de las subreparaciones [resp. superreparaciones] de G con respecto a $\mathcal{R}$ como $\mathrm{Rep}_{\subseteq}(G, \mathcal{R})$ [resp. $\mathrm{Rep}_{\supseteq}(G, \mathcal{R})$].

Ejemplo 19. En el Ejemplo 16, los data-grafos (a) y (b) de la Figura 1.3 son una superreparación y una subreparación de G con respecto a $\mathcal{R}$, respectivamente.

1.2 Computando reparaciones

En esta sección, estudiaremos la complejidad computacional del problema de hallar reparaciones dando como entrada un data-grafo finito y un conjunto finito de restricciones. Empezamos men-

cionando un dato bastante útil de Reg-GXPath demostrado en [127, Theorem 4.3.] y que usaremos a lo largo del capítulo.

Teorema 20. *Dada una expresión α de* Reg-GXPath *y un data-grafo G, es posible computar el conjunto $[\![\alpha]\!]_G$ en tiempo polinomial en el tamaño de G y α, más específicamente, en tiempo $O(|\alpha| \cdot |V|^3)$, donde V es el conjunto de nodos de G.*

Se concluye a partir de este resultado que dado un conjunto finito $\mathcal{R}$ de expresiones de Reg-GXPath, es posible verificar si G es consistente con respecto a $\mathcal{R}$ en tiempo polinomial en el tamaño de G y $\mathcal{R}$, donde el tamaño de $\mathcal{R}$ se define como $|\mathcal{R}| := \sum_{\alpha \in \mathcal{R}} |\alpha|$.

Hemos dividido esta sección en dos partes para analizar por separado los problemas relacionados a reparaciones cuando nos restringimos a subdata-grafos o superdata-grafos. Más allá de lo natural que parezca esto, la razón principal se debe a que también estudiamos el problema de reparación según distintos fragmentos de Reg-GXPath. Otra consideración a tener en cuenta es que en algunos casos se fijará el conjunto de restricciones $\mathcal{R}$, obteniendo resultados en data complejidad.

1.2.1 Subreparaciones

Denotamos por $\emptyset$ al data-grafo *vacío*. Dado que este satisface cualquier conjunto de restricciones, concluimos que cualquier data-grafo G tiene una subreparación dado cualquier conjunto de restricciones $\mathcal{R}$. Para entender la complejidad de encontrar tales subreparaciones, definimos los siguientes problemas computacionales donde $\mathcal{L}$ representa un fragmento de Reg-GXPath:

Subset Repair	
Entrada:	Un data-grafo finito G y un conjunto finito $\mathcal{R}$ de expresiones de $\mathcal{L}$.
Salida:	Una subreparación $G' \neq \emptyset$ de G con respecto a $\mathcal{R}$.

∃Subset Repair	
Entrada:	Un data-grafo finito G y un conjunto finito $\mathcal{R}$ de expresiones de $\mathcal{L}$.
Salida:	Decidir si G tiene una subreparación $G' \neq \emptyset$ con respecto a $\mathcal{R}$.

Ambos problemas dependen explícitamente de $\mathcal{L}$ por lo que en los siguientes resultados siempre se especificará el fragmento de Reg-GXPath a considerar. Note que ∃Subset Repair es el problema de decisión asociado a Subset Repair y además, ∃Subset Repair se puede reducir a Subset Repair, dando así una cota inferior para la complejidad del problema general de computar subreparaciones. También es fácil notar que ∃Subset Repair con respecto a $\mathcal{L} =$ Reg-GXPath está en NP, ya que por el Teorema 20 podemos verificar no determinísticamente en tiempo polinomial si algún subdata-grafo no vacío de G satisface $\mathcal{R}$.

Teorema 21. *Existe un conjunto finito $\mathcal{R}_p \subset$ Core-GXPath^{pos} tal que* ∃Subset Repair *es* NP-completo *en data complejidad, cuando se fija $\mathcal{R} = \mathcal{R}_p$ en la entrada.*

Demostración. Construiremos una reducción de 3SAT a ∃Subset Repair, considerando un conjunto fijo $\mathcal{R}_p$ de expresiones de camino que describiremos más adelante. Dada una fórmula proposicional ϕ en 3CNF de n variables $p_1, \ldots, p_n$ y m cláusulas $c_1, \ldots, c_m$, queremos armar un data-grafo

$G_\phi = (V, D, E)$ tal que G_ϕ tiene una subreparación no vacía con respecto a $\mathcal{R}$ si y solo si ϕ es satisfacible. Daremos el conjunto $\mathcal{R}$ luego de describir la reducción $\phi \mapsto G_\phi$.

Esta reducción depende solo de una cantidad finita de etiquetas, específicamente, el conjunto de etiquetas a usar es $\{cycle, sat, valid\} \subset \mathbb{A}$. De hecho, como los valores de datos serán completamente irrelevantes, definiremos G_ϕ simplemente como el par (V, E).

Reducción. El conjunto de nodos V lo definimos como

$$V := \{\bot_i \mid 1 \leq i \leq n\} \dot\cup \{\top_i \mid 1 \leq i \leq n\} \dot\cup \{u_j \mid 1 \leq j \leq m\}.$$

Codificaremos la información de ϕ en los ejes del data-grafo con la etiqueta *sat* de la siguiente manera: para todo par de índices i, j,

$$(\top_i, sat, u_j) \in E \text{ si y solo si el literal } p_i \text{ aparece en } c_j,$$
$$(\bot_i, sat, u_j) \in E \text{ si y solo si el literal } \neg p_i \text{ aparece en } c_j.$$

Note que las flechas con etiqueta *sat* no solo indican qué literales aparecen en determinada cláusula, sino que indican el valor de verdad que debería tener la correspondiente variable proposicional para que la cláusula se satisfaga.

En principio, buscamos que para cada índice i solo uno de los nodos $\bot_i, \top_i$ se mantenga en algún subdata-grafo de G_ϕ, en correspondencia con el posible valor de verdad que una valuación asignaría a cada variable proposicional p_i. Usaremos la etiqueta *valid* para este propósito. Para todo par $u, v \in V$, $(u, valid, v) \in E$ excepto si ocurre alguno de los siguientes casos:

$$u = \bot_i \text{ y } v = \top_i \text{ para algún } 1 \leq i \leq n,$$
$$u = \bot_i \text{ y } v = u_j \text{ para algún } 1 \leq i \leq n \text{ y } 1 \leq j \leq m,$$
$$u = \top_i \text{ y } v = u_j \text{ para algún } 1 \leq i \leq n \text{ y } 1 \leq j \leq m.$$

La etiqueta *cycle* la usaremos para garantizar que mantendremos en el dominio una cantidad mínima de nodos necesarios. Para todo $u, v \in V$, $(u, cycle, v) \in E$ si ocurre alguno de los siguientes casos:

$$\begin{array}{llll} u = u_j & \text{y} & v = u_{j+1} & \text{para algún } 1 \leq j \leq m-1, \\ u = u_m & \text{y} & v = *_1 & \text{para } * \in \{\bot, \top\}, \\ u = *_i^1 & \text{y} & v = *_{i+1}^2 & \text{para } *^1, *^2 \in \{\bot, \top\} \text{ y } 1 \leq i \leq n-1, \\ u = *_n & \text{y} & v = u_1 & \text{para } * \in \{\bot, \top\}. \end{array}$$

Note que si para cada i eliminamos uno de los nodos $\bot_i, \top_i$, se genera un único ciclo con la etiqueta *cycle*. La Figura 1.4 muestra el grafo G_ϕ para una fórmula particular ϕ. Note que $|G_\phi|$ es polinomial en el tamaño de ϕ, definido simplemente como $n + m$.

Restricciones. El conjunto $\mathcal{R}_p$ consta solo de las siguientes expresiones:

$$\beta_1 := \downarrow^+_{cycle},$$
$$\beta_2 := \downarrow_{valid} \cup \downarrow_{valid} \downarrow_{sat} .$$

La fórmula β_1 indica que *todo par de nodos debe pertenecer a la cláusura transitiva de cycle*, lo cual se garantiza directamente si el data-grafo contiene un ciclo con la etiqueta *cycle* que pasa por todos los nodos. La fórmula β_2 indica que *todos los nodos se conectan por la relación valid o por un camino de tamaño 2 con la etiqueta valid* $\circ$ *sat*.

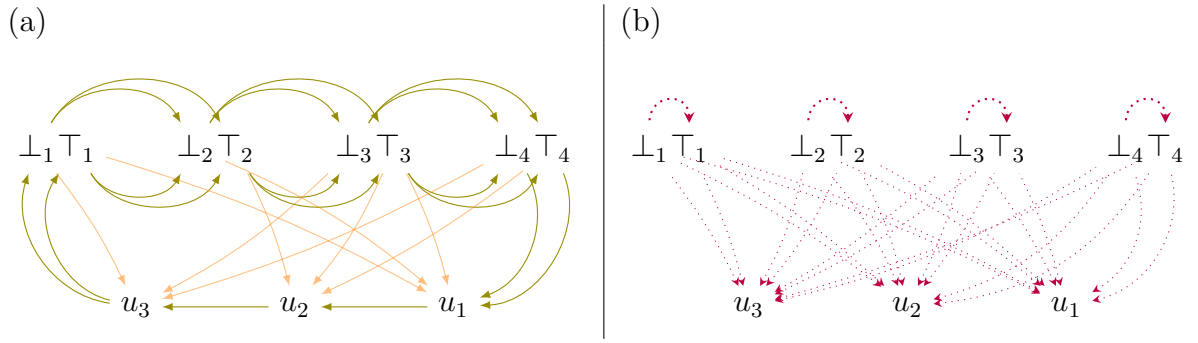

Figura 1.4: Para la fórmula $\phi = (p_1 \vee \neg p_2 \vee p_3) \wedge (p_2 \vee p_3 \vee p_4) \wedge (\neg p_1 \vee \neg p_3 \vee \neg p_4)$, el data-grafo G_ϕ se puede entender de la siguiente manera: (a) Las flechas verdes representan la relación *cycle* y las flechas naranjas la relación *sat*. (b) La relación *valid* se representa con el color morado, en este gráfico se muestran con flechas punteadas todas las relaciones que **no** aparecen en G_ϕ. Luego, G_ϕ es la superposición de (a) y el complemento de (b).

Notemos que $(G_\phi, \mathcal{R}_p)$ es inconsistente, dado que la fórmula β_2 se incumple particularmente en $(\bot_i, \top_i)$ para todo $1 \leq i \leq n$. Esto implica que en caso de existir una subreparación no vacía de G_ϕ, esta no podría contener los nodos $\bot_i$ y $\top_i$ a la vez, sin embargo, debe contener al menos uno de los dos porque necesitamos que la fórmula β_1 se siga cumpliendo. La Figura 1.5 muestra una reparación no vacía para el G_ϕ de la Figura 1.4.

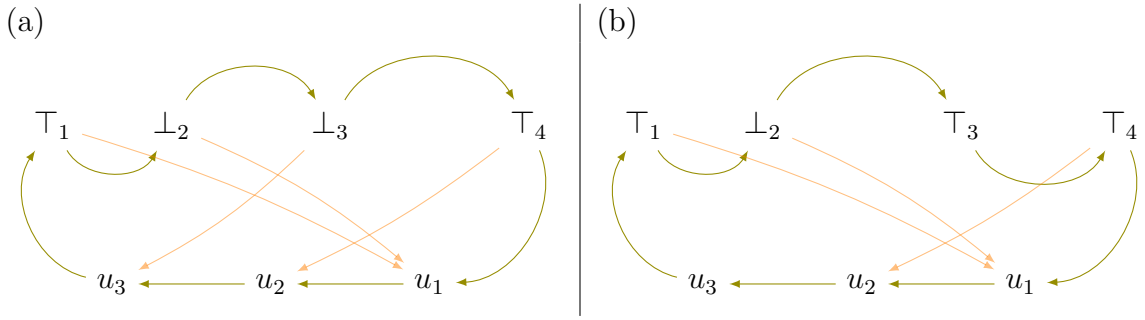

Figura 1.5: Para G_ϕ de la Figura 1.4, estos dibujos representan subdata-grafos, donde la relación *valid* no se muestra, pero se considera subyacente. (a) En este caso tenemos una subreparación de G_ϕ con respecto a $\mathcal{R}_p$, que se corresponde con la valuación $p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1$ que hace verdadera a la fórmula ϕ. (b) En este caso tenemos un subdata-grafo que no puede ser una subreparación porque la fórmula β_2 se incumple en $(\top_1, u_3)$. Note que este data-grafo está asociado a la valuación $p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 1, p_4 \mapsto 1$ que hace falsa a ϕ.

Finalizamos la prueba demostrando la siguiente proposición.

Afirmación. *Las siguientes son equivalentes:*

1. *ϕ es satisfacible.*
2. *G_ϕ tiene una subreparación no vacía con respecto a $\mathcal{R}_p$.*

1. implica 2. Sea f una valuación de $\{p_1, \ldots, p_n\}$ tal que $f \models \phi$ y definamos el subdata-grafo (no vacío) $G_{\phi,f} = (V', E')$ como sigue:

$$V' := \{\bot_i \mid 1 \leq i \leq n \text{ y } f(p_i) = 0\} \cup \{\top_i \mid 1 \leq i \leq n \text{ y } f(p_i) = 1\} \cup \{u_j \mid 1 \leq j \leq m\},$$
$$E' := \{(u, a, v) \in E \mid u, v \in V'\}.$$

Claramente β_1 se cumple porque tenemos un ciclo con la etiqueta *cycle* en $G_{\phi,f}$, garantizado por solo haber eliminado uno entre $\bot_i, \top_i$ para cada i.

Para β_2 solo falta verificar que si $*_i \in V'$ con $* \in \{\bot, \top\}$ entonces $(*_i, u_j) \in [\![\beta_2]\!]_{G_{\phi,f}}$ para todo j. Como $f \models \phi$ entonces $f \models c_j$ para todo j, pero por definición de cláusula, esto ocurre si existe un literal ℓ en c_j tal que $f \models \ell$. Supongamos que $\ell = p_k$ para algún k. Por definición de $G_{\phi,f}$, tenemos que $\top_k \in V'$ y $(\top_k, \mathit{sat}, u_j) \in E'$. Dado que $(*_i, \mathit{valid}, \top_k) \in E'$ para todo par i, k (vale incluso si $i = k$ pues $*_i = \top_k$ en este caso), obtenemos que hay un camino con la etiqueta $\mathit{valid} \circ \mathit{sat}$ de $*_i$ a u_j. Análogamente si $\ell = \neg p_k$.

Hemos obtenido que $G_{\phi,f}$ es consistente con respecto a $\mathcal{R}_p$, lo cual es suficiente para garantizar que G_ϕ tiene una subreparación no vacía. Si bien podríamos demostrar que $G_{\phi,f}$ es justamente una subreparación, dejamos esta parte de la prueba hasta aquí.

2. implica 1. Supongamos que $G' = (V', E')$ es una subreparación no vacía de G con respecto a $\mathcal{R}_p$. Primero veamos que V' satisface lo siguiente: (a) $\{u_j \mid 1 \leq j \leq m\} \subset V'$ y (b) $\bot_i \in V'$ si y solo si $\top_i \notin V'$ para todo $1 \leq i \leq n$.

En efecto, como $V' \neq \emptyset$, entonces $(u, u) \in [\![\beta_1]\!]_{G'}$ para algún $u \in V'$. Dado que la relación *cycle* no es reflexiva en G para ningún nodo tampoco lo es en G', por lo que debe existir un camino en G' (y en G) de la forma

$$v_1 \xrightarrow{\mathit{cycle}} v_2 \xrightarrow{\mathit{cycle}} \cdots v_{k-1} \xrightarrow{\mathit{cycle}} v_k,$$

con $\ell > 1$ y $v_1 = v_k = u$. Por definición de G, tal recorrido solo es posible si pasamos al menos una vez por todos los nodos de la forma u_j (sucediendo así (a)) y por al menos uno de los nodos $\bot_i$ y $\top_i$ para cada i. Ahora, si $\bot_i, \top_i \in V$ para algún i, entonces β_2 se incumple en $(\bot_i, \top_i)$, por lo que (b) debe suceder.

Definamos la valuación f como $f(p_i) = 0$ si $\bot_i \in V'$, o $f(p_i) = 1$ si $\top_i \in V'$, para todo i, la cual está bien definida por (b). Tenemos que $f \models \phi$ si y solo si $f \models c_j$ para todo j, y esto último sucede si $f \models \ell$ para algún literal ℓ en c_j. Veamos que esto último ocurre para todo j.

Sea $*_i^1 \in V'$ con $*^1 \in \{\bot, \top\}$ y fijemos j. Como $(*_i^1, u_j) \in [\![\beta_2]\!]_{G'}$, esto solo es posible si hay un camino con etiqueta $\mathit{valid} \circ \mathit{sat}$ de $*_i^1$ a u_j, es decir, existe $*_k^2$ con $*^2 \in \{\bot, \top\}$ tal que $*_i^1 \xrightarrow{\mathit{valid}} *_k^2 \xrightarrow{\mathit{sat}} u_j$ es un camino en G'. Consideremos el literal ℓ dado por $\ell = \neg p_k$ si $*^2 = \bot$ o $\ell = p_k$ si $*^2 = \top$, que por definición de G se encuentra en c_j. Dado que $f \models \ell$, obtenemos que $f \models c_j$. □

El teorema anterior dice que basta con considerar un conjunto finito de expresiones de camino positivas para que ∃Subset Repair sea intratable en data complejidad. De hecho, las fórmulas propuestas son expresiones de RPQ. Veremos que es posible obtener la misma conclusión para un conjunto fijo de expresiones de nodo.

Teorema 22. *Existe* $\mathcal{R}_n \subset$ Reg-GXPath *finito tal que* ∃Subset Repair *es* NP-completo *en data complejidad, cuando se fija* $\mathcal{R} = \mathcal{R}_n$ *en la entrada.*

Demostración. Nuevamente reducimos 3SAT a ∃Subset Repair. Dada una fórmula proposicional ϕ en 3CNF de n variables $p_1, \ldots, p_n$ y m cláusulas $c_1, \ldots, c_m$, queremos armar un data-grafo $G_\phi = (V, D, E)$ tal que G_ϕ tiene una subreparación no vacía con respecto a $\mathcal{R}$ si y solo si ϕ es satisfacible. Igual que en el teorema anterior, daremos el conjunto $\mathcal{R}$ luego de describir la reducción $\phi \mapsto G_\phi$.

Esta reducción dependerá del conjunto de etiquetas $\{assign, pos, neg, aux\} \subset \mathbb{A}$ y de valores de datos $\{\texttt{var}, \texttt{clause}, \texttt{false}, \texttt{true}\} \subset \mathbb{D}$. Note que a diferencia de la demostración anterior, ahora sí requeriremos de valores de datos.

Reducción. El conjunto de nodos V lo definimos como

$$V := \{\bot, \top\} \dot\cup \{v_i \mid 1 \leq i \leq n\} \dot\cup \{u_j \mid 1 \leq j \leq m\}.$$

Los valores de datos se asignan de la siguiente manera:

$$\begin{aligned} D(\bot) &= \texttt{false}, \quad D(\top) = \texttt{true}, \\ D(v_i) &= \texttt{var}, \text{ para todo } 1 \leq i \leq n, \\ D(u_j) &= \texttt{clause}, \text{ para todo } 1 \leq j \leq m. \end{aligned}$$

Codificaremos la información de ϕ en los ejes del data-grafo con las etiquetas pos, neg de la siguiente manera: para todo par de índices i, j,

$$\begin{aligned} &(u_j, pos, v_i) \in E \text{ si y solo si el literal } p_i \text{ aparece en } c_j, \\ &(u_j, neg, v_i) \in E \text{ si y solo si el literal } \neg p_i \text{ aparece en } c_j. \end{aligned}$$

Para todo índice i, agregamos las flechas $(v_i, assign, \bot)$ y $(v_i, assign, \top)$. En principio, buscamos que para cada índice i, el nodo v_i se conecte solo con uno de los nodos $\bot, \top$ en algún subdata-grafo de G_ϕ, en correspondencia con el posible valor de verdad que una valuación asignaría a cada variable proposicional p_i.

Por último agregamos a E los ejes que componen el siguiente ciclo con etiqueta aux y el cual usaremos para garantizar que la subreparación (en caso de existir) debe tener el mismo conjunto de nodos que G_ϕ:

$$\bot \xrightarrow{aux} \top \xrightarrow{aux} v_1 \xrightarrow{aux} \cdots \xrightarrow{aux} v_n \xrightarrow{aux} u_1 \xrightarrow{aux} \cdots \xrightarrow{aux} u_m \xrightarrow{aux} \bot. \tag{A}$$

La Figura 1.6 muestra el grafo G_ϕ para una fórmula particular ϕ. Note que $|G_\phi|$ es polinomial en el tamaño de ϕ, que se define como $n + m$.

Restricciones. El conjunto $\mathcal{R}_n$ consta solo de las siguientes expresiones:

$$\begin{aligned} \psi_1 &:= \langle \downarrow_{aux} \rangle, \\ \psi_2 &:= \neg(\texttt{var}^{=} \wedge \langle \downarrow_{assign} \neq \downarrow_{assign} \rangle), \\ \psi_3 &:= \texttt{clause}^{=} \Rightarrow \big(\langle \downarrow_{pos} \downarrow_{assign} (\texttt{true}^{=})? \rangle \vee \langle \downarrow_{neg} \downarrow_{assign} (\texttt{false}^{=})? \rangle \big). \end{aligned}$$

La fórmula ψ_1 indica que *de todo nodo sale una flecha aux*, lo cual se garantiza directamente si el data-grafo contiene un ciclo con la etiqueta aux que pasa por todos los nodos. La fórmula ψ_2 indica que *de todos los nodos con valor* `var` *no pueden salir dos flechas assign a nodos con valores distintos*. Por último, la fórmula ψ_3 indica que *toda cláusula de ϕ debe satisfacerse*. Como dato curioso, el conjunto $\mathcal{R}_n$ propuesto es de hecho un subconjunto de Core-GXPath.

Note que en la base de datos actual, ni ψ_1 ni ψ_3 se incumplen, mientras que ψ_2 se incumple en

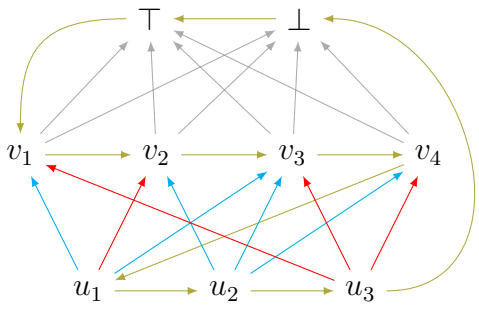

Figura 1.6: Data-grafo G_ϕ para la fórmula $\phi = (p_1 \vee \neg p_2 \vee p_3) \wedge (p_2 \vee p_3 \vee p_4) \wedge (\neg p_1 \vee \neg p_3 \vee \neg p_4)$, omitiendo los valores de datos. Las flechas verdes representan a la relación *aux*, la cual forma un ciclo que pasa por todos los nodos del data-grafo. Las flechas grises representan la relación *assign*. Las flechas rojas y azules indican la codificación de ϕ, con las flechas rojas representando a *neg* y las azules a *pos*.

v_i para todo i, por lo que G_ϕ es inconsistente con respecto a $\mathcal{R}_n$. Una subreparación no vacía de G_ϕ requeriría que para todo i alguna de las relaciones $(v_i, \mathit{assign}, \bot), (v_i, \mathit{assign}, \top)$ se elimine. La Figura 1.7 muestra una reparación no vacía para el G_ϕ de la Figura 1.6.

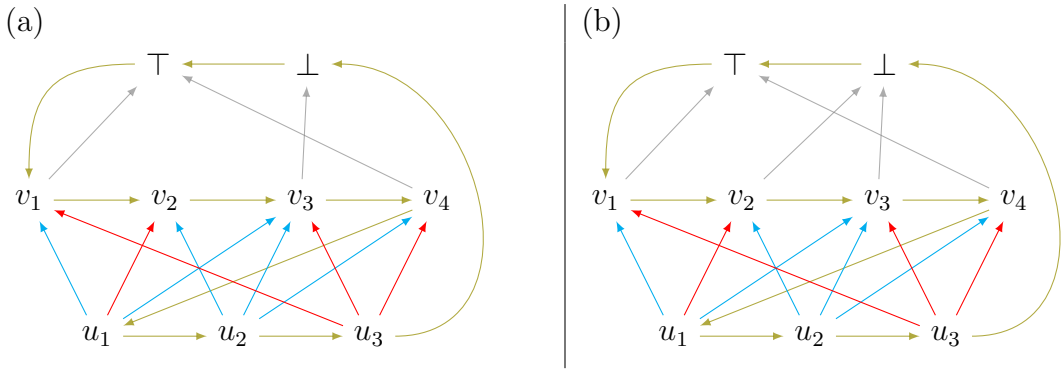

Figura 1.7: Para G_ϕ de la Figura 1.6 estos dibujos representan subdata-grafos, con los valores de datos omitidos. Tanto el data-grafo (a) como (b) son consistentes con respecto a $\mathcal{R}_n$, pero solo (b) es una subreparación y es aquella que está en correspondencia con la valuación $p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1$, mientras que (a) solo está en correspondencia con la valuación parcial $p_1 \mapsto 1, p_3 \mapsto 0, p_4 \mapsto 1$.

Finalizamos la prueba demostrando la siguiente proposición.

Afirmación. *Las siguientes son equivalentes:*

1. *ϕ es satisfacible.*
2. *G_ϕ tiene una subreparación no vacía con respecto a $\mathcal{R}_n$.*

1. implica 2. Sea f una valuación de $\{p_1, \ldots, p_n\}$ tal que $f \models \phi$ y definamos el subdata-grafo (no vacío) $G_{\phi,f} = (V, E')$ donde E' es E salvo las flechas de la forma $(v_i, \mathit{assign}, \bot)$ si $f(p_i) = 1$ y $(v_i, \mathit{assign}, \top)$ si $f(p_i) = 0$.

Dado que mantenemos todas las demás flechas, es fácil ver que $[\![\psi_1]\!]_{G_{\phi,f}} = V$ y $[\![\psi_2]\!]_{G_{\phi,f}} = V$, la primera igualdad porque mantenemos el ciclo sobre V con la etiqueta *aux* y la segunda porque

hemos removido para cada v_i exactamente una de las relaciones $(v_i, \mathit{assign}, \bot)$, $(v_i, \mathit{assign}, \top)$, por lo que ahora ningún nodo incumple la restricción ψ_2.

Para ψ_3, notemos que en cada clásula c_j existe un literal ℓ tal que $f \models \ell$. Si $\ell = p_i$ para algún i, entonces por definición de $G_{\phi,f}$ tenemos el camino $u_j \xrightarrow{pos} v_i \xrightarrow{assign} \top$, de donde $u_j \in [\![\psi_3]\!]_{G_{\phi,f}}$. Si $\ell = \neg p_i$, obtenemos un resultado análogo con el camino $u_j \xrightarrow{neg} v_i \xrightarrow{assign} \bot$.

Como $G_{\phi,f}$ es consistente con respecto a $\mathcal{R}_n$ concluimos que G_ϕ tiene una subreparación no vacía. Omitimos el detalle menor de demostrar que $G_{\phi,f}$ es una subreparación.

2. implica 1. Supongamos que $G' = (V', D', E')$ es una subreparación no vacía de G con respecto a $\mathcal{R}_n$. Necesitamos definir una valuación f sobre el conjunto $\{p_1, \ldots, p_n\}$ tal que $f \models \phi$. Primero veamos que $V' = V$. Dado que $[\![\psi_1]\!]_{G'} = V'$ y $V' \neq \emptyset$, entonces existe al menos una flecha en G' de la forma (u, aux, v), que a su vez tiene que ser parte del ciclo (A). Dado que $v \in [\![\psi_1]\!]_{G'}$ por ser G' consistente, la flecha del ciclo (A) que sigue a (u, aux, v) también tiene que estar en G', obteniendo así que todo el ciclo (A) es parte de G', lo que implica que $V' = V$. En consecuencia, $D' = D$.

Ahora, como V' contiene a todos los elementos v_i y a $\bot, \top$, debemos poder verificar que para todo i, $(v_i, \mathit{assign}, \bot)$ o $(v_i, \mathit{assign}, \top)$ está en E' (más no ambos a la vez por ψ_2). Notemos que esto se debe al hecho particular de que G' es subreparación y no solo a la consistencia con respecto a $\mathcal{R}_n$, como se ilustra en el ejemplo de la Figura 1.7(a). En efecto, supongamos que G' es consistente con respecto a $\mathcal{R}_n$ pero para algún i, ni $(v_i, \mathit{assign}, \bot)$ ni $(v_i, \mathit{assign}, \top)$ están en E'. Definamos el data-grafo $H = (V, D, E'')$, donde $E'' = E' \cup \{(v_i, \mathit{assign}, \bot)\}$. Entonces $G' \subsetneq H \subsetneq G$ y es fácil ver que H es consistente con respecto a $\mathcal{R}_n$, contradiciendo que G' es una subreparación.

Por lo anterior, podemos definir una valuación f sobre $\{p_1, \ldots, p_n\}$ dada por $f(p_i) = 0$ si $(v_i, \mathit{assign}, \bot) \in E'$, o $f(p_i) = 1$ si $(v_i, \mathit{assign}, \top) \in E'$, para todo i. Tenemos que $f \models \phi$ si y solo si $f \models c_j$ para todo j, y esto último sucede si $f \models \ell$ para algún literal ℓ en c_j. Veamos que esto último ocurre para todo j.

Como $u_j \in V'$ y $D'(u_j) = \texttt{clause}$, por ψ_3 debe existir un camino de la forma $u_j \xrightarrow{pos} v_i \xrightarrow{assign} \top$ o $u_j \xrightarrow{neg} v_i \xrightarrow{assign} \bot$ en G', para algún i. Por definición de f y G, si ocurre el primer caso es porque el literal $\ell = p_i$ aparece en c_j y $f(p_i) = 1$, o si ocurre el segundo caso es porque $\ell = \neg p_i$ aparece en c_j y $f(p_i) = 0$, pero en ambos casos $f \models \ell$ como queríamos demostrar. □

En vista de los teoremas anteriores, nos mostramos interesados en saber si se puede replicar el resultado de intratabilidad de ∃Subset Repair para un conjunto de expresiones de nodo positivas. Para responder si esto es posible estudiamos la siguiente *propiedad de monotonía* para el fragmento positivo de Reg-GXPath:

Lema 23. *Sea G un data-grafo, $G \subseteq G'$ un superdata-grafo y η una expresión de* Reg-GXPath^{pos}. *Entonces $[\![\eta]\!]_G \subseteq [\![\eta]\!]_{G'}$.*

Demostración. Procederemos por inducción estructural. Sean $G = (V, D, E)$ y $G' = (V', D', E')$ dos data-grafos, con G' superdata-grafo de G y η una expresión de Reg-GXPath^{pos}.

Los casos bases cuando η es una expresión de nodo son $\mathsf{c}^{=}$ y $\mathsf{c}^{\neq}$ para algún $\mathsf{c} \in \mathbb{D}$. Dado que $D(u) = D'(u)$ para todo $u \in V$, entonces $[\![\mathsf{c}^{=}]\!]_G \subseteq [\![\mathsf{c}^{=}]\!]_{G'}$ y $[\![\mathsf{c}^{\neq}]\!]_G \subseteq [\![\mathsf{c}^{\neq}]\!]_{G'}$.

Los casos bases cuando η es una expresión de camino son ε, $\sqcup$, y a, a^- para todo $a \in \mathbb{A}$. Dado que $V \subseteq V'$ y $E \subseteq E'$, entonces $[\![\varepsilon]\!]_G \subseteq [\![\varepsilon]\!]_{G'}$ y $[\![\sqcup]\!]_G \subseteq [\![\sqcup]\!]_{G'}$. Para una etiqueta a, tenemos que toda flecha (u, a, v) en G tambien es una flecha en G', por lo que $[\![a]\!]_G \subseteq [\![a]\!]_{G'}$ y $[\![a^-]\!]_G \subseteq [\![a^-]\!]_{G'}$.

Para el caso inductivo, supongamos que φ, ψ son expresiones de nodo de Reg-GXPath^{pos}, α, β son expresiones de camino Reg-GXPath^{pos} y que para todas estas expresiones vale la tesis del enunciado. Varias de las igualdades utilizadas a continuación surgen de la Figura 1.2. Empezamos con las expresiones de nodo que involucran los casos cuando η es una de las siguientes fórmulas: (a) $\varphi \wedge \psi$; (b) $\langle\alpha\rangle$; (c) $\langle\alpha = \beta\rangle$; (d) $\langle\alpha \neq \beta\rangle$.

(a) Directo, pues $[\![\varphi \wedge \psi]\!]_G = [\![\varphi]\!]_G \cap [\![\psi]\!]_G$ y $[\![\varphi \wedge \psi]\!]_{G'} = [\![\varphi]\!]_{G'} \cap [\![\psi]\!]_{G'}$.

(b) Sea $u \in [\![\langle\alpha\rangle]\!]_G$. Entonces existe $v \in V$ tal que $(u, v) \in [\![\alpha]\!]_G$. Como $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$ por hipótesis inductiva, obtenemos que $u \in [\![\langle\alpha\rangle]\!]_{G'}$.

(c) Sea $u \in [\![\langle\alpha = \beta\rangle]\!]_G$. Entonces existen $v_1, v_2 \in V$ tales que $(u, v_1) \in [\![\alpha]\!]_G$, $(u, v_2) \in [\![\beta]\!]_G$ y $D(v_1) = D(v_2)$. Como $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$ y $[\![\beta]\!]_G \subseteq [\![\beta]\!]_{G'}$ por hipótesis inductiva, obtenemos que $u \in [\![\langle\alpha = \beta\rangle]\!]_{G'}$.

(d) Análogo al caso anterior.

Cuando η es una expresión de camino estudiamos los siguientes casos: (e) $\alpha \circ \beta$; (f) $\alpha \cup \beta$; (g) α^*; (h) $\alpha^{n,m}$; (i) $\varphi?$.

(e) Sea $(u, v) \in [\![\alpha \circ \beta]\!]_G$. Entonces existe $v_1 \in V$ tal que $(u, v_1) \in [\![\alpha]\!]_G$ y $(v_1, v) \in [\![\beta]\!]_G$. Como $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$ y $[\![\beta]\!]_G \subseteq [\![\beta]\!]_{G'}$ por hipótesis inductiva, obtenemos que $(u, v) \in [\![\alpha \circ \beta]\!]_{G'}$.

(f) Directo, pues $[\![\alpha \cup \beta]\!]_G = [\![\alpha]\!]_G \cup [\![\beta]\!]_G$ y $[\![\alpha \cup \beta]\!]_{G'} = [\![\alpha]\!]_{G'} \cup [\![\beta]\!]_{G'}$.

(g) Sea $(u, v) \in [\![\alpha^*]\!]_G$. Entonces existen nodos $v_1, v_2, \ldots, v_n \in V$ tales que $v_1 = u$, $v_n = v$ y $(v_i, v_{i+1}) \in [\![\alpha]\!]_G$ para todo $1 \leq i \leq n-1$. Como $[\![\alpha]\!]_G \subseteq [\![\alpha]\!]_{G'}$ por hipótesis inductiva, obtenemos que $(u, v) \in [\![\alpha^*]\!]_{G'}$.

(h) Similar al caso anterior.

(i) Como $(u, u) \in [\![\varphi?]\!]_G$ si y solo si $u \in [\![\varphi]\!]_G$ y $[\![\varphi]\!]_G \subseteq [\![\varphi]\!]_{G'}$ por hipótesis inductiva, obtenemos que $[\![\varphi?]\!]_G \subseteq [\![\varphi?]\!]_{G'}$. □

Este hecho nos permite describir un procedimiento efectivo para hallar una subreparación, basado en la siguiente observación: si G no satisface una expresión de nodos positiva φ y u es un nodo de G tal que $u \notin [\![\varphi]\!]_G$, entonces no existe una subreparación de G que satisfaga a φ y que contenga a u. Esto queda formalizado con la siguiente propiedad:

Lema 24. *Sea G un data-grafo, $\mathcal{R}$ un conjunto de restricciones en* Reg-GXPath^{pos} *y v un nodo de G que incumple una expresión de nodos de $\mathcal{R}$. Entonces* $\mathrm{Rep}_{\subseteq}(G, \mathcal{R}) = \mathrm{Rep}_{\subseteq}(G_X, \mathcal{R})$*, donde G_X es el subdata-grafo de G generado por $X = V \setminus \{v\}$, con V el conjunto de nodos de G.*

Demostración. Veamos que $\mathrm{Rep}_{\subseteq}(G, \mathcal{R}) \subseteq \mathrm{Rep}_{\subseteq}(G_X, \mathcal{R})$.

Sea $G' \in \mathrm{Rep}_{\subseteq}(G, \mathcal{R})$. Notemos primero que v no es nodo de G'. Sea $\varphi \in \mathcal{R}$ la expresión de nodo que v incumple y supongamos que $v \in [\![\varphi]\!]_{G'}$. Como φ es una expresión positiva, por el Lema 23, $v \in [\![\varphi]\!]_G$, lo cual es una contradicción. Luego, G' es un subdata-grafo de G_X consistente

con respecto a $\mathcal{R}$. Además, $G' \in \text{Rep}_{\subseteq}(G_X, \mathcal{R})$, de lo contrario, existiría $G' \subsetneq H \subseteq G_X$ consistente con respecto a $\mathcal{R}$, lo que implica que G' no sería reparación de G.

Veamos ahora que $\text{Rep}_{\subseteq}(G_X, \mathcal{R}) \subseteq \text{Rep}_{\subseteq}(G, \mathcal{R})$.

Sea $G' \in \text{Rep}_{\subseteq}(G_X, \mathcal{R})$ con $G' = (V', D', E')$. Sabemos que $V' \subseteq X$. Sea $\varphi \in \mathcal{R}$ la expresión de nodo que v incumple y supongamos que $G' \notin \text{Rep}_{\subseteq}(G, \mathcal{R})$, entonces existe $G' \subsetneq H \subseteq G$ que es consistente con respecto a $\mathcal{R}$. Notemos que por monotonía de φ, v no puede ser nodo de H, es decir, que H también es subdata-grafo de G_X y por lo tanto $G' \notin \text{Rep}_{\subseteq}(G_X, \mathbb{R})$, contradicción. □

Para todo par de subdata-grafos G_1 y G_2 de un mismo data-grafo G vale la siguiente propiedad para toda expresión de nodo φ positiva:

$$[\![\varphi]\!]_{G_1} \cup [\![\varphi]\!]_{G_2} \subseteq [\![\varphi]\!]_{G_1 \cup G_2} \subseteq V_1 \cup V_2,$$

donde V_i es el conjunto de nodos de G_i para $i = 1, 2$. Como consecuencia del Lema 23, si G_1 y G_2 satisfacen φ, entonces $G_1 \cup G_2$ también satisface φ Por lo tanto, si $\mathcal{R}$ solo se compone de expresiones de nodos positivas, concluimos que para todo G hay una *única* subreparación de G con respecto a $\mathcal{R}$. El siguiente algoritmo muestra el proceso para obtener la subreparación bajo las condiciones anteriores.

Algoritmo 1: SUBSET REPAIR para conjuntos de expresiones de nodo positivas

Datos: Un data-grafo $G = (V, D, E)$ y $\mathcal{R}_n \subset \text{Reg-GXPath}^{pos}$ finito
Resultado: Una subreparación de G con respecto a $\mathcal{R}_n$

```
X ← V;
H ← G;
mientras (H, R) sea inconsistente hacer
    Y ← {u ∈ X | u incumple algún φ ∈ R_n};
    X ← X \ Y;
    H ← H_X;
fin
devolver H
```

Este método para computar una subreparación de un data-grafo G dado un conjunto $\mathcal{R}$ de expresiones de nodos positivas es correcto por el Lema 24 y culmina en cualquier entrada $(G, \mathcal{R}_n)$ ya que el data-grafo vacío siempre es consistente. El conjunto X se puede computar en tiempo polinomial y dado que hay a lo sumo $|V|$ iteraciones, concluimos lo siguiente:

Teorema 25. *Dado un data-grafo G y un conjunto $\mathcal{R}$ de expresiones de nodo positivas, es posible computar la única subreparación de G con respecto a $\mathcal{R}$ en tiempo polinomial.*

Note que los Teoremas 21 y 25 establecen un criterio frontera de intratabilidad para el problema ∃SUBSET REPAIR respecto a la clase de fórmulas $\mathcal{L} = \text{Reg-GXPath}^{pos}$, según si se consideran conjuntos de restricciones con o sin expresiones de camino. Concluimos esta sección haciendo notar que aunque el uso de expresiones de camino eleva notablemente la complejidad de ∃SUBSET REPAIR, la clase de expresiones de nodo positivas no deja de ser vastamente expresiva, como se puede observar en [127, Example 4.1].

1.2.2 Superreparaciones

Comenzamos aclarando que a diferencia de lo que sucede con las subreparaciones, no siempre existe una superreparación para cualesquiera G y $\mathcal{R}$, lo cual se puede corroborar con un simple ejemplo. Sea $G = (\{u\}, D, E)$, donde $D(u) = \mathsf{c}$, $E = \emptyset$, y $\mathcal{R}$ que solo tiene a $\varphi := \mathsf{c}^{\neq}$. Como u también es nodo de cualquier superdata-grafo G' de G, entonces $u \notin [\![\varphi]\!]_{G'}$, es decir, no existe un superdata-grafo de G que sea consistente con respecto a $\mathcal{R}$. Note que el conjunto $\mathcal{R}$ consta de una expresión de nodo muy simple perteneciente a Reg-GXPath^{pos}.

Por otra parte, podría darse el caso que un conjunto de reglas induzca una extensión consistente pero infinita (notemos que se evita el uso del término reparación en este caso). Por ejemplo, para el mismo G de antes y $\mathcal{R} = \{\psi, \alpha\}$ con $\psi := \langle \downarrow_r^- \rangle$ y $\alpha := \downarrow_r^* \Rrightarrow \bar{\varepsilon}$, podemos generar la estructura

$$\cdots v_4 \xrightarrow{r} v_3 \xrightarrow{r} v_2 \xrightarrow{r} v_1 \xrightarrow{r} u,$$

que es consistente con respecto a $\mathcal{R}$, pues todo elemento tiene un *r-padre* como pide ψ y no hay *r-ciclos* como pide α.

Para estudiar la complejidad de hallar superreparaciones, definimos los siguientes problemas computacionales:

Superset Repair	
Entrada:	Un data-grafo finito G y un conjunto finito $\mathcal{R}$ de expresiones de $\mathcal{L}$.
Salida:	Una superreparación G' de G con respecto a $\mathcal{R}$.

∃Superset Repair	
Entrada:	Un data-grafo finito G y un conjunto finito $\mathcal{R}$ de expresiones de $\mathcal{L}$.
Salida:	Decidir si G tiene una superreparación G' con respecto a $\mathcal{R}$.

Un problema computacional que usaremos en las próximas demostraciones y que también será relevante en otros capítulos es *el problema de respuestas consistentes a consultas*, denotado CQA por sus siglas en inglés y el cual fue introducido en [19]. En lo que sigue usaremos este problema como una herramienta para obtener la complejidad de nuestros problemas de reparación, pero más adelante en el Capítulo 2 nos dedicaremos a estudiar instancias de CQA en el contexto de bases de conocimiento. Existen muchas versiones de este problema que dependen de la escogencia de un dominio adecuado para los parámetros de entrada, pero en términos muy generales se trata del siguiente planteamiento:

CQA (Consistent Query Answering)	
Entrada:	Una estructura M, un conjunto finito $\Gamma \subset \mathcal{O}$, una consulta $q \in \mathcal{Q}$ de aridad k y una k-tupla $\bar{a}$ de elementos M.
Salida:	Decidir si $\bar{a} \in [\![q]\!]_K$ para todo $K \in \mathrm{Rep}(M, \Gamma)$.

Denotamos $(M, \bar{a}) \in \mathrm{CQA}(q, \Gamma)$ cuando la respuesta de CQA en la entrada $(M, \Gamma, q, \bar{a})$ es afirmativa y $(M, \bar{a}) \notin \mathrm{CQA}(q, \Gamma)$ en caso contrario. En otros contextos, este problema se denomina (en inglés) como *Ontology Mediated Query Answering* (OMQA) o también $\mathcal{Q}$ *Entailment from* $\mathcal{O}$

Knowledge Bases. Aquí el conjunto $\mathcal{O}$ representa un lenguaje de ontologías, tal cual como en este capítulo usamos Reg-GXPath (y sus fragmentos) para ese propósito; $\mathcal{Q}$ representa un lenguaje de consulta; M simboliza una estructura relacional apropiada para ambos lenguajes; $[\![q]\!]_K$ es el conjunto de k-tuplas de elementos de K donde la consulta q se cumple; y $\mathrm{Rep}(M, \Gamma)$ un conjunto de estructuras consistentes con respecto a Γ relacionadas con M y que establecen un criterio de reparación. En muchos casos esos criterios se basan en propiedades conjuntistas y para esta sección nos limitamos a tratar con casos de *superestructuras* de M, es decir, cuando $\mathrm{Rep}(M, \Gamma)$ consta de estructuras K que extienden a M minimalmente, representado como $\mathrm{Rep}_{\supseteq}(M, \Gamma)$. Bajo este criterio, nos referimos al problema CQA más específicamente como Superset CQA.

El formato con el que hemos presentado a CQA es el usual y en algunos ámbitos se clasifica como *certain semantics*, en alusión a la condición de que la tupla $\bar{a}$ debe pertenecer a todo conjunto de respuestas $[\![q]\!]_K$. Una variación de esta condición es pedir que $\bar{a}$ pertenezca a algún conjunto $[\![q]\!]_K$, lo que se denomina como *brave semantics* [43]. Respecto a un lenguaje lo suficientemente expresivo como para capturar el complemento de las consultas q, podemos pensar que los conceptos de *certain semantics* y *brave semantics* son duales. En las siguientes demostraciones trabajaremos realmente con el complemento de CQA.

Análogo al problema de reparaciones, con CQA nos restringimos a estructuras finitas (aunque se podría tener más arbitrariedad como sucederá en el capítulo siguiente). Con pocos recursos sintácticos de Reg-GXPath obtenemos el siguiente resultado negativo.

Teorema 26. *Existe un conjunto finito $\mathcal{R}_p \subset$ Reg-GXPath tal que* ∃Superset Repair *es indecidible cuando se fija $\mathcal{R} = \mathcal{R}_p$ en la entrada.*

Demostración. Construiremos una reducción del problema Superset CQA (una versión tomada de [32]) a ∃Superset Repair. En [32, Theorem 4] prueban que Superset CQA (con $\mathcal{O}$ y $\mathcal{Q}$ descritos más adelante) es indecidible para una escogencia particular de los parámetros q y Γ.

La versión de Superset CQA con la que lidiamos en esta demostración es aquella donde $\mathcal{O}$ es la familia de restricciones de palabra sobre $\mathbb{A}$ (Definición 7) y $\mathcal{Q}$ la familia de RPQs no recursivas.

En [32, Theorem 4] prueban que existe un conjunto finito Γ_1 de restricciones de palabra y una fórmula no recursiva q_1 tales que Superset CQA es indecidible incluso cuando Γ y q se fijan como Γ_1 y q_1 en la entrada. Nuestro propósito ahora es demostrar que para todo grafo $G = (V, E)$ y par $(u, v) \in V^2$ existe un data-grafo $\mathrm{d}(G)$ tal que $(G, (u, v)) \notin \mathrm{CQA}(q_1, \Gamma_1)$ si y solo si $\mathrm{d}(G)$ tiene una superreparación con respecto a $\mathcal{R}_p$, donde $\mathcal{R}_p$ es un conjunto fijo de expresiones de camino que construiremos a partir de q_1 y Γ_1. Por lo tanto, nuestro problema también es indecidible.

Reducción. Sin pérdida de generalidad, supongamos que en $\mathbb{A}$ hay una etiqueta *aux* que no se usa en las expresiones de Γ_1 ni en q_1. Sea $\mathrm{d}(G) = (V, E')$ donde $E' = E \cup \{(u, aux, v)\}$. Note que la etiqueta *aux* sirve solo para identificar al par (u, v) como parte de la entrada del problema CQA.

Restricciones. Como vimos en el Ejemplo 13 cualquier RPQ L se puede entender como una expresión de camino en Reg-GXPath y bajo esta consideración, definamos $\mathcal{R}_p$ como el conjunto de expresiones $\{L_1 \Rrightarrow L_2 \mid L_1 \subseteq L_2 \in \Gamma_1\} \cup \{\downarrow_{aux} \Rrightarrow \overline{q_1}\}$.

Las fórmulas $L_1 \Rrightarrow L_2$ tienen exactamente la misma semántica que $L_1 \subseteq L_2$, la cual expresa que *si entre dos nodos hay un camino con etiqueta L_1 también hay un camino con etiqueta L_2*. La fórmula $\downarrow_{aux} \Rrightarrow \overline{q_1}$ indica que *entre todo par de nodos conectados por un eje con etiqueta aux, no puede haber un camino con etiqueta q_1*.

Finalizamos la prueba demostrando la siguiente proposición.

Afirmación. *Las siguientes son equivalentes:*

1. $(G,(u,v)) \notin \mathrm{CQA}(q_1,\Gamma_1)$.
2. $\mathrm{d}(G)$ *tiene una superreparación con respecto a* $\mathcal{R}_p$.

1. implica 2. Existe un grafo $H \in \mathrm{Rep}_{\supseteq}(G,\Gamma_1)$ tal que $(u,v) \notin [\![q_1]\!]_H$. Sea el data-grafo $\mathrm{d}(H)$ que consta de los mismos nodos y flechas que H además de (u, aux, v). Es claro que $\mathrm{d}(H)$ es un superdata-grafo de $\mathrm{d}(G)$ que satisface todas las fórmulas $L_1 \Rrightarrow L_2$ en $\mathcal{R}_p$. Veamos que también satisface $\downarrow_{aux} \Rrightarrow \overline{q_1}$. En efecto, el único par que podría incumplir esta fórmula es (u,v), pero como $[\![q_1]\!]_H = [\![q_1]\!]_{\mathrm{d}(H)}$, obtenemos la consistencia de $\mathrm{d}(H)$ con respecto a $\mathcal{R}_p$. Más aún, $\mathrm{d}(H)$ es superreparación pero dejamos este hecho sin verificar.

2. implica 1. Sea G' una superreparación de $\mathrm{d}(G)$ con respecto a $\mathcal{R}_p$. Consideremos el grafo H que consta de los mismos nodos y flechas que G' salvo (u, aux, v). De manera similar al caso anterior, tenemos que H es un supergrafo de G que satisface Γ_1. El hecho de que $(u,v) \notin [\![q_1]\!]_H$ se debe a que G' satisface $\downarrow_{aux} \Rrightarrow \overline{q_1}$ y $[\![q_1]\!]_H = [\![q_1]\!]_{G'}$, por lo que $(G,(u,v)) \notin \mathrm{CQA}(q_1,\Gamma_1)$. □

Ahora demostraremos una versión del Teorema 22 para superreparaciones y nuevamente con la ontología restringida a expresiones de nodo, pero esta vez el resultado se cumplirá en complejidad combinada. Se mantiene como problema abierto si la indecibilidad obtenida vale también en data complejidad (en cuyo caso el Teorema 26 sería un corolario trivial), o si por el contrario, para cualquier conjunto de expresiones de nodo $\mathcal{R}$ fijo, ∃Superset Repair es decidible.

Teorema 27. ∃Superset Repair *es indecidible incluso cuando consideramos* $\mathcal{R}$ *variando como conjuntos de expresiones de nodo* $\mathcal{R}_n \subset$ Reg-GXPath.

Demostración. Ahora usaremos otra versión de Superset CQA que aparece en [151][4]. En ese artículo analizan el problema tomando $\mathcal{Q}$ como 2RPQ y $\mathcal{O}$ como $\mathcal{ALCOIF}$ (ver los conceptos de estos lenguajes en Nociones Básicas). La idea es dar una reducción de Superset CQA a ∃Superset Repair, con los lenguajes $\mathcal{Q}$ y $\mathcal{O}$ ya mencionados y usar [151, Corollary 16] donde se establece que esta versión de CQA es indecidible. Este resultado se mantiene incluso considerando solo una cantidad finita de nombres de rol y queries q de la forma $\exists x.x \xrightarrow{L} x$, con L una 2RPQ y $x \in \mathsf{Var}$.[5]

Haremos uso de todas las convenciones establecidos en el preámbulo de este trabajo dadas en Nociones Básicas, por lo que recomendamos tener frescos los conceptos de lógicas de descripción ahí presentados antes de proceder con la lectura de esta larga prueba. Para simplificar algunos detalles técnicos, asumimos que $\mathsf{N_R} \subseteq \mathbb{A}$. Veamos que bajo estas consideraciones, nuestro problema también es indecidible.

Para cada KB $\mathcal{K}$ y consulta q construiremos un data-grafo $G(\mathcal{K})$ y un conjunto de restricciones de nodo $\mathcal{R}(\mathcal{K},q)$ tales que $G(\mathcal{K})$ tiene una superreparación con respecto a $\mathcal{R}(\mathcal{K},q)$ si y solo si hay

[4]Específicamente, usaremos el problema denominado finite 2RPQ entailment from $\mathcal{ALCOIF}$ KBs, renombrado aquí como Superset CQA por consistencia notacional.

[5]Técnicamente, q es una C2RPQ por el uso del cuantificador. Aquí usamos la terminología de [151] en lugar de la establecida en la Definición 4.

un *contra-modelo finito* para $\mathcal{K}$ y q, es decir, un modelo finito de $\mathcal{K}$ que no satisface q. Note que a diferencia de las demostraciones anteriores, el conjunto de restricciones en este caso sí depende de la instancia de Superset CQA a considerar, de ahí que este teorema lo demostraremos para complejidad combinada.

Reducción. El data-grafo $G(\mathcal{K})$ usará solo las etiquetas de eje en $\Sigma_e := \mathsf{N_R} \cup \{total\}$, donde *total* es una etiqueta auxiliar que no está en $\mathsf{N_R}$. Los valores de datos que usaremos son

$$\Sigma_n := \{P_J \mid J \subseteq \mathsf{N_C} \cup \mathsf{N}_{\{\mathsf{I}\}} \text{ y } |J| < \infty\} \dot\cup \{T_J \mid J \subseteq \mathsf{N_C} \cup \mathsf{N}_{\{\mathsf{I}\}} \text{ y } |J| < \infty\}.$$

Intuitivamente, usaremos los valores P_J como una descripción preliminar de los conceptos que aparecen en la ABox de $\mathcal{K}$ y los valores T_J como una descripción total de los conceptos usados en un modelo de $\mathcal{K}$. Note que Σ_n es numerable, pues los índices J se toman como los subconjuntos finitos de un conjunto numerable. Asumimos que contamos con una forma computacional para construir $\Sigma_n \subseteq \mathbb{D}$ dado $\mathcal{K}$. A un nodo que tenga valor P_J lo llamaremos *nodo parcial* y a uno que tenga valor T_J lo llamaremos *nodo total*. Diremos que un nodo con valor de dato P_J o T_J *contiene* el nombre de concepto C si $C \in J$.

En particular, dado un KB $\mathcal{K}$ finito, el par $(G(\mathcal{K}), \mathcal{R}(\mathcal{K}, q))$ se definirá sobre el conjunto de valores de datos $\{P_J \mid J \subseteq cn(\mathcal{K}) \cup nom(\mathcal{K})\} \dot\cup \{T_J \mid J \subseteq cn(\mathcal{K}) \cup nom(\mathcal{K})\}$. En adelante, asumimos que los índices I, J de los valores de datos varían como subconjuntos de $cn(\mathcal{K}) \cup nom(\mathcal{K})$. Sea $\mathcal{K} = (\mathcal{G}, \Gamma)$ y definamos $G(\mathcal{K}) = (V, D, E)$ dado por

- $V := ind(\mathcal{K})$;
- para todo $a \in V$, $D(a) := P_J$, donde $J = \{C \in \mathsf{N_C} \mid C(a) \in \mathcal{G}\} \cup \{\{a\} : \{a\} \in nom(\mathcal{K})\}$;
- $E := \{(a, r, b) \mid r(a, b) \in \mathcal{G}\}$.

En pocas palabras, $G(\mathcal{K})$ es el grafo subyacente de $\mathcal{G}$, excepto que hemos recopilado todos los nombres de conceptos sobre un individuo a en un único valor de dato P_J. Notemos que J incluye a $\{a\}$ solo en caso de que haya una inclusión de conceptos en $\mathcal{K}$ que contenga a este nominal.

Restricciones. Presentamos ahora el conjunto de fórmulas $\mathcal{R}(\mathcal{K}, q)$, que consta de expresiones de nodo denotadas por ψ_i, las cuales presentamos junto con una corta descripción de la propiedad que queremos modelar en cada caso:

(i) La fórmula ψ_1 es la conjunción de las siguientes expresiones:

$$\psi_1^1 := \bigwedge_I (P_I^{=} \Rightarrow \langle \downarrow_{total} \big(\bigvee_{I \subseteq J} T_J^{=} \big) ? \rangle),$$

$$\psi_1^2 := \neg \langle \downarrow_{total}^{-} \downarrow_{total} \cap \bar{\varepsilon} \rangle,$$

$$\psi_1^3 := \langle \downarrow_{total} \rangle \Rightarrow \bigvee_J P_J^{=}.$$

La fórmula ψ_1^1 significa que *cada nodo parcial debe conectarse a un nodo total por medio de la relación total, preservando el valor de dato original*; la fórmula ψ_1^2 implicaría que *todo nodo parcial se conecta precisamente con solo un nodo total*; y la fórmula ψ_1^3 que *toda flecha con etiqueta total debe salir de un nodo parcial*. En conjunto, ψ_1 significa que *total es una asignación de los individuos de* $\mathcal{K}$.

Añadimos los siguientes conceptos y convenciones que serán útiles más adelante. Un nodo total conectado a un nodo parcial por medio de la relación *total* se llama *la contraparte total* del nodo parcial. Por conveniencia, decimos que un eje o flecha que sale [resp. entra] de un nodo parcial también es un eje que sale [resp. entra] de su contraparte total. Dos nodos *están relacionados con el mismo nodo total* si existe un camino entre ellos con etiqueta en

$$\varepsilon^{\mathcal{S}} := (\downarrow_{total} \cup \downarrow^{-}_{total})^{*}. \tag{B}$$

Esta expresión de camino se usará en varias partes de esta prueba.

(ii) Para cada $\varphi := \bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j$ en Γ consideremos:

$$\psi_2^{\varphi} := \neg \bigvee_{\substack{\forall i.\, A_i \in I \\ \forall j.\, B_j \notin I}} T_I^{=},$$

que indica que *todo nodo total que contiene todos los conceptos A_i también debe contener algún concepto B_j*. Así, ψ_2 es la conjunción de todas las fórmulas ψ_2^{φ}.

(iii) Para cada $\varphi := A \sqsubseteq \forall r.B$ en Γ consideremos:

$$\psi_3^{\varphi} := \neg\langle (\bigvee_{A \in I} T_I^{=})?\, \varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}} (\bigvee_{B \notin J} T_J^{=})?\rangle,$$

que indica que *un nodo total que contiene el concepto A no puede conectarse mediante r con otro nodo total que no contenga el concepto B*. La subexpresión $\varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}}$ sirve para solapar todos los casos en los que $\downarrow_r$ es una flecha que entra o sale de un nodo total, como se establece al final del inciso (i). Así, ψ_3 es la conjunción de todas las fórmulas ψ_3^{φ}.

(iv) Para cada $\varphi := A \sqsubseteq \exists r.B$ en Γ consideremos:

$$\psi_4^{\varphi} := \bigvee_{A \in I} T_I^{=} \Rightarrow \langle \varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}} (\bigvee_{B \in J} T_J^{=})?\rangle,$$

que indica que *un nodo total que contiene el concepto A debe conectarse mediante r con otro nodo total que contiene el concepto B*. Así, ψ_4 es la conjunción de todas las fórmulas ψ_4^{φ}.

(v) Para cada $\varphi := A \equiv \{a\}$ en Γ consideremos:

$$\psi_5^{\varphi} := \bigvee_{\{a\} \in I} P_I^{=} \Rightarrow \langle \downarrow_{total} (\bigvee_{A \in J} T_J^{=})?\rangle,$$

$$\psi_6^{\varphi} := \bigvee_{A \in J} T_J^{=} \Rightarrow \langle \downarrow^{-}_{total} (\bigvee_{\{a\} \in I} P_I^{=})?\rangle,$$

de modo que $\psi_5^{\varphi} \wedge \psi_6^{\varphi}$ indica que *un nodo total contiene el concepto A si y solo si es la contraparte total de un nodo parcial que contiene el nominal $\{a\}$*. Así, ψ_5 y ψ_6 son la conjunción de todas las fórmulas ψ_5^{φ} y ψ_6^{φ}, respectivamente.

(vi) Para cada $Fun(r)$ en Γ, consideremos

$$\psi_7^{r} := \neg\langle (\bigvee_{I} T_I^{=})?\, \varepsilon^{\mathcal{S}} \downarrow_r^{-} \varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}} (\bigvee_{I} T_I^{=})? \cap \bar{\varepsilon}\rangle,$$

que dado un modelo $\mathcal{I}$ de $\mathcal{K}$, simula el comportamiento funcional de $r^{\mathcal{I}}$ en el contexto de data-grafos. El problema con esta propiedad radica en que no podemos acotar la cantidad de flechas con etiqueta r que salen de los nodos parciales de un data-grafo, pero sí podemos controlar que por medio de estas flechas se llegue a un mismo nodo total forzando a que ciertos patrones de grafos no sucedan en nuestras estructuras, simulando un comportamiento funcional de la relación r. En la Figura 1.8 se ven algunos de los patrones que no queremos que aparezcan en una reparación de $G(\mathcal{K})$ y cómo la fórmula ψ_7^r lidia con cada caso particular para tratar con la condición de funcionalidad de r en el sentido de data-grafos. Luego, ψ_7 es la conjunción de todos los ψ_7^r.

(vii) Por último, debemos añadir una restricción que asegure la no satisfacibilidad de la consulta q. Para ello necesitamos la siguiente traducción especial que transforma fórmulas L de 2RPQ en fórmulas $L^{\mathcal{S}}$ de Reg-GXPath:

$$\varepsilon \mapsto \varepsilon^{\mathcal{S}},$$
$$r \mapsto \varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}}, \text{ para todo } r \in \Sigma,$$
$$r^- \mapsto \varepsilon^{\mathcal{S}} \downarrow_r^- \varepsilon^{\mathcal{S}}, \text{ para todo } r \in \Sigma,$$
$$L_1 \star L_2 \mapsto L_1^{\mathcal{S}} \star L_2^{\mathcal{S}}, \text{ para } \star \in \{\circ, \cup\},$$
$$L^* \mapsto (L^{\mathcal{S}})^*.$$

Para $q = \exists x.x \xrightarrow{L} x$, añadimos en $\mathcal{R}(\mathcal{K}, q)$ la restricción $\psi_q := \langle \varepsilon \cap \overline{L^{\mathcal{S}}} \rangle$. Note que si eliminamos cada aparición del símbolo $\downarrow_{total}$ de una palabra ω de $L^{\mathcal{S}}$, obtenemos simplemente una palabra de L. La fórmula ψ_q indica que *para cada nodo no puede haber un ciclo que empiece y termine en ese nodo y cuya etiqueta pertenezca a* $L^{\mathcal{S}}$.

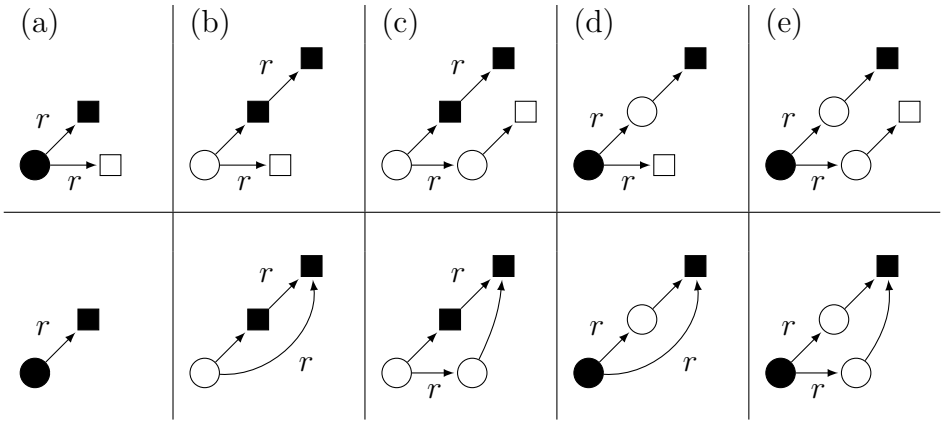

Figura 1.8: En las cajas superiores se muestran algunos patrones que no queremos que aparezcan en una reparación como se especifica en la restricción ψ_7^r. En las cajas inferiores, se muestra cómo arreglar cada problema al unir el nodo problemático □ con un nodo ■. Los nodos se interpretan como sigue: □ nodos totales no deseados; ■ nodos totales; ○ nodos parciales; ● cualquier tipo de nodo. Los ejes sin etiqueta representan ejes con etiqueta *total*, y por lo tanto los nodos □ y ■ son la contraparte total del nodo ○ incidente.

Finalizamos la prueba demostrando la siguiente proposición.

Afirmación. *Las siguientes son equivalentes:*

1. *Existe un contra-modelo $\mathcal{I}$ para $\mathcal{K}$ y q.*
2. *$G(\mathcal{K})$ tiene una superreparación con respecto a $\mathcal{R}(\mathcal{K}, q)$.*

1. implica 2. Sea $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ un contra-modelo finito para $\mathcal{K}$ y q. Definamos el data-grafo $G' = (V', D', E')$, donde $V' := V \cup \Delta^{\mathcal{I}}$, $D'|_V := D$, para todo $u \in \Delta^{\mathcal{I}}$, $D(u) = T_I$ con

$$I = \{C \in cn(\mathcal{K}) \mid u \in C^{\mathcal{I}}\} \cup \{\{a\} \in nom(\mathcal{K}) \mid a^{\mathcal{I}} = u\},$$

y E' es E extendido con las siguientes flechas:

$(a, total, a^{\mathcal{I}})$, para cada $a \in V$, y
(u, r, v), para cada par $u, v \in \Delta^{\mathcal{I}}$ tales que $(u, v) \in r^{\mathcal{I}}$.

Como los elementos $u \in \Delta^{\mathcal{I}}$ se pueden ver como nodos de $\mathcal{I}$ y G' vamos a subrayarlos para distinguir en qué contexto estamos, por ejemplo, $\underline{u}$ expresará $u \in \Delta^{\mathcal{I}}$ *como nodo de* G'. A lo largo de la prueba solo nos interesará interpretar cualquier expresión de nodo o de camino en G', por lo que simplificamos la notación $[\![\cdot]\!]_{G'}$ a la más sencilla $[\![\cdot]\!]$.

Como $\mathcal{I}$ es modelo de $\mathcal{K}$, entonces (1) si $r(a, b)$ es una aserción, entonces ambas flechas (a, r, b) y $(\underline{a}^{\mathcal{I}}, r, \underline{b}^{\mathcal{I}})$ están en E', y (2) si $\{a\}$ es un nominal en $\mathcal{K}$, entonces $[\![\bigvee_{\{a\} \in I} P_I^{=}]\!] = \{a\}$. Estas dos propiedades serán relevantes en un instante.

Demostraremos que G' satisface las expresiones en $\mathcal{R}(\mathcal{K}, q)$ una por una.

(i) Las fórmulas ψ_1^1 y ψ_1^3 se satisfacen por definición de G', mientras que ψ_1^2 se satisface porque la aplicación $\cdot^{\mathcal{I}}$ es funcional sobre el conjunto $V = ind(\mathcal{K})$, por lo que no es posible que exista en G' un camino $u \xrightarrow{total^{-}} a \xrightarrow{total} v$ con $u \neq v$. Luego, $[\![\psi_1]\!] = V'$.

(ii) Supongamos que $\varphi := \bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j$ está en Γ y sea $x \in V'$. Analicemos por casos. Si x es un nodo parcial, entonces $x \in [\![\psi_2^{\varphi}]\!]$. Si $x = \underline{u}$ y no contiene algún concepto A_i, entonces $x \in [\![\psi_2^{\varphi}]\!]$. Si $x = \underline{u}$ y contiene todos los conceptos A_i, para probar que $x \in [\![\psi_2^{\varphi}]\!]$ basta con verificar que x también contiene algún concepto B_j. Sea $D'(x) = T_J$, con $A_i \in J$ para todo i. Por definición de G', $u \in \Delta^{\mathcal{I}}$ y $u \in A_i^{\mathcal{I}}$ para todo i. Dado que $\mathcal{I} \models \Gamma$, tenemos que $u \in B_j^{\mathcal{I}}$ para algún j y por definición de D', $B_j \in J$. Luego, $[\![\psi_2]\!] = V'$.

(iii) Supongamos que $\varphi := A \sqsubseteq \forall r.B$ está en Γ y sea $x \in V'$. Si x es un nodo parcial, o un nodo total que no contiene el concepto A, entonces $x \in [\![\psi_3^{\varphi}]\!]$. Si $x = \underline{u}$ contiene el concepto A pero $\downarrow_r$ no es una flecha que sale de x (ver en inciso (i) el significado de *flecha saliendo*) entonces $x \in [\![\psi_3^{\varphi}]\!]$. El único caso restante es cuando $x = \underline{u}$ contiene el concepto A y hay flechas $\downarrow_r$ que salen de x, para el cual debemos verificar que si x alcanza un nodo y mediante un camino con etiqueta en $\varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}}$, entonces y contiene el concepto B. Note que si y es el nodo final de tal camino, entonces y es de la forma $\underline{v}$.

Dado lo anterior, por definición de G' basta considerar el siguiente par de casos: (a) (x, r, y) es flecha de G'; (b) x alcanza a y por un camino con etiqueta $\downarrow_{total}^{-} \downarrow_r \downarrow_{total}$, por lo que x e y serían las contrapartes totales de ciertos nodos parciales: $x = \underline{a}^{\mathcal{I}}$ y $y = \underline{b}^{\mathcal{I}}$ para $a, b \in V$. El caso (a) ocurre solo si $(u, v) \in r^{\mathcal{I}}$ y dado que $u \in A^{\mathcal{I}}$ y $\mathcal{I} \models \Gamma$, entonces $v \in B^{\mathcal{I}}$, como queríamos. Ahora, note que (b) ocurre solo si $r(a, b)$ es una aserción, en cuyo caso debe suceder que $(a^{\mathcal{I}}, b^{\mathcal{I}}) = (u, v) \in r^{\mathcal{I}}$ y hemos caído en el caso anterior.

(iv) Supongamos que $\varphi := A \sqsubseteq \exists r.B$ está en Γ y sea $x \in V'$. Si x es un nodo parcial, o un nodo total que no contiene el concepto A, entonces $x \in [\![\psi_4^\varphi]\!]$. Si $x = \underline{u}$ contiene el concepto A, para probar que $x \in [\![\psi_4^\varphi]\!]$ debemos verificar que x alcanza un nodo y que contiene el concepto B mediante un camino con etiqueta en $\varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}}$. En efecto, por definición de G', tenemos que $u \in A^{\mathcal{I}}$ y como $\mathcal{I} \models \Gamma$, existe $v \in \Delta^{\mathcal{I}}$ tal que $(u, v) \in r^{\mathcal{I}}$ y $v \in B^{\mathcal{I}}$. Tomando $y = \underline{v}$ obtenemos el resultado.

(v) Supongamos que $\varphi := A \equiv \{a\}$ está en Γ y sea $x \in V'$. Dado que $[\![\bigvee_{\{a\}\in I} P_I^{=}]\!] = \{a\}$ por (2), si $x \neq a$ entonces $x \in [\![\psi_5^\varphi]\!]$. Si $x = a$, solo necesitamos probar que la contraparte total de a, o sea, el nodo $\underline{a}^{\mathcal{I}}$, contiene el concepto A. Pero esto es claro del hecho que $\mathcal{I} \models \Gamma$ y por lo tanto $a^{\mathcal{I}} \in A^{\mathcal{I}}$.

Ahora, con un argumento similar, si x es un nodo parcial, o un nodo total que no contiene el concepto A, entonces $x \in [\![\psi_6^\varphi]\!]$. Pero si x contiene el concepto A, debemos verificar que x es la contraparte total de a. Por definición de G', $x = \underline{u}$ para algún $u \in \Delta^{\mathcal{I}}$, y dado que $u \in A^{\mathcal{I}}$, entonces $u = a^{\mathcal{I}}$, como queríamos.

(vi) Supongamos que $Fun(r)$ está en Γ y sea $x \in V'$. Si x es un nodo parcial o no le entra una flecha r, entonces $x \in [\![\psi_7^r]\!]$. Supongamos que $x = \underline{v}$ y que hay un camino $\underline{v} \xrightarrow{\varepsilon^{\mathcal{S}}\, r^-\, \varepsilon^{\mathcal{S}}\, r\, \varepsilon^{\mathcal{S}}} \underline{w}$ en G'. Entonces $x \in [\![\psi_7^r]\!]$ siempre que $v = w$. Por (1), tenemos que existe un nodo total $\underline{u}$ tal que $\underline{v} \xrightarrow{r^-} \underline{u} \xrightarrow{r} \underline{w}$ es un camino en G', pero esto equivale a tener $(u, v), (u, w) \in r^{\mathcal{I}}$. Como $\mathcal{I} \models \Gamma$, entonces $v = w$.

(vii) Esta parte la probaremos por el contrarrecíproco: si $[\![\psi_q]\!] \neq V'$, entonces $\mathcal{I}$ satisface la consulta q, es decir, hay un ciclo en $\mathcal{I}$ donde se lee una palabra en L.

Primero, notemos que para cada α en 2RPQ, nodos a en $V = ind(\mathcal{K})$ y x en $V' = V \cup \Delta^{\mathcal{I}}$, se tiene por definición del traductor $L^{\mathcal{S}}$ y de G' que

(a) $(a, x) \in [\![\alpha^{\mathcal{S}}]\!]$ si y solo si $(\underline{a}^{\mathcal{I}}, x) \in [\![\alpha^{\mathcal{S}}]\!]$, y

(b) $(x, a) \in [\![\alpha^{\mathcal{S}}]\!]$ si y solo si $(x, \underline{a}^{\mathcal{I}}) \in [\![\alpha^{\mathcal{S}}]\!]$.

Dado que $(\underline{a}^{\mathcal{I}}, r, \underline{b}^{\mathcal{I}})$ es un eje de G' para cada aserción $r(a, b)$ en $\mathcal{K}$, también concluimos que

(c) si $(\underline{u}, \underline{v}) \in [\![\alpha^{\mathcal{S}}]\!]$ entonces $(\underline{u}, \underline{v}) \in [\![\alpha]\!]$.

En otras palabras, si u y v son dos nodos totales conectados por un camino con etiqueta en $\alpha^{\mathcal{S}}$, entonces también están conectados por un camino que no recorre ningún eje $\downarrow_{total}$ y cuya etiqueta está en α. Antes de demostrar esta propiedad, concluimos la prueba de este inciso. Supongamos entonces que $[\![\psi_q]\!] \neq V'$ y sea $x \in V'$ un nodo que incumple ψ_q. Por definición, esto equivale a $(x, x) \in [\![L^{\mathcal{S}}]\!]$. Luego, por (a) y (b), podemos suponer que x es de la forma $\underline{u}$ y por (c) tenemos que $(x, x) \in [\![L]\!]$, lo que implica que en $\mathcal{I}$ hay un ciclo con una palabra en L.

Demostración de (c) Por inducción en la estructura de α. Sean $\underline{u}, \underline{v}$ nodos totales de G' tales que $(\underline{u}, \underline{v}) \in [\![\alpha^{\mathcal{S}}]\!]$.

Si $\alpha = \varepsilon$, entonces $\underline{u} = \underline{v}$ porque $\downarrow_{total}$ es funcional en G', por lo que $(\underline{u}, \underline{v}) \in [\![\alpha]\!]$.

Si $\alpha = \downarrow_r$, hay un par de casos para analizar. Si $\underline{u}$ o $\underline{v}$ no son la contraparte total de nodos parciales, entonces $(\underline{u}, r, \underline{v})$ es un eje de G' por definición de E', de donde $(\underline{u}, \underline{v}) \in [\![\alpha]\!]$. Si ambos son la contraparte total de nodos parciales a, b, entonces por (1), $(\underline{u}, \underline{v}) \in [\![\alpha]\!]$.

Si $\alpha = \downarrow_r^-$, análogo a lo anterior.

Si $\alpha = \alpha_1 \cup \alpha_2$, entonces $(\underline{u}, \underline{v}) \in [\![\alpha_i^{\mathcal{S}}]\!]$ para $i = 1$ o $i = 2$, de donde $(\underline{u}, \underline{v}) \in [\![\alpha_i]\!] \subseteq [\![\alpha]\!]$ por

hipótesis inductiva.

Si $\alpha = \alpha_1 \circ \alpha_2$, entonces $(\underline{u}, y) \in [\![\alpha_1^{\mathcal{S}}]\!]$ y $(y, \underline{v}) \in [\![\alpha_2^{\mathcal{S}}]\!]$ para algún $y \in V'$. Si y es un nodo total, entonces por hipótesis inductiva $(\underline{u}, y) \in [\![\alpha_1]\!]$, $(y, \underline{v}) \in [\![\alpha_2]\!]$, y por lo tanto, $(\underline{u}, \underline{v}) \in [\![\alpha]\!]$. Si y es un nodo parcial, por (a) y (b), $(\underline{u}, y^{\mathcal{I}}) \in [\![\alpha_1]\!]$, $(y^{\mathcal{I}}, \underline{v}) \in [\![\alpha_2]\!]$ y procedemos igual que antes.

Si $\alpha = \alpha_1^*$, procedemos análogamente al caso anterior, lo que concluye esta demostración.

Hemos obtenido que G' es un data-grafo finito que contiene $G(\mathcal{K})$ y satisface $\mathcal{R}(\mathcal{K}.q)$ y por lo tanto contiene una reparación de $G(\mathcal{K})$.

2. implica 1. Sea $G' = (V', D', E')$ una superreparación de $G(\mathcal{K})$ con respecto a $\mathcal{R}(\mathcal{K}, q)$. Vamos a construir un contra-modelo para $\mathcal{K}$ y q aplicando una relación de equivalencia sobre V'. Sea $\sim$ la relación sobre V' definida como: $u \sim v$ si y solo si $(u, v) \in [\![\varepsilon^{\mathcal{S}}]\!]$. Usamos la estrella de Kleene para garantizar que la relación es transitiva y reflexiva, mientras que la simetría es una consecuencia directa de la definición. Esencialmente, estamos colapsando cada nodo parcial con su contraparte total, la cual sabemos que existe pues la fórmula ψ_1 de (i) se satisface. Para cada $u \in V'$, denotamos por $[u]$ a su clase de equivalencia. Consideremos $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, donde $\Delta^{\mathcal{I}} := V/\sim$ y la función $\cdot^{\mathcal{I}}$ se define como sigue:

- $C^{\mathcal{I}} := \{[u] \in \Delta^{\mathcal{I}} \mid \exists J, \exists u' \sim u.\, D'(u') = T_J \text{ y } C \in J\}$ para cada nombre de concepto C;
- $r^{\mathcal{I}} = \{([u], [v]) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists u' \sim u, v' \sim v.\, (u', r, v') \in E'\}$ para cada nombre de rol r;
- $a^{\mathcal{I}} := [a]$ para cada nombre de individuo a.

Note que cada clase $[u]$ contiene exactamente un nodo total y podría contener muchos o ningún elemento de $ind(\mathcal{K})$ y otros nodos parciales agregados en G' al reparar $G(\mathcal{K})$. Denotaremos por $[u]_T$ al nodo total de la clase $[u]$.

Demostraremos que $\mathcal{I}$ es un contra-modelo para $\mathcal{K} = (\mathcal{G}, \Gamma)$ y q. Por definición, se tiene que $a^{\mathcal{I}} \in C^{\mathcal{I}}$ para cada aserción $C(a)$ en $\mathcal{G}$ y $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ para cada aserción $r(a, b)$ en $\mathcal{G}$. Por lo que las condiciones (I1) y (I2) se cumplen. Ahora veremos que (I3) se cumple para todos los casos de fórmulas tipo (D) que podrían estar en Γ.

Supongamos que $\bigsqcap_i A_i \sqsubseteq \bigsqcup_j B_j$ está en Γ y sea $[u] \in \Delta^{\mathcal{I}}$ tal que $[u] \in A_i^{\mathcal{I}}$ para cada i. Para el nodo $u' = [u]_T$ de G', sabemos que $D'(u') = T_J$ con $A_i \in J$ para cada i. Dado que G' satisface la restricción ψ_2 del inciso (ii), se sigue que $B_k \in J$ para algún k, lo que implica que $[u] \in B_k^{\mathcal{I}}$.

Supongamos que $A \sqsubseteq \forall r.B$ está en Γ y sean $[u], [v] \in \Delta^{\mathcal{I}}$ tales que $[u] \in A^{\mathcal{I}}$ y $([u], [v]) \in r^{\mathcal{I}}$. Entonces, $D'([u]_T) = T_I$ con $A \in I$ y existen un par de nodos de G' tales que $u' \sim u, v' \sim v$ y $(u', r, v') \in E'$. Dado que G' satisface ψ_3 del inciso (iii), si $D'(v') = T_J$ para algún índice J entonces B debe estar en J. Si $D'(v') = P_J$, entonces $[u]_T$ y $[v]_T$ están conectados por un camino con etiqueta en $\varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}}$ y por lo tanto $D'([v]_T) = T_{J'}$ con $B \in J'$. En cualquier caso, obtenemos que $[v] \in B^{\mathcal{I}}$.

Supongamos que $A \sqsubseteq \exists r.B$ está en Γ y sea $[u] \in \Delta^{\mathcal{I}}$ tal que $[u] \in A^{\mathcal{I}}$. Entonces $D'([u]_T) = T_I$ con $A \in I$ y dado que G' satisface ψ_4 del inciso (iv), existe un nodo v tal que $D'(v) = T_J$ con $B \in J$ y $[u]_T$ y v están conectados por un camino con etiqueta en $\varepsilon^{\mathcal{S}} \downarrow_r \varepsilon^{\mathcal{S}}$. Esto implica que $([u], [v]) \in r^{\mathcal{I}}$ y $[v] \in B^{\mathcal{I}}$.

Supongamos que $A \equiv \{a\}$ está en Γ y sea $[u] \in \Delta^{\mathcal{I}}$. Si $[u] \in A^{\mathcal{I}}$, entonces $D'([u]_T) = T_J$ con $A \in J$. Como G' satisface ψ_6 del inciso (v), $[u]_T$ es la contraparte total de un nodo parcial con valor de dato P_I tal que $\{a\} \in I$. Pero por definición de G' esto solo ocurre si $[u] = a^{\mathcal{I}}$. Por otra

parte, si $[u] = a^{\mathcal{I}}$, como G' satisface ψ_5 del inciso (v), debe suceder que el valor de dato $[u]_T$ es algún T_J con $A \in J$, lo que implica que $[u] \in A^{\mathcal{I}}$.

Para (I4), supongamos que $Fun(r)$ está en Γ y que $([u],[v]),([u],[w]) \in r^{\mathcal{I}}$. Debemos probar que $[v] = [w]$, o equivalentemente, que $[v]_T = [w]_T$. La Figura 1.8 refleja algunos patrones indeseables que pueden surgir al reparar G pero que la fórmula ψ_7 del inciso (vi) soluciona. Por hipótesis, existen $u_1 \sim u, u_2 \sim u, v_1 \sim [v]_T$ y $w_1 \sim [w]_T$ tales que (u_1, r, v_1) y (u_2, r, w_1) son flechas de G'. En otras palabras, tenemos que en G' existe el siguiente camino:

$$[v]_T \xrightarrow{\varepsilon^{\mathcal{S}}} v_1 \xrightarrow{r^-} u_1 \xrightarrow{\varepsilon^{\mathcal{S}}} u_2 \xrightarrow{r} w_1 \xrightarrow{\varepsilon^{\mathcal{S}}} [w]_T,$$

Como G' satisface ψ_7, este camino solo puede existir si $[v]_T = [w]_T$, como queríamos.

Finalmente, debemos chequear que $\mathcal{I}$ no satisface la consulta q y lo argumentaremos por contrarrecíproco. Supongamos que hay un ciclo en $\mathcal{I}$ donde se lee una palabra en el lenguaje L comenzando y terminando en $[u] \in \Delta^{\mathcal{I}}$. Probaremos que hay un ciclo en G' donde se lee una palabra del lenguaje $L^{\mathcal{S}}$. Más aún, este ciclo contiene al nodo $[u]_T$ y la palabra leída comienza y termina en dicho nodo. Supongamos que el ciclo en $\mathcal{I}$ es la secuencia

$$[u_0] \xrightarrow{r_1} [u_1] \xrightarrow{r_2} [u_2] \cdots [u_{n-1}] \xrightarrow{r_n} [u_n],$$

donde $[u_0] = [u_n] = [u]$ y $r_1 r_2 \cdots r_n \in L$. Por definición de $\mathcal{I}$,

$$[u_0]_T \xrightarrow{r_1^{\mathcal{S}}} [u_1]_T \xrightarrow{r_2^{\mathcal{S}}} [u_2]_T \cdots [u_{n-1}]_T \xrightarrow{r_n^{\mathcal{S}}} [u_n]_T,$$

es un camino en G', que es de hecho un ciclo pues $[u_0]_T = [u_n]_T$. Dado que $r_1^{\mathcal{S}} r_2^{\mathcal{S}} \cdots r_n^{\mathcal{S}}$ es una palabra en $L^{\mathcal{S}}$, G' no puede satisfacer ψ_q del inciso (vii) ya que $([u]_T, [u]_T) \in [\![L^{\mathcal{S}}]\!]$. □

Como se dijo al comienzo de esta sección, algunos conjuntos muy simples de restricciones podrían impedir la existencia de superreparaciones. Los Teoremas 26 y 27 indican que debemos imponer restricciones importantes de Reg-GXPath para obtener resultados de decibilidad. Ahora analizaremos qué sucede cuando nos limitamos a expresiones positivas.

Definición 28. Sea η una expresión de Reg-GXPath. Definimos el conjunto de valores de datos presentes en η, denotado $\mathbb{D}^{\eta}$, como el conjunto de los $\mathtt{c} \in \mathbb{D}$ tales que $\mathtt{c}^{=}$ o $\mathtt{c}^{\neq}$ es usado en η. Dado un conjunto $\mathcal{R}$ de expresiones, definimos $\mathbb{D}^{\mathcal{R}} := \bigcup_{\eta \in \mathcal{R}} \mathbb{D}^{\eta}$.

Si bien la noción de superreparación podría generar la aparición arbitraria de nuevos valores de datos, el siguiente lema garantiza que bajo restricciones en Reg-GXPath^{pos} podemos establecer algún tipo de control.

Lema 29. *Sea $G = (V, D, E)$ un data-grafo y $\mathcal{R}$ un conjunto de expresiones en* Reg-GXPath^{pos}*. Si hay una superreparación G' de G con respecto a $\mathcal{R}$, entonces hay una superreparación de G que solo usa valores de datos en $\mathbb{D}^{\mathcal{R}} \cup \mathrm{Im}(D) \cup \{\mathtt{c}_1, \mathtt{c}_2\}$, con $\mathtt{c}_1, \mathtt{c}_2$ valores de datos frescos.*

Demostración. Sea $G' = (V', D', E')$ una superreparación de G con respecto a $\mathcal{R}$. Vamos a asumir que en G' aparecen al menos tres valores de datos $\mathtt{c}_1, \mathtt{c}_2, \mathtt{d}$ que no están en $\mathbb{D}^{\mathcal{R}} \cup \mathrm{Im}(D)$, pues de lo contrario G' sería la superreparación del enunciado. Dividamos el conjunto V' en dos partes: V_s que contiene todos los nodos con valores de datos en $\mathbb{D}^{\mathcal{R}} \cup \mathrm{Im}(D) \cup \{\mathtt{c}_1, \mathtt{c}_2\}$ y V_d que contiene

los nodos con el resto de valores de datos usados en G'. Definimos a su vez $V_d^i = \{v^i \mid v \in V_d\}$ con $i = 1, 2$ que serán simplemente dos copias de V_d disjuntas entre sí y disjuntas con V_s.

Sea $H = (V_H, D_H, E_H)$ donde $V_H := V_s \cup V_d^1 \cup V_d^2$; $D_H|_{V_s} := D'$ y $D_H(v^i) := \mathtt{c}_i$ para $v^i \in V_d^i$; y E_H se obtiene a partir de E al substituir todas las flechas de la forma $(u, a, v), (v, a, u)$ y (v, a, w) con $v, w \in V_d$ y $u \in V_s$ por $(u, a, v^i), (v^i, a, u)$ y (v^i, a, w^j), respectivamente y para $i, j = 1, 2$. Para ver que H es una superreparación usaremos la siguiente propiedad cuya demostración se consigue más adelante.

Lema 30. *Para toda expresión η de* Reg-GXPath^{pos} *que solo usa valores de datos en $\mathbb{D}^{\mathcal{R}} \cup \mathrm{Im}(D)$, y cualesquiera $u, u' \in V_s$, $v, w \in V_d$ e $i, j = 1, 2$, se cumplen las siguientes implicaciones:*

0. *si $u \in [\![\eta]\!]_{G'}$ [o $(u, u') \in [\![\eta]\!]_{G'}$], entonces $u \in [\![\eta]\!]_H$ [resp. $(u, u') \in [\![\eta]\!]_H$];*
1. *si $v \in [\![\eta]\!]_{G'}$ entonces $v^i \in [\![\eta]\!]_H$;*
2. *si $(u, v) \in [\![\eta]\!]_{G'}$ entonces $(u, v^i) \in [\![\eta]\!]_H$;*
3. *si $(v, u) \in [\![\eta]\!]_{G'}$ entonces $(v^i, u) \in [\![\eta]\!]_H$;*
4. *si $(v, w) \in [\![\eta]\!]_{G'}$ entonces $(v^i, w^j) \in [\![\eta]\!]_H$.*

Sea φ una expresión de nodo en $\mathcal{R}$. Como $[\![\varphi]\!]_{G'} = V'$, por el lema anterior, tenemos que si $u \in V_s$ entonces $u \in [\![\varphi]\!]_H$, y si $v \in V_d$ entonces $v^1, v^2 \in [\![\varphi]\!]_H$, de donde $[\![\varphi]\!]_H = V_H$. Un argumento similar sirve para demostrar que si α es un expresión de camino en $\mathcal{R}$, entonces $[\![\alpha]\!]_H = V_H \times V_H$. □

Demostración del Lema 30. Procederemos por inducción estructural para demostrar las cinco propiedades simultáneamente. Los casos bases cuando η es una expresión de nodo son $\mathtt{d}^{=}$ y $\mathtt{d}^{\neq}$ para algún $\mathtt{d} \in \mathbb{D}^{\mathcal{R}}$ y es claro que $u \in [\![\eta]\!]_{G'}$ [o $v \in [\![\eta]\!]_{G'}$] si y solo si $u \in [\![\eta]\!]_H$ [resp. $v^i \in [\![\eta]\!]_H$]. Los casos bases cuando η es una expresión de camino son $\varepsilon, \sqcup$, y a, a^- para todo $a \in \mathbb{A}$. Para $\eta = \varepsilon$ es trivial y para los casos restantes se cumple directamente la propiedad por definición de H.

Para el caso inductivo, supongamos que φ, ψ son expresiones de nodo de Reg-GXPath^{pos}, α, β son expresiones de camino de Reg-GXPath^{pos} y que para todas estas expresiones vale la tesis de la afirmación. Empezamos con las expresiones de nodo que involucran los casos cuando η es una de las siguientes fórmulas: (a) $\varphi \wedge \psi$; (b) $\langle\alpha\rangle$; (c) $\langle\alpha = \beta\rangle$; (d) $\langle\alpha \neq \beta\rangle$. En lo que sigue haremos la demostración para el nodo $v \in V_d$, pues para $u \in V_s$ se usa un argumento similar.

(a) Directo, pues $[\![\varphi \wedge \psi]\!]_{G'} = [\![\varphi]\!]_{G'} \cap [\![\psi]\!]_{G'}$ y $[\![\varphi \wedge \psi]\!]_H = [\![\varphi]\!]_H \cap [\![\psi]\!]_H$.

(b) Si $v \in [\![\langle\alpha\rangle]\!]_{G'}$, existe $v_1 \in V'$ tal que $(v, v_1) \in [\![\alpha]\!]_{G'}$. Si $v_1 \in V_s$, por hipótesis inductiva ítem 3 tenemos que $(v^i, v_1) \in [\![\alpha]\!]_H$ y por lo tanto $v^i \in [\![\langle\alpha\rangle]\!]_H$. Análogo si $v_1 \in V_d$ aplicando el ítem 4.

(c) Si $v \in [\![\langle\alpha = \beta\rangle]\!]_{G'}$, entonces existen $v_1, v_2 \in V$ tales que $(v, v_1) \in [\![\alpha]\!]_{G'}$, $(v, v_2) \in [\![\beta]\!]_{G'}$ y $D'(v_1) = D'(v_2)$. Si $v_1, v_2 \in V_s$, entonces $D_H(v_1) = D_H(v_2)$ y por hipótesis inductiva ítem 3 tenemos que $(v^i, v_1) \in [\![\alpha]\!]_H$ y $(v^i, v_2) \in [\![\beta]\!]_H$. Los casos $v_1 \in V_s$, $v_2 \in V_d$ y $v_1 \in V_d$, $v_2 \in V_s$ son imposibles. Si $v_1, v_2 \in V_d$, entonces $D_H(v_1^j) = D_H(v_2^j) = \mathtt{c}_j$ y por hipótesis inductiva ítem 4 tenemos que $(v^i, v_1^j) \in [\![\alpha]\!]_H$ y $(v^i, v_2^j) \in [\![\beta]\!]_H$. En cualquier caso posible siempre obtenemos que $v^i \in [\![\langle\alpha = \beta\rangle]\!]_H$.

(d) Si $v \in [\![\langle\alpha \neq \beta\rangle]\!]_{G'}$, entonces existen $v_1, v_2 \in V$ tales que $(v, v_1) \in [\![\alpha]\!]_{G'}$, $(v, v_2) \in [\![\beta]\!]_{G'}$ y $D'(v_1) \neq D'(v_2)$. Si $v_1, v_2 \in V_s$, entonces $D_H(v_1) \neq D_H(v_2)$ y por hipótesis inductiva ítem 3

obtenemos el resultado. Si $v_1, v_2 \in V_d$, entonces $D_H(v_1^1) \neq D_H(v_2^2)$ y por hipótesis inductiva ítem 4 obtenemos el resultado. La situación más interesante ahora es la posibilidad de que $v_1 \in V_s$ y $v_2 \in V_d$ (o vicecersa), pues podría suceder que $D'(v_1) = \mathsf{c}_j$ con $j = 1, 2$ y tendríamos que aplicar la hipótesis inductiva con mayor conveniencia. Si $D'(v_1) = \mathsf{c}_1$ consideremos el nodo v_2^2 de H, de modo que $D_H(v_2^2) = \mathsf{c}_2 \neq \mathsf{c}_1 = D_H(v_1)$. Por hipótesis inductiva ítems 3 y 4 obtenemos respectivamente que $(v^i, v_1) \in [\![\alpha]\!]_H$ y $(v^i, v_2^2) \in [\![\beta]\!]_H$. Análogo para el resto de combinaciones posibles. En todos los casos siempre obtenemos que $v^i \in [\![\langle \alpha \neq \beta \rangle]\!]_H$.

Cuando η es una expresión de camino estudiamos los siguientes casos: (e) $\alpha \circ \beta$; (f) $\alpha \cup \beta$; (g) α^*; (h) $\alpha^{n,m}$; (i) $\varphi?$. En lo que sigue haremos la demostración para el par (u, v) con $u \in V_s$ y $v \in V_d$, excepto para la última expresión, donde esta escogencia no tiene sentido. Para el resto de casos se sigue un argumento similar.

(e) Si $(u, v) \in [\![\alpha \circ \beta]\!]_{G'}$, entonces existe $v_1 \in V'$ tal que $(u, v_1) \in [\![\alpha]\!]_{G'}$ y $(v_1, v) \in [\![\beta]\!]_{G'}$. Si $v_1 \in V_s$, por hipótesis inductiva ítems 0 y 2 obtenemos que $(u, v_1) \in [\![\alpha]\!]_H$ y $(v_1, v^i) \in [\![\beta]\!]_H$, de donde, $(u, v^i) \in [\![\alpha \circ \beta]\!]_H$. Si $v_1 \in V_d$, se obtiene el mismo resultado aplicando hipótesis inductiva ítems 2 y 4.

(f) Directo, pues $[\![\alpha \cup \beta]\!]_{G'} = [\![\alpha]\!]_{G'} \cup [\![\beta]\!]_{G'}$ y $[\![\alpha \cup \beta]\!]_H = [\![\alpha]\!]_H \cup [\![\beta]\!]_H$.

(g) Si $(u, v) \in [\![\alpha^*]\!]_{G'}$, entonces existe $n \geq 1$ tal que $(u, v) \in [\![\alpha^n]\!]_{G'}$ (note que descartamos el caso $n = 0$ por el simple hecho de que $u \neq v$ por hipótesis). Vamos a demostrar que para α^n vale la afirmación, lo cual haremos por inducción en $n \geq 1$. Para $n = 1$ estamos en la situación $(u, v) \in [\![\alpha]\!]_{G'}$, de donde $(u, v^i) \in [\![\alpha]\!]_H$ por hipótesis inductiva ítem 2. Supongamos que para $n = k$ vale la afirmación y que $(u, v) \in [\![\alpha^{k+1}]\!]_{G'}$. Como $\alpha^{k+1} = \alpha \circ \alpha^k$, con la hipótesis inductiva estamos en la situación del ítem (e), de donde $(u, v^i) \in [\![\alpha^{k+1}]\!]_H$ y por lo tanto $(u, v^i) \in [\![\alpha^*]\!]_H$.

(h) Similar al caso anterior.

(i) Como $(x, y) \in [\![\varphi?]\!]_{G'}$ si y solo si $x = y$ y $x \in [\![\varphi]\!]_{G'}$, en este caso vamos a demostrar la propiedad cuando $x = u$ y $x = v$, pero estos salen aplicando directamente hipótesis inductiva ítems 0 y 1, respectivamente.

□

El Lema 29 no garantiza que la superreparación H obtenida será de tamaño polinomial con respecto a $|G|$ y $|\mathcal{R}|$, pues su tamaño depende directamente de $|G'|$, pero podemos refinar el proceso para obtener algo más deseable.

Lema 31. *Sea $G = (V, D, E)$ un data-grafo y V_c un conjunto de nodos de G con valor de dato c y sea v_c un nodo nuevo. Consideremos el data-grafo $H = (V', D', E')$ donde $V' := (V \setminus V_\mathsf{c}) \cup \{v_\mathsf{c}\}$; $D'|_{V \setminus V_\mathsf{c}} := D$ y $D'(v_\mathsf{c}) := \mathsf{c}$; y E' se obtiene a partir de E al substituir todas las flechas de la forma $(u, a, v), (v, a, u)$ y (v, a, v') con $u \in V \setminus V_\mathsf{c}$ y $v, v' \in V_\mathsf{c}$ por $(u, a, v_\mathsf{c}), (v_\mathsf{c}, a, u)$ y $(v_\mathsf{c}, a, v_\mathsf{c})$, respectivamente. Para toda expresión η de* Reg-GXPath^{pos}*, si $G \models \eta$, entonces $H \models \eta$.*

Solo como intuición gráfica antes de dar la demostración, note que H es el colapso de todos los nodos en V_c a un nuevo representante v_c, preservando todas las relaciones ya existentes en G. A esta proceso lo denominamos contracción. La Figura 1.9 muestra un ejemplo de tal proceso. Note

que el recíproco del Lema 31 no es cierto, pues en este gráfico se ve que la contracción satisface la expresión de camino positiva $\alpha := \sqcup$ mientras que el data-grafo original no.

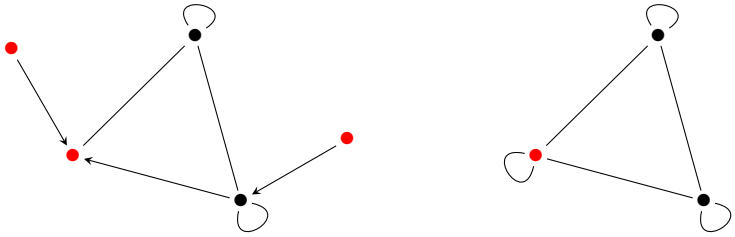

Figura 1.9: A la izquierda un data-grafo donde los nodos rojos representan que tienen el mismo valor de dato. Los ejes sin flecha representan simetría. Del lado derecho la *contracción* de todos los nodos rojos. Note que este nuevo data-grafo es reflexivo y simétrico.

Demostración del Lema 31. Vamos a demostrar primero algo un poco más general.

Afirmación. *Sea* $h : V \to V'$ *dada por* $h(u) := u$ *para todo* $u \in V \setminus V_{\mathtt{c}}$ *y* $h(u) := v_{\mathtt{c}}$ *para todo* $u \in V_{\mathtt{c}}$. *Para* η *en* Reg-GXPath^{pos}, *verifiquemos que si* $u \in [\![\eta]\!]_G$ [*o* $(u, v) \in [\![\eta]\!]_G$], *entonces* $h(u) \in [\![\eta]\!]_H$ [*resp.* $(h(u), h(v)) \in [\![\eta]\!]_H$].

Demostración. Procederemos por inducción estructural. Los casos bases cuando η es una expresión de nodo son $\mathtt{d}^{=}$ y $\mathtt{d}^{\neq}$ para algún $\mathtt{d} \in \mathbb{D}$ y es claro que $u \in [\![\eta]\!]_G$ si y solo si $h(u) \in [\![\eta]\!]_H$.

Los casos bases cuando η es una expresión de camino son $\varepsilon, \sqcup$, y a, a^- para todo $a \in \mathbb{A}$. Cuando $\eta = \varepsilon$ es trivial y los casos restantes cumplen la propiedad por definición de H.

Para el caso inductivo, supongamos que φ, ψ son expresiones de nodo de Reg-GXPath^{pos}, α, β son expresiones de camino de Reg-GXPath^{pos} y que para todas estas expresiones vale la tesis del enunciado. Empezamos con las expresiones de nodo que involucran los casos cuando η es una de las siguientes fórmulas: (a) $\varphi \wedge \psi$; (b) $\langle\alpha\rangle$; (c) $\langle\alpha = \beta\rangle$; (d) $\langle\alpha \neq \beta\rangle$.

(a) Directo, pues $[\![\varphi \wedge \psi]\!]_G = [\![\varphi]\!]_G \cap [\![\psi]\!]_G$ y $[\![\varphi \wedge \psi]\!]_H = [\![\varphi]\!]_H \cap [\![\psi]\!]_H$.

(b) Si $u \in [\![\langle\alpha\rangle]\!]_G$, existe $v \in V$ tal que $(u, v) \in [\![\alpha]\!]_G$. Por hipótesis inductiva, $(h(u), h(v)) \in [\![\alpha]\!]_H$, de donde, $h(u) \in [\![\langle\alpha\rangle]\!]_H$.

(c) Si $u \in [\![\langle\alpha = \beta\rangle]\!]_G$, entonces existen $v_1, v_2 \in V$ tales que $(u, v_1) \in [\![\alpha]\!]_G$, $(u, v_2) \in [\![\beta]\!]_G$ y $D(v_1) = D(v_2)$. Por hipótesis inductiva y definición de H tenemos que $(h(u), h(v_1)) \in [\![\alpha]\!]_H$, $(h(u), h(v_2)) \in [\![\beta]\!]_H$ y $D'(h(v_1)) = D'(h(v_2))$, de donde $h(u) \in [\![\langle\alpha = \beta\rangle]\!]_H$.

(d) Análogo al caso anterior.

Cuando η es una expresión de camino estudiamos los siguientes casos: (e) $\alpha \circ \beta$; (f) $\alpha \cup \beta$; (g) α^*; (h) $\alpha^{n,m}$; (i) $\varphi?$.

(e) Si $(u, v) \in [\![\alpha \circ \beta]\!]_G$, entonces existe $v_1 \in V$ tal que $(u, v_1) \in [\![\alpha]\!]_G$ y $(v_1, v) \in [\![\beta]\!]_G$. Por hipótesis inductiva, $(h(u), h(v_1)) \in [\![\alpha]\!]_H$ y $(h(v_1), h(v)) \in [\![\beta]\!]_H$, de donde $(h(u), h(v)) \in [\![\alpha \circ \beta]\!]_H$.

(f) Directo, pues $[\![\alpha \cup \beta]\!]_G = [\![\alpha]\!]_G \cup [\![\beta]\!]_G$ y $[\![\alpha \cup \beta]\!]_H = [\![\alpha]\!]_H \cup [\![\beta]\!]_H$.

(g) Si $(u, v) \in [\![\alpha^*]\!]_G$, entonces existen nodos $v_1, v_2, \ldots, v_n \in V$ tales que $v_1 = u$, $v_n = v$ y $(v_i, v_{i+1}) \in [\![\alpha]\!]_G$ para todo $1 \leq i \leq n-1$. Por hipótesis inductiva, $(h(v_i), h(v_{i+1})) \in [\![\alpha]\!]_H$ para todo $1 \leq i \leq n-1$. De donde $(h(u), h(v)) \in [\![\alpha^*]\!]_H$.

(h) Similar al caso anterior.

(i) Como $(u, u) \in [\![\varphi?]\!]_G$ si y solo si $u \in [\![\varphi]\!]_G$, por hipótesis inductiva, $h(u) \in [\![\varphi]\!]_H$, de donde $(h(u), h(u)) \in [\![\varphi?]\!]_H$.

Continuación de la demostración del Lema 31. Ahora, si η es una expresión de nodo y G satisface η, entonces $[\![\eta]\!]_G = V$ y por la afirmación anterior, $h(u) \in [\![\eta]\!]_H$ para todo $u \in V$. Luego, por sobreyectividad de h, obtenemos que H satisface η. Un argumento similar sirve cuando η es una expresión de camino. □

La afirmación demostrada en el lema anterior se puede adaptar para obtener que Reg-GXPath^{pos} *es invariante vía homomorfismos de data-grafos*, que es una propiedad interesante desde el punto de vista de la Teoría de Modelos. Con ayuda de estos lemas, probaremos el siguiente hecho importante.

Teorema 32. *Sea $G = (V, D, E)$ un data-grafo y $\mathcal{R}$ un conjunto de expresiones en* Reg-GXPath^{pos}*. Si hay una superreparación de G con respecto a $\mathcal{R}$, entonces hay una superreparación cuyo tamaño depende polinomialmente de $|G|$ y $|\mathcal{R}|$.*

Demostración. Sea $G' = (V', D', E')$ una superreparación de G con respecto a $\mathcal{R}$ que usa a lo sumo $|\mathbb{D}^{\mathcal{R}}| + |V| + 2$ valores de dato, el cual existe por Lema 29.

Para todo $\mathsf{c} \in \mathrm{Im}(D')$, sea $V_{\mathsf{c}} = \{v \in V' \setminus V \mid D'(v) = \mathsf{c}\}$ y sea H el data-grafo que se obtiene de contraer todos los conjuntos V_{c}. Por el Lema 31, H satisface $\mathcal{R}$. H es un superdata-grafo de G, por lo que debe contener una superreparación, así que basta ver que su tamaño está acotado polinomialmente en función de $|G|$ y $|\mathcal{R}|$ para demostrar el teorema.

Notemos que H también usa a lo sumo $|\mathbb{D}^{\mathcal{R}}| + |V| + 2$ valores de dato y, de hecho, aparte de los nodos de G, H tiene a lo sumo un nodo por cada valor de dato, de esta manera el tamaño de su dominio está acotado por $k = |\mathbb{D}^{\mathcal{R}}| + 2|V| + 2$. El tamaño máximo que puede tener H es entonces $O(k^2)$. Como en general, $|\mathbb{D}^{\mathcal{R}}| \leq |\mathcal{R}|$ y $|V| \leq |G|$ entonces $|H| = O(p(|G|, |\mathcal{R}|))$ para algún polinomio en dos variables p. □

Este teorema muestra que ∃Superset Repair está en NP cuando nos restringimos a expresiones de Reg-GXPath^{pos}, pues podemos adivinar una extensión consistente de tamaño a lo sumo polinomial en el tamaño de la entrada considerando solo dos símbolos de valor de datos nuevos, como se estableció en la demostración anterior. Más aún, si nos restringimos a expresiones de nodo, podemos adaptar el Algoritmo 1 para el caso de superreparaciones. Para ello, consideremos las siguientes definiciones. Sea $\Sigma \subset \mathbb{A}$ finito, $G = (V, D, E)$ un data-grafo finito, $\mathcal{R}$ un conjunto de restricciones de nodo en Reg-GXPath^{pos}, ambos definidos sobre $\Sigma \cup \mathbb{D}$ y $\mathsf{c}_1, \mathsf{c}_2$ un par de valores de datos que no están en $\mathbb{D}^{\mathcal{R}} \cup \mathrm{Im}(D)$, que serán los valores de datos extras requeridos para hallar una posible superreparación de G. Consideremos a parte un conjunto de nodos $V_1 = \{v_{\mathsf{c}} \mid \mathsf{c} \in \mathbb{D}^{\mathcal{R}} \dot{\cup} \{\mathsf{c}_1, \mathsf{c}_2\}\}$ que siempre se puede definir de modo que $V_1 \cap V = \emptyset$, y sea $D_1 : V_1 \to \mathbb{D}$ la función dada por $D_1(v_{\mathsf{c}}) = \mathsf{c}$. Sea el data-grafo $H = (V_H, D_H, E_H)$ que extiende a G de la siguiente manera:

- $V_H := V \dot\cup V_1$;
- para todo $u \in V_H$, $D_H(u) := D(u)$ si $u \in V$, y $D_H(u) := D_1(u)$ si $u \in V_1$;
- $E_H := \{(u, r, v) \mid u, v \in V_H,\ r \in \Sigma\}$.

Al grafo H lo denotamos complete$(G, \mathcal{R})$ y es claro que se puede construir en tiempo polinomial en el tamaño de G y $\mathcal{R}$. Todo el proceso de construcción de V_1 y complete$(G, \mathcal{R})$ asumimos que lo tenemos descrito con macros que usaremos en el Algoritmo 2. Por Teorema 32, si existe una superreparación de G con respecto a $\mathcal{R}$ entonces existe un subdata-grafo de H que es superreparación de G.

Algoritmo 2: ∃Superset Repair para conjuntos de expresiones de nodo positivas

Datos: Un data-grafo $G = (V, D, E)$ y $\mathcal{R}_n \subset$ Reg-GXPath^{pos} finito
Resultado: ¿Existe una superreparación de G con respecto a $\mathcal{R}_n$?

```
1  X ← V ∪̇ V_1;
2  H ← complete(G, R_n);
3  mientras (H, R_n) sea inconsistente hacer
4  |  Y ← {u ∈ X | u incumple algún φ ∈ R_n};
5  |  si Y ∩ V ≠ ∅ entonces
6  |  |  devolver No
7  |  fin
8  |  X ← X \ Y;
9  |  H ← H_X;
10 fin
11 devolver Sí
```

El Algoritmo 2 busca una subreparación de $H =$ complete$(G, \mathcal{R}_n)$ con respecto a $\mathcal{R}_n$ que contenga a G. Recordemos que respecto a expresiones de nodo, un data-grafo inconsistente siempre tiene una única subreparación, y este algoritmo hace lo mismo que el Algoritmo 1 en la entrada $(H, \mathcal{R}_n)$, salvo que en cada iteración verifica que no se estén eliminando elementos de G. El algoritmo es correcto por monotonía de Reg-GXPath^{pos} (Lema 23), y porque cualquier superreparación de G contenida en H debe estar contenida en la (única) subreparación de H. Notemos que a diferencia del Algoritmo 1 que directamente resuelve la versión funcional del problema de subreparación, en este caso simplemente respondemos la versión de decisión del problema de superreparación, porque con respecto a G no hay necesariamente determinismo de superreparación (dicho de otra forma, pueden haber múltiples superreparaciones de G, todas contenidas en la subreparación de H). De esta manera obtenemos el siguiente resultado.

Teorema 33. *Existe un algoritmo polinomial que responde si un data-grafo G tiene una superreparación con respecto a un conjunto $\mathcal{R}$ de expresiones de nodo positivas.*

A pesar de lo anterior expuesto, el problema ∃Superset Repair resulta intratable en general.

Teorema 34. ∃Superset Repair *respecto a expresiones positivas es* NP-completo *incluso en complejidad de expresión, es decir, cuando en la entrada $(G, \mathcal{R})$ se considera algún G fijo y $\mathcal{R}$ variando en* Reg-GXPath^{pos}.

Demostración. Construiremos una reducción de 3SAT a ∃Superset Repair. Dada una fórmula proposicional ϕ en 3CNF de n variables $x_1, \ldots, x_n$ y m cláusulas $c_1, \ldots, c_m$, queremos armar un data-grafo $G_\phi = (V, D, E)$ y un conjunto de expresiones $\mathcal{R}_\phi$ tales que G_ϕ tiene una superreparación con respecto a $\mathcal{R}_\phi$ si y solo si ϕ es satisfacible.

Para esta reducción usaremos solo una etiqueta de $\mathbb{A}$ que denotamos simplemente como $\downarrow$ y valores de datos $\{\mathtt{p}_i \mid i \in \mathbb{N}\} \dot\cup \{\mathtt{n}_i \mid i \in \mathbb{N}\} \dot\cup \{\mathtt{aux}\} \subset \mathbb{D}$. Un nodo con valor de dato $\mathtt{p}_i$ representa al literal x_i y con valor de dato $\mathtt{n}_i$ representa al literal $\neg x_i$.

Reducción. Para cualquier ϕ, fijamos G_ϕ como el data-grafo $G = (V, D, E)$, donde $V = \{v\}$, $D(v) = \mathtt{aux}$ y $E = \emptyset$. Para esta demostración hemos dejado que la codificación de ϕ se almacene en las restricciones, como mostramos a continuación.

Restricciones. Para cada $1 \leq i \leq n$, definimos la fórmula

$$\alpha_i := (\mathtt{p}_i^{=})? \downarrow^* (\mathtt{n}_i^{\neq})? \cup (\mathtt{p}_i^{\neq})? \downarrow^*,$$

indicando que *no puede haber simultáneamente nodos con valores de dato* $\mathtt{p}_i$ y $\mathtt{n}_i$. En efecto, supongamos que H es un data-grafo que satisface α_i y sean u, v nodos de H con valores de dato $\mathtt{p}_i$ y $\mathtt{n}_i$, respectivamente. Como $(u, v) \in [\![\alpha_i]\!]_H$, debe ocurrir que $(u, v) \in [\![(\mathtt{p}_i^{=})? \downarrow^* (\mathtt{n}_i^{\neq})?]\!]_H$ o $(u, v) \in [\![(\mathtt{p}_i^{\neq})? \downarrow^*]\!]_H$, pero el primer caso no puede ser por el valor de v y el segundo no puede ser por el valor de u. Note entonces que la aparición de un valor de dato $\mathtt{p}_i$ o $\mathtt{n}_i$ se puede corresponder con la idea de asignar el valor 1 o 0 (según el caso) a la variable proposicional x_i.

Sea la cláusula c_j y ℓ_1, ℓ_2, ℓ_3 los literales que contiene. Definimos el valor de dato $c_{j,k}$ con $k = 1, 2, 3$, como $\mathtt{p}_i$ si $\ell_k = x_i$, o como $\mathtt{n}_i$ si $\ell_k = \neg x_i$. Así, para cada $1 \leq j \leq m$, definimos la fórmula

$$\beta_j :=\downarrow^* (c_{j,1}^{=} \vee c_{j,2}^{=} \vee c_{j,3}^{=})? \downarrow^*,$$

indicando que *todo par de nodos se conectan por un camino que pasa por un nodo cuyo valor de dato es* $c_{j,k}$ *con* $k = 1, 2, 3$. Notemos que las expresiones α_i y β_j están de hecho en Core-GXPath^{pos}. La Figura 1.10 muestra una reparación para una fórmula ϕ particular.

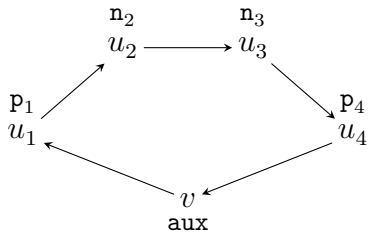

Figura 1.10: Una superrepación de G con respecto a $\mathcal{R}_\phi$ para $\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$, que se corresponde con la valuación $x_1 \mapsto 1, x_2 \mapsto 0, x_3 \mapsto 0, x_4 \mapsto 1$ que hace verdadera a la fórmula ϕ.

Note que la satisfacibilidad de cualquiera de las restricciones también implica que el data-grafo es completo con respecto a la relación $\downarrow^*$. Definimos $\mathcal{R}_\phi := \{\alpha_i \mid 1 \leq i \leq n\} \cup \{\beta_j \mid 1 \leq j \leq m\}$ y finalizamos la prueba demostrando la siguiente proposición.

Afirmación. *Las siguientes son equivalentes:*

1. *ϕ es satisfacible.*

2. G tiene una superreparación con respecto a $\mathcal{R}_\phi$.

1. implica 2. Sea f una valuación de $\{x_1, \ldots, x_n\}$ tal que $f \models \phi$ y sea $G_{\phi,f} = (V', D', E')$ con: $V' := \{v\} \cup \{u_i \mid 1 \leq i \leq n\}$; $D'(v) := D(v)$, $\quad D'(u_i) := \mathtt{p}_i$ si $f(x_i) = 1$ o $D'(u_i) := \mathtt{n}_i$ si $f(x_i) = 0$; y $E' := V \times \{\downarrow\} \times V$. Notemos que $G_{\phi,f}$ es un superdata-grafo de G y que tenemos por cada índice i exactamente un nodo con valor $\mathtt{p}_i$ o $\mathtt{n}_i$. Como $G_{\phi,f}$ es completo respecto a la relación $\downarrow$, entonces cada fórmula α_i se satisface.

Fijemos la fórmula β_j. Por hipótesis, hay un literal ℓ_k de c_j que se satisface. Supongamos que $\ell_k = x_i$ para algún i. Entonces por definición de $G_{\phi,f}$, el nodo u_i tiene valor de dato $\mathtt{p}_i$ y como $G_{\phi,f}$ es completo, para todo par de nodos u, w podemos construir un camino de $u \to^* u_i \to^* w$, como exige β_j. Análogo para $\ell_k = \neg x_i$. Entonces $G_{\phi,f}$ satisface β_j para todo j.

2. implica 1. Sea H una reparación de G con respecto a $\mathcal{R}_\phi$. Como H satisface α_i para cada i, puede suceder que hay un nodo u con valor de $\mathtt{p}_i$, o valor $\mathtt{n}_i$, o no hay ningún nodo con estos valores de dato.

Definamos la valuación f de $\{x_1, \ldots, x_n\}$ como $f(x_i) = 1$ si H tiene un nodo con valor de dato $\mathtt{p}_i$, o $f(x_i) = 0$, en caso contrario. Veamos que $f \models \phi$, que equivale a demostrar que $f \models c_j$ para todo j, que equivale a ver que c_j tiene un literal que se satisface. Como H satisface β_j, entonces para un par de nodos u, w de H, debe existir un camino $u \to^* u' \to^* w$, donde u' es un nodo con valor de dato $c_{j,k}$ con $k = 1, 2, 3$. Por definición, esto implica que el literal ℓ_k de c_j es de la forma x_i si $c_{j,k} = \mathtt{p}_i$, o de la forma $\neg x_i$ si $c_{j,k} = \mathtt{n}_i$. En cualquier caso tenemos que $f \models \ell_k$. □

El uso de una cantidad no acotada de valores de datos es en realidad un aspecto necesario para obtener la intratabilidad de ∃Superset Repair respecto a restricciones positivas. En efecto, para $\mathbb{D}_1 \subset \mathbb{D}$ finito y fijo, consideremos las ideas previas al planteamiento del Algoritmo 2 y supongamos que ahora $G = (V, D, E)$ y $\mathcal{R}$ (que en este caso también puede contener expresiones de camino) están limitados a usar solo símbolos de $\Sigma \cup \mathbb{D}_1$. Notemos que bajo esta consideración, el tamaño de $H = \text{complete}(G, \mathcal{R})$ solo depende de $|G|$ y más aún, por monotonía de Reg-GXPath^{pos}, basta con verificar que $H_X \models \mathcal{R}$ con $V \subseteq X \subseteq V \dot\cup V_1$. Notemos que este H_X se puede hallar a fuerza bruta en tiempo polinomial, porque solo hay a lo sumo $2^{|\mathbb{D}_1|}$ de tales data-grafos, es decir, una cantidad fija constante para cualquier entrada G. Finalizamos esta sección con el siguiente enunciado.

Teorema 35. ∃Superset Repair *respecto a expresiones positivas definidas sobre una cantidad finita de etiquetas fijas es* PTime *en complejidad combinada.*

Capítulo 2

Controlabilidad finita de consultas regulares mediadas por ontologías

Estudiaremos una versión del problema CQA, presentado en la Sección 1.2.2, en el contexto de bases de conocimiento y que denominamos de ahora en adelante como Ontology-Mediated Query Answering (OMQA) en concordancia con el artículo [86], en el que se basa este capítulo. Informalmente, este problema consiste en la identificación de las respuestas comunes de una consulta sobre todas las *extensiones relevantes* o *mundos posibles* de una base de conocimiento inconsistente e incompleta. En el capítulo anterior, entendíamos una extensión relevante como una superreparación (es decir, una estructura consistente y minimal) de un data-grafo inconsistente con respecto a un conjunto de restricciones. En [126] analizan el problema de respuestas consistentes de bases de conocimiento bajo reparaciones definidas por *membresías*, que es análogo a la idea de reparaciones de bases de datos definidas por diferencia simétrica. En este capítulo nos liberamos de la condición extra de minimalidad que caracteriza a las reparaciones, pero abordamos el caso de extensiones consistentes (posiblemente infinitas) de una base inconsistente con respecto a cierto tipo de restricciones que engloban varias clases conocidas de lógicas de descripción y que serán introducidas como fragmentos decidibles de la lógica de primer orden.

Uno de los problemas de mayor importancia de las últimas décadas trata sobre el manejo y mantenimiento de bases de datos masivas, para las cuales dar respuestas a consultas, incluso para las más básicas, puede resultar una tarea altamente costosa. Esto ha dado lugar al desarrollo de varias técnicas que se centran justamente en el problema de responder consultas, y particularmente, para los casos en los que hay una ontología circundante. En el esquema de la Web Semántica se encuentra la comunidad OWL (Web Ontology Language) que se enfoca en el desarrollo de sistemas automáticos para derivabilidad (*entailment*) y respuestas a consultas (*query answering*), en especial cuando estas respuestas provienen de una base de conocimientos. Algunos trabajos que surgen bajo esta perspectiva son [74, 75] donde desarrollan técnicas escalables para responder consultas mediante procesos de refinamiento y sumarización; [116] donde se establece un BPR (*Business Process Repository*) con una semántica para almacenar ontologías y razonar con ellas; [144] donde se plantea una lógica de descripción ideal para razonar con grandes volúmenes de información y un método de respuestas a consultas con mecanismos de mapeo específicos y de acceso eficiente a la data. A diferencia de los trabajos citados, el problema OMQA no solo se basa en computar el

conjunto de soluciones de una consulta, sino que con este se busca identificar con qué información ya existente en la base se pueden obtener respuestas a consultas que estarán siempre presentes en posibles extensiones consistentes, lo cual lo podemos pensar como un problema de derivabilidad en un sentido lógico, es decir, qué información es semánticamente consecuente bajo una ontología y consultas dadas. Varios trabajos destacados en la línea de bases de conocimiento son [45, 141] donde recopilan bastante teoría (incluyendo varios resultados de complejidad) desarrollada para lógicas de descripción básicas y consultas navegacionales; y [55, 138] para reglas existenciales y consultas conjuntivas Booleanas y no Booleanas.

Cuando nos restringimos a extensiones consistentes finitas, denominamos este problema *finite* OMQA, o FOMQA. Para superreparaciones, en [32] analizan el problema CQA de RPQs bajo RPCs, que resulta ser indecidible (para más detalles técnicos sobre esto, ver la demostración del Teorema 26). En este mismo trabajo obtienen un resultado tratable cuando consideran a las restricciones como reglas existenciales GAV (*global-as-view*), es decir, cuando las contenciones tienen una conjunción de átomos del lado izquierdo y un solo átomo del lado derecho. Otra instancia conocida donde se usan RPQs para el lenguaje de consulta es [151] (ver la demostración del Teorema 27), donde se estudia la indecibilidad de CQA de 2RPQ bajo restricciones $\mathcal{ALCOIF}$,[1] que es una lógica de descripción altamente expresiva. Hasta donde sabemos, no hay otros trabajos previos a [86] aparte de los mencionados que traten el problema FOMQA con (extensiones de) RPQ como el lenguaje de consulta, siendo que la mayoría de trabajos al respecto suelen fijar como lenguaje de consulta a las *conjunctive queries* (CQs), o uniones de CQs (UCQs) [25, 100, 101], o variaciones de estos que son menos expresivos que los lenguajes navegacionales usuales. Respecto a la ontología, como se puede observar en los trabajos previamente citados, se han estudiado muchas versiones de FOMQA bajo restricciones dadas por lógicas de descripción, o fragmentos decidibles de la lógica de primer orden, como los *guarded fragments* [158], que serán muy importantes en este capítulo.

En particular, nos interesa analizar el caso cuando OMQA sobre extensiones consistentes arbitrarias coincide (como problemas computacionales) con OMQA sobre extensiones consistentes finitas, identificando y clasificando los fragmentos de los lenguajes ontológicos y de consulta donde esta relación existe. A tal propiedad la denominamos *controlabilidad finita*, tal como fue introducida en [119], donde demostraron que el problema Query Answering de CQs bajo dependencias de inclusión (ID) es decidible. En ese trabajo, el problema de controlabilidad finita para ID se dejó abierto. Esta noción fue retomada en [149, 150] bajo ontologías dadas por IDs y dependencias funcionales, quedando resuelta positivamente la mencionada pregunta abierta de Johnson y Klug, al aplicar un método que denominaron *chase finito*. Este procedimiento fue generalizado en [53] donde demostraron controlabilidad finita respecto a ontologías en GFO y consultas UCQs. Desde el punto de vista de Teoría de Bases de Datos, estudiar el criterio de controlabilidad finita tiene sentido en virtud de que las bases de datos son técnicamente estructuras finitas, para lo cual establecer teorías lógicas de razonamiento y deducción puede resultar muy difícil y complicado, mientras que si se establece que para ciertos lenguajes vale la controlabilidad finita es posible en muchos casos utilizar técnicas y herramientas estándar de teoría de modelos, donde no hay

[1] En realidad, en [151] se estudia la complejidad de varias versiones del problema *Finite Entailment* que está íntimamente relacionado a CQA y FOMQA.

restricciones de cardinalidad.

En este capítulo analizamos la controlabilidad finita de OMQA de $\mathcal{Q}$ bajo $\mathcal{O}$, cuando $\mathcal{Q}$ es un fragmento de UC2RPQ y $\mathcal{O}$ un fragmento de GNFO. Los fragmentos de UC2RPQ a considerar dependerán de propiedades relacionadas al *esqueleto de una consulta*, que hemos definido como una especie de grafo subyacente donde los nodos y las aristas de este grafo se etiquetan en función de las características sintácticas de la consulta asociada, estos conceptos se presentarán en la Sección 2.2 junto con el planteamiento del resultado principal del capítulo. En el Teorema 38 hemos clasificado los fragmentos de UC2RPQ (definidos por clases de esqueletos especiales) que son finitamente controlables bajo ciertos fragmentos de GNFO. Esta clasificación es completa, en el sentido de que el teorema se ha planteado con condiciones necesarias y suficientes para garantizar la controlabilidad finita de OMQA. La demostración de la necesidad se aborda en la Sección 2.3 y la estrategia principal (Lema 40) se basa en reducir una variante de OMQA a el problema de satisfacibilidad de GNFO el cual es decidible y tiene la propiedad de satisfacibilidad finita [157], de donde se desprende nuestro resultado de controlabilidad finita. Para esta parte usamos una versión generalizada de OMQA, que denotamos gOMQA, con el propósito de definir ciertas reducciones que bajo el planteamiento usual de OMQA no es claro cómo se podría realizar. La demostración de la suficiencia se aborda en la Sección 2.4, argumentando por contrarrecíproco y la estrategia se basa en plantear contraejemplos que construimos aprovechando propiedades conocidas del *chase standard* [56], que serán repasadas en la Sección 2.1 de preliminares.

2.1 Preliminares

Símbolos, estructuras y morfismos Usaremos gran parte de las convenciones de bases de conocimientos dadas en Nociones Básicas, particularmente las referentes a conceptos estructurales como *modelos*, *grafo subyacente*, *dominio activo*, etc, pero aprovechamos este espacio introducir conceptos que son relevantes para este capítulo. En cuanto al lenguaje ontológico, no haremos uso de $\mathcal{ALCOIF}$, sino que definiremos en esta sección nuestro lenguaje de interés, GNFO.

A diferencia de los data-grafos, el tipo de estructuras utilizadas en el capítulo anterior, un modelo o ABox puede especificar interacciones entre más de dos individuos del dominio activo bajo una misma relación, es decir, no estamos restringidos a usar predicados binarios o unarios. Otra diferencia semántica notoria es que los individuos de un modelo no *contienen data*, lo que difiere del capítulo anterior donde introducíamos esta noción como una forma de etiquetado bajo la cual ciertos lenguajes tenían acceso a la información contenida en los nodos de un data-grafo para realizar verificaciones comunes como comparación de datos. En este caso, un hecho de la forma $P(a)$ con P un predicado unario y a un individuo, refleja simplemente una característica estructural del modelo en cuestión, similar a lo que sucede con las funciones de etiquetadoo de estados de un sistema de transición [52]. Si bien en la demostración del Teorema 27 vimos que un modelo finito (que solo usa predicados unarios y binarios) puede reinterpretarse como un data-grafo, donde los hechos unarios que involucran un mismo individuo se sumergen en un único valor de dato, aclaramos que esta reducción funciona en un sentido netamente computacional. En general, no es directo ni claro cómo traducir los resultados de este capítulo al esquema teórico del Capítulo 1.

Un homomorfismo de un modelo $\mathcal{A}$ a un modelo $\mathcal{B}$ es una función $h : adom(\mathcal{A}) \to adom(\mathcal{B})$ tal que si $R(\bar{a}) \in \mathcal{A}$ entonces $R(h(\bar{a})) \in \mathcal{B}$. Decimos que h preserva un conjunto $C \subseteq \mathsf{N_I}$, si $h(c) = c$ para todo $c \in C$. Un homomorfismo de una conjunción de átomos φ a un modelo $\mathcal{B}$ es una función $h : terms(\varphi) \to adom(\mathcal{B})$ que preserva todas las constantes de φ y además si $R(\bar{t})$ es un átomo de φ entonces $R(h(\bar{t})) \in \mathcal{B}$. Notemos que esta segunda noción de homomorfismo es similar a la presentada en la Definición 5, pero para expresiones en CQ. Preservar constantes bajo este tipo de homomorfismos se corresponde con la *standard name assumption*[2] que asumimos a lo largo de este capítulo.

Consultas mediadas por ontologías En adelante, consideramos una restricción como una propiedad de un modelo descrita mediante un lenguaje lógico. Dado un lenguaje de consulta $\mathcal{Q}$ y un lenguaje ontológico o de restricciones $\mathcal{O}$, el problema Ontology-Mediated Query Answering de $\mathcal{Q}$ bajo $\mathcal{O}$ se describe como[3]

OMQA	
Entrada:	Un modelo finito $\mathcal{G}$, un conjunto finito de restricciones $\Gamma \subseteq_{\text{fin}} \mathcal{O}$, una consulta $q(\bar{x}) \in \mathcal{Q}$ de aridad k, y una k-tupla $\bar{a}$ de $adom(\mathcal{G})$.
Salida:	Decidir si para todo $\mathcal{G}'$ que extiende a $\mathcal{G}$ y satisface todas las propiedades de Γ, tenemos que $\bar{a}$ está en el conjunto de respuestas $q(\mathcal{G}')$.

También estudiamos una versión generalizada de este problema, que será útil a nivel técnico para obtener algunas reducciones. Con $\mathcal{Q}$ y $\mathcal{O}$ como antes, el problema OMQA generalizado se define como

gOMQA	
Entrada:	Un modelo finito $\mathcal{G}$, un conjunto finito de restricciones $\Gamma \subseteq_{\text{fin}} \mathcal{O}$, una consulta $q(\bar{x}) \in \mathcal{Q}$ de aridad k, y X un conjunto finito de k-tuplas de $adom(\mathcal{G})$.
Salida:	Decidir si para cada $\mathcal{G}' \supseteq \mathcal{G}$ que satisface Γ existe $\bar{a} \in X$ tal que $\bar{a} \in q(\mathcal{G}')$.

Note que gOMQA restringido a conjuntos unitarios es equivalente a OMQA, de ahí el adjetivo *generalizado*. El problema finite ontology-mediated query answering (FOMQA) y su generalización (gFOMQA) se definen análogos a los problemas anteriores, excepto que ahora $\mathcal{G}'$ varía sobre todas las extensiones *finitas* y consistentes de $\mathcal{G}$.

Escribimos $\text{OMQA}_{\mathcal{Q},\mathcal{O}}$ [resp. $\text{FOMQA}_{\mathcal{Q},\mathcal{O}}$] para denotar el conjunto de 4-tuplas $(\mathcal{G}, \Gamma, q(\bar{x}), \bar{a})$[4] tales que el problema OMQA [resp. FOMQA] de $\mathcal{Q}$ bajo $\mathcal{O}$ arroja una respuesta positiva. Para

[2] Las constantes se interpretan de la misma manera en todos los modelos.

[3] Este problema es un planteamiento particular del esquema general de CQA presentado en la Sección 1.2.2.

[4] El par $(\mathcal{G}, \Gamma)$ es una base de conocimiento en la terminología usada anteriormente en este trabajo.

el caso de las versiones generalizadas, las 4-tuplas son de la forma $(\mathcal{G}, \Gamma, q(\bar{x}), X)$ con X un conjunto finito de k-tuplas y se denota al conjunto de respuestas afirmativas como $\text{gOMQA}_{\mathcal{Q},\mathcal{O}}$ [resp. $\text{gFOMQA}_{\mathcal{Q},\mathcal{O}}$].

Una instancia negativa $(\mathcal{G}, \Gamma, q(\bar{x}), \bar{a})$ $\text{OMQA}_{\mathcal{Q},\mathcal{O}}$ [resp. $\text{FOMQA}_{\mathcal{Q},\mathcal{O}}$] cumple que existe un modelo [resp. un modelo finito] $\mathcal{G}' \supset \mathcal{G}$ que satisface Γ pero $\bar{a} \notin q(\mathcal{G}')$. A tal $\mathcal{G}'$ lo llamamos contra-modelo [resp. contra-modelo finito] de $(\mathcal{G}, \Gamma, q(\bar{x}), \bar{a})$. Los contra-modelos para instancias negativas de $\text{gOMQA}_{\mathcal{Q},\mathcal{O}}$ y $\text{gFOMQA}_{\mathcal{Q},\mathcal{O}}$ se definen análogamente.

Observe que $\text{OMQA}_{\mathcal{Q},\mathcal{O}} \subseteq \text{FOMQA}_{\mathcal{Q},\mathcal{O}}$ [resp. $\text{gOMQA}_{\mathcal{Q},\mathcal{O}} \subseteq \text{gFOMQA}_{\mathcal{Q},\mathcal{O}}$] siempre, pero la contención contraria no es necesariamente cierta. Si $\text{OMQA}_{\mathcal{Q},\mathcal{O}} = \text{FOMQA}_{\mathcal{Q},\mathcal{O}}$ [resp. $\text{gOMQA}_{\mathcal{Q},\mathcal{O}} = \text{gFOMQA}_{\mathcal{Q},\mathcal{O}}$], decimos que OMQA [resp. gOMQA] de $\mathcal{Q}$ bajo $\mathcal{O}$ tiene la propiedad de controlabilidad finita.

Regular Path Queries Consideramos *lenguajes regulares* sobre cualquier subconjunto de $\mathsf{N}_\mathsf{R}^\pm$ definidos por expresiones regulares como es usual. Usaremos los aspectos establecidos en el Capítulo y en general usaremos fragmentos de UC2RPQ como el lenguaje de consulta $\mathcal{Q}$. Solo aclaramos que en una instancia $(\mathcal{G}, \Gamma, q, \bar{a})$ de OMQA (y cualquiera de sus variantes) podemos considerar q como Booleana tomando $\bar{a} = ()$.

Lenguajes de restricción Comenzamos definiendo el lenguaje ontológico más expresivo para esta parte: el fragmento guarded-negation de la lógica de primer orden. Guarded-negation first order logic (GNFO) es el fragmento de la lógica de primer orden con igualdad, dada por la siguiente gramática, donde $P \in \mathsf{Pred}$ y $\alpha \in \mathsf{Pred} \dot{\cup} \{=\}$:

$$\varphi ::= P(\bar{x}) \mid x = y \mid \exists x\, \varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \alpha(\bar{x}\bar{y}) \wedge \neg\varphi(\bar{y}). \tag{A}$$

Acá $P(\bar{x})$ y $\alpha(\bar{x}\bar{y})$ son átomos, que posiblemente contienen constantes y cuyas variables libres están entre las variables de $\bar{x}$ y $\bar{x}\bar{y}$, respectivamente. En una expresión de la forma $\alpha(\bar{x}\bar{y}) \wedge \neg\varphi(\bar{y})$ llamamos al átomo $\alpha(\bar{x}\bar{y})$ guarda. El tamaño de una fórmula φ es el número de símbolos utilizados para escribirla y se denota por $|\varphi|$.

Asumimos la semántica usual para este tipo de expresiones y dado un modelo $\mathcal{G}$ y una sentencia φ de GNFO, escribimos $\mathcal{G} \models \varphi$ para indicar que $\mathcal{G}$ satisface φ. Decimos que $\mathcal{G}$ satisface un conjunto finito Γ de sentencias en GNFO si $\mathcal{G}$ satisface $\bigwedge_{\varphi\in\Gamma} \varphi$.

Observe que en GNFO, fórmulas con a lo sumo una variable libre son cerradas bajo negación, debido a la equivalencia $\neg\varphi(x) \equiv (x = x \wedge \neg\varphi(x))$ (análogo para sentencias). GNFO posee muchas buenas propiedades computacionales y de expresividad, varias de las cuales están recopiladas en el survey de [158]. En particular, tiene la *propiedad de modelo finito* (si $\varphi \in$ GNFO es satisfacible, entonces es satisfacible en un modelo finito), y su problema de satisfacibilidad es decidible y 2ExpTime-completo [30]. Es bien conocida la relación de GNFO con varios fragmentos importantes de la lógica de primer orden como GFO y UNFO, pero en este trabajo abordaremos expresiones de otra índole y que presentamos a continuación.

Una regla existencial (también conocida como regla Datalog$^\pm$, o más tradicionalmente *dependencia generadora de tuplas* TGD) es una sentencia de primer orden de la forma

$$\forall\bar{x}\bar{y}(\varphi(\bar{x}\bar{y}) \Rightarrow \exists\bar{z}\psi(\bar{x}\bar{z})), \tag{B}$$

donde φ y ψ son conjunciones de átomos. Llamamos a φ y ψ el cuerpo y la cabeza de la regla (B), respectivamente, y a $\bar{x}$ las variables frontera. En [36] fueron definidos estos tipos de expresiones con la intensión de unificar bajo un mismo esquema formal varios tipos de restricciones (también llamadas *dependencias*) ampliamente utilizados en bases de datos relacionales. Presentamos algunas familias de reglas existenciales de interés para nuestros resultados.

- Una regla existencial es frontier-guarded si hay un átomo en el cuerpo (al que llamamos guarda) que contiene a todas las variables frontera; denotamos FG al conjunto de todas las reglas existenciales frontier-guarded.
- Una regla existencial es de frontier 1 si tiene a lo sumo una variable frontera; denotamos F1 al conjunto de todas las reglas existenciales de frontera 1.
- Una regla existencial es term-frontier 1 si es de frontera 1 y su cuerpo y átomo tienen a lo sumo un término en común (i.e. podría tratarse de una variable o una constante). En particular, la regla $(\forall x)R(c,x) \Rightarrow S(c,x)$ donde c es una constante, es de frontera 1, pero no es term-frontera 1. Denotamos TF1 al conjunto de todas las reglas existenciales de term-frontera 1.
- Una inclusion dependency es una regla existencial sin constantes cuya cabeza y cuerpo consisten de un solo átomo; denotamos por ID al conjunto de todas las dependencias de inclusión.

Si bien en GNFO podemos disponer de cuantificadores existenciales con libertad, no es el caso de los cuantificadores universales, que por lo general solo pueden introducirse ante la presencia de una guarda. Para ver que una expresión en FG se ajusta al esquema (A), notemos primero que una expresión de la forma (B) es equivalente a:

$$\exists u \left(u = u \wedge \neg\exists \bar{x}\bar{y}(\varphi(\bar{x}\bar{y}) \wedge \neg\exists \bar{z}\psi(\bar{x}\bar{z}))\right), \tag{C}$$

donde u es una variable cualquiera que se usa para introducir la negación externa mediante la guarda $u = u$. Si suponemos que (B) es FG entonces φ debe tener un subátomo $\alpha(\bar{x}\bar{w})$ (con las variables de $\bar{w}$ todas en $\bar{y}$), el cual puede asociarse con $\neg\exists\bar{z}\psi(\bar{x}\bar{z})$ para justificar la aparición de esta segunda negación, por lo que (C) sería en este caso una expresión de GNFO. Notemos que el tamaño de la expresión (C) es polinomial con respecto al tamaño de la fórmula (B). De manera similar podemos ver que las clases ID, F1, TF1 también son definibles en GNFO, a través de traducciones polinomiales.

OMQA de UC2RPQ bajo GNFO Como ya se discutió, el problema OMQA de CRPQ bajo GNFO es decidible incluso cuando el lenguaje de restricciones se extiende con operadores de punto fijo (GNFP) [30] y ya vimos que con GNFO se pueden expresar reglas existenciales con guardas, incluyendo las frontier-guarded y varias otras subclases.

Proposición 36. *El problema* OMQA *de* UC2RPQ *bajo* GNFO *es decidible.*

Demostración. Esto es consecuencia de un resultado de [40], donde definen el lenguaje GNFP-UP que extiende a GNFP (y por lo tanto, también a GNFO), captura a UC2RPQ y cuyo problema de satisfacibilidad asociado es decidible.

Esta reducción sigue un argumento similar al utilizado en las demostraciones de los Teoremas 26 y 27. Dada una instancia $\mathcal{I} = (\mathcal{G}, \Gamma, q(\bar{x}), \bar{a})$ del problema OMQA de UC2RPQ bajo GNFO, definiremos una fórmula $\phi_{\mathcal{I}}$ que será satisfacible si y solo si $\mathcal{I}$ tiene un contra-modelo.

Para $\mathcal{G}$ definimos $\phi_{\mathcal{G}} := \bigwedge_{P(\bar{a})} P(\bar{a})$, que es una sentencia que solo usa términos constantes y por lo tanto está en GNFP-UP. Para Γ definimos $\phi_\Gamma := \bigwedge_{\varphi \in \Gamma} \varphi$ que está trivialmente en GNFP-UP. En [40, Example 4], muestran cómo traducir una consulta en C2RPQ a una expresión equivalente en GNFP-UP, lo cual puede extenderse a uniones de C2RPQs. Para $q(\bar{x})$ y $\bar{a}$, definimos $\phi_{q,\bar{a}}$ como la traducción de la sentencia $\neg q(\bar{a})$.

Definimos $\phi_{\mathcal{I}} := \phi_{\mathcal{G}} \wedge \phi_\Gamma \wedge \phi_{q,\bar{a}}$ para la cual se cumple

Afirmación. *Las siguientes son equivalentes:*

1. $\phi_{\mathcal{I}}$ *es satisfacible.*
2. $\mathcal{I}$ *tiene un contra-modelo.*

1. implica 2. Existe un modelo $\mathcal{G}'$ y un homomorfismo $h : terms(\phi_{\mathcal{I}}) \to adom(\mathcal{G})$. Como h preserva constantes entonces $\mathcal{G}'$ extiende a $\mathcal{G}$ por satisfacer $\phi_{\mathcal{G}}$. Es directo que $\mathcal{G}'$ satisface Γ por definición de ϕ_Γ. Dado que $\phi_{q,\bar{a}}$ y $\neg q(\bar{a})$ poseen la misma semántica, entonces $\mathcal{G}'$ satisface $\phi_{q,\bar{a}}$ si y solo si $\bar{a} \notin q(\mathcal{G}')$, lo que implica que $\mathcal{G}'$ es un contra-modelo de $\mathcal{I}$.
2. implica 1. El argumento es análogo a lo anterior. □

Sin embargo, el respectivo problema FOMQA permanece abierto y por los momentos no es posible aplicar una demostración análoga a la anterior. Se sabe que el problema de satisfacibilidad finita para GNFP es decidible [29], pero no es claro si el problema de satisfacibilidad finita para GNFP-UP también lo es. Para el caso de restricciones definidas por reglas existenciales con guardas tenemos el siguiente resultado.

Proposición 37. [24] *El problema* OMQA *de* UC2RPQ *bajo reglas existenciales con guarda es* 2ExpTime*-completo en complejidad combinada y* PTime*-completo en data complejidad.*

Chase En la literatura relacionada con teoría de bases de datos, teoría de dependencias y teoría de la demostración, el *chase* puede hacer referencia a un par de conceptos íntimamente relacionados. El chase puede referirse a (1) ciertos procedimientos algorítmicos usados para decidir (o aceptar) instancias del *problema de implicación de dependencias*, o (2) modelos *universales* que surgen de extender algún otro modelo inconsistente respecto a un conjunto de restricciones. La conexión entre ambos conceptos radica en que los procesos aludidos en (1) por lo general involucran la generación de los modelos mencionados en (2), lo cual se puede formalizar incluso en los casos cuando no hay garantía de la terminación del chase como algoritmo, dando como resultado en estos casos un modelo universal infinito. En el ámbito de reglas existenciales (y *reglas de igualdad* que no serán definidas en este trabajo) fue aplicado originalmente este procedimiento en [36, 37] donde se analizaron distintos casos del problema de implicación respecto a varias subclases de dependencias. Como proceso algorítmico existen muchísimas adaptaciones, varias de las cuales fueron estudiadas y definidas en [70], donde además analizan el problema de terminación del chase. En este trabajo requeriremos el chase bajo reglas existenciales en el sentido de modelo universal como se define en [56], cuya construcción describimos a continuación mediante un proceso denominado *restricted TGD chase rule*.

Sea N_{null} un conjunto infinito de *nulls* indexados, que utilizaremos para describir el dominio activo de los modelos universales aparte de las constantes N_I. A diferencia de las constantes sobre

las cuales asumimos la standard name assumption, dos *nulls* distintos pueden corresponderse con una misma constante mediante algún homomorfismo o asignación.

Dado un conjunto Γ de reglas existenciales, $Chase(\mathcal{G}, \Gamma)$ es un modelo posiblemente infinito definido como la unión $\bigcup_{i \geq 0} \mathcal{G}_i$ tal que $\mathcal{G}_0 = \mathcal{G}$, y para cada $i > 0$, $\mathcal{G}_i$ es un modelo que extiende a $\mathcal{G}_{i-1}$ con una cantidad finita de nuevos átomos que certifican la cabeza de las reglas, cuando sea que el cuerpo se satisfaga. Concretamente, sea S el conjunto de todos los pares (ψ, h) tales que existe una regla $\forall \bar{x}\bar{y}(\varphi(\bar{x}\bar{y}) \Rightarrow \exists \bar{z}\psi(\bar{y}\bar{z}))$ en Γ y un homomorfismo h de $\varphi(\bar{x}\bar{y})$ a $\mathcal{G}_{i-1}$ tal que $\mathcal{G}_{i-1} \not\models \exists \bar{z}\psi(h(\bar{y})\bar{z})$. En este caso decimos que la tupla $h(\bar{y})$ activa la regla $\forall \bar{x}\bar{y}(\varphi(\bar{x}\bar{y}) \Rightarrow \exists \bar{z}\psi(\bar{y}\bar{z}))$. Para cada par (ψ, h) de S, sea $g_{\psi,h} : \bar{z} \to \mathsf{N}_{null} \setminus adom(\mathcal{G}_{i-1})$ cualquier función inyectiva de las variables en $\bar{z}$ a *nulls* de modo que la imagen de cualesquiera dos de tales funciones tengan intersección vacía. Definimos $\mathcal{G}_i$ como la extensión de $\mathcal{G}_{i-1}$ con todos los hechos en $\psi(h(\bar{y}), g_{\psi,h}(\bar{z}))$, para cada $(\psi, h) \in S$. Notemos que para todo $t \in adom(\mathcal{G}_{i-1})$, al generar $\mathcal{G}_i$ solo se agregan nuevos hechos que involucran a t en caso de que exista una tupla $\bar{a}$ de $adom(\mathcal{G}_{i-1})$ que contenga a t y active una regla de Γ. Observe que $\mathcal{G}_{i-1} = \mathcal{G}_i$ si y solo si $\mathcal{G}_{i-1} \models \Gamma$. Además, cada $\mathcal{G}_i$ es único salvo por renombramiento de los términos en $\mathsf{N}_{null} \setminus adom(\mathcal{G})$ y por lo tanto, $Chase(\mathcal{G}, \Gamma)$ también es único. Llamamos a $\mathcal{G}_i$ el i-ésimo paso del chase, denotado $Chase^i(\mathcal{G}, \Gamma)$. A todo este proceso se le suele llamar el Chase 'standard' o 'restrictivo'. Hay tres propiedades fundamentales de $Chase(\mathcal{G}, \Gamma)$ que usaremos en nuestras pruebas:

(a) $Chase(\mathcal{G}, \Gamma)$ satisface Γ.

(b) $Chase(\mathcal{G}, \Gamma)$ es un *modelo universal* en el siguiente sentido: si $\mathcal{G}' \supseteq \mathcal{G}$ y $\mathcal{G}'$ satisface Γ, entonces hay un homomorfismo de $Chase(\mathcal{G}, \Gamma)$ a $\mathcal{G}'$ que preserva a $adom(\mathcal{G})$.

(c) Para los casos particulares cuando $\Gamma \subseteq$ TF1, para cada $i \geq 0$ y $t \in adom(Chase^i(\mathcal{G}, \Gamma))$, hay un $j \geq i$ tal que todos los hechos que contienen a t en $Chase(\mathcal{G}, \Gamma)$ ya están presentes en $Chase^j(\mathcal{G}, \Gamma)$; en particular, esto significa que $Chase(\mathcal{G}, \Gamma)$ es de *ramificación finita*.

Las propiedades (a) y (b) son bien conocidas y comúnmente citadas, tanto para los casos de chase finito como infinito [54, 79, 140], mientras que la propiedad (c) ocurre por el siguiente hecho: cuando un término $t \in adom(Chase^i(\mathcal{G}, \Gamma))$ activa una regla existencial TF1[5] $\forall x\bar{y}(\varphi(x\bar{y}) \Rightarrow \exists \bar{z}\psi(x\bar{z}))$ de Γ (con x el único elemento frontera de la regla), entonces en $Chase^j(\mathcal{G}, \Gamma)$ con $j > i$, la regla no se vuelve a activar en t. Sea j lo suficientemente grande para garantizar que hasta el j-ésimo paso del chase, todas las reglas de Γ que se pudieron haber activado en t, se activaron y se solucionaron en el siguiente paso. Por definición de $Chase(\mathcal{G}, \Gamma)$, todos los hechos que contienen a t, ya deben estar en $Chase^j(\mathcal{G}, \Gamma)$. La Figura 2.1 muestra que la propiedad de ramificación finita se pierde incluso para reglas en F1.

2.2 Resultados Principales

Como se anticipó al comienzo de este capítulo, nuestros resultados caracterizan las clases de fórmulas en UC2RPQ finitamente controlables basadas en la forma de sus grafos subyacentes. Sin embargo, como queremos ser mucho más descriptivos, modificamos la noción usual de grafo

[5] Otro caso por considerar es cuando t es una constante y la regla es de la forma $\forall \bar{y}(\varphi(t\bar{y}) \Rightarrow \exists \bar{z}\psi(t\bar{z}))$, pero el razonamiento es el mismo al del párrafo.

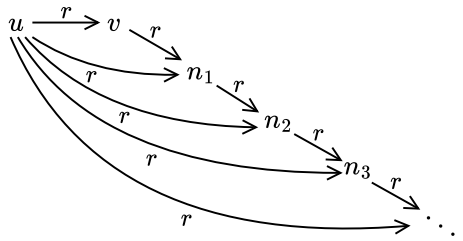

Figura 2.1: *Chase*$(\mathcal{G}, \Gamma)$ para $\mathcal{G} = \{r(u, v)\}$ y $\Gamma = \{(\forall x)(r(u, x) \Rightarrow (\exists z)r(x, z) \wedge r(u, z)\})$. El i-ésimo paso del chase se identifica con el modelo generado por $\mathcal{G}$ y los primeros i *nulls*: $n_1, \ldots, n_i$. Notemos que Γ se compone de una única fórmula en F1 que no está en TF1 por tener como términos frontera a la constante u y a la variable x. El chase resultante no es de ramificación finita pues para todo i tenemos que $r(u, n_i) \in$ *Chase*$(\mathcal{G}, \Gamma)$.

subyacente para que indiquen cuáles nodos provienen de una variable libre y cuáles ejes tienen un lenguaje finito como etiqueta. Llamaremos a este grafo el *esqueleto* de un C2RPQ. El propósito es definir una clase de esqueletos $\mathcal{S}$ tal que (i) cada consulta con esqueleto en $\mathcal{S}$ es finitamente controlable; y (ii) para cada esqueleto $sk \notin \mathcal{S}$ existe una consulta que no es finitamente controlable y cuyo esqueleto es *sk*. Definiremos una clase $\mathcal{S}_1$ con tal propiedad para cada lenguaje de restricciones que contienen dependencias de inclusión y que estén contenidas en GNFO; y otra clase $\mathcal{S}_2$ estrictamente más grande que $\mathcal{S}_1$ y que satisface la propiedad para reglas existenciales F1.

Definiciones Un multigrafo es una tupla $M = (V, E, \eta)$ donde V es un conjunto finito de vértices, E es un conjunto finito de ejes etiquetados, y $\eta : E \to V \times V$ es una función que asocia a cada eje el par que indica el vértice de origen y el de llegada. El grafo subyacente no dirigido G_M de M es el grafo no dirigido simple que tiene a V como conjunto de vértices y $\{\{v, v'\} : e \in E, \eta(e) = (v, v')\}$ como conjunto de ejes. Un camino simple no dirigido (resp. ciclo no dirigido) de M es una sucesión (posiblemente vacía) de ejes que induce un camino simple (resp. un ciclo) en G_M.

Un esqueleto es una terna (M, ν, μ) donde $M = (V, E, \eta)$ es un multigrafo, $\nu : V \to \{b, f\}$ y $\mu : E \to \{\infty, < \infty\}$. En este contexto, decimos que un vértice $v \in V$ es libre (*free* en inglés) o ligado (*bound* en inglés) dependiendo de si $\nu(v) = f$ o $\nu(v) = b$. La distancia entre dos vértices v, v' es finito si existe un camino no dirigido $e_1, \ldots, e_n$ de M tal que $\mu(e_i) =< \infty$ para todo i. Un ciclo no dirigido $e_1, \ldots, e_n$ de M es infinito si $\mu(e_i) = \infty$ para algún i y es ligado si para cada i tal que $\eta(e_i) = (v_1, v_2)$ tenemos que $\nu(v_1) = \nu(v_2) = b$.

Dado un C2RPQ q y un esqueleto $sk = (M, \nu, \mu)$, decimos que *sk* es el esqueleto de q si para $M = (V, E, \eta)$ tenemos que V es el conjunto de variables de q, E es el conjunto de átomos de q, $\eta(e) = (v_1, v_2)$ si y solo si el átomo e es de la forma $v_1 \xrightarrow{L} v_2$ para algún L, $\nu(v) = b$ si y solo si v está ligado en q, y $\nu(e) = \infty$ si y solo si el átomo e es de la forma $v_1 \xrightarrow{L} v_2$ con L un lenguaje *infinito*. Ver Figura 2.2 para un ejemplo de una consulta y su esqueleto.

Usualmente nos referiremos a los elementos de V y E como vértices y átomos de *sk*. El esqueleto de una fórmula en UC2RPQ es el conjunto de esqueletos de las C2RPQ que contiene. Dado un conjunto de esqueletos $\mathcal{S}$, definimos la clase UC2RPQ($\mathcal{S}$) como el conjunto de todas las consultas en UC2RPQ cuyos esqueletos están en $\mathcal{S}$.

Definimos $\mathcal{S}_1$ como el conjunto de todos los esqueletos tales que cada ciclo infinito no dirigido

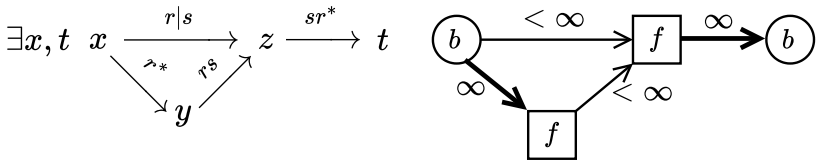

Figura 2.2: A la izquierda una consulta C2RPQ con variables libres y, z, a la derecha el esqueleto correspondiente. En los próximos gráficos identificamos los ejes con etiqueta ∞ con aquellas flechas de mayor grosor, los nodos con etiqueta b con nodos circulares y aquellos con etiqueta f con nodos cuadrados. En un esqueleto la dirección de las flechas puede no ser completamente relevante pero se mantienen para especificar de manera más sencilla los contraejemplos que se construirán en la Sección 2.4.

contiene un nodo libre. Por ejemplo, el esqueleto de la Figura 2.2 está en $\mathcal{S}_1$. Definimos $\mathcal{S}_2$ como el conjunto de todos los esqueletos tales que cada ciclo infinito no dirigido y ligado contiene un vértice a distancia finita de uno libre. Ver Figura 2.3 para algunos ejemplos. Por definición, $\mathcal{S}_1 \subsetneq \mathcal{S}_2$.

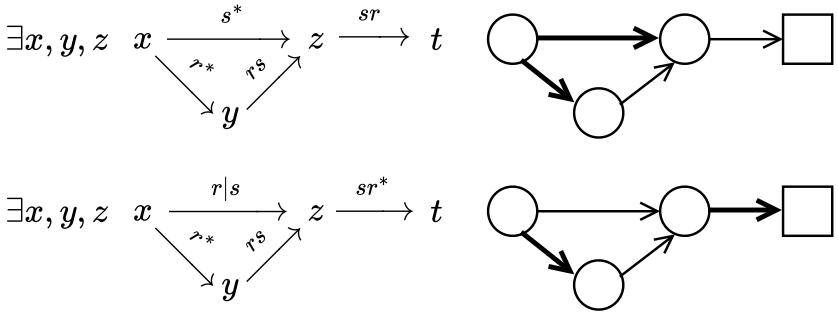

Figura 2.3: A la izquierda un par de consultas y a la derecha sus respectivos esqueletos. El esqueleto de arriba pertenece a $\mathcal{S}_2$ pero no a $\mathcal{S}_1$. El esqueleto de abajo no está en $\mathcal{S}_2$.

Nuestros resultados se resumen de la siguiente manera.

Teorema 38. *Para cada clase $\mathcal{S}$ de esqueletos, para cada $\mathcal{O}_0 \in \{\mathrm{ID}, \mathrm{F1}\}$, para cada lenguaje de consulta* $\mathrm{CRPQ}(\mathcal{S}) \subseteq \mathcal{Q} \subseteq \mathrm{UC2RPQ}(\mathcal{S})$*, tenemos que*

1. *Si $\mathcal{O}_0 \subseteq \mathcal{O} \subseteq$ GNFO, entonces* OMQA *(resp.* gOMQA*) de $\mathcal{Q}$ bajo $\mathcal{O}$ es finitamente controlable si y solo si $\mathcal{S} \subseteq \mathcal{S}_1$;*
2. OMQA *(resp.* gOMQA*) de $\mathcal{Q}$ bajo* TF1 *es finitamente controlable si y solo si $\mathcal{S} \subseteq \mathcal{S}_2$.*

Observe que el Teorema 38 se plantea como una equivalencia general y esta es la razón por la que $\mathcal{O}$ tiene tanto una cota inferior $\{\mathrm{ID}, \mathrm{F1}\}$ como una cota superior GNFO en las hipótesis del ítem 1. Alternativamente, se podría replantear como:
(1.a) Si $\mathcal{O} \subseteq$ GNFO y $\mathcal{S} \subseteq \mathcal{S}_1$, entonces OMQA (resp. gOMQA) de $\mathcal{Q}$ bajo $\mathcal{O}$ es finitamente controlable.
(1.b) Si $\mathcal{O}_0 \subseteq \mathcal{O}$ y $\mathcal{S} \not\subseteq \mathcal{S}_1$, entonces OMQA (resp. gOMQA) de $\mathcal{Q}$ bajo $\mathcal{O}$ no es finitamente controlable.

Notemos que los ítems del Teorema 38 también reflejan entre sí una especie de dualidad de poder expresivo en el siguiente sentido: entre más complejas sean las consultas (como las del ítem 2 en comparación a las de 1), más restringida debe ser la ontología (en 1 hay más opciones para $\mathcal{O}$ que en 2).

Estrategia de la prueba. Dividiremos la demostración del Teorema 38 en dos secciones. Las implicaciones de derecha a izquierda de ambos ítems del Teorema 38 (demostradas en la Sección 2.3) se cumplen por reducciones al problema de satisfacibilidad de GNFO. Dado que GNFO tiene la propiedad de modelo finito, se obtiene la controlabilidad finita. Estas reducciones no son directas y se dan mediante el siguiente esquema, donde $\mathcal{S}_0$ es una clase de esqueletos por definir que está estrictamente contenida en $\mathcal{S}_1$:

(i) gOMQA de UC2RPQ($\mathcal{S}_2$) bajo reglas TF1 se reduce a gOMQA de UC2RPQ($\mathcal{S}_1$) bajo reglas TF1 (Proposición 42);

(ii) gOMQA de UC2RPQ($\mathcal{S}_1$) bajo GNFO se reduce a gOMQA de UC2RPQ($\mathcal{S}_0$) bajo GNFO (Proposición 41);

(iii) gOMQA de UC2RPQ($\mathcal{S}_0$) bajo GNFO se reduce al problema de satisfacibilidad (finita) de GNFO (Lema 40), y por lo tanto, es finitamente controlable.

En particular, para (i) necesitamos trabajar con la versión *generalizada* de OMQA. De hecho, no es claro para nosotros cómo reducir consultas de UC2RPQ($\mathcal{S}_2$) a UC2RPQ($\mathcal{S}_1$) sin pasar por la versión generalizada y es por ello que hemos definido esta versión de OMQA. Una conclusión rápida de complejidad que podemos obtener con estas reducciones es la siguiente:

Corolario 39. *El problema* (F)OMQA *de* UC2RPQ($\mathcal{S}_0$) *bajo reglas en* GNFO *es decidible y* 2ExpTime *en complejidad combinada.*

La cota de complejidad del Corolario 39 es una consecuencia directa del hecho de que la reducción del Lema 40 es polinomial, en combinación con que el problema de satisfacibilidad de GNFO es 2ExpTime-completo [30]. Las reducciones de los ítems (ii) y (iii) también son polinomiales, por lo que podemos obtener la misma cota para las versiones de OMQA ahí mencionados.

Las implicaciones de izquierda a derecha del Teorema 38 (demostradas en la Sección 2.4) se cumplen por contraejemplos. Es decir, para $(\mathcal{S}, \mathcal{O}) \in \{(\mathcal{S}_1, \text{F1}), (\mathcal{S}_1, \text{ID}), (\mathcal{S}_2, \text{TF1})\}$, y algún $sk \notin \mathcal{S}$, exhibiremos un modelo $\mathcal{G}$, una tupla de elementos $\bar{t}$, una consulta $q \in$ CRPQ con esqueleto sk y un conjunto de restricciones Γ en $\mathcal{O}$ para los cuales $\bar{t}$ está en el conjunto de respuestas $q(\mathcal{G}')$, para toda extensión finita $\mathcal{G}' \supseteq \mathcal{G}$ que satisface Γ, pero $\bar{t}$ no está en el conjunto de respuestas $q(\mathit{Chase}(\mathcal{G}, \Gamma))$.

2.3 Reducciones a GNFO

En esta sección, demostraremos las implicaciones de derecha a izquierda de ambos ítems del Teorema 38. Observe que para esta dirección es suficiente demostrar la controlabilidad finita de gOMQA para UC2RPQ($\mathcal{S}_1$) (resp. UC2RPQ($\mathcal{S}_2$)) bajo GNFO (resp. TF1). Para obtener ambos resultados, necesitamos definir una clase auxiliar de esqueletos. Sea $\mathcal{S}_0$ el conjunto de todos los esqueletos que no contienen un ciclo no dirigido infinito. Notemos que $\mathcal{S}_0$ es un subconjunto propio de $\mathcal{S}_1$.

Lema 40. gOMQA *de* UC2RPQ($\mathcal{S}_0$) *bajo* GNFO *es finitamente controlable.*

El Lema 40 es nuestro resultado técnico principal y su prueba se deja para el final de la sección. Usando esto, podemos demostrar la implicación de derecha a izquierda del ítem 1 del Teorema 38.

Proposición 41. gOMQA *de* UC2RPQ($\mathcal{S}_1$) *bajo* GNFO *es finitamente controlable.*

Demostración. La idea es bastante simple: romper todos los ciclos en la consulta introduciendo nuevas variables libres y aplicar el Lema 40.

Sea $q(\bar{x})$ en UC2RPQ($\mathcal{S}_1$) una consulta con k variables libres $\bar{x} = (x_1, \ldots, x_k)$. Supongamos que la variable x_i ocurre m_i veces en $q(\bar{x})$. Para $k' = \sum_i m_i$ y la tupla $\bar{y} = (y_1^1, \ldots, y_1^{m_1}, \ldots, y_k^1, \ldots, y_k^{m_k})$ de dimensión k', definimos la consulta $q'(\bar{y})$ como el resultado de reemplazar la j-ésima ocurrencia de x_i en $q(\bar{x})$ por y_i^j (que asumimos que no está ocurriendo en q), para $i = 1, \ldots, k$. Cualquier ciclo en $q(\bar{x})$ tiene una variable libre y este ciclo no aparece en $q'(\bar{y})$, dado que en $q'(\bar{y})$ cada variable libre se usa exactamente una vez. Por lo tanto, $q'(\bar{y}) \in$ UC2RPQ($\mathcal{S}_0$). La Figura 2.4 muestra el proceso para una consulta particular.

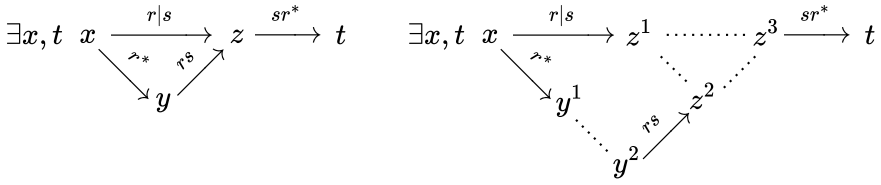

Figura 2.4: A la izquierda la consulta $q(y, z)$ de la Figura 2.2 cuyo esqueleto está en $\mathcal{S}_1$. A la derecha la consulta $q'(y^1, y^2, z^1, z^2, z^3)$ que resulta de aplicar el método de reescritura de la demostración. Las líneas punteadas solo especifican aquellas nuevas variables que son copias de y, z respectivamente y que se interpretarán como la misma constante bajo la asignación ν'. Note que la conexidad del esqueleto original no se preserva necesariamente con el método descrito.

Observe que para cualquier modelo $\mathcal{G}$ y valuación $\nu : \bar{x} \to adom(\mathcal{G})$, tenemos que $\mathcal{G}, \nu \models q$ si y solo si $\mathcal{G}, \nu' \models q'$, donde ν' es la valuación definida por $\nu'(y_i^j) = \nu(x_i)$. Por lo tanto, gOMQA para $q(\bar{x})$ y X es equivalente a gOMQA para $q'(\bar{y})$ y X', con

$$X' = \{(\underbrace{c_1, \ldots, c_1}_{m_1 \text{ veces}}, \ldots, \underbrace{c_k, \ldots, c_k}_{m_k \text{ veces}}) \mid (c_1, \ldots, c_k) \in X\}.$$

Luego, por Lema 40, se sigue el resultado. □

Para el ítem 2 del Teorema 38 reduciremos el problema al ítem 1 por medio de dos hechos: (i) estamos trabajando con la versión *generalizada* gOMQA de OMQA; y (ii) que el Chase bajo reglas TF1 tiene la propiedad (c).

Proposición 42. gOMQA *de* UC2RPQ($\mathcal{S}_2$) *bajo* TF1 *es finitamente controlable.*

Demostración. Reduciremos el problema gOMQA para UC2RPQ($\mathcal{S}_2$) al problema gOMQA para UC2RPQ($\mathcal{S}_1$) (ambos bajo reglas en TF1). Sea $q(\bar{x}) \in$ UC2RPQ($\mathcal{S}_2$) donde $\bar{x} = (x_1, \ldots, x_k)$ son las k variables libres de q, sea $\mathcal{G}$ un modelo finito, sea Γ un conjunto finito de reglas en TF1, y sea X un conjunto finito de k-tuplas de constantes.

Sea Z el conjunto más pequeño de variables de $q(\bar{x})$ que contienen a todas las variables libres y que satisfacen que para cualquier átomo $x \xrightarrow{L} y$ de q donde L es finito, tenemos que $x \in Z$ si y solo si $y \in Z$. Supongamos que Z contiene r variables ligadas $y_1, \ldots, y_r$ de $q(\bar{x})$ y consideremos ahora la consulta $q'(\bar{x}\bar{y})$, con $\bar{y} = (y_1, \ldots, y_r)$, que resulta de convertir a todas las variables en $Z \setminus \bar{x}$ libres. Por ejemplo, en la primera consulta de la Figura 2.3 se liberarían las variables y, z mientras que x permanecería ligada. Observemos que $q'(\bar{x}\bar{y}) \in \text{UC2RPQ}(\mathcal{S}_1)$.

Sea m un número mayor que la longitud de cualquier lenguaje finito en $q(\bar{x})$. Sea U el conjunto de todas las constantes y *nulls* de $\mathit{Chase}(\mathcal{G}, \Gamma)$ a distancia $\leq m$ de una constante en $\mathcal{G}$. Por la propiedad (c), U es finito. Sea $\mathcal{G}^+ = \mathit{Chase}^i(\mathcal{G}, \Gamma)$, con i suficientemente grande para que $\mathcal{G}^+$ contenga a U. Notemos que $\mathcal{G}^+$ contiene a $\mathcal{G}$. Consideremos ahora el conjunto de tuplas de aridad $k + r$ de átomos en $\mathcal{G}^+$, definido como $X' = X \times U^r$. Ahora demostraremos que $(\mathcal{G}, \Gamma, q(\bar{x}), X)$ es una instancia positiva de gOMQA si y solo si también lo es $(\mathcal{G}^+, \Gamma, q'(\bar{x}\bar{y}), X')$. Concluimos la demostración por Proposición 41, por el hecho de que $q'(\bar{x}\bar{y}) \in \text{C2RPQ}(\mathcal{S}_1)$ y que TF1 $\subseteq$ GNFO.

Para la implicación de izquierda a derecha, sea $\mathcal{G}' \supseteq \mathcal{G}^+$ un modelo que satisface Γ. Dado que $\mathcal{G}' \supseteq \mathcal{G}$ sabemos por hipótesis que para alguna k-tupla $\bar{a} \in X$, tenemos que $\mathcal{G}', \{\bar{x} \mapsto \bar{a}\} \models q(\bar{x})$. Esto implica que hay una r-tupla de constantes $\bar{b}$ tal que $\mathcal{G}', \{\bar{x}\bar{y} \mapsto \bar{a}\bar{b}\} \models q'(\bar{x}\bar{y})$. Por la selección de $\bar{y}$, cualquier constante c de $\bar{b}$ está a distancia a lo sumo m de una constante en $\mathcal{G}$ y luego $c \in U$. Por lo tanto, $\bar{a}\bar{b} \in X'$ como queremos.

Para la implicación de derecha a izquierda, observamos primero que $\mathit{Chase}(\mathcal{G}, \Gamma)$ satisface Γ por propiedad (a) del Chase y $\mathit{Chase}(\mathcal{G}, \Gamma) \supseteq \mathcal{G}^+$ por construcción. Por hipótesis sabemos que para alguna k-tupla $\bar{a}$ y alguna r-tupla $\bar{b} = (b_1, \ldots, b_r) \in U^r$ tales que $\bar{a}\bar{b} \in X'$ se cumple que $\mathit{Chase}(\mathcal{G}, \Gamma), \{\bar{x}\bar{y} \mapsto \bar{a}\bar{b}\} \models q'(\bar{x}\bar{y})$. Sea $\mathcal{G}' \supseteq \mathcal{G}$ un modelo que satisface a Γ. Por la propiedad (b) del Chase, tenemos que hay un homomorfismo $h : \mathit{Chase}(\mathcal{G}, \Gamma) \to \mathcal{G}'$ tal que $h(a) = a$ para cada $a \in \mathit{adom}(\mathcal{G})$. Dado que cualquier consulta en UC2RPQ se preserva bajo homomorfismos, tenemos que $\mathcal{G}' \models q'(\bar{a}, h(b_1), \ldots, h(b_r))$ y esto implica que $\mathcal{G}' \models q(\bar{a})$. □

Para finalizar esta sección daremos la demostración del Lema 40, que se obtiene mediante una reducción al problema de satisfacibilidad de GNFO.

Demostración del Lema 40. Para simplificar consideramos que $q(\bar{x})$ está en $\text{CRPQ}(\mathcal{S}_0)$, pero el argumento se puede adaptar fácilmente para $q(\bar{x})$ en $\text{UC2RPQ}(\mathcal{S}_0)$. Sea $\mathcal{G}$ un modelo finito, Γ un conjunto finito de sentencias en GNFO, $q(\bar{x})$ en $\text{CRPQ}(\mathcal{S}_0)$ una consulta con k variables libres $\bar{x} = (x_1, \ldots, x_k)$ y sea $X \subseteq_{\text{fin}} (\mathit{adom}(\mathcal{G}))^k$ un conjunto de k-tuplas. Construiremos una sentencia φ en GNFO tal que

> φ es satisfacible (resp. finitamente satisfacible) si y solo si hay un modelo (resp. modelo finito) $\mathcal{G}' \supseteq \mathcal{G}$ que satisface Γ y tal que $X \cap q(\mathcal{G}') = \emptyset$. (†)

Dado que GNFO tiene la propiedad de modelo finito, estaría lista la demostración. Primero introduciremos algunas nociones necesarias para la prueba. Luego definiremos φ, verificaremos que está en GNFO y que satisface (†).

Regiones. Veamos algunas nociones sobre el esqueleto sk de q. Una región de $sk = (M, \nu, \mu)$ es el esqueleto de un subgrafo conexo de M. Una región finita rg de sk es una región en la cual

$\mu(e) =< \infty$ para cada átomo e en rg. Aunque $sk \in \mathcal{S}_0$ podría contener ciclos, la idea es pensar en sk como un árbol dirigido con etiquetas, identificando cada región finita maximal con un único nodo, que se conecta con otros nodos solo por medio de lenguajes infinitos. En adelante, por región finita nos referimos a una región finita *maximal* de sk y para simplificar y sin pérdida de generalidad, asumimos que sk es conexo. Cuando vemos las conexiones entre regiones finitas de $sk \in \mathcal{S}_0$, estas solo se pueden dar por medio de ejes etiquetados con un lenguaje infinito. En este sentido, el grafo resultante luce como un árbol, como se ve en el ejemplo de la Figura 2.5. La definición de φ se vale de esta estructura para describir, desde las hojas de este árbol hacia arriba, la ejecución de los autómatas correspondientes a los lenguajes regulares que aparecen en q. En el siguiente párrafo introducimos algo de notación para nombrar diferentes partes de este árbol, que se usarán en la definición formal de φ.

Decimos que dos regiones finitas de sk están *conectadas* si hay un eje infinito e tal que su vértice de origen está en una de las regiones y el vértice de llegada está en la otra región. Note que por definición de $\mathcal{S}_0$, una región no se conecta consigo mismo y dos regiones se conectan con a lo sumo un lenguaje infinito (de otro modo sk contendría un ciclo infinito). Además el grafo simple cuyos nodos son las regiones finitas y las aristas son los ejes infinitos que conectan regiones finitas, es un árbol (de otro modo, la existencia de un ciclo en el grafo junto con la conectividad de las regiones, implicaría que hay un ciclo infinito en sk).

Designamos una región $root$ de sk como la *raíz*, y usamos la nomenclatura usual de árboles (padre, hijo, etc.). Para una región rg, denotamos por $\mathrm{ch}(rg)$ al conjunto de todos sus hijos. Observe que si rg es el padre de rg' (es decir, si $rg' \in \mathrm{ch}(rg)$), entonces hay exactamente una variable y' en rg', exactamente una variable y en rg y exactamente un lenguaje regular infinito L tales que $y \xrightarrow{L} y'$ o $y' \xrightarrow{L} y$ está en sk. Denotaremos a las variables y' e y por $\mathrm{src}_{rg'}$ y $\mathrm{tgt}_{rg'}$, respectivamente. Definimos $L_{rg'}$ como L en el caso $y' \xrightarrow{L} y$, y como L^- en el caso $y \xrightarrow{L} y'$. La interfase-a-padre de una región $rg \neq root$ es src_{rg}, es decir, el nodo que conecta a una región con su región padre. La interfase-a-hijo de una región rg es $\{\mathrm{tgt}_{rg'} \mid rg' \in \mathrm{ch}(rg)\}$, es decir, los nodos que conectan a una región con sus regiones hijos. La contracción por regiones de sk con raíz $root$ es el digrafo $\mathcal{T}$ con forma de árbol con etiquetas, cuyo conjunto de nodos $V_{\mathcal{T}}$ es el conjunto de regiones finitas maximales de sk, el conjunto de ejes etiquetados $E_{\mathcal{T}} \subseteq V_{\mathcal{T}} \times V_{\mathcal{T}}$ se define como $(rg', rg) \in E_{\mathcal{T}}$ si y solo si rg es el padre de rg', y para $(rg', rg) \in E_{\mathcal{T}}$ definimos la etiqueta $\ell(rg', rg) = L_{rg'}$. Note que cada eje de $\mathcal{T}$ apunta hacia la raíz $root$. Ver Figura 2.5 para un ejemplo.

A veces nos referimos a una región rg también como un nodo de $\mathcal{T}$. Para facilitar la prueba, reemplazamos algunos átomos $x_i \xrightarrow{L_i} y_i$ con $y_i \xrightarrow{L_i^-} x_i$ en $q(\bar{x})$, para que la nueva dirección concuerde con las de $\mathcal{T}$.

Expresando regiones en primer orden. Para cada átomo $x \xrightarrow{L} y$ de q con L finito, hay una fórmula equivalente $\chi_L(x, y)$ en la lógica de primer orden con igualdad que es positiva, existencial y computable en tiempo lineal. La existencia de esta fórmula se puede garantizar por inducción en la expresión L usando la gramática de las RPQ de la Definición 1 obviando la estrella de Kleene.

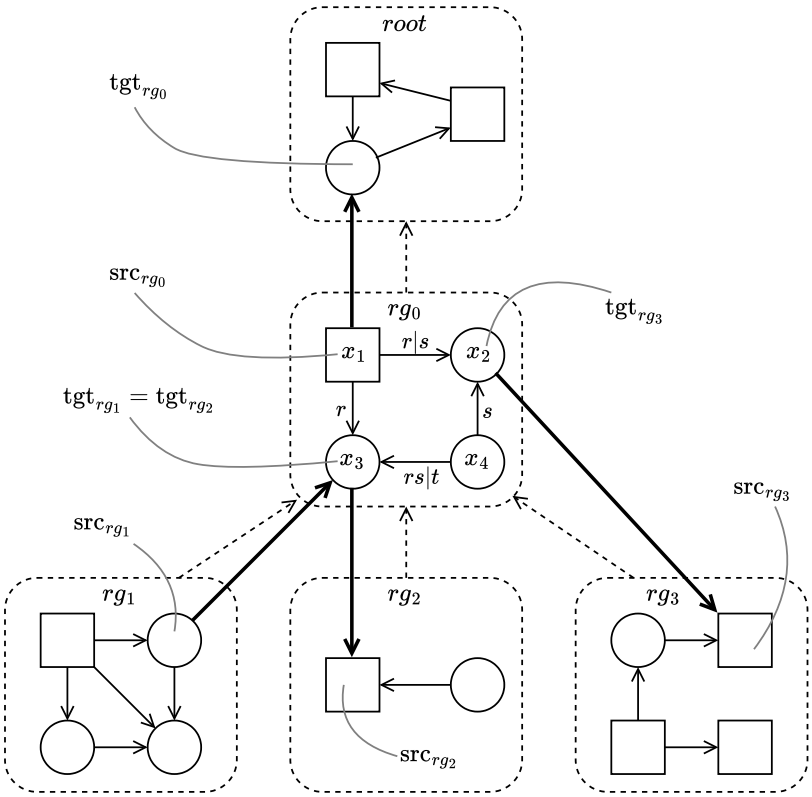

Figura 2.5: Contracción por regiones de un esqueleto en $\mathcal{S}_0$ de alguna consulta q implícita. Las regiones, que también son los nodos de este árbol, están representados por las cajas con borde rayado. Las aristas de este árbol se corresponden con los átomos infinitos de q, que conectan las distintas regiones. El gráfico ilustra también cuáles variables de q son las interfaces con padres e hijos de cada región. Para la región rg_0 tenemos que $\psi_{rg_0} = \exists x_4(\chi_{rs|t}(x_4, x_3) \wedge \chi_s(x_4, x_2) \wedge \chi_r(x_1, x_3) \wedge \chi_{r|s}(x_1, x_2))$

Para cada región $rg \neq root$ de sk definimos la fórmula

$$\psi_{rg}(\bar{b}_{rg}, \bar{w}_{rg}, \text{src}_{rg}) \stackrel{\text{def}}{=} \exists \bar{z}_{rg} \bigwedge_{\substack{t \xrightarrow{L} t' \text{ es un} \\ \text{átomo en } rg}} \chi_L(t, t'), \tag{D}$$

donde $\bar{b}_{rg}$ es la tupla de variables ligadas de rg que están en la interfase-a-hijo de rg, $\bar{w}_{rg}$ es la tupla de variables libres de rg, src_{rg} es la interfase-a-padre de rg, y $\bar{z}_{rg}$ es la tupla de variables ligadas en rg que no están en $\bar{b}_{rg}$. Ver Figura 2.5 para un ejemplo. Cuando $rg = root$ definimos ψ_{rg} de manera similar, excepto que la variable src_{rg} (la cual no está definida) se omite y escribimos sencillamente $\psi_{root}(\bar{b}_{root}, \bar{w}_{root})$. Si no hay átomos en rg definimos ψ_{rg} simplemente como $\top$.

Las variable libres de ψ_{rg} están entre las de $\bar{b}_{rg}, \bar{w}_{rg}, \text{src}_{rg}$. Observe que bajo esta notación la tupla de variables libres podría tener repeticiones: en el caso cuando la interfase-a-padre de rg es también una interfase-a-hijo (es decir, si $\text{src}_{rg} \in \bar{b}_{rg}$), o si src_{rg} es libre (ver Figura 2.5). Note que ψ_{rg} es básicamente la subfórmula de $q(\bar{x})$ cuyo correspondiente esqueleto es rg, excepto que las variables ligadas en $\bar{b}_{rg}$ aparecen libres en ψ_{rg}, i.e., rg es el esqueleto de $\exists \bar{b}_{rg} \psi_{rg}$.

Sea $adom(\mathcal{G}) = \{v_1, \ldots, v_n\}$. Denotamos por $\bar{v}$ a la tupla $(v_1, \ldots, v_n)$. Para cada variable y de $q(\bar{x})$ y k-tupla de constantes $\bar{a} = (a_1, \ldots, a_k) \in (adom(\mathcal{G}))^k$, definimos $\bar{a}[y]$ como la constante a_i cuando $y = x_i$ es una variable libre de $q(\bar{x})$, y como el nombre de variable y en otro caso. Para cada $\bar{a} \in X$, definimos $\psi^{\bar{a}}_{rg}$ como el resultado de reemplazar cada variable $y \in \bar{w}_{rg}$ con $\bar{a}[y]$ en ψ_{rg}. Las variables libres de $\psi^{\bar{a}}_{rg}$ están entre $\bar{b}_{rg}, \mathrm{src}_{rg}$. Explícitamente usamos la notación $\psi^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg})$ para destacar este hecho, aunque src_{rg} podría no estar en esta fórmula.

La fórmula φ. Supongamos que $q(\bar{x}) = \exists \bar{y} \bigwedge_{i \in I} t_i \xrightarrow{L_i} t'_i$. Para cada lenguaje regular L_i, denotamos al autómata finito no determinístico que lo acepta como $\mathcal{A}_i$ y asumimos que tiene un estado inicial q_0 y un estado final q_f. Si para la región rg, $L_{rg} = L_i$, entonces identificamos $\mathcal{A}_{rg}$ con $\mathcal{A}_i$ (recordemos que hemos cambiado la dirección de estos lenguajes para que concuerden con la dirección de los ejes del árbol $\mathcal{T}$, por lo que además tendríamos que $t_i = \mathrm{src}_{rg}$ y $t'_i = \mathrm{tgt}_{rg}$).

Definimos φ sobre la signatura que extiende la de $\mathcal{G}$ (tanto la de constantes como la de predicados) con relaciones unarias $s^{\bar{a}}_{\mathcal{A}_i}$, para cada estado s de $\mathcal{A}_i$ ($i \in I$), y para cada $\bar{a} \in X$. Concretamente, la sentencia φ se define como

$$\varphi \stackrel{\text{def}}{=} \varphi_1 \wedge \varphi_2 \wedge \bigwedge_{\bar{a} \in X} (\varphi^{\bar{a}}_3 \wedge \varphi^{\bar{a}}_4 \wedge \varphi^{\bar{a}}_5).$$

La idea es que si φ se cumple en una estructura $\mathcal{M}$ entonces φ_1 expresa que $\mathcal{G} \subseteq \mathcal{M}$, y φ_2 que $\mathcal{M}$ satisface las restricciones. $\varphi^{\bar{a}}_3$ expresa que cualquier región $rg \neq root$ que puede realizarse en $\mathcal{M}$ y que es una hoja en $\mathcal{T}$ es forzada a empezar el autómata $\mathcal{A}_{rg}$ de su interfase-a-padre; cualquier región que puede realizarse en $\mathcal{M}$ también es forzada a empezar, con la condición de que los autómatas de todos sus hijos hayan alcanzado el estado final. $\varphi^{\bar{a}}_4$ expresa que los estados del autómata de cualquier lenguaje infinito en $q(\bar{x})$ son forzados a difundirse sobre $\mathcal{M}$ de acuerdo a ciertas reglas. Finalmente, $\varphi^{\bar{a}}_5$ expresa que si todos los nodos en la interfase-a-hijo de $root$ están en el estado final de los lenguajes correspondientes, entonces $root$ no se realiza en $\mathcal{M}$.

Observe que las sentencias φ_1 y φ_2 se codifican en GNFO, como se realizó en la demostración de la Proposición 36. Mostramos ahora cómo definir las subsentencias restantes.

$\varphi^{\bar{a}}_3$: Iniciación de autómatas. Para cualquier $\bar{a} \in X$ y región rg, sea

$$\theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg}) \stackrel{\text{def}}{=} \psi^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg}) \wedge \bigwedge_{rg' \in \mathrm{ch}(rg)} q^{\bar{a}}_{f\,\mathcal{A}_{rg'}}(\bar{a}[\mathrm{tgt}_{rg'}]). \tag{E}$$

Cuando rg es una hoja, la gran conjunción de (E) se define como $\top$. Cuando $rg = root$, simplemente escribimos $\theta^{\bar{a}}_{root}(\bar{b}_{root})$. Para cualquier $\bar{a} \in X$ y $rg \neq root$, definimos

$$\varphi^{\bar{a}}_3 \stackrel{\text{def}}{=} \bigwedge_{rg \neq root} \forall \bar{b}_{rg} \forall \mathrm{src}_{rg} (\theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg}) \Rightarrow q^{\bar{a}}_{0\,\mathcal{A}_{rg}}(\bar{a}[\mathrm{src}_{rg}])).$$

La idea de $\varphi^{\bar{a}}_3$ es que para cada región $rg \neq root$, se debe forzar a que se active el cómputo del autómata correspondiente a la interfase-a-padre de rg, cuando todos los nodos de la interfase-a-hijo de rg llegaron al estado final, y la restricción dada por rg sea satisfecha. Esto se hace para *cualquier* posible asignación de las variables en $\bar{b}_{rg}$ y de src_{rg}, cuando este aparece libre en $\psi^{\bar{a}}_{rg}$. Observe que si rg' es un hijo de rg, tenemos que $\bar{a}(\mathrm{tgt}_{rg'})$ es una constante en $adom(\mathcal{G})$ en caso de que $\mathrm{tgt}_{rg'}$ esté

libre, y es una variable en $\bar{b}_{rg}$ en otro caso. Una situación similar ocurre con src_{rg}: en caso de que src_{rg} sea libre entonces $\bar{a}(\text{src}_{rg}) \in adom(\mathcal{G})$ y el cuantificador $\forall \text{src}_{rg}$ es espurio; sino, $\bar{a}(\text{src}_{rg}) = \text{src}_{rg}$ y está universalmente cuantificado. Como consecuencia de esto, obtenemos que $\varphi_3^{\bar{a}}$ es la conjunción de reglas existenciales TF1, por lo que está en GNFO.

$\varphi_4^{\bar{a}}$: Cómputos de autómatas. Para cualquier $\bar{a} \in X$, definimos

$$\varphi_4^{\bar{a}} \stackrel{\text{def}}{=} \forall x, y \bigwedge_{\substack{L_i \text{ en } q(\bar{x}), |L_i|=\infty, \\ t \xrightarrow{\alpha} s \text{ en } \mathcal{A}_i}} \big(t^{\bar{a}}_{\mathcal{A}_i}(x) \wedge \chi_\alpha(x,y) \implies s^{\bar{a}}_{\mathcal{A}_i}(y)\big).$$

La fórmula $\varphi_4^{\bar{a}}$ fuerza a que se esparzan los estados de los lenguajes infinitos en $q(\bar{x})$ como relaciones unarias sobre los nodos del modelo (los lenguajes finitos ya son tratados en $\varphi_3^{\bar{a}}$). Como χ_α es un átomo, entonces $\varphi_4^{\bar{a}}$ está en TF1 y por lo tanto está en GNFO.

$\varphi_5^{\bar{a}}$: La raíz no termina. Finalmente, para cada $\bar{a} \in X$, definimos $\varphi_5^{\bar{a}} \stackrel{\text{def}}{=} \neg\exists \bar{b}_{root} \theta^{\bar{a}}_{root}(\bar{b}_{root})$. Niega la posibilidad de que todos los nodos en la interfase-a-hijo de *root* puedan alcanzar el estado final y que la región *root* se pueda realizar. Como la negación que aparece en $\varphi_5^{\bar{a}}$ se usa sobre una sentencia existencial y $\theta^{\bar{a}}_{root}$ es una conjunción de átomos, entonces esta fórmula está también en GNFO.

Verificación de (†). Ya verificamos que φ está en GNFO, por ser conjunción de fórmulas en GNFO.

Implicación de izquierda a derecha de (†) Sea $\mathcal{H}$ un modelo que satisface φ y sea $\mathcal{H}'$ el *reducto* de $\mathcal{H}$ a los símbolos de predicado en $\mathcal{G}$. Dado que $\mathcal{H} \models \varphi_1 \wedge \varphi_2$, $\mathcal{H}' \supseteq \mathcal{G}$ y $\mathcal{H}'$ satisface Γ. Es suficiente demostrar que $\bar{a} \notin q(\mathcal{H}')$ para todo $\bar{a} \in X$, dado que $\mathcal{H}$ es finito si y solo si $\mathcal{H}'$ lo es.

Asumamos por contradicción que $\bar{a} \in X$ es una k-tupla de constantes tal que $\mathcal{H}' \models q(\bar{a})$. Entonces hay una valuación ν en $\mathcal{H}'$ que interpreta todas las variables $\bar{b}_{rg}, \text{src}_{rg}$, para todas las regiones rg en sk, de modo que $\mathcal{H}, \nu \models \psi^{\bar{a}}_{rg}(\bar{b}_{rg}, \text{src}_{rg})$ para cualquier rg. Más aún, si rg es el padre de rg' entonces hay un camino con etiqueta w en $\mathcal{H}'$ de $\nu(\text{src}_{rg'})$ a $\nu(\text{tgt}_{rg'})$, con $w \in L_{rg'}$.

Se puede demostrar por inducción (simultánea) en $\mathcal{T}$ que para cualquier región rg:

1. $\mathcal{H}, \nu \models \theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \text{src}_{rg})$,
2. $q_{0\mathcal{A}_{rg}}^{\bar{a}}$ es verdadero en $\mathcal{H}$ en el nodo $\nu(\bar{a}[\text{src}_{rg}])$, y
3. $q_{f\mathcal{A}_{rg}}^{\bar{a}}$ es verdadero en $\mathcal{H}$ en el nodo $\nu(\bar{a}[\text{tgt}_{rg}])$.

El caso básico es cuando rg es una hoja de $\mathcal{T}$, para el cual obtenemos directamente 1 ya que $\theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \text{src}_{rg}) = \psi^{\bar{a}}_{rg}(\bar{b}_{rg}, \text{src}_{rg})$. Como $\mathcal{H} \models \varphi_3^{\bar{a}}$ y $\mathcal{H}, \nu \models \theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \text{src}_{rg})$, entonces $\mathcal{H}, \nu \models q_{0\mathcal{A}_{rg}}^{\bar{a}}(\bar{a}[\text{src}_{rg}])$, lo que se traduce al ítem 2. Dado que src_{rg} y tgt_{rg} son únicos, entonces $\nu(\text{src}_{rg}) = \nu(\bar{a}[\text{src}_{rg}])$ y $\nu(\text{tgt}_{rg}) = \nu(\bar{a}[\text{tgt}_{rg}])$ y sabemos que existe un camino de $\nu(\text{src}_{rg})$ a $\nu(\text{tgt}_{rg})$ con etiqueta $w \in L_{rg}$. Como $\mathcal{H} \models \varphi_4^{\bar{a}}$, los predicados $s^{\bar{a}}_{\mathcal{A}_{rg}}$ asociados a los estados s del autómata $\mathcal{A}_{rg}$ se esparcen en el dominio activo de $\mathcal{H}$, tal como indican las transiciones de $\mathcal{A}_r$ y ya que w es una palabra aceptada por $\mathcal{A}_{rg}$ y $q_{0\mathcal{A}_{rg}}^{\bar{a}}$ es verdadero en el nodo $\nu(\text{src}_{rg})$, entonces $q_{f\mathcal{A}_{rg}}^{\bar{a}}$ es verdadero en el nodo $\nu(\text{tgt}_{rg})$,

cumpliéndose 3. El paso inductivo se demuestra con un argumento similar bajo la hipótesis de que para todos los hijos de una región rg valen 1, 2 y 3.

En particular, el ítem 1 es verdadero para $rg = root$. Dado que $root$ no tiene interfase-a-padre, esto significa que $\mathcal{H}, \nu \models \theta^{\bar{a}}_{root}(\bar{b}_{root})$ y esto contradice que $\mathcal{H} \models \varphi^{\bar{a}}_5$.

Implicación de derecha a izquierda de (†) Asumimos que hay un modelo $\mathcal{H} \supseteq \mathcal{G}$ que satisface Γ tal que $\mathcal{H} \not\models q(\bar{a})$ para cada $\bar{a} \in X$ y definamos el modelo de primer orden $\mathcal{H}'$ como el inducido por $\mathcal{H}$ más alguna interpretación para las relaciones unarias $s^{\bar{a}}_{\mathcal{A}_{rg}}$, para cada $\bar{a} \in X$, región rg de sk y estado s de $\mathcal{A}_{rg}$ de manera que $\mathcal{H}' \models \varphi$.

Definimos la interpretación de $s^{\bar{a}}_{\mathcal{A}_{rg}}$ sobre el dominio activo de $\mathcal{H}$ recursivamente como sigue:

i. Si rg es una hoja en sk entonces $q^{\bar{a}}_{0\mathcal{A}_{rg}}$ es verdadero en $\mathcal{H}'$ en todos los elementos d de $\mathcal{H}'$ tales que para alguna asignación ν sobre las variables $\bar{b}_{rg}, \mathrm{src}_{rg}$, tenemos 1) $\mathcal{H}, \nu \models \theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg})$ y 2) $d = \nu(\mathrm{src}_{rg})$. En pocas palabras, para las regiones rg que sean hojas estamos inicializando los cómputos del autómata $\mathcal{A}_{rg}$ siempre que sea posible.

ii. $s^{\bar{a}}_{\mathcal{A}_{rg}}$ es verdadero en todos los elementos d de $\mathcal{H}'$ tales que hay una transición $t \xrightarrow{\alpha} s$ en $\mathcal{A}_{rg}$ y un elemento c de $\mathcal{H}'$ que satisface $t^{\bar{a}}_{\mathcal{A}_{rg}}$ y $c \xrightarrow{\alpha} d$ en $\mathcal{H}'$.

iii. Si rg no es una hoja en sk ni $root$ entonces $q^{\bar{a}}_{0\mathcal{A}_{rg}}$ es verdadero en $\mathcal{H}'$ en todos los elementos d de $\mathcal{H}'$ tales que para alguna asignación ν sobre las variables $\bar{b}_{rg}, \mathrm{src}_{rg}$, tenemos 1), 2) como en el ítem i. y 3) para todo hijo rg' de rg tenemos que $\nu(\mathrm{tgt}_{rg'})$ satisface $q^{\bar{a}}_{f\mathcal{A}_{rg'}}$. Nuevamente, estamos inicializando los cómputos de $\mathcal{A}_{rg}$ siempre que sea posible.

Por construcción es claro que $\mathcal{H}' \models \varphi_1 \wedge \varphi_2$ y también es claro que para cualquier $\bar{a} \in X$ tenemos que $\mathcal{H}' \models \varphi^{\bar{a}}_4$ por ii. También es directo que si para algún $\bar{a} \in X$, alguna valuación ν y alguna región rg, existen constantes d y e en $\mathcal{H}'$ donde son verdaderos los conceptos $q^{\bar{a}}_{0\mathcal{A}_{rg}}$ y $q^{\bar{a}}_{f\mathcal{A}_{rg}}$, respectivamente, entonces hay un camino de d a e cuya etiqueta está en el lenguaje L_{rg}.

Sea $\bar{a} \in X$ y ν una valuación que interpreta todas las variables de $\bar{b}_{rg}, \mathrm{src}_{rg}$, para todas las regiones rg en sk. Las reglas i. y iii. garantizan que $\mathcal{H}', \nu \models \theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg}) \Longrightarrow q^{\bar{a}}_{0\mathcal{A}_{rg}}(\bar{a}[\mathrm{src}_{rg}])$ para todo $rg \neq root$, por lo que $\mathcal{H}' \models \varphi^{\bar{a}}_3$. Para ver que $\mathcal{H}' \models \varphi^{\bar{a}}_5$, analizamos algunos casos.

- Supongamos que ν es tal que para alguna región rg, ocurre que $\mathcal{H}', \nu \not\models \theta^{\bar{a}}_{rg'}(\bar{b}_{rg'}, \mathrm{src}_{rg'})$ con $rg' \in \mathrm{ch}(rg)$, entonces $\mathcal{H}', \nu \not\models \theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg})$. En efecto, como $\mathcal{H}' \models \varphi^{\bar{a}}_3$, en $d = \nu(\mathrm{src}_{rg'})$ no puede ser verdadero el predicado $q^{\bar{a}}_{0\mathcal{A}_{rg'}}$, por lo que $\mathcal{A}_{rg'}$ nunca inicializó bajo esta valuación (en general, todos los predicados $s^{\bar{a}}_{\mathcal{A}_{rg'}}$ tienen interpretación vacía en $\mathcal{H}'$). En consecuencia, el predicado $q^{\bar{a}}_{f\mathcal{A}_{rg'}}$ tampoco es verdadero en $\nu(\mathrm{tgt}_{rg})$ y así $\mathcal{H}', \nu \not\models \theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg})$ (ver la definición (E) para mayor claridad). Luego, $\mathcal{H}', \nu \not\models \theta^{\bar{a}}_{root}(\bar{b}_{root})$.
- Supongamos que para todo $rg \neq root$, $\mathcal{H}', \nu \models \theta^{\bar{a}}_{rg}(\bar{b}_{rg}, \mathrm{src}_{rg})$. Para todo $rg' \in \mathrm{ch}(root)$, tenemos que $q^{\bar{a}}_{0\mathcal{A}_{rg'}}$ es verdadero en $\nu(\mathrm{src}_{rg'})$. Si además ocurre que para todo $rg' \in \mathrm{ch}(root)$ es verdadero el concepto $q^{\bar{a}}_{f\mathcal{A}_{rg'}}$ en $\nu(\mathrm{tgt}_{rg'})$, entonces $\mathcal{H}', \nu \not\models \psi^{\bar{a}}_{root}(\bar{b}_{root})$, de lo contrario, el reducto de $\mathcal{H}'$ a los símbolos de $\mathcal{G}$, es decir, el modelo $\mathcal{H}$, satisfacería $q(\bar{a})$. Luego, $\mathcal{H}', \nu \not\models \theta^{\bar{a}}_{root}(\bar{b}_{root})$.

Esto demuestra que $\mathcal{H}' \models \varphi^{\bar{a}}_5$, lo que finaliza la prueba de este teorema. □

2.4 Los casos que no son finitamente controlables

Primero veremos que los casos que no son finitamente controlables se pueden hallar para cualquier esqueleto fuera de $\mathcal{S}_1$ bajo reglas FG, incluso si solo consideramos reglas sin constantes. Esto significa que el ítem 1 del Teorema 38 se cumple para cualquier fragmento $\mathcal{T}$ de GNFO que contiene a FG sin constantes.

Proposición 43. *Si $\mathcal{S} \subsetneq \mathcal{S}_1$, entonces* OMQA *de* CRPQ($\mathcal{S}$) *bajo* FG *no es finitamente controlable. Esto se cumple incluso para un conjunto que solo contiene una regla sin constantes.*

Demostración. Demostraremos que si $sk \notin \mathcal{S}_1$ existe un modelo $\mathcal{G}$, una consulta $q \in \text{CRPQ}(\{sk\})$, una tupla $\bar{a}$ de constantes de $\mathcal{G}$ y un conjunto finito Γ de reglas FG tales que

1. $q(\bar{a})$ es falso en algún modelo infinito que extiende a $\mathcal{G}$ y que satisface Γ (en particular, esto se cumple en $\mathit{Chase}(\mathcal{G}, \Gamma)$), y
2. $q(\bar{a})$ es verdadero en todas las extensiones finitas de $\mathcal{G}$ que satisfacen Γ.

Como en la Proposición 42, usaremos el Chase. El esqueleto sk da la forma de q, donde solo tenemos que especificar los lenguajes que queremos usar como etiquetas, respetando la cardinalidad (finita/infinita) de acuerdo al esqueleto. Supongamos que sk tiene variables libres $\bar{x} = x_1, \ldots, x_k$. Debe haber algún eje infinito en un ciclo ligado. En este eje, ponemos s^+, y en el resto de los ejes de este ciclo colocamos t^* para cada eje infinito y ε para cada eje finito. Para el resto de los ejes de sk colocamos r^* o $r \mid \varepsilon$, dependiendo de si son infinitos o finitos, respectivamente. Ver Figura 2.6 para un ejemplo particular.

El conjunto de reglas FG se define como el conjunto que consta solamente de la siguiente fórmula:

$$\forall x, y((x \xrightarrow{r} y \wedge x \xrightarrow{r} x) \Rightarrow \exists z(y \xrightarrow{s} z \wedge x \xrightarrow{r} z \wedge z \xrightarrow{r} x)).$$

Sea $\mathcal{G} = \{v \xrightarrow{r} v\}$ y ν la valuación dada por $\nu(x_i) = v$ para todo i. Se puede demostrar que $\mathit{Chase}(\mathcal{G}, \Gamma)$ es el modelo infinito mostrado en la Figura 2.6. Observe que todos los ejes de $\mathcal{G}$ están etiquetados con r o s, por lo que si $\mathit{Chase}(\mathcal{G}, \Gamma), \nu \models q$, debería haber un ciclo no trivial con etiqueta en el lenguaje s^+ en $\mathit{Chase}(\mathcal{G}, \Gamma)$, lo cual no es cierto. Entonces $\mathit{Chase}(\mathcal{G}, \Gamma), \nu \not\models q$, cumpliéndose el ítem 1.

Para 2, sea $\mathcal{G}'$ un modelo finito que extiende a $\mathcal{G}$ y que satisface Γ. Sea $h : \mathit{Chase}(\mathcal{G}, \Gamma) \to \mathcal{G}'$ un homomorfismo. Debe haber algún $i < j$ tal que $h(n_i) = h(n_j)$. Consideremos la siguiente asignación ν' de variables de q a vértices de $\mathcal{G}'$: todas las variables libres son mapeadas a $h(v) = v$, y todas las demás variables son mapeadas a $h(n_i)$. Demostramos que cada átomo en q es verdadero en $\mathcal{G}'$ bajo ν'.

Supongamos que $x \xrightarrow{L} y$ es un átomo en q. Si x y y son libres en q, entonces L es $r \mid \varepsilon$ o r^*, y $\nu'(x) = \nu'(y) = h(v)$. Por lo que el átomo es verdadero, pues $\varepsilon \in L$. Si x es libre e y ligado en q, entonces L es $r \mid \varepsilon$ o r^*, y $\nu'(x) = h(v)$ y $\nu'(y) = h(n_i)$. El átomo es verdadero, ya que $r \in L$ y en $\mathit{Chase}(\mathcal{G}, \Gamma)$ existe el eje $v \xrightarrow{r} n_i$. Si x está ligado e y es libre, el argumento es similar al caso anterior. Si x e y están ambos ligados en q, entonces $\nu'(x) = \nu'(y) = h(n_i)$. Por construcción, ocurre que $\varepsilon \in L$, en cuyo caso el átomo es verdadero, o $L = s^+$, en cuyo caso el átomo también es verdadero, ya que existe un ciclo no trivial $h(n_i) \xrightarrow{s} h(n_{i+1}) \xrightarrow{s} \cdots \xrightarrow{s} h(n_j) = h(n_i)$ en $\mathcal{G}'$. □

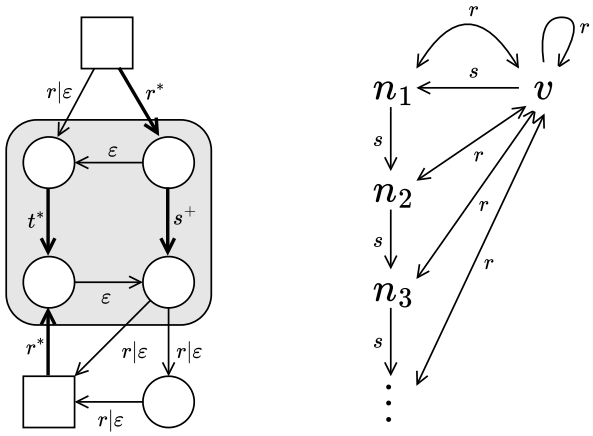

Figura 2.6: A la izquierda la consulta que se construye dado un esqueleto particular *sk* que no está en $\mathcal{S}_1$. A la derecha $Chase(\mathcal{G}, \Gamma)$ para los $\mathcal{G}, \Gamma$ como en las demostraciones de las Proposiciones 43 y 44.

Aceptando el uso de constantes en las reglas, se puede probar de manera similar lo siguiente.

Proposición 44. *Si* $\mathcal{S} \not\subseteq \mathcal{S}_1$, *entonces* OMQA *de* CRPQ($\mathcal{S}$) *bajo* F1 *no es finitamente controlable.*

Demostración. Procedemos como en la prueba de la Proposición 43, solo que ahora el conjunto Γ consta de la regla

$$\forall y(v \xrightarrow{r} y \Rightarrow \exists z(y \xrightarrow{s} z \wedge v \xrightarrow{r} \wedge z \xrightarrow{r} v)),$$

que usa la constante v. El Chase generado es el mismo que en la prueba de la Proposición 43, ver Figura 2.6. □

Observemos que en la demostración anterior el uso de la constante v en las reglas es completamente necesario, ya que por el ítem 2 del Teorema 38, si no permitimos constantes estaríamos en la clase TF1 y bajo esta ontología el problema es finitamente controlable para consultas en UC2RPQ($\mathcal{S}_2$). Notemos que justamente el esqueleto *sk* de la Figura 2.6 pertenece a $\mathcal{S}_2$.

Proposición 45. *Si* $\mathcal{S} \not\subseteq \mathcal{S}_1$, *entonces* OMQA *de* CRPQ($\mathcal{S}$) *bajo* ID *no es finitamente controlable.*

Demostración. Razonamos nuevamente como en la demostración de la Proposición 43. Dado *sk*, fijamos q con esqueleto *sk* definiendo sus átomos como en esa prueba (ver Figura 2.6). Definimos $\mathcal{G} = \{R(v, v, w), v \xrightarrow{s} w, v \xrightarrow{r} v\}$, y la valuación ν para cada variable libre x_i de q se define como $\nu(x_i) = v$. El conjunto de reglas es

$$\Gamma = \{R(x, y, z) \Rightarrow R(x, z, t),\ R(x, y, z) \Rightarrow y \xrightarrow{s} z,$$
$$R(x, y, z) \Rightarrow x \xrightarrow{r} z,\ R(x, y, z) \Rightarrow z \xrightarrow{r} x\}$$

(las variables x, y, z, t están cuantificadas implícitamente). Tenemos que $Chase(\mathcal{G}, \Gamma)$ es el modelo de la Figura 2.7 extendido con los hechos $\{R(v, v, w), R(v, w, n_1)\} \cup \{R(v, n_i, n_{i+1}) \mid i \geq 1\}$. El resto de la prueba es igual al de la Proposición 43. □

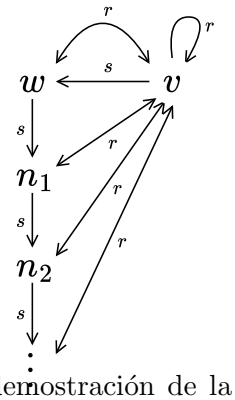

Figura 2.7: *Chase*$(\mathcal{G}, \Gamma)$ para $\mathcal{G}, \Gamma$ en la demostración de la Proposición 45. Notemos que el gráfico muestra realmente a *graph*$(\mathcal{G})$, como se definió en la Sección 2.1. Los hechos que dependen del predicado R quedan implícitos y son como se describen en la demostración.

Este es el único resultado en este capítulo donde requerimos salir del esquema de grafos y utilizar un predicado ternario para el planteamiento del conjunto de restricciones. Note que gran parte de la estrategia de estas demostraciones depende de que podamos establecer los hechos $v \xrightarrow{r} n_i$ y $n_i \xrightarrow{r} v$ para lidiar con los átomos finitos de la consulta q al ser mapeados en $\mathcal{G}'$. Dado que las IDs no admiten constantes y no más de un átomo en el cuerpo y la cabeza de sus fórmulas, una relación ternaria es la alternativa apropiada para generar estos hechos, ya que de esta manera podemos aludir a la relación de la constante v con otro par de constantes sin usar a v explícitamente en las fórmulas. Veamos por último el contraejemplo correspondiente al ítem 2 del Teorema 38.

Proposición 46. *Si* $\mathcal{S} \not\subseteq \mathcal{S}_2$*, entonces* OMQA *de* CRPQ$(\mathcal{S})$ *bajo* TF1 *no es finitamente controlable.*

Demostración. De nuevo, como en la demostración de la Proposición 43. Definimos q con esqueleto sk etiquetando como antes, pero esta vez el ciclo distinguido C solo tiene variables ligadas, las cuales están todas a distancia infinita de cada variable libre. Tomamos $\mathcal{G}$ y ν también como en esa demostración.

El conjunto Γ ahora consta solamente de la regla

$$\forall x, y(x \xrightarrow{r} y \Rightarrow \exists z(y \xrightarrow{r} z \wedge z \xrightarrow{r} y \wedge y \xrightarrow{s} z)).$$

Se sigue que *Chase*$(\mathcal{G}, \Gamma)$ es el modelo finito de la Figura 2.8, y como en las demostraciones anteriores tenemos que *Chase*$(\mathcal{G}, \Gamma) \not\models q$.

Sea $\mathcal{G}'$ un modelo finito que extiende a $\mathcal{G}$ y satisface Γ. Sea h : *Chase*$(\mathcal{G}, \Gamma) \to \mathcal{G}'$ un homomorfismo y sea $i < j$ tal que $h(n_i) = h(n_j)$. Consideremos la siguiente asignación ν': si x es una variable libre o ligada a distancia finita de una variable libre, $\nu'(x) = h(v) = v$. Para el resto de las variables x, $\nu'(x) = h(n_i)$.

Supongamos que $x \xrightarrow{L} y$ es un átomo en q. Si x e y son variables a distancia finita de una variable libre, entonces $x \xrightarrow{L} y$ es verdadero en $\mathcal{G}', \nu'$ ya que por construcción tenemos que $\varepsilon \in L$. Si x está a distancia finita de una variable libre e y no (o viceversa), entonces L es infinito. Notemos que ni x ni y pertenecen a C. Por construcción, $L = r^*$ y entonces el átomo $x \xrightarrow{L} y$ es verdadero en $\mathcal{G}', \nu'$. Supongamos que x e y son variables a distancia infinita de cualquier variable libre. Entonces $\nu'(x) = \nu'(y) = h(n_i)$. Si x e y están ambos fuera de C, entonces $\varepsilon \in L$ y estamos listos. Si uno

de los dos está C y el otro no, el argumento es similar. Si tanto x como y están en C, entonces $\varepsilon \in L$ o $L = s^+$. En ambos casos el átomo $x \xrightarrow{L} y$ es verdadero en $\mathcal{G}', \nu'$. □

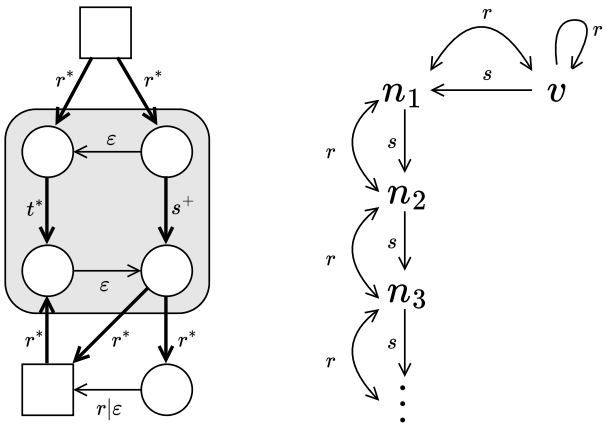

Figura 2.8: A la izquierda la consulta que se construye dado un esqueleto particular *sk* que no está en $\mathcal{S}_2$. A la derecha $Chase(\mathcal{G}, \Gamma)$ para $\mathcal{G}, \Gamma$ como en la demostración de la Proposición 46.

Capítulo 3

Extensiones de PDL mediante propiedades de grafos

La Lógica Proposicional Dinámica, o PDL por sus siglas en inglés, se propuso originalmente como una lógica modal para razonar sobre el comportamiento de programas [88, 89]. Este lenguaje se compone de dos tipos de expresiones: fórmulas, que se interpretan como relaciones unarias sobre estructuras de Kripke (generalizando a la lógica modal); y programas, que se interpretan como relaciones binarias sobre estructuras de Kripke, las cuales se definen siguiendo un esquema de recursión mutua (Sección 3.2). Algunos de los resultados principales obtenidos por Fischer y Ladner son de carácter computacional, dando procedimientos y las respectivas complejidades para decidir model checking y satisfacibilidad de PDL, así como de carácter semántico, demostrando propiedades como el Teorema de Modelo Pequeño (toda fórmula de tamaño n satisfacible se satisface en un modelo de tamaño a lo sumo 2^n), que implica que PDL posee la propiedad de modelo finito, de la que hablamos en el capítulo anterior como una propiedad que se cumple para el lenguaje GNFO. Poco después estos resultados fueron complementados por [146] donde se da una cota superior para la complejidad del problema de satisfacibilidad de PDL en base a procedimientos determinísticos, y por [39] donde se dan resultados análogos para PDL bajo semántica de modelos determinísticos. Otros trabajos que siguen una línea similar a los ya citados son [160], donde extienden a PDL con operadores de *loop* y *converse*, preservando la decibilidad del lenguaje pero no así la propiedad de modelo finito; [66] donde extienden a PDL con el operador *strong loop*, que junto al operador *converse* referenciamos más adelante en este trabajo bajo el nombre de loop-CPDL (Sección 3.3); [65] donde definen IPDL como PDL más el operador de *intersección de programas*, y demuestran que el problema de satisfacibilidad de IPDL es 2ExpTime. Este último resultado fue complementado por [125] donde prueban que el problema de satisfacibilidad de IPDL es de hecho 2ExpTime-completo; cuando se considera intersección junto a *converse* se obtiene ICPDL (Sección 3.3) cuyo problema de satisfacibilidad es decidible vía una reducción a MSO [133], pero este resultado fue mejorado unos años más tarde por [103] donde demuestran que es 2ExpTime-completo. En relación a los lenguajes que hemos utilizado en los capítulos previos, RPQ (Ejemplo 13) se encuentra naturalmente contenido en PDL, y Reg-GXPath y otros lenguajes tratados en [128] comparten varias similitudes semánticas con PDL, que permiten establecer relaciones con las cotas de complejidad de PDL obtenidas por Fischer y Ladner.

Debido en parte a este tipo de adaptaciones, PDL ha sido utilizado en distintas áreas de las ciencias computacionales, en particular, en lógicas de descripción [99], lógicas epistémicas [166], síntesis de programación [143], o incluso ha sido considerado como lenguaje de consulta para bases de dato semi-estructuradas [8, 162], entre otras muchas aplicaciones. En [85] abordamos un enfoque más arraigado a la Teoría de Modelos y propusimos una extensión de PDL, denotada como CPDL$^+$, para la cual demostramos una serie de propiedades sintácticas y semánticas que muestran su amplio poder expresivo, a la vez que se preservan varias propiedades computacionales ya presentes en PDL, todo lo cual será expuesto a lo largo de este capítulo.

Como ya hemos visto, PDL ha sido extendido desde su concepción con varios operadores sintácticos que aumentan su poder expresivo, esto debido tanto a la sencillez de su sintaxis como a la amplia variedad de propiedades fundamentales que posee. Bajo esta noción, gran parte del material existente se podría dividir en dos vertientes cuyos propósitos son: (i) estudiar extensiones más expresivas y decidibles de PDL respecto a operadores compatibles con la sintaxis y semántica usual de PDL, (ii) analizar el límite de la decibilidad de PDL cuando se extiende con operadores no regulares. En particular, los resultados mostrados en este capítulo caen en la primera de estas vertientes. Algunos ejemplos del segundo punto son [51,114,115,129], que si bien difieren de nuestro objetivo, varias de las técnicas utilizadas en estos trabajos son similares a las que aplicaremos más adelante. En la Sección 3.2 introducimos una operación recursiva que denominamos *programa conjuntivo*, que es una forma de realizar un tipo generalizado de clausura conjuntiva sobre PDL, similar a como se construye C2RPQ a partir de 2RPQ (ver Ejemplo 13 o Sección 2.1). Más importante aún, veremos en la Sección 3.3 que otros operadores ya mencionados como *strong loop* e intersección de programas también son abarcables por los programas conjuntivos mediante traducciones polinomiales, lo que garantiza que CPDL$^+$ es un lenguaje que unifica bajo un mismo operador varias clases de lenguajes de consulta, incluso algunas que son incomparables entre sí. Nuestra estrategia en este punto se basa en garantizar la equivalencia de los lenguajes mencionados a fragmentos de CPDL$^+$ que dependen de propiedades topológicas de grafos impuestas sobre los grafos subyacentes de los programas conjuntivos.

Esta nueva extensión de PDL surgió al intentar establecer un criterio de *bisimulación* para ICPDL mediante un juego de múltiples piedras, similar a los planteados para la lógica infinitaria [121]. Las conclusiones de esta idea se abordarán en la Sección 3.4, donde obtenemos una caracterización de varios fragmentos de CPDL$^+$ definidos por propiedades de treewidth acotado (*bounded treewidth*), que es un parámetro que indica qué tan similar es un grafo a un árbol (Sección 3.1). Los fragmentos en cuestión los denotamos por CPDL$^+$(TW$_k$) con $k \in \mathbb{N}$, los cuales forman una *jerarquización estrictamente creciente* de CPDL$^+$, es decir, se cumple que

$$\text{CPDL}^+ = \bigcup_{k \in \mathbb{N}} \text{CPDL}^+(\text{TW}_k)$$

y CPDL$^+$(TW$_{k+1}$) es más expresivo que CPDL$^+$(TW$_k$) para todo $k \geq 2$. Esta última propiedad se demuestra formalmente en la Sección 3.5, por medio de los criterios de simulación planteados en la sección previa y por una propiedad de modelo de ancho de árbol acotado (Corolario 69), lo que funciona como una generalización de un resultado análogo obtenido en [103] para ICPDL y modelos de treewidth 2. Por último, en la Sección 3.6, demostramos que el problema de satisfacibilidad de CPDL$^+$ es decidible, siguiendo un esquema similar al de [103] donde demuestran la decibilidad del

problema de satisfacibilidad de ICPDL por medio de una reducción a un problema de autómatas. La Figura 3.1 muestra algunos de los resultados obtenidos.

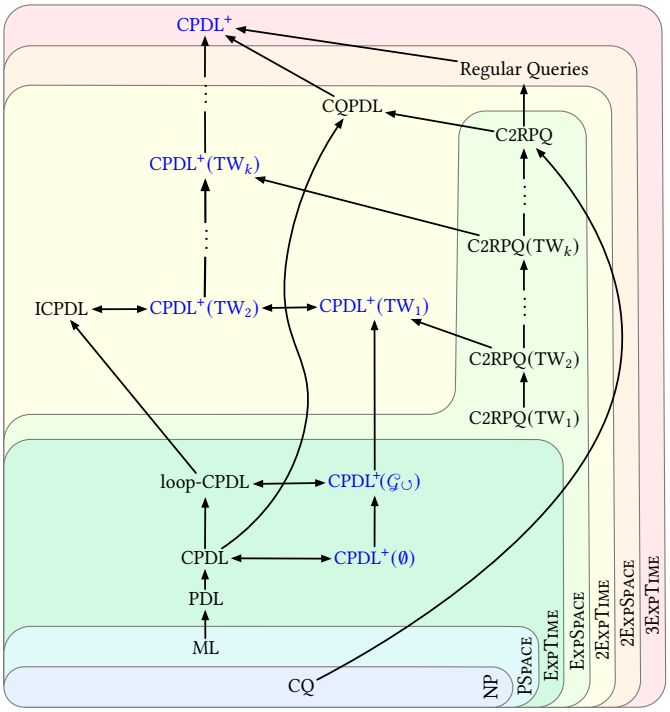

Figura 3.1: Mapa de expresividad y complejidad para lenguajes basados en formalismos de PDL y CQ. Las flechas van en dirección al lenguaje más expresivo, cuyas relaciones se dan por medio de traducciones polinomiales, excepto la que sale de Regular Queries [148], que es exponencial. En los casos de CQ, C2RPQ, Regular Queries y CQPDL nos restringimos a consultas conexas de aridad 1 para establecer la comparación del poder expresivo. Por $\text{C2RPQ}(\text{TW}_k)$ denotamos al conjunto de consultas (conexas) C2RPQ cuyos grafos subyacentes tienen treewidth $\leq k$. Las cotas de complejidad corresponden a las de los problemas de razonamiento asociados a cada formalismo lógico: satisfacibilidad para lógicas cerradas por negación, y *containment* para lenguajes de la familia CQ/C2RPQ. En azul destacamos la familia de lenguajes introducidos en este capítulo.

3.1 Preliminares

Usamos la notación estándar $\bar{u}$ para denotar una tupla de elementos de algún conjunto X y $\bar{u}[i]$ para denotar su i-ésima componente. Si $\bar{q} = (q_1, \ldots, q_k)$ es una tupla y $1 \leq i \leq k$, por $\bar{q}[i \mapsto r]$ denotamos a la tupla $(q_1, \ldots, q_{i-1}, r, q_{i+1}, \ldots, q_k)$. Para una función $f : X \to Y$ usualmente escribimos $f(\bar{u})$ para denotar $(f(\bar{u}[1]), \ldots, f(\bar{u}[k]))$, donde k es la dimensión de $\bar{u}$.

Un grafo (simple, no dirigido) es un par $G = (V, E)$ donde V es un conjunto no vacío de vértices y E es un conjunto de aristas, es decir, conjuntos no vacíos de vértices de tamaño a lo sumo 2. Nos referiremos a V y E como $V(G)$ y $E(G)$, respectivamente. Por lo general asumimos que los grafos son finitos (es decir, que $V(G)$ es finito), salvo que se especifique lo contrario.

Veamos algunos conceptos de grafos que se usarán bastante en este capítulo. Un clique o grafo completo es un grafo G tal que para todo $u, v \in V(G)$, $\{u, v\} \in E(G)$. Dado un grafo G y $B \subseteq V(G)$, decimos que B es subclique de G si $\{\{x, y\} \mid x, y \in B,\ x \neq y\} \subseteq E(G)$. Un árbol es un grafo conexo sin ciclos (y en particular, sin aristas de la forma $\{v\}$).

Una descomposición de árbol de un grafo G es un par $(T, \mathbf{v})$ donde T es un árbol y $\mathbf{v}$ es una función que asigna a cada vértice de T, llamado bolsa, un conjunto de vértices de G, es decir, $\mathbf{v} : V(T) \to \wp(V(G))$. Cuando $x \in \mathbf{v}(b)$ decimos que la bolsa $b \in V(T)$ contiene al vértice x. El tamaño de una bolsa b es el cardinal de $\mathbf{v}(b)$. Además, se deben satisfacer las siguientes tres propiedades:

TD1 cada vértice x de G está contenido en al menos una bolsa de T;

TD2 para cada eje $\{x, y\}$ de G hay al menos una bolsa de T que contiene tanto a x como a y; y

TD3 para cada vértice x de G, el conjunto de bolsas de T que contienen a x define un subgrafo conexo (un *subárbol*) de T.

La descomposición de árbol de un grafo infinito es un árbol infinito y una función que satisfacen las condiciones anteriores. El ancho de $(T, \mathbf{v})$ es el máximo (o supremo, si T es infinito) de los valores $|\mathbf{v}(b)| - 1$ cuando b varía en $V(T)$. El treewidth de G, denotado $\mathrm{tw}(G)$, es el mínimo de todos los anchos de las descomposiciones de árbol de G. Denotamos por TW_k al conjunto de todos los grafos (finitos) de ancho $\leq k$. Ver Figura 3.2 para ejemplos de descomposiciones de árbol.

Intuitivamente, el treewidth de un grafo se entiende como un parámetro que indica qué tan similar es el grafo a un árbol; entre más bajo el valor, más parecido a un árbol es. En particular, TW_1 consta de todos los árboles (y bosques) finitos. Otra propiedad útil para entender algunos ejemplos posteriores es que todo clique de tamaño k está en TW_{k-1}, y si un grafo G contiene un clique de tamaño k entonces $\mathrm{tw}(G) \geq k-1$. Esto último es debido a que los vértices que conforman el clique deben aparecer juntos necesariamente en alguna bolsa de cualquier descomposición de árbol de G.

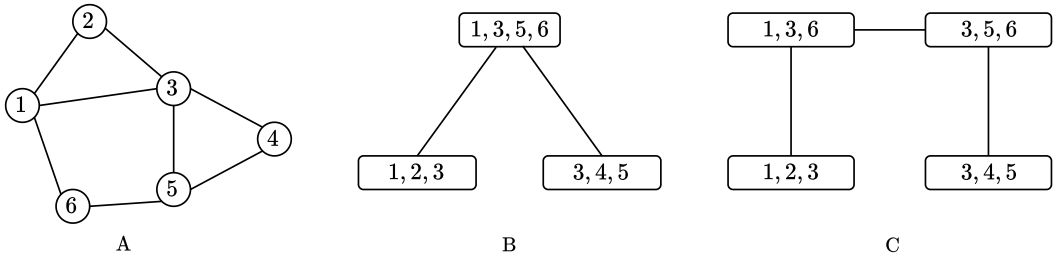

Figura 3.2: A. Un grafo simple; B. Una descomposición de árbol de ancho 3; C. Una descomposición de árbol que demuestra que el treewidth del grafo A es 2 (no puede ser 1 por no ser un árbol). Note que los cliques formados por los vértices $1, 2, 3$ y $3, 4, 5$ deben conformar al menos una bolsa en cualquier descomposición de árbol.

Sean $\mathbb{P}$ y $\mathbb{A}$ conjuntos infinitos numerables y disjuntos dos a dos, que contienen proposiciones atómicas y programas atómicos, respectivamente. Una estructura de Kripke es una tupla

$$K = (X, \{\to_a \mid a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\}), \tag{A}$$

donde X es un conjunto no vacío de mundos denotado por $W(K)$, $\rightarrow_a \subseteq X \times X$ es una relación de transición para cada $a \in \mathbb{A}$, y $X_p \subseteq X$ es una relación unaria para cada $p \in \mathbb{P}$. La distancia entre dos mundos de K es simplemente la longitud del camino más corto en el grafo subyacente de K. Decimos que K es de grado finito si cada mundo de K tiene un número finito de vecinos a distancia 1.

Una subestructura de K es una estructura de Kripke $K' = (X', \{\rightarrow'_a |\ a \in \mathbb{A}\}, \{X'_p \mid p \in \mathbb{P}\})$ tal que $X' \subseteq X$; $\rightarrow'_a \subseteq \rightarrow_a$ para todo $a \in \mathbb{A}$; y $X'_p \subseteq X_p$ para todo $p \in \mathbb{P}$. Dado un subconjunto no vacío $X' \subseteq X$, la subestructura inducida por X' es la subestructura K' de K tal que $W(K') = X'$ y es maximal con esta propiedad.

Sean $K = (X, \{\rightarrow_a |\ a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\})$ y $K' = (X', \{\rightarrow'_a |\ a \in \mathbb{A}\}, \{X'_p \mid p \in \mathbb{P}\})$. Recordemos que un homomorfismo entre K y K' es una función $f : X \to X'$ tal que para todos $w, w' \in X$ tenemos que $w \in X_p$ implica que $f(w) \in X'_p$ y $(w, w') \in \rightarrow_a$ implica que $(f(w), f(w')) \in \rightarrow'_a$, para cada $p \in \mathbb{P}$ y $a \in \mathbb{A}$. Una función $f : \hat{X} \to X'$ con $\hat{X} \subseteq X$ es un homomorfismo parcial si es un homomorfismo de $\hat{K}$ a K', donde $\hat{K}$ es la subestructura de K inducida por $\hat{X}$. Sea $Hom_k(K, K')$ el conjunto de todos los pares de tuplas $(\bar{u}, \bar{v}) \in W(K)^k \times W(K')^k$ tales que $\{\bar{u}[i] \mapsto \bar{v}[i] \mid 1 \leq i \leq k\}$ es un homomorfismo parcial de K en K'. Por conveniencia, asumimos que $Hom_k(K, K')$ posee todas las tuplas homomorfas de menor orden, o en otras palabras, que $Hom_k(K, K') \subseteq Hom_{k+1}(K, K')$ para todo $k \geq 1$.

Por último, definimos el treewidth de una estructura de Kripke como el treewidth de su grafo subyacente. Otras definiciones usuales de grafos como clique, conexidad, camino, etc, también son aplicables a estructuras de Kripke. Varias de estas definiciones ya fueron utilizadas en capítulos anteriores.

3.2 PDL y sus extensiones

Las expresiones de CPDL pueden ser fórmulas φ o programas π, definidos por la siguiente gramática, donde p varía en $\mathbb{P}$ y a en $\mathbb{A}$:

$$\begin{aligned} \varphi &::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\pi\rangle \\ \pi &::= \varepsilon \mid a \mid \bar{a} \mid \pi \cup \pi \mid \pi \circ \pi \mid \pi^* \mid \varphi? \end{aligned} \tag{B}$$

Definimos la semántica de programas $[\![\pi]\!]_K$ y de fórmulas $[\![\varphi]\!]_K$ en una estructura de Kripke K como (A), donde $[\![\pi]\!]_K \subseteq X \times X$ y $[\![\varphi]\!]_K \subseteq X$, como sigue:

$$\begin{aligned} [\![p]\!]_K &:= X_p \text{ para } p \in \mathbb{P}, \\ [\![\neg\varphi]\!]_K &:= X \setminus [\![\varphi]\!]_K, \\ [\![\varphi_1 \wedge \varphi_2]\!]_K &:= [\![\varphi_1]\!]_K \cap [\![\varphi_2]\!]_K, \\ [\![\langle\pi\rangle]\!]_K &:= \{u \in X \mid \exists v \in X.(u, v) \in [\![\pi]\!]_K\}, \\ [\![\varepsilon]\!]_K &:= \{(u, u) \mid u \in X\}, \\ [\![a]\!]_K &:= \rightarrow_a , \\ [\![\bar{a}]\!]_K &:= \{(v, u) \in X^2 \mid (u \rightarrow_a v)\}, \\ [\![\pi_1 \star \pi_2]\!]_K &:= [\![\pi_1]\!]_K \star [\![\pi_2]\!]_K \text{ para } \star \in \{\cup, \circ\}, \end{aligned}$$

$$[\![\pi^*]\!]_K := \text{la clausura reflexiva y transitiva de } [\![\pi]\!]_K,$$
$$[\![\varphi?]\!]_K := \{(u,u) \mid u \in X, u \in [\![\varphi]\!]_K\}.$$

Escribimos $K, u \models \varphi$ para $u \in [\![\varphi]\!]_K$ y $K, u, v \models \pi$ para $(u,v) \in [\![\pi]\!]_K$ y escribimos $\varphi_1 \equiv \varphi_2$ (resp. $\pi_1 \equiv \pi_2$) si $[\![\varphi_1]\!]_K = [\![\varphi_2]\!]_K$ (resp. $[\![\pi_1]\!]_K = [\![\pi_2]\!]_K$) para cada estructura de Kripke K, en cuyo caso decimos que φ_1, φ_2 (resp. π_1, π_2) son equivalentes. A los programas de la forma $\varphi?$ se les denomina *test*. PDL es el fragmento de expresiones en CPDL que no usan programas atómicos de la forma $\bar{a}$.

Algunas otras relaciones que se pueden introducir bajo la semántica dada y que extienden la sintaxis (B) de manera natural son las constantes Booleanas, $\texttt{true} = p \vee \neg p$ con $p \in \mathbb{P}$ y $\texttt{false} = \neg\texttt{true}$; la disyunción de fórmulas $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$; y la modalidad usual $\langle\pi\rangle\varphi = \langle\pi \circ \varphi?\rangle$. También podemos reescribir algunas operaciones ya establecidas como $\varphi \wedge \psi \equiv \langle\varphi?\rangle\psi \equiv \langle\varphi? \circ \psi?\rangle$, que pueden ser útiles más adelante.

CPDL se puede extender de distintas maneras agregando operadores de fórmulas o programas con semánticas particulares. Algunos de los lenguajes más conocidos y utilizados los describimos a continuación.

ICPDL se define como CPDL más intersección de programas $\pi ::= \pi \cap \pi$ cuya semántica está dada por $[\![\pi_1 \cap \pi_2]\!]_K := [\![\pi_1]\!]_K \cap [\![\pi_2]\!]_K$. Para un programa en ICPDL π definimos su inversa π^{-1} como sigue:

$$\begin{aligned} *^{-1} &= * && * \in \{\varepsilon, \varphi?\} \\ a^{-1} &= \bar{a} && a \in \mathbb{A} \\ \bar{a}^{-1} &= a && a \in \mathbb{A} \\ (\pi_1 \star \pi_2)^{-1} &= \pi_2^{-1} \star \pi_1^{-1} && \star \in \{\circ, \cup, \cap\} \\ (\pi^*)^{-1} &= (\pi^{-1})^* \end{aligned}$$

Por inducción estructural, se puede demostrar fácilmente que

Proposición 47. *π es un programa de* ICPDL *si y solo si π^{-1} es un programa de* ICPDL*, y $(u,v) \in [\![\pi]\!]_K$ si y solo si $(v,u) \in [\![\pi^{-1}]\!]_K$ para toda estructura de Kripke K.*

loop-CPDL se define como CPDL más el operador de fórmulas $\varphi ::= \text{loop}(\pi)$ cuya semántica está dada por $[\![\text{loop}(\pi)]\!]_K := \{x \mid (x,x) \in [\![\pi]\!]_K\}$. Notemos que loop-CPDL se puede considerar como un fragmento de ICPDL pues las expresiones $\text{loop}(\pi)$ y $\langle\pi \cap \varepsilon\rangle$ poseen la misma semántica.

Hasta aquí hemos tratado con conceptos ya conocidos y que forman parte de la literatura desde hace un tiempo. A continuación, veremos la teoría desarrollada en [85].

Añadiendo esteroides Consideremos el conjunto de variables Var. Un átomo de programa es una expresión de la forma $\pi(x,x')$ donde $x, x' \in \mathsf{Var}$[1] y π es un programa de (alguna extensión de) CPDL. La idea es usar la "atomización" de programas para definir nuestra extensión de CPDL mediante un operador iterativo especial. Primero, dado un átomo de programa $\pi(x,x')$ definimos $vars(\pi(x,x')) := \{x,x'\}$ y para un conjunto C de átomos de programa definimos $vars(C) := \bigcup_{A \in C} vars(A)$.

[1] Es posible que $x = x'$.

CPDL$^+$ se define como CPDL más programas de la forma $\pi ::= C[x_s, x_t]$ donde:

(1) C es un conjunto finito no vacío de átomos de programa;

(2) $x_s, x_t \in \mathit{vars}(C)$; y

(3) el grafo subyacente $\mathbf{G}_C$ de C es conexo[2], donde $\mathbf{G}_C$ se define como $(V(\mathbf{G}_C), E(\mathbf{G}_C))$, donde $V(\mathbf{G}_C) = \mathit{vars}(C)$ y $E(\mathbf{G}_C) = \{\mathit{vars}(A) \mid A \in C\}$.

Llamamos a este tipo de expresiones programas conjuntivos. Dado que $\{x_s, x_t\} \subseteq \mathit{vars}(C)$, también definimos $\mathit{vars}(C[x_s, x_t]) := \mathit{vars}(C)$. Por otra parte, para $\pi = C[x_s, x_t]$ se define el grafo subyacente $\mathbf{G}_\pi$ igual a $\mathbf{G}_C$ añadiendo el eje $\{x_s, x_t\}$. Ver Figura 3.3 para ejemplos de grafos subyacentes. Note que por recursividad, los conjuntos C se pueden componer a su vez de programas conjuntivos atomizados.

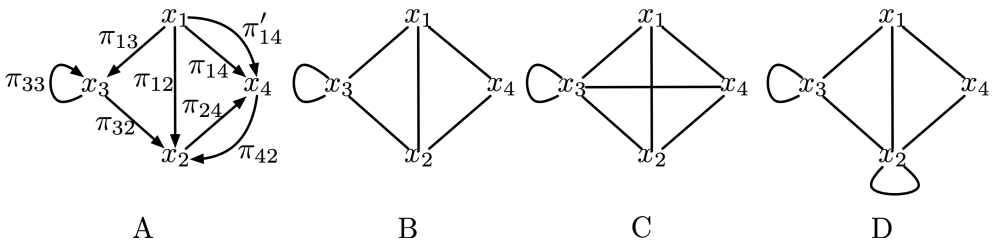

Figura 3.3: A. Representación gráfica del conjunto C que consta de los átomos de programa $\pi_{13}(x_1, x_3), \pi_{12}(x_1, x_2), \pi_{14}(x_1, x_4), \pi'_{14}(x_1, x_4), \pi_{33}(x_3, x_3), \pi_{32}(x_3, x_2), \pi_{24}(x_2, x_4), \pi_{42}(x_4, x_2)$; B. $\mathbf{G}_{\pi_\mathrm{B}}$ con $\pi_\mathrm{B} = C[x_1, x_2]$; C. $\mathbf{G}_{\pi_\mathrm{C}}$ con $\pi_\mathrm{C} = C[x_3, x_4]$; D. $\mathbf{G}_{\pi_\mathrm{D}}$ con $\pi_\mathrm{D} = C[x_2, x_2]$. Observe que $\mathbf{G}_{\pi_\mathrm{B}}, \mathbf{G}_{\pi_\mathrm{D}} \in \mathrm{TW}_2$, pero $\mathbf{G}_{\pi_\mathrm{C}} \in \mathrm{TW}_3 \setminus \mathrm{TW}_2$.

La semántica para programas conjuntivos está dada por interpretaciones. Decimos que una función $f : \mathit{vars}(C) \to W(K)$ es una asignación de satisfacción de C sobre K si $(f(x), f(x')) \in [\![\pi]\!]_K$ para todo átomo de programa $\pi(x, x') \in C$. La semántica $[\![C[x_s, x_t]]\!]_K$ de un programa conjuntivo en una estructura de Kripke K es el conjunto de todos los pares $(f(x_s), f(x_t)) \in W(K) \times W(K)$ donde f es una asignación de satisfacción de C. Para un par de mundos $u, v \in W(K)$ tenemos que $K, u, v \models C[x_s, x_t]$ si y solo si existe una asignación de satisfacción f de C sobre K tal que $f(x_s) = u$ y $f(x_t) = v$. La propiedad (3) permite demostrar la siguiente proposición:

Proposición 48. *Para toda estructura de Kripke K, mundos $u, v \in W(K)$ y programa π de* CPDL$^+$*, si $K, u, v \models \pi$ entonces u, v pertenecen a una misma componente conexa de K.*

Demostración. Para los casos básicos ε y $a, \bar{a}$ con $a \in \mathbb{A}$ esto es obvio. El resto de los casos se deducen directamente de la hipótesis inductiva. Haremos solo el análisis para programas conjuntivos: sea $\pi = C[x_s, x_t]$ y supongamos que $K, u, v \models \pi$. Entonces existe una asignación de satisfacción f de C sobre K tal que $f(x_s) = u$ y $f(x_t) = v$. Por definición, tenemos que $K, f(x), f(x') \models \pi'$ para todo $\pi'(x, x') \in C$, y por hipótesis inductiva, los mundos $f(x), f(x')$ están en una misma componente conexa de K. Como $\mathbf{G}_C$ es conexo y la propiedad de conexidad es transitiva y simétrica,

[2]Para algunos resultados que se demostrarán más adelante podríamos prescindir de la restricción de conexidad para el grafo subyacente de programas conjuntivos, pero la incluimos para establecer relaciones con otras lógicas ya conocidas, como ICPDL y loop-CPDL.

entonces todos los elementos en $\{f(x) \mid x \in vars(C)\}$ pertenecen a una misma componente conexa de K, en particular, esto vale para u y v. □

De la demostración anterior también podemos deducir que si $K, u, v \models \pi$, entonces u y v están conectados por un camino en K que solo usa programas atómicos que aparecen en π.

Note que los programas conjuntivos se interpretan como relaciones *binarias* y los lenguajes lógicos que introduciremos a continuación se limitan a este tipo de expresiones, pero fácilmente podemos extender las ideas ya vistas para lidiar con relaciones de mayor aridad: dado C un conjunto de átomos de programa y $\bar{z}$ una n-tupla de $vars(C)$, la expresión $C[\bar{z}]$ se denomina programa generalizado, cuya semántica viene dada naturalmente como el conjunto $[\![C[\bar{z}]]\!]_K$ que consta de todas las tuplas $f(\bar{z})$ donde f es una asignación de satisfacción de C. El grafo subyacente $\mathbf{G}_{C[\bar{z}]}$ se define igual a $\mathbf{G}_C$ añadiendo los ejes $\{y, y'\}$ para todo par de variables y, y' que aparecen en $\bar{z}$, es decir, la tupla $\bar{z}$ genera un subclique en $\mathbf{G}_{C[\bar{z}]}$. Sobre los programas generalizados también vale la propiedad de la Proposición 48, bajo la suposición de que $\mathbf{G}_C$ sea conexo. Como ya se dijo, este tipo de expresiones no formarán parte de los lenguajes a tratar, pero serán de ayuda en algunas demostraciones.

Para una clase $\mathcal{G}$ de grafos definimos CPDL$^+(\mathcal{G})$ como el fragmento de CPDL$^+$ cuyos programas conjuntivos tienen alguna forma permitida en $\mathcal{G}$, o más formalmente, CPDL$^+(\mathcal{G})$ es CPDL$^+$ restringido a los programas conjuntivos $\pi = C[x_s, x_t]$ tales que $\mathbf{G}_\pi$ es isomorfo a un elemento de $\mathcal{G}$. Por lo general trabajaremos con clases de grafos cerradas por isomorfismos pero en algunos ejemplos usaremos clases finitas de grafos, por simpleza.

Algunas veces haremos alusión a ICPDL$^+$ que es igual a CPDL$^+$ extendido con intersección de programas ($\cap$), cuya semántica es similar a la dada para ICPDL. Observe que cualquier expresión de ICPDL también es una expresión de ICPDL$^+$ que no utiliza programas conjuntivos y cuya semántica se preserva; por lo tanto, ICPDL$^+$ extiende a ICPDL. Análogamente, ICPDL$^+(\mathcal{G})$ se define como CPDL$^+(\mathcal{G})$ extendido con intersección de programas.

Para una expresión e de ICPDL$^+$, denotamos por sub(e) al conjunto de todas las subexpresiones de e que se define como el conjunto más pequeño que satisface lo siguiente:

- $e \in \text{sub}(e)$,
- si $\psi \wedge \psi' \in \text{sub}(e)$ entonces $\{\psi, \psi'\} \subseteq \text{sub}(e)$,
- si $\neg\psi \in \text{sub}(e)$ entonces $\psi \in \text{sub}(e)$,
- si $\langle\pi\rangle \in \text{sub}(e)$ entonces $\pi \in \text{sub}(e)$,
- si $\star \in \{\circ, \cup, \cap\}$ y $\pi \star \pi' \in \text{sub}(e)$, entonces $\{\pi, \pi'\} \subseteq \text{sub}(e)$,
- si $\pi^* \in \text{sub}(e)$, entonces $\pi \in \text{sub}(e)$,
- si $\psi? \in \text{sub}(e)$ entonces $\psi \in \text{sub}(e)$,
- si $C[x_s, x_t] \in \text{sub}(e)$ y $\pi(x, y) \in C$, entonces $\pi \in \text{sub}(e)$.

Si una fórmula ψ está en sub(e) decimos que es una subfórmula de e, y si un programa π está en sub(e) decimos que es un subprograma de e. Una expresión en CPDL$^+$ o en ICPDL$^+$ es positiva si no contiene subfórmulas del tipo $\neg\psi$.

Definimos el tamaño $|e|$ de expresiones e de ICPDL$^+$ por inducción como sigue:

$$
\begin{aligned}
&|\varepsilon| = |p| = |a| = |\bar{a}| := 1 && p \in \mathbb{P} \text{ y } a \in \mathbb{A},\\
&|\pi_1 \star \pi_2| := 1 + |\pi_1| + |\pi_2| && \star \in \{\cup, \cap, \circ\}\\
&|\neg\varphi| = |\varphi?| := 1 + |\varphi|\\
&|\langle\pi\rangle| := 1 + |\pi|\\
&|C[x_s, x_t]| := 1 + \sum_{\pi(x,x') \in C} \big(|\pi| + 2\big)
\end{aligned}
$$

Notemos que en el caso de programas conjuntivos $C[x_s, x_t]$, su tamaño depende implícitamente del cardinal de C.

La introducción de ICPDL$^+$ se hace por razones técnicas, puesto que no añade poder expresivo a CPDL$^+$, como se demuestra a continuación.

Proposición 49. ICPDL$^+$ *y* CPDL$^+$ *son equivalentes en términos de poder expresivo. Más aún, para cualquier clase* $\mathcal{G}$ *que contiene al grafo que consta de un solo eje entre dos nodos distintos se tiene que* ICPDL$^+(\mathcal{G})$ *y* CPDL$^+(\mathcal{G})$ *son también equivalentes en términos de poder expresivo.*

Demostración. $\pi_1 \cap \pi_2$ es equivalente a $\{\pi_1(x,y), \pi_2(x,y)\}[x,y]$ con $x \neq y$. □

Para un par de lenguajes lógicos $\mathcal{L}_1$ y $\mathcal{L}_2$ que admiten expresiones de fórmulas y programas, decimos que $\mathcal{L}_2$ es (al menos) tan expresivo como $\mathcal{L}_1$, denotado $\mathcal{L}_1 \leqq \mathcal{L}_2$, si hay una función Tr que manda fórmulas de $\mathcal{L}_1$ a fórmulas de $\mathcal{L}_2$ y programas de $\mathcal{L}_1$ a programas de $\mathcal{L}_2$ que preserva equivalencias, es decir, tal que para toda estructura de Kripke K, fórmula φ de $\mathcal{L}_1$ y programa π de $\mathcal{L}_1$, ocurre que $[\![\varphi]\!]_K = [\![\operatorname{Tr}(\varphi)]\!]_K$ y $[\![\pi]\!]_K = [\![\operatorname{Tr}(\pi)]\!]_K$. A tales funciones las llamamos traducciones. Escribimos $\mathcal{L}_1 \lneqq \mathcal{L}_2$ para denotar que $\mathcal{L}_1 \leqq \mathcal{L}_2$ y $\mathcal{L}_2 \nleqq \mathcal{L}_1$. Escribimos $\mathcal{L}_1 \equiv \mathcal{L}_2$ para denotar que $\mathcal{L}_1 \leqq \mathcal{L}_2$ y $\mathcal{L}_2 \leqq \mathcal{L}_1$, en cuyo caso decimos que $\mathcal{L}_1$ y $\mathcal{L}_2$ son equiexpresivos. Por ejemplo, de la Proposición 49 obtenemos que ICPDL$^+$ y CPDL$^+$ son equiexpresivos.

La Lógica de Menor Punto Fijo (o LFP por sus siglas en inglés) resulta relevante en este contexto, ya que es bien conocido que con LFP podemos expresar la clausura reflexiva y transitiva de un conjunto, mientras que las demás operaciones de programas como unión, inversión, concatenación, intersección y los programas conjuntivos se pueden interpretar fácilmente con conectivos Booleanos y cuantificadores de primer orden. Por lo tanto, tenemos el siguiente resultado.

Proposición 50. ICPDL$^+ \leqq$ LFP.

A continuación veremos algunas relaciones más interesantes entre varios de los lenguajes ya definidos. Dado que las traducciones pueden fungir como reducciones entre problemas de decisión, muchas veces nos referiremos a estas como *traducciones polinomiales* (resp. *exponenciales*), implicando que vienen acompañadas de un algoritmo determinístico que las computa en tiempo polinomial (resp. exponencial) en el tamaño de la entrada, aunque esto quedará implícito a lo largo del capítulo. En general, nos limitaremos a demostrar que si Tr es una traducción de expresiones del lenguaje $\mathcal{L}_1$ a $\mathcal{L}_2$, entonces el tamaño de $\operatorname{Tr}(e)$ depende polinomialmente (resp. exponencialmente) del tamaño de e.

3.3 Relación de CPDL$^+$ con loop-CPDL e ICPDL

Sea $\mathcal{G}_{\circlearrowleft} := \{G\}$ la clase que solo posee un grafo $G = (\{v\}, \{\{v\}\})$ que consta de un solo nodo v y un lazo en v.

Proposición 51. loop-CPDL *y* CPDL$^+(\mathcal{G}_{\circlearrowleft})$ *son equiexpresivos, vía traducciones polinomiales.*

Demostración. La traducción Tr_1 de loop-CPDL a CPDL$^+(\mathcal{G}_{\circlearrowleft})$ se basa en reemplazar recursivamente cada ocurrencia de una subfórmula $\mathrm{loop}(\pi)$ donde π no usa el operador loop, por $\langle\{\pi(x,x)\}[x,x]\rangle$. Más detalladamente,

$$\begin{aligned}
\mathrm{Tr}_1(*) &= * && * \in \mathbb{P} \cup \{\varepsilon\} \\
\mathrm{Tr}_1(\varphi \wedge \psi) &= \mathrm{Tr}_1(\varphi) \wedge \mathrm{Tr}_1(\psi) \\
\mathrm{Tr}_1(\neg\varphi) &= \neg\, \mathrm{Tr}_1(\varphi) \\
\mathrm{Tr}_1(\langle\pi\rangle) &= \langle \mathrm{Tr}_1(\pi)\rangle \\
\mathrm{Tr}_1(\varphi?) &= \mathrm{Tr}_1(\varphi)? \\
\mathrm{Tr}_1(\pi_1 \star \pi_2) &= \mathrm{Tr}_1(\pi_1) \star \mathrm{Tr}_1(\pi_2) && \star \in \{\circ, \cup\} \\
\mathrm{Tr}_1(\pi^*) &= \mathrm{Tr}_1(\pi)^* \\
\mathrm{Tr}_1(\mathrm{loop}(\pi)) &= \langle\{\mathrm{Tr}_1(\pi)(x,x)\}[x,x]\rangle && \text{(C)}
\end{aligned}$$

Para verificar que Tr_1 preserva la semántica de las expresiones, todos los casos son directos por hipótesis inductiva, aunque desarrollaremos (C) solo para ilustrar el concepto de semántica para programas conjuntivos.

Sea K una estructura de Kripke y $u \in [\![\mathrm{loop}(\pi)]\!]_K$. Por definición, $u \in [\![\mathrm{loop}(\pi)]\!]_K$ si y solo si $(u,u) \in [\![\pi]\!]_K$ y por hipótesis inductiva, esto último ocurre si y solo si $(u,u) \in [\![\,\mathrm{Tr}_1(\pi)]\!]_K$. Definamos $C = \{\mathrm{Tr}_1(\pi)(x,x)\}$ y notemos que la interpretación f tal que $f(x) = u$ es una asignación de satisfacción de C y por lo tanto, $(u,u) \in [\![C[x,x]]\!]_K$, o equivalentemente, $u \in [\![\langle C[x,x]\rangle]\!]_K$, como queríamos demostrar.

Definiendo el tamaño de la fórmula $\mathrm{loop}(\pi)$ como $|\pi| + 1$, se obtiene que $\mathrm{Tr}_1(\pi)$ se computa en tiempo lineal en el tamaño de π.

Para la traducción Tr_1' de CPDL$^+(\mathcal{G}_{\circlearrowleft})$ a loop-CPDL observemos primero que todo programa conjuntivo es de la forma $\{\pi_1(x,x), \ldots, \pi_n(x,x)\}[x,x]$. Si esta expresión es tal que ningún π_i contiene un programa conjuntivo, entonces es equivalente a $(\mathrm{loop}(\pi_1) \wedge \cdots \wedge \mathrm{loop}(\pi_n))?$. Por lo que basta realizar un reemplazo recursivo como en el caso anterior. La traducción Tr_1' se define igual a Tr_1 salvo el caso de la igualdad (C) que se sustituye por

$$\mathrm{Tr}_1'(C[x,x]) = \left(\bigwedge_{\pi(x,x) \in C} \mathrm{loop}(\mathrm{Tr}_1'(\pi)) \right)? \qquad \text{(D)}$$

Veamos que Tr_1' preserva la semántica de los programas conjuntivos. Sea K una estructura de Kripke y $u \in W(K)$ tal que $(u,u) \in [\![C[x,x]]\!]_K$. Luego, $(u,u) \in [\![\pi]\!]_K$ para todo $\pi(x,x) \in C$ y por hipótesis inductiva, esto ocurre si y solo si $(u,u) \in [\![\,\mathrm{Tr}_1'(\pi)]\!]_K$ para todo $\pi(x,x) \in C$. Entonces, $u \in [\![\mathrm{loop}(\mathrm{Tr}_1'(\pi))]\!]_K$ para todo $\pi(x,x) \in C$ y de acá es directo que $(u,u) \in [\![\,\mathrm{Tr}_1'(C[x,x])]\!]_K$, como queríamos demostrar.

Igual que con Tr_1, también sucede que Tr_1' se computa en tiempo lineal. □

Sea $\mathcal{G}_\cap := \{G\}$ la clase que solo posee un grafo $G = (\{v, v'\}, \{\{v, v'\}\})$ que consta de un solo eje entre un par de nodos distintos. La idea es demostrar la siguiente proposición.

Proposición 52. ICPDL *y* $\mathrm{CPDL}^+(\mathcal{G}_\cap)$ *son equiexpresivos, mediante traducciones que se pueden computar en tiempo polinomial.*

Sin embargo, en lugar de dar las traducciones en una sola demostración como se hizo para la Proposición 51, vamos a demostrar una serie de lemas que introducirán una clase de fragmentos importantes de CPDL$^+$ más adelante.

Lema 53. $\mathrm{ICPDL} \leqq \mathrm{CPDL}^+(\mathcal{G}_\cap)$, *mediante una traducción polinomial.*

Demostración. La traducción Tr_2 de fórmulas de ICPDL a fórmulas de $\mathrm{CPDL}^+(\mathcal{G}_\cap)$ y de programas de ICPDL a programas de $\mathrm{CPDL}^+(\mathcal{G}_\cap)$ se define como Tr_1 de la Proposición 51 exceptuando la regla (C) que se sustituye por la siguiente regla con $x \neq y$:

$$\mathrm{Tr}_2(\pi_1 \cap \pi_2) = \{\mathrm{Tr}_2(\pi_1)(x, y), \mathrm{Tr}_2(\pi_2)(x, y)\}[x, y]. \tag{E}$$

Por hipótesis inductiva y la Proposición 49 restringida a programas de ICPDL, obtenemos lo deseado. □

Recordemos que TW_k es la clase de todos los grafos que tienen una descomposición de árbol de ancho $\leq k$. Claramente, $\mathrm{CPDL}^+(\mathrm{TW}_k) \leqq \mathrm{CPDL}^+(\mathrm{TW}_{k+1})$ para todo $k \in \mathbb{N}$. Notemos también que $\mathrm{CPDL}^+(\mathcal{G}_\circlearrowleft)$ y $\mathrm{CPDL}^+(\mathcal{G}_\cap)$ son fragmentos de $\mathrm{CPDL}^+(\mathrm{TW}_1)$, porque las clases $\mathcal{G}_\circlearrowleft$ y $\mathcal{G}_\cap$ constan ambos de grafos en TW_1. Unificando todo esto con el Lema 53 queda lo siguiente:

Corolario 54. $\mathrm{ICPDL} \leqq \mathrm{CPDL}^+(\mathcal{G}_\cap) \leqq \mathrm{CPDL}^+(\mathrm{TW}_1) \leqq \mathrm{CPDL}^+(\mathrm{TW}_2)$.

Si demostramos la equiexpresabilidad de ICPDL con $\mathrm{CPDL}^+(\mathrm{TW}_2)$ queda demostrada la Proposición 52. Para ello veamos primero algunos resultados técnicos. El próximo lema establece que cualquier programa conjuntivo $C[x, y]$ en $\mathrm{CPDL}^+(\mathrm{TW}_k)$ puede pensarse como uno cuyo grafo subyacente tiene una descomposición de árbol de ancho $\leq k$ y tal que todas sus bolsas son subcliques de $C[x, y]$.

Lema 55. *Dado un programa conjuntivo* $C'[x, y]$ *en* $\mathrm{CPDL}^+(\mathrm{TW}_k)$, *se puede computar en tiempo polinomial otro programa* $C[x, y]$ *en* $\mathrm{CPDL}^+(\mathrm{TW}_k)$ *tal que* $C[x, y] \equiv C'[x, y]$, $\mathit{vars}(C) = \mathit{vars}(C')$ *y* $\mathbf{G}_{C[x,y]}$ *tiene una descomposición de árbol* $(T, \mathbf{v})$ *de ancho* $\leq k$ *tal que para cualquier* $b \in V(T)$, $\mathbf{v}(b)$ *es subclique de* $\mathbf{G}_{C[x,y]}$.

Demostración. Sea $(T, \mathbf{v})$ una descomposición de árbol de ancho $\leq k$ de $\mathbf{G}_{C'[x,y]}$ y consideremos $A = \mathbb{A} \cap \mathrm{sub}(C'[x, y])$ como el conjunto de programas atómicos que aparecen en C'. Definamos C como $C' \cup D$, donde

$$D = \{((\cup_{a \in A} a) \cup (\cup_{a \in A} \bar{a}))^*(z_1, z_2) \mid b \in V(T),\ z_1, z_2 \in \mathbf{v}(b),\ z_1 \neq z_2\}.$$

Es claro que $\mathit{vars}(D) = \mathit{vars}(C')$ y que $(T, \mathbf{v})$ es una descomposición de árbol de $\mathbf{G}_D$ donde cada bolsa es un subclique en $\mathbf{G}_D$. Dado que $\mathbf{G}_{C'}$ es conexo, por la Proposición 48 y su nota posterior,

sucede que $C[x,y] \equiv C'[x,y]$. Más aún, $(T, \mathbf{v})$ también es una descomposición de árbol de $\mathbf{G}_{C[x,y]}$. Dado que D tiene a lo sumo $|vars(C')|$ átomos de programa y computar una descomposición de árbol de ancho $\leq k$ se puede hacer en tiempo lineal [47], entonces toda la construcción de C se puede realizar en tiempo polinomial. □

El siguiente lema es la clave para obtener una traducción de CPDL$^+$(TW$_2$) a ICPDL:

Lema 56. *Sea C un conjunto finito de átomos de programa de la forma $\pi(z_1, z_2)$, donde π es un programa de* ICPDL*, $z_1, z_2 \in \mathsf{Var}$, $\mathbf{G}_C$ es un clique y $|vars(C)| \leq 3$. Entonces:*

(1) Para cualesquiera $x, y \in vars(C)$, $x \neq y$, existe un programa de ICPDL *$\pi_{C[x,y]}$ tal que $\pi_{C[x,y]} \equiv C[x,y]$.*

(2) Para cualquier $x \in vars(C)$, hay un programa de ICPDL *$\pi_{C[x,x]}$ tal que $\pi_{C[x,x]} \equiv C[x,x]$.*

Además, estas traducciones se pueden hacer en tiempo polinomial.

Demostración. Para $z_1, z_2 \in vars(C)$, con $z_1 \neq z_2$, sea

$$\Pi_{z_1 z_2} = \left(\bigcap_{\pi(z_1,z_2) \in C} \pi \right) \cap \left(\bigcap_{\pi(z_2,z_1) \in C} \pi^{-1} \right)$$

$$\Pi_{z_1} = \left(\bigwedge_{\pi(z_1,z_1) \in C} \langle \pi \cap \varepsilon \rangle \right)?$$

donde π^{-1} es la inversa de π y una conjunción con rango vacío se define por $\langle \varepsilon \rangle$. Note que siempre que $z_1 \neq z_2$, $\Pi_{z_1 z_2}$ nunca es una intersección vacía, porque $\mathbf{G}_C$ es un clique, es decir, siempre hay un átomo de programa $\pi(z_1, z_2)$ o $\pi(z_2, z_1)$ en C. Para $x \neq y$ definamos los programas de ICPDL $\pi_{C[x,y]}$ y $\pi_{C[x,x]}$ como sigue:

- Si $vars(C) = \{x, y, z\}$ con $x \neq z \neq y$, entonces

$$\pi_{C[x,y]} = \Pi_x \circ (\Pi_{xy} \cap (\Pi_{xz} \circ \Pi_z \circ \Pi_{zy})) \circ \Pi_y$$
$$\pi_{C[x,x]} = \langle \pi_{C[x,y]} \rangle?$$

- Si $vars(C) = \{x, y\}$, entonces

$$\pi_{C[x,y]} = \Pi_x \circ \Pi_{xy} \circ \Pi_y$$
$$\pi_{C[x,x]} = \langle \pi_{C[x,y]} \rangle?$$

- Si $vars(C) = \{x\}$, entonces

$$\pi_{C[x,x]} = \Pi_x$$

Se puede demostrar que $\pi_{C[x,y]} \equiv C[x,y]$ y $\pi_{C[x,x]} \equiv C[x,x]$. En efecto, supongamos que $vars(C) = \{x, y, z\}$ y sea K una estructura de Kripke y u, v mundos de K. Supongamos primero que $(u, v) \in [\![C[x,y]]\!]_K$. Por definición, existe una asignación de satisfacción $f : \{x, y, z\} \to W(K)$ tal que $f(x) = u$, $f(y) = v$ y $(f(z_1), f(z_2)) \in [\![\pi]\!]_K$ para todo $\pi(z_1, z_2) \in C$. En particular, esto implica que $(f(z_1), f(z_2)) \in [\![\Pi_{z_1 z_2}]\!]$ para todo $z_1, z_2 \in vars(C)$, de donde se obtiene que

$(u, v) = (f(x), f(y)) \in [\![\pi_{C[x,y]}]\!]_K$. Recíprocamente, si $(u, v) \in [\![\pi_{C[x,y]}]\!]_K$, como las fórmulas Π_{z_1} son tests por definición de $\pi_{C[x,y]}$ obtenemos que

$$(u, u) \in [\![\Pi_x]\!]_K, \quad (u, v) \in [\![\Pi_{xy} \cap (\Pi_{xz} \circ \Pi_z \circ \Pi_{zy})]\!]_K, \quad (v, v) \in [\![\Pi_y]\!]_K.$$

Desarrollando la inclusión de en medio, obtenemos que existe un mundo $w \in W(K)$ tal que

$$(u, v) \in [\![\Pi_{xy}]\!]_K, \quad (u, w) \in [\![\Pi_{xz}]\!]_K, \quad (w, w) \in [\![\Pi_z]\!]_K, \quad (w, v) \in [\![\Pi_{zy}]\!]_K.$$

Claramente, la función $f : \{x, y, z\} \to W(K)$ dada por $f(x) = u, f(y) = v, f(z) = w$ es una asignación de satisfacción de C sobre K. El resto de los casos por tratar salen de manera completamente análoga. □

La Figura 3.4 ilustra un ejemplo de cómo construir los $\pi_{C[x,y]}$ y $\pi_{C[x,x]}$ del Lema 56. El caso general podría involucrar situaciones un poco más complejas, como que en el programa conjuntivo hay varios átomos de programa $\pi_1(z_1, z_2), \ldots, \pi_n(z_1, z_2)$ para un mismo par de variables z_1, z_2, o de la forma $\pi(z_1, z_1)$ que no son mostrados en el ejemplo. A veces incluso se podría necesitar 'invertir' algún subprograma para obtener el programa ICPDL deseado, como sucede en la definición de la fórmula $\Pi_{z_1 z_2}$ de la demostración anterior.

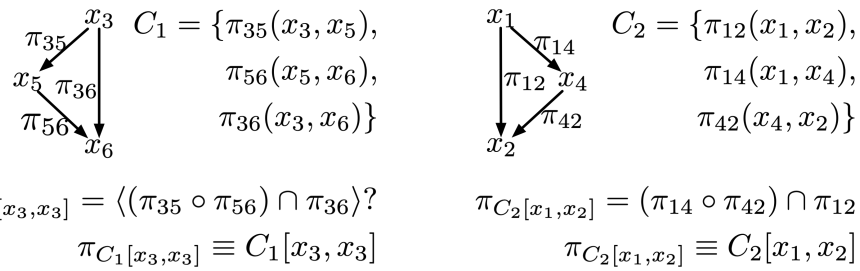

Figura 3.4: Ejemplo de las traducciones hechas en la demostración del Lema 56 donde se asume que cada subprograma π_{ij} es un programa de ICPDL.

Usando el Lema 56 se puede mostrar la traducción de ICPDL$^+$(TW$_2$) a ICPDL$^+$. Por razones técnicas, en este punto decidimos trabajar con ICPDL$^+$ en lugar de CPDL$^+$ ya que simplificará las cuentas.

Lema 57. ICPDL$^+$(TW$_2$) $\leqq$ ICPDL.

Demostración. Sea e un expresión en ICPDL$^+$(TW$_2$) y sean $c_1(e) = \bigcup_{C[x,y] \in \text{sub}(e)} vars(C)$, $c_2(e)$ el conjunto de subexpresiones de e que no son expresiones en ICPDL, y $c_3(e)$ la complejidad sintáctica de e. Finalmente, sea $c(e) = (|c_1(e)| + |c_2(e)|, c_3(e))$. Definiremos una traducción Tr(e) de fórmulas de ICPDL$^+$(TW$_2$) a fórmulas de ICPDL y de programas de ICPDL$^+$(TW$_2$) a programas de ICPDL dada por recursión en el orden lexicográfico de c.

Para los casos aparte de los programas conjuntivos, Tr se define como Tr$_1$ de la Proposición 51, reemplazando además la condición $\star \in \{\circ, \cup\}$ por $\star \in \{\circ, \cup, \cap\}$. Notemos que si e' es una subexpresión de e, entonces $|c_1(e')| + |c_2(e')| \leq |c_1(e)| + |c_2(e)|$ y $c_3(e') < c_3(e)$, así que por la segunda componente de c e hipótesis inductiva podemos garantizar que Tr está bien definida y preserva la semántica de las expresiones.

Para el caso de un programa conjuntivo $C[x,y]$, supongamos que $C = \{\pi_i(x_i, y_i) \mid i = 1, \ldots, n\}$ y $\mathbf{G}_{C[x,y]} \in \mathrm{TW}_2$. Si todos los π_i son programas de ICPDL y $\mathit{vars}(C) \leq 3$, entonces C satisface las hipótesis del Lema 56. En caso de que $x = y$, existe un programa de ICPDL $\pi_{C[x,x]} \equiv C[x,x]$ y definimos $\mathrm{Tr}(C[x,y]) = \pi_{C[x,x]}$. En caso de que $x \neq y$, existe un programa de ICPDL $\pi_{C[x,y]} \equiv C[x,y]$ y definimos $\mathrm{Tr}(C[x,y]) = \pi_{C[x,y]}$.

En caso contrario, hay un π_i que no es un programa de ICPDL o $|\mathit{vars}(C)| > 3$. Definimos $\mathrm{Tr}(C[x,y]) = \mathrm{Tr}(D[x,y])$ para un conjunto de átomos D que estamos por definir.

(1) Si π_i no es un programa de ICPDL, tomamos $D = C \setminus \{\pi_i(x_i, y_i)\} \cup \{\mathrm{Tr}(\pi_i)(x_i, y_i)\}$, donde por hipótesis inductiva $\mathrm{Tr}(\pi_i)$ es un programa de ICPDL equivalente a π_i. Observe que en este caso (a) $c_1(D[x,y]) \subseteq c_1(C[x,y])$, ya que por construcción no hemos agregado nuevas variables a D; y (b) $c_2(D[x,y]) \subsetneq c_2(C[x,y])$, ya que en D hemos sustituido al menos una expresión de C que no es un programa de ICPDL y no agregamos ninguno nuevo. Por lo tanto, $c(D[x,y]) < c(C[x,y])$.

(2) Sino, todos los π_i son programas de ICPDL pero $\mathit{vars}(C) > 3$. Sea $(T, \mathbf{v})$ una descomposición de árbol de ancho ≤ 2 del grafo subyacente de $C[x,y]$ con una bolsa raíz que contiene a x y a y (lo cual es posible fijar porque $\mathbf{G}_{C[x,y]}$ contiene al eje $\{x,y\}$), tal que las hojas de T son bolsas de tamaño al menos 2 (y a lo sumo 3) que no estén incluidos en la bolsa padre. Por el Lema 55 podemos asumir sin pérdida de generalidad que para toda bolsa b, $\mathbf{v}(b)$ es un subclique de $\mathbf{G}_{C[x,y]}$. Como $\mathit{vars}(C) > 3$, entonces T debe constar de al menos dos bolsas. Escojamos cualquier hoja b de T y sea b' el padre de b en T y denotemos $B = \mathbf{v}(b)$ y $B' = \mathbf{v}(b')$. Consideremos C_B como el conjunto de todos los átomos de programa $A \in C$ tales que $\mathit{vars}(A) \subseteq B$. Observe que C_B satisface las hipótesis del Lema 56.

- Supongamos que $B' \cap B = \{z_1\}$ y $B \supseteq \{z_1, z_2\}$ con $z_1 \neq z_2$. Tomamos D como el conjunto de todos los átomos de programa en C excepto aquellos en C_B, más el programa de ICPDL $\pi_{C_B[z_1,z_1]} \equiv C_B[z_1, z_1]$ del Lema 56. Note que $z_2 \in \mathit{vars}(C) \setminus \mathit{vars}(D)$.
- Supongamos que $B' \cap B = \{z_1, z_2\}$ y $B \supseteq \{z_1, z_2, z_3\}$ con $z_i \neq z_j$ para $i \neq j$. Tomamos D como el conjunto de todos los átomos de programa en C excepto aquellos en C_B, más el programa de ICPDL $\pi_{C_B[z_1,z_2]} \equiv C_B[z_1, z_2]$ del Lema 56. Note que $z_3 \in \mathit{vars}(C) \setminus \mathit{vars}(D)$.

Observe que en ambos puntos $c_1(D[x,y]) \subsetneq c_1(C[x,y])$, y $c_2(D[x,y]) \subseteq c_2(C[x,y])$ (de hecho, $c_2(C[x,y]) = \emptyset$) por lo que $c(D[x,y]) < c(C[x,y])$.

Es directo que $\mathrm{Tr}(C[x,y]) \equiv \mathrm{Tr}(D[x,y])$ en todos los casos analizados. Luego, por hipótesis inductiva aplicada a D, cualquier expresión de ICPDL$^+(\mathrm{TW}_2)$ se ha transformado a una expresión equivalente en ICPDL. Esto concluye la construcción de la traducción. Solo falta ver que existe una fórmula en ICPDL equivalente a $\pi = C[x,y]$ cuyo tamaño es polinomial respecto a $|\pi|$, lo cual realizaremos en una proposición aparte. □

La Figura 3.5 ilustra un ejemplo de traducción de un programa conjuntivo $C[x_1, x_2]$ donde C solo contiene átomos de programa de la forma $\pi(z_1, z_2)$ con π en ICPDL (es decir, el caso (2) de la demostración anterior). Como indica la demostración, dada la descomposición de árbol de $\mathbf{G}_{C[x_1,x_2]}$, se pueden remover sucesivamente las hojas de este árbol hasta obtener una única bolsa. Cada vez que una hoja b se remueve, todos los átomos de programa en C que solo usan variables de

$B = \mathbf{v}(b)$ se remueven para ser sustituidos por un programa en ICPDL que contiene la información dada por B.

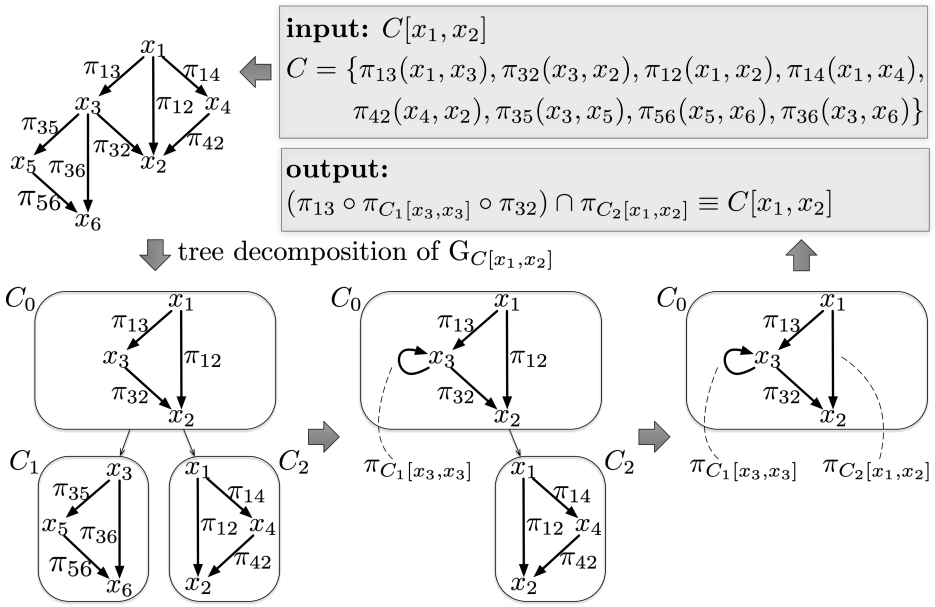

Figura 3.5: Un programa conjuntivo $C[x_1, x_2]$ que se compone solamente de programas de ICPDL y los pasos dados en la demostración del Lema 57 ítem (2) para construir una expresión equivalente de ICPDL dada una descomposición de árbol de treewidth 2 de $\mathbf{G}_{C[x_1,x_2]}$. Note que el árbol de la imagen muestra información adicional que no es parte de la definición de una descomposición de árbol. Ver Figura 3.4 para la definición de $\pi_{C[x_3,x_3]}$ y $\pi_{C[x_1,x_2]}$.

Lema 58. *Para todo programa conjuntivo $C'[x, y]$ de* ICPDL$^+$(TW$_2$)*, existe una fórmula equivalente π de* ICPDL *cuyo tamaño es polinomial en el tamaño de $C'[x_1, x_2]$.*

Demostración. Daremos primero un proceso de traducción para un programa conjuntivo $C'[x, y]$ que satisface el caso (2) de la demostración anterior, es decir, cuando ocurre que C' solo consta de átomos de programa $\pi'(z_1, z_2)$ con π' en ICPDL. Consideremos el Algoritmo 3, que toma como entrada uno de tales programas conjuntivos y devuelve un π en ICPDL equivalente.

El primer par de líneas del algoritmo se pueden realizar en tiempo polinomial porque una descomposición de árbol de ancho ≤ 2 se puede computar en tiempo lineal [47] y $C[x, y]$ se puede definir por Lema 55. Para evaluar los casos de los condicionales asumimos de antemano que T satisface las condiciones impuestas en el ítem (2) (las bolsas de T son de tamaño al menos 2 y ninguna bolsa está contenida en su bolsa padre), lo cual claramente no altera el tiempo de ejecución del algoritmo que seguirá siendo polinomial.

Para un programa π de ICPDL, denotamos por $t(\pi)$ a su complejidad sintáctica (o tamaño) y para un conjunto D de átomos de programa de la forma $\pi(z, z')$ con π en ICPDL, definimos $t(D) = \sum_{\pi(z,z')\in D} t(\pi)$.[3]

[3]Este es un parámetro distinto al del tamaño de un programa conjuntivo.

Algoritmo 3: Traducción de ICPDL$^+$(TW$_2$) a ICPDL para el caso (2)

Datos: $C'[x,y]$ en ICPDL$^+$(TW$_2$) donde C' solo tiene programas $\pi'(z_1,z_2)$ con π' en ICPDL y $z_1, z_2 \in \mathsf{Var}$

Resultado: Un programa π de ICPDL tal que $\pi \equiv C'[x,y]$

```
sea (T, v) una descomposición de árbol de ancho ≤ 2 de G_{C'[x,y]};
sea C[x,y] ≡ C'[x,y] tal que (T, v) es una descomposición de árbol de G_{C[x,y]} y para todo
  b ∈ V(T) es un clique en G_{C[x,y]};
mientras T no sea una bolsa hacer
    sea B una hoja de T;
    sea B' el padre de B en T;
    sea C_B := {π'(z1,z2) ∈ C | {z1,z2} ∈ B} ;
    si B' ∩ B = {z1} y B \ B' ≠ ∅ entonces
        C := C \ C_B ∪ {π_{C_B[z1,z1]}}
    fin
    si B' ∩ B = {z1,z2} (z1 ≠ z2) y B = {z1,z2,z3} (zi ≠ zj) entonces
        C := C \ C_B ∪ {π_{C_B[z1,z2]}}
    fin
    Remover B de T;
fin
si x ≠ y entonces
    devolver π_{C[x,y]}
en otro caso
    devolver π_{C[x,x]}
fin
```

La construcción de los programas de ICPDL $\pi_{C_B[z_1,z_1]}$ y $\pi_{C_B[z_1,z_2]}$ que surgen de aplicar el Lema 56 se pueden realizar en tiempo $\mathcal{O}(t(C_B))$. En efecto, al inspeccionar la definición de estos programas como fue dada en la demostración del Lema 56, se puede observar que existe una constante k tal que para cualesquiera B y $z_1, z_2 \in \mathit{vars}(C_B)$ ($z_1 = z_2$ incluso),

$$t(\pi_{C_B[z_1,z_2]}) \leq k|C_B| + t(C_B).$$

Luego, para la salida $\pi_{C[x,y]}$ del Algoritmo 3 se tiene que $t(\pi_{C[x,y]}) \leq k|C| + t(C)$. Esto ya prueba el enunciado de esta proposición para el caso de programas conjuntivos como en (2), pero aclaramos además que el Algoritmo 3 se ejecuta en tiempo polinomial, ya que como T es de tamaño polinomial en el tamaño de $\mathbf{G}_{C'[x,y]}$, solo tiene una cantidad polinomial de bolsas, por lo que el ciclo del algoritmo solo realiza una cantidad polinomial de iteraciones.

Para el caso general, sean π un programa conjuntivo de ICPDL$^+$(TW$_2$) y $C'[x,y] \in \mathrm{sub}(\pi)$, donde C' solo consta de átomos de programa $\pi'(z_1,z_2)$ con π' un programa en ICPDL. Ejecutamos el Algoritmo 3 en la entrada $C'[x,y]$ para obtener un programa de ICPDL equivalente $\pi_{C'[x,y]}$. Sustituimos $C'[x,y]$ por $\pi_{C'[x,y]}(x,y)$ en π y repetimos el procedimiento. El resultado final es claramente un programa de ICPDL de tamaño polinomial en el tamaño de π. □

Dado que CPDL$^+$(TW$_2$) $\leqq$ ICPDL$^+$(TW$_2$), hemos obtenido la Proposición 52, pero más precisamente, podemos concluir el siguiente corolario debido a los Lemas 53, 54, 57 y 58:

Corolario 59. ICPDL, CPDL$^+(\mathcal{G}_\cap)$, CPDL$^+$(TW$_1$) *y* CPDL$^+$(TW$_2$) *son equiexpresivos, mediante traducciones polinomiales.*

3.4 Criterios de simulación para CPDL$^+$(TW$_k$)

Siguiendo la línea de la sección anterior donde se utilizaron los fragmentos CPDL$^+$(TW$_k$) para caracterizar algunas extensiones conocidas de CPDL, estamos interesados en seguir explorando el alcance del poder expresivo de estos lenguajes. En este sección, caracterizaremos a CPDL$^+$(TW$_k$) mediante una forma de juego de (bi)simulación usando piedras, similar a los juegos de caracterización para fragmentos de la lógica de primer orden con finitas variables [118].

3.4.1 Relación de simulación

Definiremos la noción de k-simulación entre un par (K, K') de estructuras de Kripke mediante un juego de dos jugadores al estilo de Ehrenfeucht-Fraïssé que denotamos $\mathbf{G}[\rightharpoonup_k]$. El tablero del juego tiene como conjunto de posiciones $S \cup D$, con

$$S = \{s\} \times Hom_k(K, K')$$
$$D = \{d_1, \ldots, d_k\} \times (W(K)^k \times W(K')^k)$$

donde a Spoiler le corresponden las posiciones de S y a Duplicador las de D.[4] El conjunto de movimientos o jugadas válidas $\mathbf{G}[\rightharpoonup_k]$ es el conjunto más pequeño que satisface lo siguiente:

J1 Hay un movimiento de $(s, \bar{u}, \bar{v})$ a $(d_i, \bar{u}', \bar{v})$ si $\bar{u}' = \bar{u}[i \mapsto w]$, donde w es un mundo de K a distancia ≤ 1 de $\bar{u}[j]$, para algún $1 \leq j \leq k$ con $i \neq j$; y

J2 Hay un movimiento de $(d_i, \bar{u}', \bar{v})$ a $(s, \bar{u}', \bar{v}')$ si $\bar{v}' = \bar{v}[i \mapsto w]$, donde w es un mundo de K' a distancia ≤ 1 de $\bar{v}[j]$, para algún $1 \leq j \leq k$ con $i \neq j$.

Dada una posición de tipo $(s, \bar{u}, \bar{v})$ o $(d_i, \bar{u}, \bar{v})$, decimos que $\bar{u}$ (resp. $\bar{v}$) es una configuración en K (resp. K'). Visto como un juego, podemos pensar que Spoiler tiene a su disposición k piedras dispuestas sobre algunos mundos de K, lo que determina una configuración $\bar{u}$ en K y Duplicador tiene a su disposición k piedras dispuestas sobre algunos mundos de K', lo que determina una configuración $\bar{v}$ en K'. Dadas estas configuraciones, una jugada tipo **J1** válida de $(s, \bar{u}, \bar{v})$ a $(d_i, \bar{u}', \bar{v})$ quiere decir que Spoiler movió la i-ésima piedra (que se hallaba en $\bar{u}[i]$) al mundo w, obteniendo entonces la nueva configuración $\bar{u}' = \bar{u}[i \mapsto w]$ y d_i indica que ahora le toca jugar a Duplicador y de ser posible debe mover su i-ésima piedra mediante una jugada válida tipo **J2**.

La condición de victoria para Duplicador es cualquier juego infinito, que es una forma de "condición de salvación", que implica determinación (posicional) del juego. Para más información sobre este tipo de juegos referimos la lectura de [108, Chapter 2].

[4] Usamos la notación $(s, \bar{u}, \bar{v})$ en lugar de $(s, (\bar{u}, \bar{v}))$ para los elementos de S y $(d_i, \bar{u}, \bar{v})$ en lugar de $(d_i, (\bar{u}, \bar{v}))$ para los elementos de D.

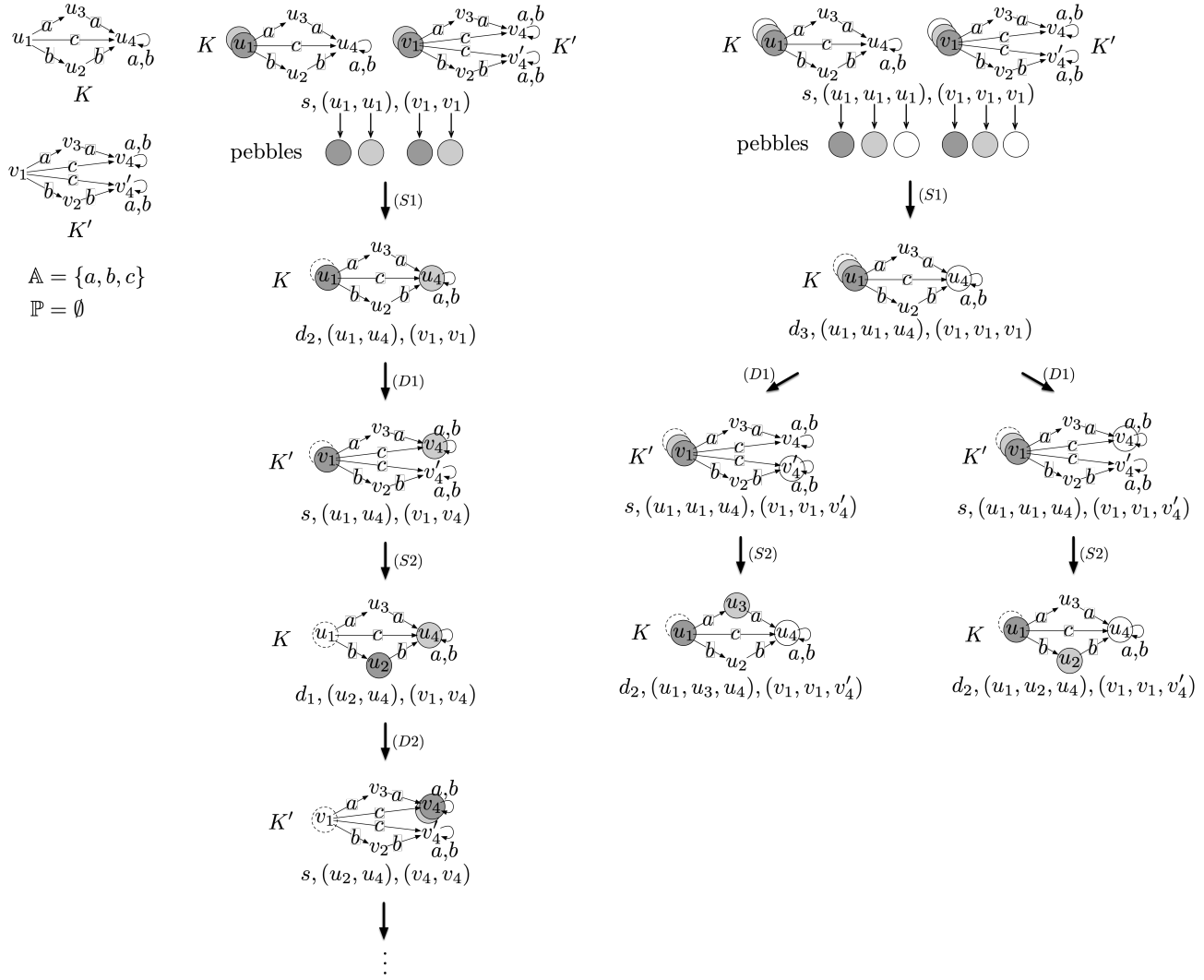

Figura 3.6: *Izquierda*: un par de estructuras de Kripke K y K' definidos con finitos átomos. *Centro*: un ejemplo de un juego $\mathbf{G}[\rightharpoonup_2]$ sobre K y K'. Gráficamente, los vectores en las posiciones se pueden pensar como movimientos de piedras que se encuentran sobre los mundos de las estructuras de Kripke. La notación (Si) indica que jugó Spoiler moviendo la piedra i (representadas por distintos colores especificados en la imagen), y análogo para (Di) indicando el movimiento respuesta de Duplicador. El orden de los movimientos es: (S1), (D1), (S2), (D2). De hecho, Duplicador siempre puede responder cualquier movimiento de Spoiler en $\mathbf{G}[\rightharpoonup_2]$, por lo que $K, u_1 \rightharpoonup_2 K', v_1$. *Derecha*: una representación de árbol de una estrategia ganadora para Spoiler en $\mathbf{G}[\rightharpoonup_3]$ sobre K y K'. Se muestran todas las posibles respuestas de Duplicador, el cual no puede responder después de (S2) en ambas ramas, por lo que $K, u_1 \not\rightharpoonup_3 K', v_1$. Observe que $K, u_1, u_1 \models \pi$ y $K', v_1, v_1 \not\models \pi$ para $\pi = \{c(x_s, x_4), a(x_s, x_3), a(x_3, x_4), b(x_s, x_2), b(x_2, x_4)\}[x_s, x_s]$. Para no sobrecargar la imagen, en ambos desarrollos de juegos solo mostramos la configuración de la jugada hecha por el jugador en turno.

Dadas dos estructuras de Kripke K, K' y tuplas de mundos $\bar{u} \in W(K)^k$ y $\bar{v} \in W(K')^k$, decimos que $K', \bar{v}$ k-simula a $K, \bar{u}$, denotado $K, \bar{u} \rightharpoonup_k K', \bar{v}$, si

C1 $(s, \bar{u}, \bar{v})$ es una posición válida de $\mathbf{G}[\rightharpoonup_k]$ en (K, K') (es decir, $(\bar{u}, \bar{v})$ induce un homomorfismo parcial), y

C2 Duplicador tiene una estrategia ganadora desde $(s, \bar{u}, \bar{v})$.

siempre que todos los mundos en $\bar{u}$ estén en una misma componente conexa de K. Observe que si hay elementos de $\bar{u}$ en distintas componentes conexas de K, entonces $K, \bar{u} \rightharpoonup_k K', \bar{v}$ para todo $\bar{v} \in W(K')^k$. Si todos los elementos de $\bar{u}$ están en una misma componente conexa de K y $K, \bar{u} \rightharpoonup_k K', \bar{v}$, entonces los mundos de $\bar{v}$ también deben pertenecer a una misma componente conexa de K'. Además, por la condición **C2**, ocurre que si Spoiler pasa de la configuración $\bar{u}$ a $\bar{u}'$ aplicando la regla **J1**, entonces Duplicador necesariamente debe poder realizar un movimiento **J2** donde $\bar{v}'$ es tal que $K, \bar{u}' \rightharpoonup_k K', \bar{v}'$. Ver la Figura 3.6 para ejemplos de victorias de Spoiler y de Duplicador.

Nos permitimos el siguiente abuso de notación: Si $\bar{u}_1$ y $\bar{u}_2$ son n-tuplas con $n < k$, escribir $K, \bar{u}_1 \rightharpoonup_k K', \bar{u}_2$ quiere decir realmente que $K, \bar{u}'_1 \rightharpoonup_k K', \bar{u}'_2$, donde $\bar{u}'_i$ es la k-tupla $\bar{u}'_i[j] = \bar{u}_i[j]$ para todo $1 \leq j \leq n$ y $\bar{u}'_i[j] = \bar{u}_i[n]$ para todo $n+1 \leq j \leq k$. En términos de juegos, la tupla $\bar{u}_1$ indica que Spoiler tiene sus primeras n piedras dispuestas en los mundos de $\bar{u}_1$ y las $k-n$ piedras restantes están apiladas con la n-ésima piedra (análogo para Duplicador). Así por ejemplo, si escribimos $K, v \rightharpoonup_k K', v'$ tenemos que Spoiler (resp. Duplicador) tiene todas sus k piedras apiladas sobre el mundo v de K (resp. sobre el mundo v' de K'), o si tenemos $K, u, v \rightharpoonup_k K', u', v'$ entonces Spoiler tiene la primera piedra sobre u y las $k-1$ piedras restantes sobre v (análogo para Duplicador).

Demostraremos ahora uno de los resultados más importantes de este capítulo.

Teorema 60. *Sea $k \geq 2$. Dadas dos estructuras de Kripke K, K' donde K' es de grado finito, y mundos $u, v \in W(K)$ y $u', v' \in W(K')$, las siguientes son equivalentes:*

(1) para cada fórmula positiva φ de $\mathrm{CPDL}^+(\mathrm{TW}_k)$, $K, v \models \varphi$ implica $K', v' \models \varphi$;

(2) $K, v \rightharpoonup_{k+1} K', v'$;

y las siguientes son equivalentes:

(1) para cada programa positivo π de $\mathrm{CPDL}^+(\mathrm{TW}_k)$, $K, u, v \models \pi$ implica $K', u', v' \models \pi$;

(2) $K, u, v \rightharpoonup_{k+1} K', u', v'$.

Más aún, la hipótesis de grado finito solo es necesaria para las implicaciones (1) a (2).

Demostración. *(1) implica (2)* Argumentamos por contrarrecíproco y analizamos primero la implicación para programas.

Supongamos que $K, u, v \not\rightharpoonup_{k+1} K', u', v'$, es decir, u, v están en una misma componente conexa de K y no ocurre **C1** o **C2**. Debemos hallar un programa π en $\mathrm{CPDL}^+(\mathrm{TW}_k)$ tal que $K, u, v \models \pi$ pero $K', u', v' \not\models \pi$. Cuando no ocurre **C1**, es decir, cuando $((u, v), (u', v')) \notin \mathit{Hom}_k(K, K')$, el resultado es bastante directo, basta analizar las finitas posibilidades para que esto último suceda, lo cual ocurre cuando:

- existe un átomo $a \in \mathbb{A}$ tal que $(u, v) \in [\![a]\!]_K$ y $(u', v') \notin [\![a]\!]_{K'}$, por lo que tomamos $\pi = a$; o

- existe $p \in \mathbb{P}$ tal que $u \in [\![p]\!]_K$ y $u' \notin [\![p]\!]_{K'}$ (resp. $v \in [\![p]\!]_K$ y $v' \notin [\![p]\!]_{K'}$), por lo que tomamos $\pi = p? \circ \pi'$ (resp. $\pi = \pi' \circ p?$) donde π' es un camino que conecta a u con v en K.

Cuando no ocurre **C2**, tenemos que hay una estrategia ganadora de Spoiler en un número acotado de rondas en $\mathbf{G}[\rightharpoonup_{k+1}]$ comenzando desde la posición $(s, (u, v), (u', v'))$. Consideremos esta estrategia descrita como un árbol de altura finita cuyos vértices están etiquetados con posiciones de $\mathbf{G}[\rightharpoonup_{k+1}]$. En particular, pedimos que este árbol satisfaga que:

(i) la raíz debe etiquetarse con $(s, (u, v), (u', v'))$,

(ii) cualquier vértice con etiqueta $(d_i, \bar{u}, \bar{v})$ tiene un hijo con etiqueta $(s, \bar{u}, \bar{v}')$ para cada posible movimiento de $(d_i, \bar{u}, \bar{v})$ a $(s, \bar{u}, \bar{v}')$ en $\mathbf{G}[\rightharpoonup_{k+1}]$,

(iii) cualquier vértice con etiqueta $(s, \bar{u}, \bar{v})$ tiene exactamente un hijo $(d_i, \bar{u}', \bar{v})$, tal que el movimiento de $(s, \bar{u}, \bar{v})$ a $(d_i, \bar{u}', \bar{v})$ es válido en $\mathbf{G}[\rightharpoonup_{k+1}]$,

(iv) cada hoja debe tener etiqueta de tipo $(d_i, \bar{u}, \bar{v})$ y no hay movimiento posible en $\mathbf{G}[\rightharpoonup_{k+1}]$ desde esa posición.

Observe que como K' es de grado finito, hay solo un número finito de movimientos que se pueden hacer en $\mathbf{G}[\rightharpoonup_{k+1}]$ desde cualquier posición correspondiente a Duplicador, por lo que el árbol de estrategia de Spoiler tiene ramificación finita y por lo tanto este árbol es finito.

De esta estrategia ganadora de Spoiler, consideremos el árbol T y función $\lambda : V(T) \to W(K)^{k+1}$ que resultan de:

(1) remover todos los vértices y etiquetados con posiciones de Spoiler exceptuando la raíz (es decir, solo mantenemos la configuración inicial (u, v) que le corresponde a Spoiler), y añadir un eje entre el padre de y y el (único) hijo de y, y

(2) proyectar la etiqueta de cada vértice en la componente que corresponde a $W(K)^{k+1}$ (es decir, $\lambda(x) = \bar{u}$ si la etiqueta de x en el árbol de estrategia era de la forma $(s, \bar{u}, \bar{v})$ o $(d_i, \bar{u}, \bar{v})$).

De este árbol se puede construir un programa conjuntivo $C[x_s, x_t]$ en CPDL$^+$(TW$_k$) que contiene todos los *tipos de homomorfismos* de las tuplas. Sea Q el conjunto de todos los pares (y, i), donde y es un vértice de T y $1 \leq i \leq k+1$, y sea $\approx$ la relación de equivalencia más fina sobre Q tal que: (a) $(y, i) \approx (y, i')$ si $\lambda(y)[i] = \lambda(y)[i']$, y (b) $(y, i) \approx (y', i)$ si y' es un hijo de y y $\lambda(y)[i] = \lambda(y')[i]$.[5] El programa conjuntivo C usará una variable por cada clase de equivalencia de $\approx$, las cuales denotamos como $[y, i]$ para cada par (y, i) en Q, y contiene a los siguientes átomos de programa:

(i) $p?([y, i], [y, i])$ si $\lambda(y)[i] \in [\![p]\!]_K$, y

(ii) $a([y, i], [y', j])$ si $y = y'$ y $(\lambda(y)[i], \lambda(y)[j]) \in [\![a]\!]_K$.

La variable x_s (resp. x_t) es la clase $[y, i]$ donde y es la raíz de T y $\lambda(y)[i] = u$ (resp. $\lambda(y)[i] = v$). Ver Figura 3.7 para una ilustración. Existe la posibilidad de que no se hayan agregado átomos de programa que contengan a x_s o x_t, o también podría suceder que $\mathbf{G}_C$ quede disconexo. En tales casos, extendemos a C con el programa conjuntivo $\pi'(x_s, x_t)$ donde π' es un camino que conecta a u con v en K (similar a lo realizado cuando no ocurre **C1**).

Demostraremos ahora algunas propiedades de $C[x_s, x_t]$.

[5] Intuitivamente, la regla (a) relaciona las piedras de Spoiler que ocupan una misma posición en una configuración determinada y (b) indica la permanencia de una piedra en una misma posición de una jugada a otra.

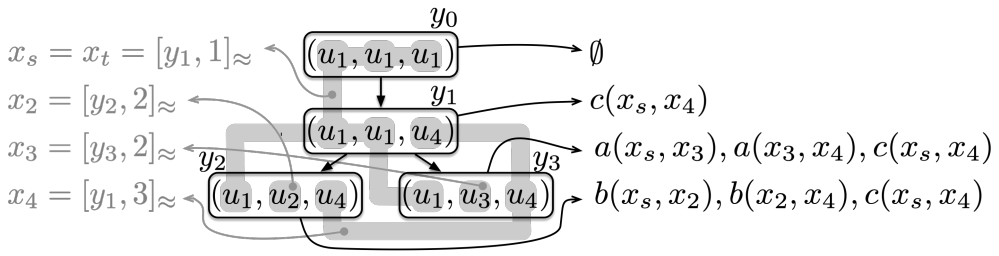

Figura 3.7: Construcción del programa conjuntivo $C[x_s, x_t]$ que resulta de la estrategia ganadora de Spoiler con 3 piedras de la Figura 3.6.

Afirmación. $C[x_s, x_t] \in \text{CPDL}^+(\text{TW}_k)$.

Demostración. Por construcción, $\mathbf{G}_C$ es conexo. Por otra parte, $\mathbf{G}_{C[x_s,x_t]}$ está en TW$_k$ porque (T, λ) induce la descomposición de árbol (T, λ') de ancho $\leq k$, donde $\lambda'(y) = \{[y, i] \mid 1 \leq i \leq k+1\}$ para cada $y \in V(T)$. □

Afirmación. $K, u, v \models C[x_s, x_t]$ *y* $K', u', v' \not\models C[x_s, x_t]$.

Demostración. Para ver que $K, u, v \models C[x_s, x_t]$ consideremos la asignación $f : \mathit{vars}(C) \to W(K)$ dada por $f([y, i]) = \lambda(y)[i]$, la cual está bien definida porque para todo par $(y, i), (y', i') \in Q$, $(y, i) \approx (y', i')$ implica $\lambda(y)[i] = \lambda(y')[i']$ por definición de $\approx$. Luego, f es una asignación de satisfacción de C.

Veamos que $K', u', v' \not\models C[x_s, x_t]$ por contradicción. Si $K', u', v' \models C[x_s, x_t]$, consideremos $f' : \mathit{vars}(C) \to W(K')$ la correspondiente asignación de satisfacción, la f del párrafo anterior y la descomposición de árbol (T, λ') de la afirmación anterior. Para cada $y \in V(T)$ definamos C_y como el conjunto de todos los átomos de programa $\pi(x, x')$ en C tales que $x, x' \in \lambda'(y)$. Tenemos que f (resp. f') es una asignación de satisfacción de C si y solo si f (resp. f') es una asignación de satisfacción de C_y para todo $y \in V(T)$. Definamos además para cada $y \in V(T)$ las tuplas

$$\bar{u}_y = (f([y, 1]), \ldots, f([y, k+1]))$$
$$\bar{v}_y = (f'([y, 1]), \ldots, f'([y, k+1]))$$

de mundos de K y K', respectivamente. Como C_y codifica la subestructura de K generada por la tupla $\bar{u}_y$, cuando sustituimos la componente $\bar{u}_y[i]$ por la correspondiente variable $[y, i]$,[6] obtenemos entonces que $(\bar{u}_y, \bar{v}_y) \in \mathit{Hom}_k(K, K')$ para todo $y \in V(T)$.

Sea $y \to y'$ una arista de T. Vamos a ver que esta arista se corresponde con algunos movimientos válidos en $\mathbf{G}[\rightharpoonup_{k+1}]$ que se utilizan en el árbol de estrategia de Spoiler. Si y es el nodo raíz de T, $\bar{u}_y$ (resp. $\bar{v}_y$) se corresponde con la configuración u, v de Spoiler (resp. u', v' de Duplicador), y por lo tanto, la arista $y \to y'$ proviene de la primera jugada realizada por Spoiler en su árbol de estrategia. Esto implica que $\bar{u}_y$ y $\bar{u}_{y'}$ son iguales salvo en alguna componente $1 \leq i \leq k+1$. Como $(\bar{u}_{y'}, \bar{v}_{y'}) \in \mathit{Hom}_k(K, K')$, entonces $\bar{v}_y$ y $\bar{v}_{y'}$ son iguales salvo en la componente i-ésima. Por

[6] Este fue el proceso realizado durante la construcción del programa $C[x_s, x_t]$.

definición de $\mathbf{G}[\rightharpoonup_{k+1}]$ tenemos que los movimientos

$$\begin{array}{l} \text{de } (s, \bar{u}_y, \bar{v}_y) \text{ a } (d_i, \bar{u}_{y'}, \bar{v}_y) \qquad \text{y} \\ \text{de } (d_i, \bar{u}_{y'}, \bar{v}_y) \text{ a } (s, \bar{u}_{y'}, \bar{v}_{y'}) \end{array} \tag{F}$$

son válidos, y por lo tanto, deben aparecer en el árbol de estrategia de Spoiler. Más general, supongamos que a las configuraciones $\bar{u}_y$ y $\bar{v}_y$ llegamos por medio de una serie de jugadas que aparecen en el árbol de estrategia de Spoiler. Igual que antes, la diferencia entre $\bar{u}_y$ y $\bar{u}_{y'}$, y entre $\bar{v}_y$ y $\bar{v}_{y'}$ es debido a una misma componente i, por lo que en $\mathbf{G}[\rightharpoonup_{k+1}]$ son válidos algunos movimientos tipo (F) que además deben aparecer en el árbol de estrategia de Spoiler por definición.

Notemos que el análisis anterior es válido incluso si y' es una hoja de T, pero esto es una contradicción, porque en el árbol de estrategia de Spoiler no deberían existir nodos etiquetados con jugadas hechas por Duplicador por debajo de las de tipo $(d_i, \bar{u}_{y'}, \bar{v}_y)$. □

Observemos que técnicamente C podría tener *infinitos* átomos de programa, esto porque entre dos mundos de una estructura de Kripke de grado finito podrían haber infinitas transiciones. Sin embargo, del hecho que $K', u', v' \not\models C[x_s, x_t]$ y de que K' es de grado finito, se puede extraer un conjunto finito $C' \subseteq C$ tal que $K', u', v' \not\models C'[x_s, x_t]$ como describimos a continuación.

Consideremos el conjunto F de funciones $f : \mathit{vars}(C) \to W(K')$ tales que (i) $f(x_s) = u'$, $f(x_t) = v'$, y (ii) para todo $x \neq x_s, x_t$, $f(x)$ está a distancia a lo sumo $|\mathit{vars}(C)|$ de u' o v'. Como K' es de grado finito, F es finito. Para cada $f \in F$, escogemos un átomo de programa $a_f(x_f, y_f)$ de C que no se cumple en K' bajo la asignación f, es decir, tal que $(f(x_f), f(y_f)) \notin [\![a_f]\!]_{K'}$. Sea C' el conjunto de todos estos programas, más una cantidad finita de átomos de programa de C para asegurar que (a) $\mathit{vars}(C') = \mathit{vars}(C)$ y (b) $\mathbf{G}_{C'}$ es conexo. Notemos que C' es finito y que $K', u', v' \not\models C'[x_s, x_t]$, ya que si existiera una asignación de satisfacción $f : \mathit{vars}(C') \to W(K')$, entonces $f \in F$ y C' tendría que incluir el átomo $a_f(x_f, y_f)$ que no se satisface en K' por definición.

El caso de $K, u \not\rightharpoonup_{k+1} K', u'$ es análogo, simplemente se construye $C[x_s, x_t]$ como antes pero se fija x_t igual a x_s. La fórmula final es entonces $\langle C[x_s, x_s] \rangle$.

(2) implica (1) Esta implicación la demostraremos por inducción estructural en la expresión y de hecho veremos algo un poco más general que lo planteado en el enunciado del teorema. Para cualquier $(\bar{u}, \bar{v}) \in \bigcup_{1 \leq n \leq k+1} (W(K)^n \times W(K')^n)$,[7] si $K, \bar{u} \rightharpoonup_{k+1} K', \bar{v}$ y $K, \bar{u} \models e$ para una expresión positiva e de CPDL$^+$(TW$_k$), veamos que $K', \bar{v} \models e$.

- ▷ $\underline{e = p \text{ para } p \in \mathbb{P}}$ Como $K, v \rightharpoonup_{k+1} K', v'$, hay un homomorfismo parcial tal que $v \mapsto v'$, y por lo tanto, si $K, v \models p$ entonces $K', v' \models p$.
- ▷ $\underline{e = a \text{ para } a \in \mathbb{A}}$ Como $K, u, v \rightharpoonup_{k+1} K', u', v'$, hay un homomorfismo parcial tal que $u \mapsto u'$ y $v \mapsto v'$, y por lo tanto, si $K, u, v \models a$ entonces $K', u', v' \models a$.
- ▷ $\underline{e = \bar{a} \text{ para } a \in \mathbb{A}}$ Análogo al caso anterior.
- ▷ $\underline{e = \varphi_1 \wedge \varphi_2}$ Si $K, v \models \varphi_1 \wedge \varphi_2$ entonces $K, v \models \varphi_i$ para cada $i = 1, 2$. Luego, por hipótesis inductiva tenemos que $K', v' \models \varphi_i$ para $i = 1, 2$, lo cual implica que $K', v' \models \varphi_1 \wedge \varphi_2$.
- ▷ $\underline{e = \pi_1 \cup \pi_2}$ Análogo al caso anterior.

[7] Tomar la unión sobre $1 \leq n \leq k+1$ tiene sentido para tratar con el último caso.

- ▷ $\underline{e = \varphi?}$ Si $K, u, v \models \varphi?$ entonces $u = v$ y $K, u \models \varphi$. Por hipótesis inductiva, $K', u' \models \varphi$. Como $K, u, v \rightharpoonup_{k+1} K', u', v'$, hay un homomorfismo parcial tal que $u \mapsto u'$ y $v \mapsto v'$, pero como $u = v$, entonces $u' = v'$, lo cual implica que $K', u', v' \models \varphi?$.
- ▷ $\underline{e = \langle\pi\rangle}$ Si $K, v \models \langle\pi\rangle$, entonces existe un mundo $\tilde{v}$ en K tal que $K, v, \tilde{v} \models \pi$. Dado que: (1) v y $\tilde{v}$ son parte de una misma componente conexa de K, (2) $K, v \rightharpoonup_{k+1} K', v'$, y (3) $k + 1 \geq 3$, se sigue que hay un mundo $\tilde{v}'$ en K' tal que $K, v, \tilde{v} \rightharpoonup_{k+1} K', v', \tilde{v}'$. Más precisamente, $\tilde{v}'$ se puede obtener usando la estrategia ganadora de Duplicador, pues si Spoiler realiza una serie de movimientos para pasar de la configuración v a la configuración $v, \tilde{v}$, estas jugadas deben ser replicadas por Duplicador en K', pudiendo así pasar de la configuración v' a la configuración $v', \tilde{v}'$ y de modo que se preserve la relación $\rightharpoonup_{k+1}$. Ahora, por hipótesis inductiva, se obtiene que $K', v', \tilde{v}' \models \pi$, de donde, $K', v' \models \langle\pi\rangle$.
- ▷ $\underline{e = \pi^*}$ Supongamos que $K, u, v \models \pi^*$. Veamos que podemos reducir este caso al de programas conjuntivos. Analizamos esto por el número de iteraciones. El caso $K, u, v \models \pi^0$ sucede si y solo si $u = v$, en cuyo caso $u' = v'$ por definición de $\rightharpoonup_{k+1}$, y así $K', u', v' \models \pi^0$. Supongamos que $K, u, v \models \pi^n$ con $n \geq 1$. Si consideramos el programa conjuntivo $\tilde{\pi} = C[x_s, x_t]$, donde $C = \{\pi(x_s, y_1), \pi(y_1, y_2), \ldots, \pi(y_{n-1}, x_t)\}$, tenemos que π^n y $\tilde{\pi}$ son equivalentes. Como las subexpresiones (propias) de $\tilde{\pi}$ son iguales a las subexpresiones (propias) de π^*, basta con que apliquemos la hipótesis inductiva a $\tilde{\pi}$.
- ▷ $\underline{e = \pi_1 \circ \pi_2}$ Igual que antes, lo reduciremos al caso de programas conjuntivos, considerando ahora que $\pi_1 \circ \pi_2$ equivale a $C[x_s, x_t]$ con $C = \{\pi_1(x_s, y), \pi_2(y, x_t)\}$.
- ▷ $\underline{e = C[x_s, x_t]}$ Este es el caso más interesante de esta implicación y de hecho, demostraremos su validez valiéndonos de los programas generalizados $\hat{C}[\bar{z}]$, donde $\bar{z} \in \mathit{vars}(\hat{C})^n$ con $2 \leq n \leq k+1$. Notemos que en $\bar{z}$ puede haber repetición de variables. Por definición de grafo subyacente, si consideramos una descomposición de árbol de $\mathbf{G}_{\hat{C}[\bar{z}]}$, todas las variables en $\bar{z}$ deben estar contenidas en alguna bolsa, puesto que $\bar{z}$ genera un subclique de tamaño t en $\mathbf{G}_{\hat{C}[\bar{z}]}$. Además, si suponemos que $\mathbf{G}_{\hat{C}[\bar{z}]} \in \mathrm{TW}_k$, entonces sin pérdida de generalidad podemos considerar una descomposición de árbol de $\mathbf{G}_{\hat{C}[\bar{z}]}$ con una bolsa que solo contiene las variables de $\bar{z}$ y a la que denominamos por conveniencia *bolsa raíz*. Demostraremos ahora algunas propiedades.

Afirmación. *Para todo $\hat{C} \subseteq C$ y $\bar{z} \in \mathit{vars}(\hat{C})^n$ con $2 \leq n \leq k+1$, tales que $\mathbf{G}_{\hat{C}[\bar{z}]} \in \mathrm{TW}_k$, y para todo $\bar{v} \in W(K)^n$ y $\bar{v}' \in W(K')^n$: si $K, \bar{v} \rightharpoonup_{k+1} K', \bar{v}'$ y $\bar{v} \in [\![\hat{C}[\bar{z}]]\!]_K$, entonces $\bar{v}' \in [\![\hat{C}[\bar{z}]]\!]_{K'}$.*

Demostración. Sea $\pi = \hat{C}[\bar{z}]$ y $(T, \mathbf{v})$ una descomposición de árbol de $\mathbf{G}_\pi$ con bolsa raíz b (es decir, en $\mathbf{v}(b)$ solo están las variables de $\bar{z}$). Supongamos que $K, \bar{v} \rightharpoonup_{k+1} K', \bar{v}'$ y $\bar{v} \in [\![\pi]\!]_K$, y veamos que $\bar{v}' \in [\![\pi]\!]_{K'}$. Sea $h : \mathit{vars}(\hat{C}) \to W(K)$ una asignación de satisfacción de $\hat{C}$ tal que $\bar{v} = h(\bar{z})$. Debemos mostrar una asignación de satisfacción $h' : \mathit{vars}(\hat{C}) \to W(K')$ tal que $\bar{v}' = h(\bar{z})$.

Supongamos que la raíz b de T tiene ℓ hijos $b_1, \ldots, b_\ell$ y sean $T_1, \ldots, T_\ell$ los correspondientes subárboles. Sea $\bar{y}_i$ cualquier vector de variables (distintas dos a dos) de $\mathbf{v}(b_i) \cap \mathbf{v}(b)$, es decir, las variables en común que están en la bolsa raíz b y su i-ésimo hijo b_i. Asumimos que en $\bar{y}_i$ hay a lo sumo k variables distintas, ya que de lo contrario $\mathbf{v}(b_i) = \mathbf{v}(b)$, pero sin pérdida de generalidad, podemos suponer que en $(T, \mathbf{v})$ no pasa esto.

Sea $\bar{v}_0 = \bar{v}$ y $\bar{v}_i = h(\bar{y}_i)$ para todo $1 \leq i \leq \ell$. Definamos C_0 como el conjunto de todos los átomos de programa $\pi'(x, x')$ de $\hat{C}$ tales que $x, x' \in \mathbf{v}(b)$[8], y para todo $1 \leq i \leq \ell$, sea $C_i \subseteq \hat{C} \setminus C_0$ el conjunto de átomos de programa $\pi'(x, x')$ de $\hat{C} \setminus C_0$ tales que $x, x' \in \mathbf{v}(b')$ para alguna bolsa b' de T_i. Se sigue que $K, \bar{v} \models \hat{C}[\bar{z}]$ si y solo si $K, \bar{v}_i \models C_i[\bar{y}_i]$ para todo $0 \leq i \leq \ell$. Tenemos dos posibilidades: $C_0 = \hat{C}$ o $C_0 \subsetneq \hat{C}$.

$\underline{C_0 = \hat{C}}$ Este caso implica que todas las variables de $\hat{C}$ son aquellas de $\bar{z}$. Consideremos la función $h' : vars(\hat{C}) \to K'$ tal que $h'(\bar{z}[j]) = \bar{v}'[j]$. Primero, h' está bien definida, ya que si para una variable x de $\bar{z}$ se tiene que $x = \bar{z}[j] = \bar{z}[j']$ con $j \neq j'$, entonces $\bar{v}'[j] = \bar{v}'[j']$ pues $\bar{v}[j] = h(\bar{z}[j]) = h(\bar{z}[j']) = \bar{v}[j']$ y $(\bar{v}, \bar{v}')$ define un homomorfismo parcial por hipótesis. Segundo, h' es una asignación de satisfacción, pues para todo átomo de programa $\pi'(x, x') \in \hat{C}$ se tiene que (a) $(h(x), h(x')) \in [\![\pi']\!]_K$; y (b) como $k+1 \geq 3$, $K, h(x), h(x') \rightharpoonup_{k+1} K', h'(x), h'(x')$[9], entonces por hipótesis inductiva en π' tenemos que $(h'(x), h'(x')) \in [\![\pi']\!]_{K'}$, como queríamos demostrar.

$\underline{C_0 \subsetneq \hat{C}}$ En este caso tenemos que $|C_i| < |\hat{C}|$ para todo $0 \leq i \leq \ell$. Para cada i, si n_i es la dimensión de la tupla $\bar{v}_i$, entonces para todo $0 \leq j \leq n_i$, si $\bar{v}_i[j] = \bar{v}[k]$, definimos $\bar{v}'_i[j] = \bar{v}'[k]$, obteniendo así la n_i-tupla $\bar{v}'_i$ de mundos de K'. Note que $\bar{v}'_0 = \bar{v}'$.

Podemos aplicar la hipótesis inductiva sobre cada C_i ya que (a) $K, \bar{v}_i \models C_i[\bar{y}_i]$; y (b) como $k+1 \geq 3$, $K, \bar{v}_i \rightharpoonup_{k+1} K', \bar{v}'_i$.[10] Así, $K', \bar{v}'_i \models C_i[\bar{y}_i]$ para todo $1 \leq i \leq \ell$. Sea $h'_i : vars(C_i) \to W(K')$ una asignación de satisfacción tal que $h'_i(\bar{y}_i) = \bar{v}'_i$ y consideremos la función $h' := h'_0 \cup \cdots \cup h'_\ell$. Primero, h' está bien definida, pues si x está en $\bar{y}_i$ con $i \neq 0$, entonces para ciertos j y k tenemos que $h'_i(x) = \bar{v}'_i[j] = \bar{v}'[k] = h'_0(\bar{z}[k])$, es decir, todas las funciones h'_i coinciden con h'_0 en las variables que tienen en común. Segundo, h' es una asignación de satisfacción de $\hat{C}$, ya que $K', \bar{v}' \models \hat{C}[\bar{z}]$ si y solo si $K', \bar{v}'_i \models C_i[\bar{y}_i]$ para todo $0 \leq i \leq \ell$. □

Hemos demostrado que la propiedad del teorema vale para todos los subprogramas generalizados de $e = C[x_s, x_t]$ que satisfacen las condiciones de la afirmación anterior, y en particular, también vale para el mismo e, quedando demostrado así el teorema. □

3.4.2 Relación de bisimulación

Definimos ahora la noción de bisimulación sobre (K, K'), como antes, a través de un juego de dos jugadores $\mathbf{G}[\rightleftharpoons_k]$, similar a lo que hicimos para el juego $\mathbf{G}[\rightharpoonup_k]$. Asumamos a lo largo de esta sección y sin pérdida de generalidad que K y K' tienen conjuntos de mundos disjuntos. Esta vez el tablero de juego de $\mathbf{G}[\rightleftharpoons_k]$ tiene como conjunto de posiciones $S \cup D$, con

$$S = \{s\} \times (Hom_k(K, K') \dot{\cup} Hom_k(K', K)),$$
$$D = \{d_1, \ldots, d_k\} \times ((W(K)^k \times W(K')^k) \dot{\cup} (W(K')^k \times W(K)^k)),$$

[8] Cabe la posibilidad de que $C_0 = \emptyset$, pero por conexidad de $\mathbf{G}_{\hat{C}}$ podemos asumir que hay al menos un átomo de programa en C_0.

[9] Como los mundos en $\bar{v}$ están en la misma componente conexa de K y contamos con al menos 3 piedras para jugar, siempre es posible para Spoiler pasar de la configuración $\bar{v}$ a la configuración $h(x), h(x')$, lo cual Duplicador puede copiar pasando de la configuración $\bar{v}'$ a $h'(x), h(x')$.

[10] Análogo al caso anterior, Spoiler puede pasar de la configuración $\bar{v}$ a $\bar{v}_i$ y Duplicador responde pasando de la configuración $\bar{v}'$ a $\bar{v}'_i$ para todo $1 \leq i \leq \ell$.

donde a Spoiler le corresponden las posiciones de S y a Duplicador las de D. El conjunto de movimientos o jugadas válidas $\mathbf{G}[\rightleftharpoons_k]$ es el conjunto más pequeño que satisface lo siguiente:

J1 Hay un movimiento de $(s, \bar{u}, \bar{v})$ a $(d_i, \bar{u}', \bar{v})$ si $\bar{u}' = \bar{u}[i \mapsto w]$, donde w es un mundo a distancia ≤ 1 de $\bar{u}[j]$, para algún $1 \leq j \leq k$ con $i \neq j$;

J2 Hay un movimiento de $(d_i, \bar{u}', \bar{v})$ a $(s, \bar{u}', \bar{v}')$ si $\bar{v}' = \bar{v}[i \mapsto w]$, donde w es un mundo a distancia ≤ 1 de $\bar{v}[j]$, para algún $1 \leq j \leq k$ con $i \neq j$; y

J3 Hay un movimiento de $(s, \bar{u}, \bar{v})$ a $(s, \bar{v}, \bar{u})$ si $\bar{u}[1] = \cdots = \bar{u}[k]$ (y por lo tanto, $\bar{v}[1] = \cdots = \bar{v}[k]$).

Note que las reglas anteriores no especifican dónde están ubicadas las tuplas $\bar{u}$ y $\bar{v}$, pero se infiere que un movimiento tipo **J3** realizado por Spoiler indica que este ha cambiado de modelo, teniendo ahora la opción de mover las piedras que se encuentran en K', en cuyo caso, Duplicador debe responder con un movimiento en K. Nuevamente, la condición de victoria para Duplicador es cualquier juego infinito.

A partir de esto, definimos las siguientes nociones de juego: las de *k-semi-simulación* y *k-bisimulación*. Dadas dos estructuras de Kripke K, K' y tuplas de mundos $\bar{u} \in W(K)^k$ y $\bar{v} \in W(K')^k$, decimos que hay una k-semi-simulación de $K, \bar{u}$ a $K', \bar{v}$, denotado $K, \bar{u} \rightrightarrows_k K', \bar{v}$, si

C1 $(s, \bar{u}, \bar{v})$ es una posición válida de $\mathbf{G}[\rightleftharpoons_k]$ (es, decir, $(\bar{u}, \bar{v})$ induce un homomorfismo parcial),

C2 Duplicador tiene una estrategia ganadora desde $(s, \bar{u}, \bar{v})$,

siempre que todos los mundos en $\bar{u}$ estén en una misma componente conexa de K. Decimos que hay una k-bisimulación entre $K, \bar{u}$ y $K', \bar{v}$, denotado $K, \bar{u} \rightleftharpoons_k K', \bar{v}$, si $K, \bar{u} \rightrightarrows_k K', \bar{v}$ y $K', \bar{v} \rightrightarrows_k K, \bar{u}$.[11] Por la regla de juego **J3**, es directo que si $u \in W(K)$, $v \in W(K')$ y $K, u \rightrightarrows_k K', v$, entonces $K', v \rightrightarrows_k K, u$.

Si Duplicador tiene una estrategia ganadora, es claro que también la tiene incluso si Spoiler nunca utiliza la nueva regla **J3**, por lo que también vale directamente lo siguiente.

Proposición 61. *Si $K, \bar{u} \rightrightarrows_k K', \bar{v}$ entonces $K, \bar{u} \rightharpoonup_k K', \bar{v}$.*

Recordemos que en el Teorema 60 la relación de k-simulación lidia con las expresiones positivas de CPDL$^+$(TW$_k$). La intención de introducir estos nuevos juegos es para abarcar ahora las demás expresiones que utilizan el símbolo de negación de fórmulas $\neg$. Para la relación de k-semi-simulación tenemos lo siguiente:

Teorema 62. *Sea $k \geq 2$. Dadas dos estructuras de Kripke K, K' donde ambas son de grado finito, y mundos $u, v \in W(K)$ y $u', v' \in W(K')$, las siguientes son equivalentes:*

(1) para cada fórmula φ de CPDL$^+$(TW$_k$)*, $K, v \models \varphi$ implica $K', v' \models \varphi$;*

(2) $K, v \rightrightarrows_{k+1} K', v'$;

y las siguientes son equivalentes:

(1) para cada programa π de CPDL$^+$(TW$_k$)*, $K, u, v \models \pi$ implica $K', u', v' \models \pi$;*

(2) $K, u, v \rightrightarrows_{k+1} K', u', v'$.

Más aún, la hipótesis de grado finito solo es necesaria para las implicaciones (1) a (2).

[11] Acá también asumimos las mismas consideraciones notacionales descritas para k-simulación.

Demostración. *(1) implica (2)* Procedemos por contrarrecíproco como en la demostración del Teorema 60. Supongamos que $K, u, v \not\rightleftharpoons_{k+1} K', u', v'$. El caso en el que u y v se encuentran en diferentes componentes conexas de K o cuando $((u,v),(u',v')) \notin Hom_k(K,K')$ son triviales, así que asumamos lo contrario. Esto significa que Spoiler tiene una estrategia ganadora en un número acotado de rondas en $\mathbf{G}[\rightleftharpoons_{k+1}]$ comenzando desde la posición $(s,(u,v),(u',v'))$.

La estrategia ganadora de Spoiler es de nuevo un árbol finito, cuyos vértices son etiquetados con posiciones de $\mathbf{G}[\rightleftharpoons_{k+1}]$. En particular: (i) la raíz debe etiquetarse con $(s,(u,v),(u',v'))$, (ii) cualquier vértice con etiqueta $(d_i,\bar{u},\bar{v})$ tiene un hijo con etiqueta $(s,\bar{u},\bar{v}')$ para cada posible movimiento de $(d_i,\bar{u},\bar{v})$ a $(s,\bar{u},\bar{v}')$ en $\mathbf{G}[\rightleftharpoons_{k+1}]$, (iii) cualquier vértice con etiqueta $(s,\bar{u},\bar{v})$ tiene exactamente un hijo, cuya etiqueta es consistente con un movimiento de $\mathbf{G}[\rightleftharpoons_{k+1}]$ (recordemos que Spoiler cuenta con dos tipos de movimientos esta vez, así que esta etiqueta podría ser de tipo $(d_i,\bar{u}',\bar{v})$ o $(s,\bar{v},\bar{u})$, según la jugada que haya realizado Spoiler), (iv) para cada hoja que debe tener etiqueta de tipo $(d_i,\bar{u},\bar{v})$, no hay movimiento posible en $\mathbf{G}[\rightleftharpoons_{k+1}]$ desde esa posición. Como K y K' son de grado finito, hay solo un número finito de movimientos que se pueden hacer desde cualquier posición en $\mathbf{G}[\rightleftharpoons_{k+1}]$, por lo que el árbol de estrategia de Spoiler tiene ramificación finita y por lo tanto este árbol es finito.

De esta estrategia de Spoiler, sea el árbol T y función $\lambda : V(T) \to (W(K)^{k+1} \dot{\cup} W(K')^{k+1})$ que resultan de: (1) remover todos los vértices y etiquetados con posiciones de Duplicador exceptuando las hojas, y añadir un eje entre el padre de y y los hijos de y, y (2) proyectar la etiqueta de cada vértice en la componente que corresponde a la primera estructura (es decir, $\lambda(x) = \bar{u}$ si la etiqueta de x en el árbol de estrategia era de la forma $(s,\bar{u},\bar{v})$ o $(d_i,\bar{u},\bar{v})$). En este árbol, diremos que un eje $y' \to y$ es de intercambio si se corresponde con una jugada de tipo **J3** hecha por Spoiler en su árbol de estrategia. Sin pérdida de generalidad, podemos imponer que no haya secuencias de ejes de intercambio salvo de tamaño 1, puesto que en términos de juegos, si hubiese dos ejes de intercambio seguidos esto querría decir que Spoiler saltó de una estructura a otra solo para regresar a la anterior en la siguiente jugada, lo cual es fútil para una estrategia ganadora.[12] Construiremos un programa conjuntivo $\pi = C[x_s, x_t]$ tal que $K, u, v \models \pi$ y $K', u', v' \not\models \pi$, mediante un proceso de podado del árbol T (o análogamente, si $u = v$ podemos definir la expresión $\langle C[x_s, x_s]\rangle$).

Para todo $y \in V(T)$, denotamos T_y al subárbol de T que tiene a y como raíz. Tratemos primero con los ejes de intercambio $y' \to y$ tales que T_y *no* tiene ejes de intercambio. Para todo $1 \leq i \leq j \leq k+1$ sucede que $\lambda(y)[i] = \lambda(y)[j]$, porque y surge de haber aplicado la regla de juego **J3**. Si a cada T_y le aplicamos el mismo proceso que al árbol T de la demostración del Teorema 60 obtendremos un programa conjuntivo $C_y[x,x]$ asociado que pertenece a CPDL$^+$(TW$_k$).[13]

Consideremos el árbol T' que resulta de remover los ejes de intercambio $y' \to y$ tales que T_y no tiene ejes de intercambio (no estamos podando todo el árbol $T_{y'}$, sino solamente el eje $y' \to y$ y el árbol T_y). En T' aplicamos nuevamente el proceso anterior, ligeramente modificado: para cada eje de intercambio $\hat{y}' \to \hat{y}$ tal que $T'_{\hat{y}}$ no tiene ejes de intercambio, definimos el programa conjuntivo

[12] Sin esta restricción, podríamos lidiar sintácticamente con esta situación con una "doble negación", pero preferimos simplificar la prueba asumiendo que Spoiler ganó de la manera más sucinta posible.

[13] Análogo a la demostración del Teorema 60, C_y codifica las jugadas realizadas por Spoiler desde la configuración $\lambda(y)$. En este caso, x se define como la única variable que aparece en la bolsa raíz de la descomposición de árbol asociada a C_y.

$C_{\hat{y}}[x,x]$ que codifica las jugadas de Spoiler desde la configuración $\lambda(\hat{y})$ hasta las hojas de $T'_{\hat{y}}$, más algunos programas conjuntivos que surgen de los ejes de intercambio $y' \to y$ de T tales que y' es hoja de T'. Los programas agregados son de la forma $\pi_y = (\neg\langle C_y[x,x]\rangle)?(z,z)$, donde z es la (única) variable que se puede definir con la bolsa y'.

Aplicamos este proceso reiteradamente hasta alcanzar un subárbol de T que no tiene ejes de intercambio, y definimos $\pi = C[x_s, x_t]$, donde x_s, x_t son las variables que surgen de tener en la raíz de T la configuración u, v. De manera análoga a la demostración del Teorema 60, se puede asumir que todos los conjuntos de programas conjuntivos definidos son finitos, sabiendo que K y K' son ambos de rango finito. Además, por inducción en el número de ejes de intercambio, podemos concluir que π está en CPDL$^+$(TW$_k$), ya que los programas conjuntivos π_y no alteran el treewidth de los grafos subyacentes involucrados.

Falta ver que $K, u, v \models \pi$ y que $K', u', v' \not\models \pi$. Sea $\hat{T}$ el subárbol de T que se obtiene de podar en cada rama los ejes de intercambio más cercanos a la raíz de T, (es decir, $\hat{T}$ es el subárbol maximal que muestra todas las jugadas de Spoiler en K antes de realizar un movimiento tipo **J3**) y sea $\hat{\pi} = \hat{C}[x_s, x_t]$ el programa conjuntivo que lo codifica. Por construcción de $\pi = C[x_s, x_t]$, tenemos que C es igual $\hat{C}$, más los programas $\pi_{\hat{y}} = (\neg\langle C_{\hat{y}}[x,x]\rangle)?(z,z)$, donde $\hat{y}$ es tal que $\hat{y}' \to \hat{y}$ es un eje de intercambio de T con $\hat{y}'$ una hoja de $\hat{T}$ y z es la (única) variable que se puede definir con la bolsa $\hat{y}'$. Hay dos posibilidades: $\hat{T} = T$, o bien $\hat{T}$ es un subárbol propio de T.

$\hat{T} = T$ Este caso implica que T no tiene ejes de intercambio, o equivalentemente, que Spoiler nunca hizo un movimiento tipo **J3** en su árbol de estrategia. Entonces π es el mismo programa que se construye en la demostración del Teorema 60, para el cual vale la propiedad.

$\hat{T}$ es subárbol propio de T Este caso implica que tenemos programas de la forma $\pi_{\hat{y}}$ en C y debemos mostrar una asignación de satisfacción $f : \mathit{vars}(C) \to W(K)$. Igual que en la demostración del Teorema 60, podemos asumir que los elementos de $\mathit{vars}(C)$ son las clases de equivalencia $[y,i]$, donde y es un elemento de $V(\hat{T})$ y $1 \leq i \leq k+1$. Así que consideremos $f([y,i]) = \lambda(y)[i]$, la cual está bien definida y devuelve las configuraciones de Spoiler en su árbol de estrategia como se especifican en $\hat{T}$. Solo debemos verificar que $K, f(z) \models \langle \pi_{\hat{y}} \rangle$, puesto que el resto de programas en C se satisfacen bajo la valuación f por construcción.

Sea el eje de intercambio $\hat{y}' \to \hat{y}$, el cual se corresponde con una jugada tipo **J3** en el árbol de estrategia de Spoiler. Como la configuración de Spoiler al llegar al nodo y' es $f(z)$, entonces existe un mundo w en K' tal que la jugada que se realizó en específico fue un movimiento de $(s, f(z), w)$ a $(s, w, f(z))$. Como $T_{\hat{y}}$ es el árbol que surge de considerar la estrategia ganadora de Spoiler desde el nodo $\hat{y}$ con etiqueta $(s, w, f(z))$ y $T_{\hat{y}}$ tiene menos ejes de intercambio que T, por hipótesis inductiva, podemos concluir que $K', w \models \langle C_{\hat{y}}[x,x]\rangle$ y $K, f(z) \not\models \langle C_{\hat{y}}[x,x]\rangle$, lo cual equivale a tener que $K, f(z) \models \langle \pi_{\hat{y}} \rangle$ y $K', w \not\models \langle \pi_{\hat{y}} \rangle$.

Notemos que en este último caso hemos demostrado también que $K', u', v' \not\models \pi$.

(2) implica (1) La prueba se realiza por inducción estructural en las expresiones, como en la demostración del Teorema 60. Los casos básicos salen por la Proposición 61. Debemos analizar un caso nuevo:

- ▷ $\varphi = \neg\psi$ Como $K, v \Rightarrow_{k+1} K', v'$, aplicando la regla **J3**, debe suceder que $K', v' \Rightarrow_{k+1} K, v$, y por hipótesis inductiva, $K', v' \models \psi$ implica $K, v \models \psi$, que es equivalente a decir que $K, v \models \varphi$ implica $K', v' \models \varphi$.

El resto de los casos se deducen exactamente como en la demostración del Teorema 60, salvo que la hipótesis inductiva se formula sobre expresiones que pueden contener el símbolo de negación. □

Como CPDL$^+(\mathcal{G})$ es cerrado por negación de fórmulas, entonces $K, v \rightleftharpoons_{k+1} K', v'$ equivale a que para toda fórmula φ en CPDL$^+$(TW$_k$), $K, v \models \varphi$ *si y solo si* $K', v' \models \varphi$. Para obtener un resultado similar para programas, solo necesitamos la clausura simétrica de esta noción, que viene dada por la relación de k-bisimulación.

Teorema 63. *Sea $k \geq 2$. Dadas dos estructuras de Kripke K, K' donde ambas son de grado finito, y mundos $u, v \in W(K)$ y $u', v' \in W(K')$, las siguientes son equivalentes:*
(1) para cada fórmula φ de CPDL$^+$(TW$_k$)*, $K, v \models \varphi$ si y solo si $K', v' \models \varphi$;*
(2) $K, v \rightleftharpoons_{k+1} K', v'$;
y las siguientes son equivalentes:
(1) para cada programa π de CPDL$^+$(TW$_k$)*, $K, u, v \models \pi$ si y solo si $K', u', v' \models \pi$;*
(2) $K, u, v \rightleftharpoons_{k+1} K', u', v'$.
Más aún, la hipótesis de grado finito solo es necesaria para las implicaciones (1) a (2).

3.4.3 Conexión con la lógica modal clásica

Culminamos esta parte de juegos de bisimulación mostrando que es posible adaptar nuestros esquemas (Secciones 3.4 y 3.4.2) al juego de bisimulación de la lógica modal básica, que describimos a continuación. Sean las estructuras de Kripke

$$\begin{aligned} K &= (X, \{\rightarrow_a \mid a \in \mathbb{A}\}, \{X_p \mid p \in \mathbb{P}\}) \\ K' &= (X', \{\rightarrow'_a \mid a \in \mathbb{A}\}, \{X'_p \mid p \in \mathbb{P}\}). \end{aligned} \tag{G}$$

Para $v \in X$ y $v' \in X'$, decimos que K', v' ML-simula a K, v, denotado $K, v \rightharpoonup K', v'$, si existe $Z \subseteq X \times X'$ tal que vZv' y para todo uZu' tenemos que

(1) $u \in X_p$ implica que $u' \in X'_p$ para todo $p \in \mathbb{P}$; y

(2) si $u \rightarrow_a w$ entonces existe $w' \in X'$ tal que $u' \rightarrow'_a w'$ y wZw'.

Es bien conocido que si K' es de grado finito entonces $K, v \rightharpoonup K', v'$ ocurre si y solo si cualquier fórmula modal positiva verdadera en v también lo es en v' [46, Theorem 2.78].

Veremos ahora que la k-simulación $\rightharpoonup_k$ se puede reducir a la ML-simulación $\rightharpoonup$. Para K como en (G) y $k > 0$ definamos la siguiente estructura de Kripke:

$$S_k(K) = (S_{K,k}, \{\rightarrow_i^{K,k} \mid i \in \mathbb{A}_k\}, \{S_r^{K,k} \mid r \in \mathbb{P}_{K,k}\}),$$

donde

- $S_{K,k} = X^k$,
- $\mathbb{A}_k = \{1, \ldots, k\}$,
- $\mathbb{P}_{K,k} = (\mathbb{P} \times \{1, \ldots, k\}) \cup (\mathbb{A} \times \{1, \ldots, k\}^2) \cup \{1, \ldots, k\}^2$,
- $\bar{u} \rightarrow_i^{K,k} \bar{v}$ si y solo si hay un $1 \leq j \leq k$ con $i \neq j$, y $w \in X$ a distancia ≤ 1 de $\bar{u}[j]$ tal que $\bar{v} = \bar{u}[i \mapsto w]$, y

- para $p \in \mathbb{P}$, $\bar{v} \in S^{K,k}_{(p,i)}$ si y solo si $\bar{v}[i] \in X_p$; para $a \in \mathbb{A}$, $\bar{v} \in S^{K,k}_{(a,i,j)}$ si y solo si $\bar{v}[i] \to_a \bar{v}[j]$; y $\bar{v} \in S^{K,k}_{(i,j)}$ si y solo si $\bar{v}[i] = \bar{v}[j]$.[14]

Teorema 64. *Para estructuras de Kripke K, K', $\bar{w} \in W(K)^k$ y $\bar{w}' \in W(K')$: $K, \bar{w} \rightharpoonup_k K', \bar{w}'$ si y solo si $S_k(K), \bar{w} \rightharpoonup S_k(K'), \bar{w}'$.*

Demostración. Sean K, K' como en (G). Se puede demostrar que para cualesquiera $\bar{u} \in X^k$ y $\bar{u}' \in X'^k$, vale que $S_k(K), \bar{u} \rightharpoonup S_k(K'), \bar{u}'$ es equivalente a $(\bar{u}, \bar{u}') \in Hom_k(K, K')$ por ítem (1).

Supongamos que $S_k(K), \bar{w} \rightharpoonup S_k(K'), \bar{w}'$ vía la relación Z. Primero observemos que $(s, \bar{w}, \bar{w}')$ es una posición válida de $\mathbf{G}[\rightharpoonup_k]$ sobre (K, K'). Veamos que Duplicador tiene una estrategia ganadora en $\mathbf{G}[\rightharpoonup_k]$ desde $(s, \bar{w}, \bar{w}')$ que es el resultado de "copiar" a Z de la siguiente manera: supongamos que $\bar{u} Z \bar{v}$ y Spoiler realiza el movimiento de $(s, \bar{u}, \bar{v})$ a $(d_i, \bar{u}', \bar{v})$, donde $\bar{u}' = \bar{u}[i \mapsto w]$ y w es un mundo de K a distancia ≤ 1 de $\bar{u}[j]$, para algún $1 \leq j \leq k$ con $i \neq j$. Dado que $\bar{u} \to_i^{K,k} \bar{u}'$ y $\bar{u} Z \bar{v}$, por ítem (2) de $\rightharpoonup$ sobre $S_k(K)$ y $S_k(K')$, existe $\bar{v}' \in X'^k$ tal que $\bar{u}' \to_i^{K',k} \bar{v}'$ y $\bar{u}' Z \bar{v}'$. Por definición, $(s, \bar{u}', \bar{v}')$ es una posición válida y Duplicador hace el movimiento de $(d_i, \bar{u}', \bar{v})$ a $(s, \bar{u}', \bar{v}')$. Todo esto vale desde la posición $(s, \bar{w}, \bar{w}')$ y por lo tanto Duplicador tiene una estrategia ganadora.

Para la otra dirección, supongamos que Duplicador tiene una estrategia ganadora desde $(s, \bar{w}, \bar{w}')$, descrito por un árbol T sin hojas cuyos vértices están etiquetados con posiciones de $\mathbf{G}[\rightharpoonup_k]$. En particular, este árbol cumple que: (i) la raíz está etiquetada por $(s, \bar{w}, \bar{w}')$, (ii) cualquier vértice con una etiqueta tipo $(d_i, \bar{u}, \bar{v})$ tiene exactamente un hijo con etiqueta $(s, \bar{u}, \bar{v}')$ cuya etiqueta es consistente con un movimiento de $\mathbf{G}[\rightharpoonup_k]$, y (iii) cualquier vértice con etiqueta tipo $(s, \bar{u}, \bar{v})$ tiene un hijo por cada posible movimiento de $\mathbf{G}[\rightharpoonup_k]$. Sea $Z \subseteq X^k \times X'^k$ dado por la propiedad $\bar{u} Z \bar{v}$ si y solo si $(s, \bar{u}, \bar{v})$ es una etiqueta de algún nodo de T. Es directo que Z satisface los ítems (1) y (2), y que $S_k(K), \bar{v} \rightharpoonup S_k(K'), \bar{v}'$ vía Z. □

Análogamente, podemos establecer una relación entre k-semi-simulación y ML-simulación, adaptando un poco la metodología anterior. Sean K y K' como en (G) tales que además $X \cap X' = \emptyset$. Definimos las estructuras de Kripke $Q_k(K, K')$ y $Q'_k(K, K')$ como sigue:

$$Q_k(K, K') = (Q, \{\to_i \mid i \in \mathbb{A}_Q\}, \{Q_q \mid q \in \mathbb{P}_Q\}),$$

donde

- $Q = X^k \cup X'^k$,
- $\mathbb{A}_Q = \{1, \ldots, k\} \cup (X \times X') \cup (X' \times X)$,
- $\mathbb{P}_Q = (\mathbb{P} \times \{1, \ldots, k\}) \cup (\mathbb{A} \times \{1, \ldots, k\}^2) \cup \{1, \ldots, k\}^2$,
- para cada $1 \leq i \leq k$, $\bar{u} \to_i \bar{v}$ si y solo si hay un $Y \in \{X, X'\}$ tal que $\bar{u} \in Y^k$ y hay un $1 \leq j \leq k$ con $i \neq j$, y $w \in Y$ a distancia ≤ 1 de $\bar{u}[j]$ tal que $\bar{v} = \bar{u}[i \mapsto w]$,
- para cada $(v_1, v_2) \in (X \times X') \cup (X' \times X)$, $\bar{u}_1 \to_{(v_1, v_2)} \bar{u}_2$ si y solo si $\bar{u}_1[i] = v_1$ y $\bar{u}_2[i] = v_2$ para todo $1 \leq i \leq k$ (notar que en caso de que $v_1 \in X$, entonces $\bar{u}_1 \in X^k$ y $\bar{u}_2 \in X'^k$; y en caso de que $v_1 \in X'$, entonces $\bar{u}_1 \in X'^k$ y $\bar{u}_2 \in X^k$), y

[14] La estructura $S_k(K)$ describe los movimientos válidos realizados por algún jugador (Spoiler o Duplicador), donde las etiquetas en $\mathbb{A}_k$ codifican las movidas de las piedras y las etiquetas en $\mathbb{P}_{K,k}$ codifican las subestructuras de K generadas por cada tupla $\bar{u} \in X^k$.

- para $(\tilde{X}, \tilde{\to}) \in \{(X, \to), (X', \to')\}$, si $\bar{v} \in \tilde{X}^k$, entonces para $p \in \mathbb{P}$, $\bar{v} \in Q_{(p,i)}$ si y solo si $\bar{v}[i] \in \tilde{X}_p$; para $a \in \mathbb{A}$, $\bar{v} \in Q_{(a,i,j)}$ si y solo si $\bar{v}[i] \tilde{\to}_a \bar{v}[j]$; y $\bar{v} \in Q_{(i,j)}$ si y solo si $\bar{v}[i] = \bar{v}[j]$,

y

$$Q'_k(K, K') = (Q', \{\to'_i \mid i \in \mathbb{A}_Q\}, \{Q'_q \mid q \in \mathbb{P}_Q\}),$$

donde

- $Q' = Q$,
- para cada $1 \leq i \leq k$, $\bar{u} \to'_i \bar{v}$ se define como en $Q_k(K', K')$ reemplazando $\to_i$ por $\to'_i$,
- para cada $(v_1, v_2) \in (X \times X') \cup (X' \times X)$, $\bar{u}_2 \to_{(v_1,v_2)} \bar{u}_1$ si y solo si $\bar{u}_1 \to_{(v_1,v_2)} \bar{u}_2$ en $Q_k(K, K')$,
- y $Q'_{(p,i)}, Q'_{(a,i,j)}$ y $Q'_{(i,j)}$ se definen como en $Q_k(K, K')$ reemplazando $Q_{(p,i)}$ por $Q'_{(p,i)}$, $Q_{(a,i,j)}$ por $Q'_{(a,i,j)}$ y $Q_{(i,j)}$ por $Q'_{(i,j)}$.

  Bajo este esquema, podemos demostrar lo siguiente.

Teorema 65. *Para estructuras de Kripke K, K', $\bar{w} \in W(K)^k$ y $\bar{w}' \in W(K')$: $K, \bar{w} \rightleftharpoons_k K', \bar{w}'$ si y solo si $Q_k(K, K'), \bar{w} \rightharpoonup Q'_k(K, K'), \bar{w}'$.*

Demostración. Similar a la demostración del Teorema 64, solo aclaramos que la inclusión de los símbolos $\to_i$ con $i \in (X \times X') \cup (X \times X')$ sirven (como se indica en la definición de las estructuras $Q_k(K, K')$ y $Q'_k(K, K')$) para lidiar con las jugadas tipo **J3** que puede realizar Spoiler durante alguna partida. □

De este teorema se obtiene que

Corolario 66. *Para estructuras de Kripke K, K', $\bar{w} \in W(K)^k$ y $\bar{w}' \in W(K')$: $K, \bar{w} \rightleftharpoons_k K', \bar{w}'$ si y solo si $Q_k(K, K'), \bar{w} \rightharpoonup Q'_k(K, K'), \bar{w}'$ y $Q'_k(K, K'), \bar{w}' \rightharpoonup Q_k(K, K'), \bar{w}$.*

Consideremos los siguientes problemas de decisión:

k-SIMULATION	
Entrada:	K, K' estructuras de Kripke, $\bar{u} \in W(K)^k$, $\bar{u}' \in W(K')^k$
Salida:	Decidir si $K, \bar{u} \rightharpoonup_k K', \bar{u}'$

k-BISIMULATION	
Entrada:	K, K' estructuras de Kripke, $\bar{u} \in W(K)^k$, $\bar{u}' \in W(K')^k$
Salida:	Decidir si $K, \bar{u} \rightleftharpoons_k K', \bar{u}'$

Restringidos a estructuras finitas podemos valernos de algunos resultados conocidos para obtener cotas de complejidad para ambos problemas. En [120] demuestran que el problema de bisimulación entre los estados de unos FSPs[15] se puede decidir en tiempo polinomial. Bajo este mismo esquema, en [28] demuestran que este problema de bisimulación es PTime-completo.

Observe que cuando K es una estructura finita, entonces tanto $S_k(K, K')$ como $Q_k(K, K')$ son también finitos y de tamaño polinomial con respecto a K.

[15] Un *Finite State Process* (FSP) se puede pensar como un autómata finito no determinista con posibles transiciones vacías. En este sentido, los FSPs incluyen a las estructuras de Kripke finitas como submodelos. El esquema de simulación entre FSPs sigue las mismas reglas descritas para ML-simulación dadas al comienzo de esta sección.

Corolario 67. *Bajo estructuras de Kripke finitas, los problemas* k-SIMULATION *y* k-BISIMULATION *se puede resolver en tiempo polinomial.*

3.5 Separabilidad

Ya vimos que los lenguajes $\mathrm{CPDL}^+(\mathrm{TW}_1)$ y $\mathrm{CPDL}^+(\mathrm{TW}_2)$ son equiexpresivos. En esta sección analizaremos qué sucede entre $\mathrm{CPDL}^+(\mathrm{TW}_k)$ y $\mathrm{CPDL}^+(\mathrm{TW}_{k+1})$ para $k \geq 2$. Los próximos resultados serán una consecuencia directa de la siguiente "propiedad de modelo tipo árbol".

Proposición 68. *Para cada* $k \geq 3$*, estructura de Kripke* K *y mundo* $u \in W(K)$*, existe una estructura de Kripke* $\hat{K}$ *de treewidth* $\leq k-1$ *y un mundo* $\hat{u} \in W(\hat{K})$ *tales que* $K, u \rightleftharpoons_k \hat{K}, \hat{u}$*. Además, si* K *es numerable,* $\hat{K}$ *tiene una descomposición de árbol numerable de ancho* $\leq k-1$*.*

Demostración. Consideremos el conjunto $\mathcal{S}$ de los subconjuntos no vacíos de a lo sumo $k-1$ mundos de K. Sea T el árbol infinito con nodos $V(T) = \{u\} \cdot \mathcal{S}^*$ (es decir, los nodos son secuencias finitas sobre $\mathcal{S}$ que comienzan con $\{u\}$) y con ejes $\{x, y\}$ si x es un prefijo de y de longitud $|y| - 1$. Decimos que $\{u\}$ es el nodo raíz de T y lo denotamos r. Denotemos por $\lambda(x)$ al último conjunto de $x \in V(T)$ (es decir, $\lambda(y \cdot S) = S$ para cada $S \in \mathcal{S}$ y $y \in \mathcal{S}^*$).

Para un par de mundos v, v' de K y un par de nodos x, x' de T, escribamos como $\approx$ a la clausura reflexiva, simétrica y transitiva de

$$\{((v, x), (v', x')) \mid x' \text{ es hijo de } x \text{ en } T,\ v = v' \text{ y } v \in \lambda(x) \cap \lambda(x')\}.$$

La clase de equivalencia del par (v, x) bajo $\approx$ se denota $[v, x]$.[16] Definimos $\hat{K}, \hat{u}$ como

(1) $W(\hat{K}) = \{[v, x] \mid v \in W(K),\ x \in V(T),\ v \in \lambda(x)\}$;

(2) para cada programa atómico $a \in \mathbb{A}$ tenemos que $[\![a]\!]_{\hat{K}}$ consiste de todos los pares $([v, x], [v', x])$ tales que $(v, v') \in [\![a]\!]_K$;

(3) para cada proposición atómica $p \in \mathbb{P}$ tenemos que $[\![p]\!]_{\hat{K}}$ consiste de todos los mundos $[v, x]$ tales que $v \in [\![p]\!]_K$; y

(4) $\hat{u} = [u, r]$.

Finalmente, consideremos la función $\mathbf{v} : V(T) \to \wp(W(\hat{K}))$ que asigna a cada $x \in V(T)$ el conjunto $\{[v, x] \mid v \in \lambda(x)\}$.

Afirmación. $(T, \mathbf{v})$ *es una descomposición de árbol de* $\hat{K}$ *de treewidth a lo sumo* $k-1$*.*

Demostración. Para cada bolsa x de T, $|\mathbf{v}(x)| = |\lambda(x)| \leq k-1$. Por definición de $\approx$, para todo mundo $[v, x] \in W(\hat{K})$, el subgrafo de T generado por todas las bolsas y tales que $[v, x] \in \mathbf{v}(y)$ (o equivalentemente, tales que $[v, x] = [v, y]$) es conexo, cumpliéndose **TD3**. Por definición de $\hat{K}$, todos sus nodos y ejes están contenidos en alguna bolsa de T, cumpliéndose **TD1** y **TD2**. □

Vamos a ver que por construcción, $\hat{K}$ es k-bisimilar a K.

[16] Intuitivamente, con la relación $\approx$ identificamos los subárboles maximales de T donde un mundo v aparece en $\lambda(x)$ para todo nodo x del subárbol.

Afirmación. $K, u \rightleftharpoons_k \hat{K}, \hat{u}$.

Demostración. Demostraremos algo más general. Para cualquier $(k-1)$-tupla $\bar{u}$ de mundos de K y cualquier nodo de T de la forma $x = w \cdot \{\bar{u}[1], \ldots, \bar{u}[k-1]\}$, tenemos que $K, \bar{u} \rightleftharpoons_k \hat{K}, \bar{\hat{u}}$, donde $\bar{\hat{u}}[i] = [\bar{u}[i], x]$ con $1 \leq i \leq k-1$.

Notemos que $\{\bar{u}[i] \mapsto \bar{\hat{u}}[i]\}_{i<k}$ es (a) una función (si $\bar{u}[i] = \bar{u}[j]$, entonces $[\bar{u}[i], x] = [\bar{u}[j], x]$, por reflexividad de $\approx$), y (b) biyectiva (si $\bar{\hat{u}}[i] = \bar{\hat{u}}[j]$ entonces $(\bar{u}[i], x) \approx (\bar{u}[j], x)$ y por definición de $\approx$, $\bar{u}[i] = \bar{u}[j]$). Más aún, por definición de $\hat{K}$, tanto $\{\bar{u}[i] \mapsto \bar{\hat{u}}[i]\}_{i<k}$ como $\{\bar{\hat{u}}[i] \mapsto \bar{u}[i]\}_{i<k}$ definen homomorfismos parciales de $Hom_k(K, \hat{K})$ y $Hom_k(\hat{K}, K)$, respectivamente. Con esto ya tenemos una de las condiciones para que se cumpla $K, \bar{u} \rightrightarrows_k \hat{K}, \bar{\hat{u}}$ y $\hat{K}, \bar{\hat{u}} \rightrightarrows_k K, \bar{u}$.

Por lo anterior, queda claro que Spoiler siempre puede realizar movimientos tipo **J3** cuando la configuración se lo permite. Ahora veremos que para cualquier movimiento tipo **J1** que realice Spoiler en K o en $\hat{K}$, Duplicador le responde.

Sea $u'_i \in W(K)$ a distancia ≤ 1 de algún $\bar{u}[j]$ con $i \neq j$, y consideremos el mundo en $\hat{K}$ dado por $\bar{\hat{u}}[i \mapsto \hat{u}'_i]$, donde $\hat{u}'_i = [u'_i, x']$ y $x' = x \cdot (\{\bar{u}[j] \mid j \neq i\} \cup \{u'_i\})$. Como x' es hijo de x en T, entonces para todo $j \neq i$, $[\bar{u}[j], x] = [\bar{u}[j], x']$ y por lo demostrado en la parte anterior, tenemos que $(\bar{u}[i \mapsto u'_i], \bar{\hat{u}}[i \mapsto \hat{u}'_i]) \in Hom_k(K, \hat{K})$.

Simétricamente, sea $\hat{u}'_i \in W(\hat{K})$ a distancia ≤ 1 de algún $\bar{\hat{u}}[j]$ con $i \neq j$. Por definición de $\hat{K}$, existe un mundo u'_i de K y una bolsa x' tales que: (i) $\hat{u}'_i = [u'_i, x']$, (ii) $\bar{\hat{u}}[j] = [\bar{u}[j], x] = [\bar{u}[j], x']$, y (iii) u'_i está a distancia ≤ 1 de $\bar{u}[j]$. Más aún, para todo $j' \neq i$ y $a \in \mathbb{A}$, si $(\hat{u}'_i, \bar{\hat{u}}[j']) \in [\![a]\!]_{\hat{K}}$ (resp. $(\bar{\hat{u}}[j'], \hat{u}'_i) \in [\![a]\!]_{\hat{K}}$) entonces $(u'_i, \bar{u}[j']) \in [\![a]\!]_K$ (resp. $(\bar{u}[j'], u'_i) \in [\![a]\!]_K$).[17] Por lo tanto, tenemos que $(\bar{\hat{u}}[i \mapsto \hat{u}'_i], \bar{u}[i \mapsto u'_i]) \in Hom_k(\hat{K}, K)$. □

Por último, observemos que $(T, \mathbf{v})$ y $\hat{K}$ son numerables si K es numerable. □

CPDL$^+$ es definible con la lógica de menor punto fijo (Proposición 50), y por lo tanto hereda la *propiedad de modelo numerable de Löwenheim-Skolem*: si una fórmula de CPDL$^+$ es satisfacible, entonces es satisfacible en una estructura numerable. De donde obtenemos la siguiente propiedad, consecuencia del Teorema 63 y de la Proposición 68:

Corolario 69. *Para todo $k \geq 2$, CPDL$^+$(TW$_k$) tiene la "propiedad de modelo de treewidth k": si una fórmula $\varphi \in$ CPDL$^+$(TW$_k$) es satisfacible, entonces es satisfacible en una estructura de Kripke numerable de treewidth a lo sumo k.*

Con estos resultados demostraremos el siguiente teorema de separación:

Teorema 70. *Para cada $k \geq 3$, CPDL$^+$(TW$_{k-1}$) $\lneq$ CPDL$^+$(TW$_k$).*

Demostración. En esencia, probaremos que la presencia de un clique de tamaño $(k+1)$ se puede expresar en CPDL$^+$(TW$_k$) pero no en CPDL$^+$(TW$_{k-1}$), para todo $k \geq 3$. Sea la siguiente fórmula

$$\xi_{k+1} := \langle C[x_1, x_{k+1}] \rangle \wedge \neg \langle C'[x_1, y] \rangle, \tag{H}$$

donde $C = \{a(x_i, x_j) \mid 1 \leq i < j \leq k+1\}$ y $C' = \{a(x_1, y), a(y, y)\}$, para algún $a \in \mathbb{A}$ fijo. Notemos que ξ_{k+1} es una fórmula en CPDL$^+$(TW$_k$) ya que $\mathbf{G}_C$ es un clique de tamaño $k+1$ y

[17] De nuevo, por definición de $\hat{K}$, cualquier relación entre nodos de $\hat{K}$ debe provenir de una relación ya existente en K.

todo clique tiene treewidth igual a su tamaño, y $\mathbf{G}_{C'}$ tiene treewidth 1. Veamos que ξ_{k+1} no puede expresarse en CPDL$^+$(TW$_{k-1}$).

Primero, si K es una estructura de Kripke y $u_1 \in W(K)$ son tales que $K, u_1 \models \xi_{k+1}$, deben existir mundos $u_2, \ldots, u_{k+1}$ de K, todos distintos entre sí y de u_1, tales que $(u_i, u_j) \in [\![a]\!]_K$ para todo $1 \leq i < j \leq k+1$, es decir, el conjunto $\{u_1, \ldots, u_{k+1}\}$ genera una subestructura de K que restringida a la etiqueta a forman un clique (dirigido). De hecho, ξ_{k+1} es satisfacible.

Ahora, supongamos que ξ_{k+1} puede expresarse en CPDL$^+$(TW$_{k-1}$). Como ξ_{k+1} es satisfacible, por el Corolario 69, esta fórmula se satisface en una estructura de Kripke K numerable de treewidth a lo sumo k. Por lo anterior, en K debe existir una clique de tamaño $k+1$, pero esto no es posible por propiedades de descomposición de árbol. □

Algo a tener en cuenta es que si bien la Proposición 68 funciona para $k \geq 2$, no es cierta la *propiedad de modelo de treewidth* 1 (o sea, no vale Corolario 69 para $k = 1$), porque la caracterización entre $\rightleftharpoons_k$ y CPDL$^+$(TW$_{k-1}$) del Teorema 63 se cumple a partir de $k = 3$. Más aún, dado que CPDL$^+$(TW$_1$) $\equiv$ CPDL$^+$(TW$_2$) (Corolario 59), tenemos que ξ_3, que es la fórmula (H) para $k = 2$, es de hecho expresable en CPDL$^+$(TW$_1$). Con todo esto hemos demostrado también que la igualdad

$$\text{CPDL}^+ = \bigcup_{k=1}^{\infty} \text{CPDL}^+(\text{TW}_k)$$

representa una jerarquización estrictamente creciente (a partir de $k = 3$) de lenguajes cada más expresivos, y cada uno caracterizado por su propio juego de bisimulación.

3.6 Satisfacibilidad de CPDL$^+$

El *problema de satisfacibilidad* para un fragmento $\mathcal{C}$ de CPDL$^+$ se describe como

SATISFIABILITY FOR $\mathcal{C}$	
Entrada:	Una expresión e que es una fórmula o un programa de $\mathcal{C}$
Salida:	Decidir si existe una estructura de Kripke K y mundos $u, v \in W(K)$ tales que $u \in [\![e]\!]_K$ (en caso de que e sea fórmula) o $(u, v) \in [\![e]\!]_K$ (en caso de que e sea programa)

La decibilidad de este problema para CPDL$^+$(TW$_k$) se sigue del Corolario 69, del hecho de que las expresiones de CPDL$^+$ se pueden traducir efectivamente a la Lógica de Segundo Orden Monádico (MSO por sus siglas en inglés) [133], y el conocido resultado de que satisfacibilidad de MSO sobre estructuras de treewidth acotado es decidible (Teorema de Courcelle [63]). Ya que cualquier expresión de CPDL$^+$ pertenece a CPDL$^+$(TW$_k$) para algún k, se sigue que satisfacibilidad de CPDL$^+$ es también decidible.

Quisiéramos ahora precisar la complejidad de los problemas de satisfacibilidad. Para ello seguiremos el esquema de la demostración de [103, Theorem 4.8] sobre satisfacibilidad de ICPDL y lo adaptaremos para el caso más general de CPDL$^+$. Usaremos las mismas definiciones y nomenclatura de [103] y las siguientes demostraciones serán bastante parecidas a las del artículo citado pero destacaremos las partes donde nuestras pruebas se diferencian.

Para las demostraciones realizaremos una serie de reducciones como sigue:

- Del problema de satisfacibilidad de CPDL$^+$(TW$_k$) sobre estructuras arbitrarias a estructuras de treewidth k (trivial por Corolario 69).
- Del problema de satisfacibilidad de CPDL$^+$(TW$_k$) sobre estructuras de treewidth k al problema de satisfacibilidad de CPDL$^+$(TW$_k$) sobre "árboles regulares" (Lema 76).
- Del problema de satisfacibilidad de CPDL$^+$ sobre árboles regulares al *problema emptiness de un modelo de autómata sobre árboles*.

Los problemas a introducir dependen de un tipo especial de autómatas denominados TWAPTAs (por las siglas en inglés de *two-way alternating parity tree automaton*), por lo que explicaremos primero algunos conceptos relacionados a estos, tomados de [103].

Sean Σ_N un alfabeto de etiquetas de nodo y Σ_E un alfabeto de etiquetas de eje, ambos finitos y no vacíos. Un árbol con etiquetas (Σ_N, Σ_E), o simplemente, un (Σ_N, Σ_E)-árbol es una función parcial $T : \Sigma_E^* \to \Sigma_N$ cuyo dominio, denotado $\mathrm{dom}(T)$, es *cerrado por prefijos* (si $w_1 \cdot w_2 \in \mathrm{dom}(T)$, entonces $w_1 \in \mathrm{dom}(T)$). Esta condición también permite ver a T con una estructura subyacente de árbol:

(i) los elementos de $\mathrm{dom}(T)$ son los nodos de T; y

(ii) si $w \in \Sigma_E^*$ y $d \in \Sigma_E$ son tales que $wd \in \mathrm{dom}(T)$, entonces w es el padre de wd.

Si $wd \in \mathrm{dom}(T)$, decimos que es el d-sucesor de w, y recíprocamente, que w es el d-predecesor de wd; bajo este contexto, si $u = w$ y $v = wd$, podemos pensar que (u, d, v) es un eje etiquetado de T. Notemos que estos árboles siempre tienen como raíz a la palabra vacía, denotada como ε. Si $\mathrm{dom}(T) = \Sigma_E^*$ decimos que T es completo. En el resto de la sección trabajaremos con árboles completos y usamos $\mathit{tree}(\Sigma_N, \Sigma_E)$ para denotar el conjunto de todos los (Σ_N, Σ_E)-árboles completos. Si Σ_E no es importante o se sobreentiende, escribiremos simplemente Σ_N-árboles.

Para un conjunto finito X, denotamos por $\mathcal{B}^+(X)$ al conjunto de todas las *fórmulas Booleanas positivas* donde los elementos de X se usan como variables. Las constantes `true` y `false` se admiten como expresiones Booleanas positivas válidas, es decir, $\texttt{true}, \texttt{false} \in \mathcal{B}^+(X)$ para cualquier X. En general, podemos asumir que los elementos de $\mathcal{B}^+(X)$ están en *forma normal disyuntiva*. El dual de un elemento de $\mathcal{B}^+(X)$ se define como la fórmula Booleana positiva que se obtiene de sustituir el símbolo $\wedge$ por $\vee$, y viceversa. En principio, el dual de una fórmula Booleana en forma normal disyuntiva está en forma normal conjuntiva, pero se puede reescribir a una fórmula equivalente en forma normal disyuntiva *de tamaño exponencialmente mayor* al de la original. Por otra parte, `true` y `false` son duales entre sí. Como es usual hacer, un subconjunto $Y \subseteq X$ se puede interpretar como una valuación, simplemente asignando el valor `true` a los elementos de Y y `false` a los elementos de $X \setminus Y$. Notemos que `true` es el único elemento de $\mathcal{B}^+(X)$ que se satisface para la valuación $Y = \emptyset$.

Para un alfabeto de ejes Σ_E, denotamos $\overline{\Sigma_E} = \{\bar{a} \mid a \in \Sigma_E\}$, que se considera como una copia

disjunta de Σ_E. Para $u \in \Sigma_E^*$ y $d \in \Sigma_E \cup \overline{\Sigma_E} \cup \{\varepsilon\}$ definimos:

$$u \cdot d = \begin{cases} ud & \text{si } d \in \Sigma_E \\ u & \text{si } d = \varepsilon \\ v & \text{si existen } v \in \Sigma_E^*, a \in \Sigma_E \text{ tales que } u = va \text{ y } d = \bar{a} \\ \text{indefinido} & \text{en otro caso.} \end{cases}$$

Informalmente, los TWAPTAs son autómatas de recorrido que en cada transición pueden leer la etiqueta en Σ_N del nodo actual y moverse a un hijo chequeando la etiqueta en Σ_E especificada, o moverse (si es posible) al padre chequeando la etiqueta en Σ_E especificada, o mantenerse en el mismo nodo. Más aún, este modelo de autómata incluye alternancia, así que las transiciones se pueden pensar como *combinaciones Booleanas positivas* de este tipo de movimientos no determinísticos. El criterio de aceptación está basado en lo que se denomina como *condición de paridad*.

Formalmente, un TWAPTA sobre $\mathit{tree}(\Sigma_N, \Sigma_E)$ es una tupla $\mathcal{T} = (S, \delta, s_0, \mathit{Acc})$, donde

- S es un conjunto finito no vacío de *estados*,
- $\delta : S \times \Sigma_N \to \mathcal{B}^+(\mathit{mov}(\Sigma_E))$ es la *función de transición*, donde $\mathit{mov}(\Sigma_E) = S \times (\Sigma_E \cup \overline{\Sigma_E} \cup \{\varepsilon\})$ es el conjunto de *movimientos*,
- $s_0 \in S$ es el *estado inicial*, y
- $\mathit{Acc} : S \to \mathbb{N}$ es la *función de prioridad*.

Para $s \in S$ y $d \in \Sigma_E \cup \overline{\Sigma_E} \cup \{\varepsilon\}$, escribimos el correspondiente movimiento como $\langle s, d\rangle$. Intuitivamente, $\langle s, d\rangle$ con $d \in \Sigma_e$ significa que el autómata envía una copia de sí mismo en el estado s al d-sucesor del nodo actual del recorrido. Similarmente, $\langle s, \bar{d}\rangle$ significa que se envía una copia al d-predecesor (en caso de existir), y $\langle s, \varepsilon\rangle$ significa que nos mantenemos en la misma posición.

El comportamiento o cómputo de los TWAPTAs se define en términos de *lecturas*. Sea $\mathcal{T}$ un TWAPTA, $s \in S$ un estado de $\mathcal{T}$, $T \in \mathit{tree}(\Sigma_N, \Sigma_E)$ y $u \in \Sigma_E^*$ un nodo. Una (s, u)-lectura de $\mathcal{T}$ sobre T es un $(S \times \Sigma_E^*)$-árbol T_R (no necesariamente completo) tal que

- $T_R(\varepsilon) = (s, u)$, y
- para todo $\alpha \in \mathrm{dom}(T_R)$, si $T_R(\alpha) = (p, v)$ y $\delta(p, T(v)) = \theta$, entonces hay un subconjunto $Y \subseteq \mathit{mov}(\Sigma_E)$ que satisface θ y tal que para todo $(p', d) \in Y$, $v \cdot d$ está definido y existe un sucesor β de α en T_R con $T_R(\beta) = (p', v \cdot d)$.

Decimos que una (s, u)-lectura T_R es exitosa si para todo camino infinito $\alpha_1\alpha_2 \cdots$ en T_R (que se asume comienza desde la raíz), el siguiente número es par:

$$\min\{\mathit{Acc}(s') \mid s' \in S \text{ con } T_R(\alpha_i) \in \{s'\} \times \Sigma_E^* \text{ para infinitos } i\}.$$

Para $s \in S$ y s_0 el estado inicial de $\mathcal{T}$, definimos

$$[\![\mathcal{T}, s]\!] := \{(T, u) \in \mathit{tree}(\Sigma_N, \Sigma_E) \times \Sigma_E^* \mid \text{existe una } (s, u)\text{-lectura exitosa de } \mathcal{T} \text{ sobre } T\},$$
$$[\![\mathcal{T}]\!] := [\![\mathcal{T}, s_0]\!].$$

El lenguaje $L(\mathcal{T})$ aceptado por $\mathcal{T}$ se define como

$$L(\mathcal{T}) := \{T \in \mathit{tree}(\Sigma_N, \Sigma_E) \mid (T, \varepsilon) \in [\![\mathcal{T}]\!]\}.$$

Para un TWAPTA $\mathcal{T} = (S, \delta, s_0, Acc)$, definimos su tamaño como $|\mathcal{T}| := |S|$ y definimos su índice $i(\mathcal{T})$ como $\max\{Acc(s) \mid s \in S\}$. El tamaño $|\delta|$ de la función de transición δ es la suma de las longitudes de todas las fórmulas Booleanas positivas que aparecen en el rango de δ. Si $L(\mathcal{T}_1)$ y $L(\mathcal{T}_2)$ son lenguajes con $\mathcal{T}_1, \mathcal{T}_2$ un par de TWAPTAs sobre $tree(\Sigma_N, \Sigma_E)$, entonces se puede construir (con la técnica estándar de lenguajes regulares) un TWAPTA $\mathcal{T}$ tal que $L(\mathcal{T}) = L(\mathcal{T}_1) \cap L(\mathcal{T}_2)$, y esta construcción se puede hacer en tiempo lineal en función de $\mathcal{T}_1$ y $\mathcal{T}_2$.

El respectivo problema *emptiness* para TWAPTAs se define como aquel que dado un TWAPTA $\mathcal{T}$, se debe decidir si $L(\mathcal{T}) \neq \emptyset$. El siguiente teorema implica que el problema *emptiness* para TWAPTAs, es decidible en ExpTime.

Teorema 71. [103, Theorem 3.1] *Dado un* TWAPTA $\mathcal{T}$ *con función de transición* δ*, se puede chequear en tiempo* $\exp(|\mathcal{T}| + i(\mathcal{T})) \cdot |\delta|^{\mathcal{O}(1)}$ *si* $L(\mathcal{T}) \neq \emptyset$*, donde* $\exp(n) = 2^{n^{\mathcal{O}(1)}}$.

Este enunciado es en realidad una adaptación de varios resultados obtenidos en [167] respecto a *two-way automatas* que es un concepto distinto al de TWAPTAs, pero lo suficientemente similar para garantizar la validez del Teorema 71.

En la Sección 3.6.2 haremos una construcción inductiva de fórmulas a TWAPTAs y para ello requeriremos usar un tipo de autómatas no determinísticos denominados NFAs cuya definición la tomamos también de [103]. Formalmente, un NFA $\mathcal{A}$ sobre un TWAPTA $\mathcal{T}$ (cuyo conjunto de estados es S) es una tupla $(Q, p_0, q_0, \rightarrow_{\mathcal{A}})$, donde Q es un conjunto finito de estados, p_0 y q_0 son dos estados seleccionados, denominados *inicio* y *final*, respectivamente, y $\rightarrow_{\mathcal{A}}$ es un conjunto de transiciones etiquetadas de las siguientes formas:

(1) $q \xrightarrow{a}_{\mathcal{A}} q'$,

(2) $q \xrightarrow{\bar{a}}_{\mathcal{A}} q'$,

(3) $q \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q'$ con $s \in S$,

donde $q, q' \in Q$ y $a \in \Sigma_E$.

Las transiciones del tercer tipo son llamadas *transiciones de testeo*. Los NFAs definen relaciones binarias sobre el conjunto de nodos de un (Σ_N, Σ_E)-árbol completo. Para hacer esto explícito, dado $T \in tree(\Sigma_N, \Sigma_E)$ definimos $\Rightarrow_{\mathcal{A},T} \subseteq (\Sigma_E^* \times Q) \times (\Sigma_E^* \times Q)$ como la relación más pequeña tal que para todo $u \in \Sigma_E^*$, $a \in \Sigma_E$, $p, q \in Q$ y $s \in S$, tenemos que:[18]

(1) $(u, p) \Rightarrow_{\mathcal{A},T} (ua, q)$ si $p \xrightarrow{a}_{\mathcal{A}} q$,

(2) $(ua, p) \Rightarrow_{\mathcal{A},T} (u, q)$ si $p \xrightarrow{\bar{a}}_{\mathcal{A}} q$,

(3) $(u, p) \Rightarrow_{\mathcal{A},T} (u, q)$ si $p \xrightarrow{\mathcal{T},s}_{\mathcal{A}} q$ y $(T, u) \in [\![\mathcal{T}, s]\!]$.

Dado un NFA $\mathcal{A} = (Q, p_0, q_0, \rightarrow_{\mathcal{A}})$ sobre un TWAPTA, definimos su tamaño como $|\mathcal{A}| := |Q|$. Note que el tamaño del TWAPTA sobre el que se define $\mathcal{A}$ no se considera en su tamaño.

Por último, sea $[\![\mathcal{A}]\!] := \{(T, u, v) \mid T \in tree(\Sigma_N, \Sigma_E), u, v \in \Sigma_E^*,$ y $(u, p_0) \Rightarrow_{\mathcal{A},T}^* (v, q_0)\}$. Note que en esta definición el estado inicial de $\mathcal{T}$ no tiene un rol importante, así que por lo general en estos casos nos referiremos a un TWAPTA simplemente como una terna $\mathcal{T} = (S, \delta, Acc)$.

[18] Otra forma de interpretar a la relación $\Rightarrow_{\mathcal{A},T}$ es que los estados en Q se propagan sobre los nodos del árbol T como indica $\mathcal{A}$. Un proceso similar se realizó en el Capítulo 2 en la demostración del Lema 40.

3.6.1 Satisfacibilidad sobre árboles ω-regulares

Consideremos el siguiente problema

ω-REGULAR TREE SATISFIABILITY	
Entrada:	Una fórmula φ de CPDL$^+$ y un TWAPTA $\mathcal{T}$
Salida:	Decidir si φ es satisfacible con respecto a $L(\mathcal{T})$, es decir, si existe un árbol en $L(\mathcal{T})$ que visto como una estructura de Kripke es un modelo de φ

Para esta parte asumimos que las fórmulas φ y los árboles se describen respecto a conjuntos $A \subseteq \mathbb{A}$ y $P \subseteq \mathbb{P}$ finitos y no vacíos. Más precisamente, si $\mathrm{prog}(\varphi) := \{a \in \mathbb{A} \mid a \text{ ocurre en } \varphi\}$ y $\mathrm{prop}(\varphi) := \{p \in \mathbb{P} \mid p \text{ ocurre en } \varphi\}$ entonces $\mathrm{prog}(\varphi) \subseteq A$, $\mathrm{prop}(\varphi) \subseteq P$ y los TWAPTAs $\mathcal{T}$ computan sobre $(\wp(P), A)$-árboles. Tales árboles T se pueden identificar con una estructura de Kripke

$$T(K) := (\mathrm{dom}(T), \{\rightarrow_a \mid a \in A\}, \{T_p \mid p \in P\}), \tag{I}$$

donde $\rightarrow_a = \{(u, ua) \mid ua \in \mathrm{dom}(T)\}$ para todo $a \in A$ y $T_p = \{u \in \mathrm{dom}(T) \mid p \in T(u)\}$ para todo $p \in P$. Observe que las estructuras de Kripke derivadas de esta manera son determinísticas, es decir, la relación de transición $\rightarrow_a$ es una función parcial para todo $a \in A$.

Decimos que la fórmula φ es satisfacible con respecto a $L(\mathcal{T})$ si existe $T \in L(\mathcal{T})$ tal que $\varepsilon \in [\![\varphi]\!]_{T(K)}$. Ahora probaremos que si φ es una fórmula en CPDL$^+$(TW$_k$) con $k \geq 2$, existen φ' de CPDL$^+$ y un TWAPTA $\mathcal{T}$ tales que φ es satisfacible si y solo si φ' es satisfacible con respecto a $L(\mathcal{T})$.

Para la reducción trabajaremos con $(\wp(P), A)$-árboles, donde P y A son los siguientes conjuntos finitos de proposiciones atómicas y programas atómicos:[19]

$$P = \{t\} \dot\cup\, \mathrm{prop}(\varphi) \dot\cup\, (\{0, 1, \cdots, k\} \times \mathrm{prog}(\varphi) \times \{0, 1, \cdots, k\}),$$
$$A = \{a, b, 0, 1, \cdots, k\}.$$

La idea es que cada árbol T aceptado por $\mathcal{T}$ codifique una estructura de Kripke K junto con una descomposición de árbol *buena* de K de ancho $\leq k$. Una descomposición de árbol $(U, \mathbf{v})$ se dice que es buena si

1) U es un árbol binario y

2) si $\{b, b'\}$ es una arista de U entonces $\mathbf{v}(b) \subseteq \mathbf{v}(b')$ o $\mathbf{v}(b') \subseteq \mathbf{v}(b)$.

Considerar solo estructuras de Kripke numerables no es ningún problema pues contamos con el Corolario 69 y además

Lema 72. [103, Lemma 4.1] *Si K tiene una descomposición de árbol numerable de ancho $\leq k$, entonces K también tiene una descomposición de árbol buena de ancho $\leq k$.*

Siguiendo el esquema de [103], definiremos cuándo un $(\wp(P), A)$-árbol es *válido*. La idea original hecha para treewidth 2 y la intuición detrás de estos conceptos se discuten en [103], nosotros

[19] Estos conjuntos se pueden construir polinomialmente en función de $|\varphi|$.

hemos tomado esas ideas y las hemos extendido de forma directa para algún valor treewidth $k \geq 2$ fijo. En la siguiente definición se describirá tanto un $(\wp(P), A)$-árbol como una función $\mathbf{v} : \{a, b\}^* \to \{0, 1, \ldots, k\}$ cuya finalidad se aclarará cuando decodifiquemos la estructura de Kripke correspondiente.

Un $(\wp(P), A)$-árbol completo T se dice válido si para todo $v \in A^*$ ocurre lo siguiente:

a) si $v \in \{a, b\}^*$ e $i \in \{0, 1, \ldots, k\}$, entonces $T(vi) = \emptyset$ o $\{t\} \subseteq T(vi) \subseteq \{t\} \cup P$; en estos casos fijamos $\mathbf{v}(v) = \{i \mid t \in T(vi)\}$;

b) si $v \in \{a, b\}^*$, entonces $T(v) \subseteq \mathbf{v}(v) \times \text{prog}(\varphi) \times \mathbf{v}(v)$;

c) si $v \in \{a, b\}^*$ y $c \in \{a, b\}$, entonces $\mathbf{v}(v) \subseteq \mathbf{v}(vc)$ o $\mathbf{v}(vc) \subseteq \mathbf{v}(v)$;

d) si $v \notin \{a, b\}^* \cup \{a, b\}^*\{0, 1, \ldots, k\}$, entonces $T(v) = \emptyset$.

Lema 73. *Existe un* TWAPTA $\mathcal{T}$ *tal que* $T \in L(\mathcal{T})$ *si y solo si* T *es un* $(\wp(P), A)$*-árbol válido.*

Demostración. Sea $\mathcal{T} = (S, \delta, s_0, Acc)$ tal que $S = \{s_0, s_1\}$, $Acc(s_i) = i$. La función de transición δ será descrita por partes, especificando las acciones que realiza al ser evaluado en un $(\wp(P), A)$-árbol completo T:[20]

- si $v \in \{a, b\}^*$,

$$\delta(s, T(v)) = \begin{cases} \bigwedge_{r \in A}(s_0, r), & \text{si } s = s_0 \text{ y en } v \text{ se cumplen } a)\text{–}c), \\ (s_1, i), & \text{si } s = s_0 \text{ y en } v \text{ no se cumple } a) \text{ para } i \in \{0, \ldots, k\} \\ (s_1, a) \wedge (s_1, b), & \text{si } s = s_1 \text{ o en } v \text{ no se cumple } b) \text{ o } c) \end{cases}$$

- si $v \in \{a, b\}^*$ e $i \in \{0, 1, \ldots, k\}$,

$$\delta(s, T(vi)) = \begin{cases} \bigwedge_{r \in A}(s_0, r), & \text{si } s = s_0, \\ \bigwedge_{r \in A}(s_1, r), & \text{en caso contrario} \end{cases}$$

- si $v \notin \{a, b\}^* \cup \{a, b\}^*\{0, 1, \ldots, k\}$,

$$\delta(s, T(v)) = \begin{cases} \bigwedge_{r \in A}(s_0, r), & \text{si } s = s_0 \text{ y } T(v) = \emptyset, \\ \bigwedge_{r \in A}(s_1, r), & \text{en caso contrario} \end{cases}$$

Notemos que $\mathcal{T}$ descrito de esta forma implica directamente que si T es válido entonces existe una (s_0, ε)-lectura exitosa de $\mathcal{T}$ sobre T, específicamente, el $(S \times A^*)$-árbol completo T_R, tal que $T_R(v) = (s_0, v)$ para todo $v \in A^*$. Por otra parte si T no es válido y T_R es una (s_0, ε)-lectura de $\mathcal{T}$ sobre T, entonces debe existir un $v \in \text{dom}(T_R) \subseteq A^*$ tal que $\delta(s_0, T(v)) = \theta$, donde θ es una conjunción de átomos de la forma (s_1, r) con $r \in A$. Por definición de $\mathcal{T}$ ocurre que para todo elemento vu, con $u \in A^+$, $T_R(vu) = (s_1, vu)$, por lo que T_R tiene una rama infinita donde s_0 aparece finitas veces y s_1 aparece infinitas veces, no cumpliéndose así la condición de paridad para que T_R sea una lectura exitosa. □

[20] Técnicamente, la función δ debería depender solamente de la etiqueta $T(v)$ y no referir al elemento v. Decidimos describir a δ de esta forma por claridad argumentativa. Toda mención a v puede ser sustituida con una propiedad referente a etiquetas en $S \times \wp(P)$.

En adelante asumimos que el $\mathcal{T}$ del Lema 73 está fijo. Dado un $(\wp(P), A)$-árbol T válido, definiremos una estructura de Kripke que denotamos por $K(T)$. Primero, consideremos el conjunto

$$P_T = \{u \in \{a,b\}^*\{0,1,\ldots,k\} \mid t \in T(u)\}$$

y sea $\sim$ la relación de equivalencia más chica sobre P_T que contiene a todos los pares $(ui, uci) \in P_T \times P_T$, donde $u \in \{a,b\}^*$, $c \in \{a,b\}$ y $0 \leq i \leq k$. Denotamos por $[u]$ a la clase de equivalencia de $u \in P_T$ y por X a $P_T/\sim$. Luego,[21]

$$K(T) := (X, \{\rightarrow_r \mid r \in \mathrm{prog}(\varphi)\}, \{X_p \mid p \in \mathrm{prop}(\varphi)\}), \tag{J}$$

donde

$$\rightarrow_r = \{([ui],[uj]) \mid u \in \{a,b\}^*, (i,r,j) \in T(u)\}, \text{ y}$$
$$X_p = \{[u] \in X \mid p \in T(u)\}.$$

En lo que sigue asumimos que las estructuras de Kripke solo usan símbolos de $\mathrm{prop}(\varphi) \cup \mathrm{prog}(\varphi)$, excepto si son de la forma $T(K)$ que usan símbolos de $\wp(P) \cup A$. El siguiente resultado es una adaptación de [103, Lemma 4.5.].

Lema 74. *Si K es una estructura de Kripke numerable que tiene una descomposición de árbol buena de width a lo sumo k, entonces K es isomorfo a $K(T)$ para algún $(\wp(P), A)$-árbol T válido. Recíprocamente, para todo $(\wp(P), A)$-árbol T válido, $K(T)$ tiene treewidth a lo sumo k.*

Demostración. Sea K una estructura de Kripke numerable y $(U, \mathbf{v})$ una descomposición de árbol buena de K con ancho $\leq k$. Por comodidad y sin pérdida de generalidad, asumimos que U es un árbol binario numerable completo[22] con dominio $\mathrm{dom}(U) = \{a,b\}^*$ y raíz ε, y que para cada bolsa u existe una función inyectiva $f_u : \mathbf{v}(u) \to \{0,1,\ldots,k\}$ tal que si u' es hijo de u en U, entonces para todo $v \in \mathbf{v}(u) \cap \mathbf{v}(u')$, $f_u(v) = f_{u'}(v)$. Notemos que por definición de descomposición de árbol, para todo mundo v de K, $f_u(v)$ siempre da el mismo valor para toda bolsa u que contiene a v.

Consideremos el siguiente $(\wp(P), A)$-árbol completo T:

- si $u \in \{a,b\}^*$, entonces $T(u) = \{(f_u(v), r, f_u(v')) \mid v, v' \in \mathbf{v}(u), r \in \mathrm{prog}(\varphi) \text{ y } v \xrightarrow{r} v'\}$,
- si $u \in \{a,b\}^*$ e $i \in \{0,1,\ldots,k\}$

$$\begin{cases} \emptyset, & \text{si no existe } v \in \mathbf{v}(u) \text{ tal que } f_u(v) = i \\ \{t\} \cup \{p \in \mathrm{prop}(\varphi) \mid f^{-1}(i) \in X_p\}, & \text{en caso contrario} \end{cases}$$

- si $u \notin \{a,b\}^* \cup \{a,b\}^*\{0,1,\ldots,k\}$, entonces $T(u) = \emptyset$.

Que T sea válido se deduce directamente de la condición 2) que U satisface. Notemos que para todo mundo v de K y bolsa u que contiene a v, si $i = f_u(v)$ entonces $t \in T(ui)$, en otras palabras, por definición de $K(T)$ (ver J), $[ui] = [u'i]$ para cualesquiera bolsas u, u' que contienen a v. Esto último determina una función $f : K \to K(T)$ dada por $f(v) = [uf_u(v)]$, donde u es cualquier bolsa de U que contiene a v. Que f sea un isomorfismo es directo de la definición de $K(T)$.

[21] No confundir $K(T)$ con $T(K)$ definido en (I).

[22] Si U tuviese hojas, podemos extenderlo añadiéndole un par de hijos a cada hoja asignándoles la misma imagen por medio de $\mathbf{v}$ que tiene el padre.

Para el recíproco, supongamos que T es un $(\wp(P), A)$-árbol válido y construyamos para $K(T)$ una descomposición de árbol $(U, \mathbf{v})$ de ancho $\leq k$. Tomemos U como el árbol binario completo con dominio $\{a, b\}^*$ y $\mathbf{v}$ dada por

$$\mathbf{v}(u) = \{[ui] \mid i \in \{0, 1, \ldots, k\} \text{ y } t \in T(ui)\},$$

para todo $u \in \{a, b\}^*$. Notemos que para todo mundo $[ui]$ de $K(T)$ existe una bolsa en U que lo contiene (específicamente u) y por lo tanto se cumple **TD1**; todos los ejes de $K(T)$ son de la forma $([ui], r, [uj])$, y por la misma razón de antes obtenemos que se cumple **TD2**; por definición de la relación $\sim$ ocurre que para todo mundo $[ui]$ de $K(T)$, el conjunto $\{u' \in \{a, b\}^* \mid u'i \sim ui\}$ define un subárbol de T, lo cual implica que U satisface **TD3**. Luego, $(U, \mathbf{v})$ es una descomposición de árbol de $K(T)$ de ancho $\leq k$. Que $(U, \mathbf{v})$ sea buena se deduce de la propiedad *c)* que T satisface. □

Solo nos queda definir la fórmula φ'. Para ello consideremos el programa auxiliar

$$\pi^1_\sim = \bigcup_{i \in \{0,1,\ldots,k\}} t? \circ \bar{i} \circ (a \cup b \cup \bar{a} \cup \bar{b}) \circ i \circ t?$$

y fijemos $\pi_\sim = t? \circ (\pi^1_\sim)^*$. Es fácil ver que para cualquier $(\wp(P), A)$-árbol T válido, $[\![\pi_\sim]\!]_{T(K)}$ es igual a la relación $\sim$.

Para la fórmula φ de CPDL$^+$ definamos $\hat{\varphi}$ que se obtiene haciendo el siguiente reemplazo sintáctico en φ:

- cada programa atómico $r \in \text{prog}(\varphi)$ se sustituye por

$$\hat{r} = \bigcup_{i,j \in \{0,1,\ldots,k\}} \pi_\sim \circ \bar{i} \circ (i, r, j)? \circ j \circ \pi_\sim$$

- y cada proposición atómica $p \in \text{prop}(\varphi)$ por $\hat{p} = \langle \pi_\sim \rangle p$.

Por inducción estructural se puede demostrar el siguiente resultado que es una adaptación trivial de [103, Lemma 4.6].

Lema 75. *Para toda fórmula φ de* CPDL$^+$*, $(\wp(P), A)$-árbol válido T y $u \in P_T$ se tiene que $u \in [\![\hat{\varphi}]\!]_T(K)$ si y solo si $[u] \in [\![\varphi]\!]_{K(T)}$.*

Definamos $\varphi' = \langle (0 \cup 1 \cup \cdots \cup k) \circ t? \rangle \hat{\varphi}$. Para esta expresión obtenemos el siguiente resultado que es una adaptación de [103, Lemma 4.7].

Lema 76. *φ es satisfacible en una estructura de treewidth a lo sumo k si y solo si φ' es satisfacible con respecto a $\mathcal{T}$. Más aún,* $\text{CW}(\varphi') = \text{CW}(\varphi)$.

Demostración. La igualdad de los *conjunctive widths* es directa, por lo que solo probaremos la doble implicación.

Supongamos que φ es una fórmula de CPDL$^+(\text{TW}_k)$ satisfacible. Por el Corolario 69, φ es satisfacible en K, u donde K es una estructura de Kripke numerable de treewidth a lo sumo k, que a su vez es isomorfa a un $K(T)$ con T un $(\wp(P), A)$-árbol válido (Lema 74), y u es un mundo de K. Sin pérdida de generalidad, podemos suponer que el isomorfismo de K a $K(T)$ mapea u a $[\varepsilon i]$ para

algún $i \in \{0, 1, \ldots, k\}$ y que $t \in T(\varepsilon j)$ si y solo si $j = i$.[23] Basta ver entonces que $\varepsilon \in [\![\varphi']\!]_{T(K)}$, lo cual se reduce a ver que $\varepsilon i \in [\![\hat{\varphi}]\!]_{T(K)}$. Por el Lema 75, esto es lo mismo que $[\varepsilon i] \in [\![\varphi]\!]_{K(T)}$, lo cual ocurre por el isomorfismo entre K y $K(T)$.

Si φ' es satisfacible con respecto a $\mathcal{T}$, existe un $(\wp(P), A)$-árbol válido T tal que $\varepsilon \in [\![\varphi']\!]_{T(K)}$, y por lo tanto, para algún $i \in \{0, 1, \ldots, k\}$, ocurre que $\varepsilon i \in [\![\hat{\varphi}]\!]_{T(K)}$. Por el Lema 75, esto es lo mismo que $[\varepsilon i] \in [\![\varphi]\!]_{K(T)}$, es decir, φ es satisfacible en el modelo $K(T), [\varepsilon i]$. □

Esta reducción es polinomial solo si el valor de k se considera fijo, en caso contrario la reducción es exponencial.

3.6.2 De CPDL$^+$ a autómatas

Como en la sección anterior, fijemos conjuntos finitos $P \subseteq \mathbb{P}$ y $A \subseteq \mathbb{A}$. Sea φ una fórmula en CPDL$^+$ tal que $\text{prog}(\varphi) \subseteq A$ y $\text{prop}(\varphi) \subseteq P$ y sea $\mathcal{T}$ un TWAPTA sobre $\mathit{tree}(\wp(P), A)$. Vimos que el problema ω-REGULAR TREE SATISFIABILITY se basa en decidir si dados tales φ y $\mathcal{T}$, se tiene que φ es satisfacible con respecto a $L(\mathcal{T})$. Demostraremos que este problema está en 2ExpTime mediante una reducción computable en tiempo exponencial al problema *emptiness* para TWAPTAs. En realidad trabajaremos con ICPDL$^+$ en lugar de CPDL$^+$ para adecuarnos mejor a la reducción hecha en [103] para ICPDL.

El ingrediente principal de la reducción es una traducción definida por inducción mutua que asigna (i) fórmulas de ICPDL$^+$ a TWAPTAs y (ii) programas de ICPDL$^+$ a NFAs sobre TWAPTAs.

En nuestro contexto de árboles, para una fórmula ψ y un programa π de ICPDL$^+$, definamos los conjuntos $[\![\psi]\!]$ y $[\![\pi]\!]$ como sigue:

$$[\![\psi]\!] := \{(T, u) \mid T \in \mathit{tree}(\wp(P), A), u \in A^*, u \in [\![\psi]\!]_{T(K)}\},$$
$$[\![\pi]\!] := \{(T, u, v) \mid T \in \mathit{tree}(\wp(P), A), u, v \in A^*, (u, v) \in [\![\pi]\!]_{T(K)}\}.$$

El propósito de esta parte es mostrar cómo convertir

- cada fórmula ψ en un TWAPTA $\mathcal{T}(\psi)$ tal que $[\![\mathcal{T}(\psi)]\!] = [\![\psi]\!]$ y
- cada programa π en un TWAPTA $\mathcal{T}(\pi)$ y un NFA $\mathcal{A}(\pi)$ sobre $\mathcal{T}(\pi)$ tales que $[\![\mathcal{A}(\pi)]\!] = [\![\pi]\!]$.

La construcción se hará por inducción sobre la estructura de las expresiones. La definición del TWAPTA $\mathcal{T}(\psi)$ para cada fórmula ψ y del TWAPTA $\mathcal{T}(\pi)$ y NFA $\mathcal{A}(\pi)$ para cada programa π se hará como en [103, §3.2], pero en nuestro desarrollo debemos agregar el caso de los programas conjuntivos. Comencemos definiendo los TWAPTAs de fórmulas.

(a) $\underline{\psi = p \in P}$ Tomamos $\mathcal{T}(\psi) = (\{s_0\}, \delta, s_0, s_0 \mapsto 0)$, donde para todo $\gamma \subseteq P$ tenemos que $\delta(s_0, \gamma) = \texttt{true}$ si $p \in \gamma$ y $\delta(s_0, \gamma) = \texttt{false}$ en caso contrario. Claramente, $[\![\mathcal{T}(\psi)]\!] = [\![\psi]\!]$.

(b) $\underline{\psi = \neg\theta}$ $\mathcal{T}(\psi)$ se obtiene de $\mathcal{T}(\theta)$ aplicando un proceso de complementación estándar, donde todas las fórmulas Booleanas positivas del lado derecho de la función de transición se dualizan y la condición de aceptación se complementa incrementando la prioridad de cada estado por uno. Este caso se analiza y formaliza en [139].

[23] Estamos asumiendo que u es el único mundo de K que aparece en la bolsa raíz de la descomposición de árbol usada para definir T. Esto es posible realizarlo siempre haciendo unas modificaciones menores en la demostración del Lema 74.

(c) $\psi = \langle\pi\rangle$ En [103] definen el TWAPTA asociado a $\langle\pi\rangle\varphi$, que abarca nuestro caso cuando tomamos $\varphi = \top$, por lo que detallamos la construcción hecha para el caso más general $\psi = \langle\pi\rangle\varphi$. Por inducción, asumimos que se ha construido

$$\text{un NFA} \quad \mathcal{A}(\pi) = (Q, p_0, q_0, \rightarrow_{\mathcal{A}}) \text{ sobre un TWAPTA} \quad \mathcal{T}(\pi) = (S_1, \delta_1, Acc_1)$$

tal que $[\![\pi]\!] = [\![\mathcal{A}(\pi)]\!]$. También se ha construido un TWAPTA

$$\mathcal{T}(\varphi) = (S_2, \delta_2, s_2, Acc_2)$$

tal que $[\![\varphi]\!] = [\![\mathcal{T}(\varphi)]\!]$. Podemos asumir que Q, S_1 y S_2 son disjuntos entre sí. El TWAPTA $\mathcal{T}(\psi)$ simulará primero a $\mathcal{A}(\pi)$ y luego a $\mathcal{T}(\varphi)$. Sea el TWAPTA $\mathcal{T}(\psi) = (S, \delta, p_0, Acc)$, con $S = Q \cup S_1 \cup S_2$. Para los estados en S_1 y S_2, las transiciones de $\mathcal{T}(\psi)$ son como las de $\mathcal{T}(\pi)$ y $\mathcal{T}(\varphi)$, respectivamente. Para simular a $\mathcal{A}(\pi)$ notamos primero que es fácil ejecutar transiciones de tipo

$$q \xrightarrow{a}_{\mathcal{A}(\pi)} r \quad \text{y} \quad q \xrightarrow{\bar{a}}_{\mathcal{A}(\pi)} r,$$

pues basta que $\mathcal{T}(\psi)$ navegue hacia arriba o hacia abajo del árbol como se requiera. Para ejecutar una transición $q \xrightarrow{\mathcal{T}(\pi),s}_{\mathcal{A}(\pi)} r$, se procede *universalmente* para simular a $\mathcal{T}(\pi)$ en el estado s y *luego* simular a $\mathcal{A}(\pi)$ en el estado r. Para asegurar que la simulación de $\mathcal{A}(\pi)$ termina, la función de prioridad de $\mathcal{T}(\psi)$ asigna 1 a todos los estados de Q.[24] Para empezar la simulación de $\mathcal{T}(\varphi)$ después de que la simulación de $\mathcal{A}(\pi)$ haya terminado, añadimos una ε-transición de q_0 a s_2. Formalmente, para $q \in Q$ y $\gamma \subseteq P$, definimos

$$\delta(q, \gamma) = \bigvee\{\langle r, a\rangle \mid q \xrightarrow{a}_{\mathcal{A}(\pi)} r\} \vee \bigvee\{\langle r, \bar{a}\rangle \mid q \xrightarrow{\bar{a}}_{\mathcal{A}(\pi)} r\} \vee$$
$$\bigvee\{\langle s, \varepsilon\rangle \wedge \langle r, \varepsilon\rangle \mid q \xrightarrow{\mathcal{T}(\pi),s}_{\mathcal{A}(\pi)} r\}$$

y con una disyunción adicional $\langle s_2, \varepsilon\rangle$ si $q = q_0$. La función de prioridad Acc se define como $Acc(s) = 1$ si $s \in Q$ y $Acc(s) = Acc_i(s)$ cuando $s \in S_i$ con $i \in \{1, 2\}$. Es directo chequear que $[\![\mathcal{T}(\psi)]\!] = [\![\psi]\!]$.

(d) $\psi = \varphi_1 \wedge \varphi_2$ Dado que $\varphi_1 \wedge \varphi_2 \equiv \langle\varphi_1?\rangle\varphi_2$, podemos ver a este tipo de fórmulas como parte del caso anterior y de las construcciones para programas que haremos a continuación.

Pasamos ahora a traducir programas cuyo proceso requiere la construcción de un NFA $\mathcal{A}$ sobre un TWAPTA $\mathcal{T}$. La parte que difiere de lo realizado en [103] es el caso (i), pero igual que hicimos con las fórmulas, reproducimos la idea de la traducción de programas para el resto de los casos:

(e) $\pi = a$ ó $\pi = \bar{a}$ con $a \in A$ El NFA $\mathcal{A} = \mathcal{A}(\pi)$ tiene una única transición entre los estados seleccionados p_0 y q_0, la cual es $p_0 \xrightarrow{a}_{\mathcal{A}} q_0$ ó $p_0 \xrightarrow{\bar{a}}_{\mathcal{A}} q_0$, respectivamente. Claramente, $[\![\pi]\!] = [\![\mathcal{A}]\!]$. Dado que $\mathcal{A}$ no tiene transiciones de testeo, no importa definir un TWAPTA $\mathcal{T}(\pi)$ en este caso. Para estimar el tamaño del autómata construido, asumimos que $\mathcal{T}(\pi)$ tiene un solo estado y que su índice es 1.

(f) $\pi = \varphi?$ Asumimos que existe un TWAPTA $\mathcal{T}(\varphi) = (S, \delta, s_0, Acc)$ tal que $[\![\varphi]\!] = [\![\mathcal{T}(\varphi)]\!]$. El TWAPTA $\mathcal{T}(\pi)$ es $\mathcal{T}(\varphi)$ (sin el estado inicial). El NFA $\mathcal{A}(\pi)$ tiene solo dos estados selecciona-

[24] Esta condición implica que si T tiene una (p_0, u)-lectura exitosa T_R por medio de $\mathcal{T}(\psi)$, esta se construye usando solo una cantidad finita de estados de Q en cualquier camino infinito de T_R.

dos p_0 y q_0 con la transición $p_0 \xrightarrow{\mathcal{T}(\pi),s_0} q_0$. Por lo tanto, tenemos que

$$[\![\pi]\!] = \{(T,u,u) \mid (T,u) \in [\![\mathcal{T}(\varphi)]\!]\} = [\![\mathcal{A}(\pi)]\!].$$

(g) $\pi = \pi_1 \cup \pi_2$, $\pi = \pi_1\pi_2$ ó $\pi = \pi_1^*$ Definimos $\mathcal{A}(\pi)$ usando las construcciones estándar de los autómatas para unión, concatenación y estrella de Kleene. En los primeros dos casos, se define $\mathcal{T}(\pi)$ como la unión disjunta de $\mathcal{T}(\pi_1)$ y $\mathcal{T}(\pi_2)$, mientras que para $\pi = \pi_1^*$ fijamos $\mathcal{T}(\pi) = \mathcal{T}(\pi_1)$.

(h) $\pi = \pi_1 \cap \pi_2$ Este en el caso más complicado y largo tratado en [103], por lo que referimos directamente a la lectura del artículo.

(i) $\pi = C[x_s, x_t]$ Este caso difiere de lo hecho en [103]. Por hipótesis inductiva, asumimos que para cada átomo de programa $\pi'(x,x') \in C$ hay un TWAPTA $\mathcal{T}(\pi')$ y un NFA $\mathcal{A}(\pi')$ sobre $\mathcal{T}(\pi')$ tal que $[\![\mathcal{A}(\pi')]\!] = [\![\pi']\!]$. Sin pérdida de generalidad, supongamos que todos los conjuntos de estados son disjuntos entre sí, por lo que para hacer notoria esta distinción, nos referiremos a los NFAs y a los TWAPTAs de los átomos de programa como $\mathcal{A}(\pi'(x,x'))$ y $\mathcal{T}(\pi'(x,x'))$ y sea Q la unión de todos los conjuntos de estados de $\mathcal{A}(\pi'(x,x'))$, con $\pi'(x,x') \in C$.

Para cualquier $\pi'(x,x') \in C$ y estados p,q de $\mathcal{A}(\pi'(x,x'))$ consideremos un nuevo tipo de *programa* denotado $p \dashrightarrow q$ con la semántica $[\![p \dashrightarrow q]\!] := [\![\mathcal{A}_{p,q}(\pi'(x,x'))]\!]$, donde $\mathcal{A}_{p,q}(\pi'(x,x'))$ es el resultado de establecer a p y q como los estados inicial y final, respectivamente, del NFA $\mathcal{A}(\pi'(x,x'))$ sobre el TWAPTA $\mathcal{T}(\pi'(x,x'))$. Notemos que, formalmente, $p \dashrightarrow q$ depende tanto de $\mathcal{A}(\pi'(x,x'))$ como de $\mathcal{T}(\pi'(x,x'))$, pero dado que los conjuntos de estados son disjuntos, los autómatas pueden recuperarse inequívocamente, y por lo tanto quedan implícitos en la definición. Un programa ICPDL sobre Q[25] es cualquier programa descrito por la sintaxis

$$\pi ::= \varepsilon \mid \pi \cup \pi \mid \pi \cap \pi \mid \pi \circ \pi \mid \pi^* \mid p \dashrightarrow q,$$

donde $p,q \in Q$. Todas estas expresiones con la semántica esperada. Veamos ahora cómo traducir $C[x_s,x_t]$ a un programa ICPDL sobre Q equivalente y de ahí basta con aplicar la traducción de los casos previos.

Supongamos que $vars(C) = \{x_1,\dots,x_n\}$ y consideremos variables auxiliares $y_1,\dots,y_{n-1}$ distintas de los x_i. T_C denota cualquier árbol cuyos nodos son $U = \{x_1,\dots,x_n,y_1,\dots,y_{n-1}\}$, y tal que todas las variables x_i son hojas de T_C y el subgrafo generado por las variables y_j forman un árbol binario. Además, las aristas de T_C pueden tener una etiqueta ε o no, cuyo propósito se explicará más adelante. Los nodos de una arista con etiqueta ε se dice que son *similares*, y esta relación la cerramos por transitividad. Para cualquier par de nodos $x,y \in U$, escribamos $x \rightsquigarrow_{T_C} y$ para denotar al (único) camino simple de x a y en T_C. Sea $st : U \to \wp(Q)$ una función que asigna cada variable y al conjunto de estados $Q_y \subseteq Q$ tal que para cada átomo de programa $\pi'(x,x') \in C$:

- si el camino $x \rightsquigarrow_{T_C} x'$ contiene a y, entonces Q_y contiene exactamente un estado de $\mathcal{A}(\pi'(x,x'))$, y si p_0,q_0 son los estados inicial y final de $\mathcal{A}(\pi'(x,x'))$ entonces $p_0 \in Q_y$ si $y = x$ o $q_0 \in Q_y$ si $y = x'$;

[25] En algunos casos diremos "programa ICPDL sobre autómatas", sobreentendiéndose que Q es la unión (disjunta) de todos los conjuntos de estados de los autómatas involucrados.

- si $x = x' = y$, entonces Q_y contiene solo el estado final de $\mathcal{A}(\pi'(x, x'))$;
- de otro modo, Q_y no contiene estados de $\mathcal{A}(\pi'(x, x'))$.

Las aristas con etiqueta ε sirven con el siguiente propósito: si en T_C hay una arista $\{y, y'\}$ con etiqueta ε entonces $st(y) = st(y')$.

La intuición es que con T_C adivinamos la *forma* como C podría ser mapeado a algún árbol de $tree(\wp(P), A)$, y con st adivinamos los estados intermedios en los caminos que certifican la satisfacción de cada átomo de programa en C. Las aristas con etiquetas ε indican cuáles variables se superponen al ser interpretadas en un algún árbol de $tree(\wp(P), A)$.

Sea $\mathcal{S}$ el conjunto de todos los pares (T_C, st) que verifican las condiciones anteriores (más adelante veremos que el tamaño de $\mathcal{S}$ se puede acotar apropiadamente). Sea $(T_C, st) \in \mathcal{S}$ y $\{y, y'\}$ cualquier arista de T_C. Si esta arista tiene etiqueta ε, definimos el programa $\rho(y, y')$ como $\varepsilon(y, y')$. En caso contrario, si existen estados $q \in st(y)$ y $q' \in st(y')$ pertenecientes a algún NFA común $\mathcal{A}(\pi'(x, x'))$, tenemos que en el camino $x \rightsquigarrow_{T_C} x'$ ocurre que (a) se visita primero a y y luego a y', o (b) se visita primero a y' y luego a y. Y así, asociamos a cada tupla (y, y', q, q') el átomo de programa $\rho(y, y')$ con $\rho = q \dashrightarrow q'$ si sucede (a), o el átomo de programa $\rho(y', y)$ con $\rho = q' \dashrightarrow q$ si sucede (b). Si $\pi'(y, y) \in C$ y p_0, q_0 son los estados inicial y final de $\mathcal{A}(\pi'(y, y))$, asociamos a la tupla (y, y, p_0, q_0) el átomo de programa $\rho(y, y)$ con $\rho = p_0 \dashrightarrow q_0$. Para unos T_C y st escogidos, consideremos $C_{T_C,st}$ como el conjunto de todos los átomos de programa $\rho(y, y')$ descritos de esta manera. Observemos que por definición

(1) $vars(C_{T_C,st}) = vars(C) \dot{\cup} \{y_1 \ldots, y_{n-1}\}$ con $n = |vars(C)|$,

(2) el número de átomos de programa de $C_{T_C,st}$ es a lo sumo polinomial en el número de átomos de programa de C (es de hecho cuadrático, pues por cada programa $\pi(x, x') \in C$ se agrega solo un átomo de programa $\rho(y, y')$ por cada posible eje del camino $x \rightsquigarrow_{T_C} x'$ y T_C tiene exactamente $2n - 2$ aristas),

(3) el grafo subyacente de $C_{T_C,st}$ tiene treewidth 1 (pues es igual a T_C si ignoramos las etiquetas), y por lo tanto, el grafo subyacente de $C_{T_C,st}[x_s, x_t]$ tiene treewidth ≤ 2,

y también

(4) $\bigcup_{(T_C,st)\in\mathcal{S}} [\![C_{T_C,st}[x_s, x_t]]\!] = [\![C[x_s, x_t]]\!]$, lo cual verificamos a continuación.

Demostración. Sea $T \in tree(\wp(P), A)$ y $u, v \in A^*$. Si existe $(T_C, st) \in \mathcal{S}$ tal que $(T, u, v) \in [\![C_{T_C,st}[x_s, x_t]]\!]$ es bastante directo que $(T, u, v) \in [\![C[x_s, x_t]]\!]$. Basta ver que si f es una asignación de satisfacción de $C_{T_C,st}$ sobre $T(K)$ con $f(x_s) = u$ y $f(x_t) = v$, entonces $f' : vars(C) \to A^*$ dada por $f'(x) = f(x)$, es también una asignación de satisfacción de C en $T(K)$. Supongamos que $\pi(x, x') \in C$ con $x \neq x'$, que $\mathcal{A}(\pi(x, x'))$ es su respectivo autómata con p_0, q_0 los estados inicial y final, y que el camino $x \rightsquigarrow_{T_C} x'$ es de la forma

$$z_0 = x \to z_1 \to \cdots \to z_m \to x' = z_{m+1},$$

donde $z_j \in \{y_1, \ldots, y_{n-1}\}$ para todo $j = 1, \ldots, m$. Por definición de st, tenemos que $p_0 \in st(x)$, $q_0 \in st(x')$ y para toda arista $\{z_i, z_{i+1}\}$ que no tiene etiqueta ε en T_C, existen estados $q \in st(z_i), q' \in st(z_{i+1})$ de $\mathcal{A}(\pi(x, x'))$ tales que $(f(z_i), f(z_{i+1})) \in [\![\mathcal{A}_{q,q'}(\pi(x, x'))]\!]$.

Conectando todo esto llegamos a que

$$(f'(x), f'(x')) = (f(x), f(x)) \in [\![\mathcal{A}(\pi(x, x'))]\!],$$

y por hipótesis inductiva, $(f'(x), f'(x')) \in [\![\pi]\!]$. El caso cuando $x = x'$ es trivial.

Supongamos ahora que $(T, u, v) \in [\![C[x_s, x_t]]\!]$ y veamos que existe $(T_C, st) \in \mathcal{S}$ tal que $(T, u, v) \in [\![C_{T_C,st}[x_s, x_t]]\!]$. Sea $f : \mathit{vars}(C) \to A^*$ una asignación de satisfacción tal que $f(x_s) = u$, $f(x_t) = v$ y $(f(x), f(x')) \in [\![\pi']\!]_{T(K)}$ para todo $\pi'(x, x') \in C$. Más aún, probaremos que T_C se puede armar con la siguiente propiedad que llamaremos *similitud*: para todo $x_i, x_j, x_k \in \mathit{vars}(C)$, para llegar de $f(x_i)$ a $f(x_j)$ hay que pasar necesariamente por $f(x_k)$ si y solo si en T_C para llegar de x_i a x_j hay que pasar por un nodo $y \in \{y_1, \ldots, y_{n-1}\}$ que es similar a x_k (es decir, el camino $y \rightsquigarrow_{T_C} x_k$ solo tiene aristas con etiqueta ε). Diremos que los nodos $f(x_i)$ y $f(x_j)$ son *cercanos* si $f(x_i) = f(x_j)$ o no existe x_k tal que $f(x_k)$ está en el camino más corto entre $f(x_i)$ y $f(x_j)$ en T.

Definiremos T_C por inducción en $|\mathit{vars}(C)|$. Si $\mathit{vars}(C) = \{x\}$ tomamos T_C como el grafo con un solo nodo x. Si $\mathit{vars}(C) = \{x, x'\}$, basta con tomar T_C como el árbol con aristas $\{x, y_1\}, \{x', y_1\}$, donde una de estas tiene etiqueta ε y la segunda también si ocurre que $f(x) = f(x')$. Ahora, supongamos que $|\mathit{vars}(C)| > 2$ y sea x una variable tal que $f(x)$ no tiene descendientes en T que sean imagen de f. Aplicando la hipótesis inductiva respecto al conjunto $\mathit{vars}(C) \setminus \{x\}$, obtenemos un árbol T'_C con las siguientes propiedades:

- los nodos de T'_C son $\mathit{vars}(C) \setminus \{x\}$ y las variables auxiliares $\{y_1, \ldots, y_{n-2}\}$,
- las variables $\{y_1, \ldots, y_{n-2}\}$ inducen un árbol binario y las variables en $\mathit{vars}(C) \setminus \{x\}$ son hojas de T'_C,
- T'_C tiene la propiedad de similitud.

El árbol T_C se define con una alteración sencilla de T'_C. Sean $z_1, \ldots, z_m$ todos los elementos en $\mathit{vars}(C) \setminus \{x\}$ tales que $f(x)$ y $f(z_i)$ son cercanos. Notemos que por como fue escogido x, todos las imágenes $f(z_i)$ también son cercanas entre sí. Si $f(z_i) = f(x)$ para algún i, consideremos la arista $\{y, z_i\}$ de T'_C con $y \in \{y_1, \ldots, y_{n-2}\}$, y definamos T_C como T'_C, excepto que $\{y, z_i\}$ se sustituye con $\{y, y_{n-1}\}$ (que estará etiquetada con ε según si $\{y, z_i\}$ lo está) y $\{y_{n-1}, z_i\}, \{y_{n-1}, x\}$ ambas etiquetadas con ε. Por otra parte, si $f(z_i) \neq f(x)$ para todo i, consideremos el subárbol de T'_C más chico que solo contiene a las hojas $z_1, \ldots, z_m$ (este árbol existe pues como ya se dijo todos los $f(z_i)$ son cercanos entre sí). Sea $\{y, y'\}$ una arista de este subárbol que no está etiquetada con ε y tal que $y \in \{y_1, \ldots, y_{n-2}\}$. En este caso, definimos T_C como T'_C excepto que $\{y, y'\}$ se sustituye con las aristas $\{y, y_{n-1}\}, \{y_{n-1}, y'\}$ y $\{y_{n-1}, x\}$, ninguna de estas etiquetadas con ε. El caso en el que la arista $\{y, y'\}$ no etiquetada no exista, implica que $f(z_i) = f(z_j)$ para todo par de índices i, j, y basta con tomar cualquier arista $\{y, y'\}$ y hacer la misma sustitución de antes salvo que $\{y, y_{n-1}\}, \{y_{n-1}, y'\}$ se etiquetan con ε pero $\{y_{n-1}, x\}$ no.

Hasta ahora queda descrito el árbol T_C. Para definir st, primero extenderemos f a una función $g : \mathit{vars}(C) \dot\cup \{y_1, \ldots, y_{n-1}\} \to A^*$ tal que para todo par de variables x, y, si $\{x, y\}$ es una arista de T_C con etiqueta ε entonces $g(x) = g(y)$ y si $y \in \{y_1, \ldots, y_{n-1}\}$ entonces $g(y)$ es el antepasado común más cercano de todos los $g(x) = f(x)$ con $x \in \mathit{vars}(C)$ que sean descendientes de y en T_C. Supongamos que $\pi(x, x') \in C$, y como

$(f(x), f(x')) \in [\![\pi]\!]_{T(K)}$, por hipótesis inductiva, $(T, f(x), f(x')) \in [\![\mathcal{A}(\pi)]\!]$. Por definición de g, si y está en el camino $x \rightsquigarrow_{T_C} x'$, entonces en el camino que conecta a $f(x)$ con $f(x')$ en T se tiene que pasar necesariamente por $g(y)$. Todo esto implica que la relación $(T, f(x), f(x')) \in [\![\mathcal{A}(\pi)]\!]$ se puede descomponer en muchas de la forma

$$\begin{gathered}(T, g(x), g(z_1)) \in [\![\mathcal{A}_{p_0,q_1}(\pi)]\!],\\ (T, g(z_1), g(z_2)) \in [\![\mathcal{A}_{q_1,q_2}(\pi)]\!],\\ \vdots\\ (T, g(z_m), g(x')) \in [\![\mathcal{A}_{q_m,q_0}(\pi)]\!],\end{gathered}$$

donde los z_i son los nodos que componen el camino $x \rightsquigarrow_{T_C} x'$ y los q_i son estados de $\mathcal{A}(\pi)$. Definimos st de modo que bajo las condiciones anteriores $p_0 \in st(x)$, $q_0 \in st(x')$ y $q_i \in st(z_i)$ para todo i.

Por construcción, obtenemos que $(T, u, v) \in [\![C_{T_C,st}[x_s, x_t]]\!]$ donde la respectiva asignación de satisfacción es la función g. □

Para cualquier $\hat{C}[x_s, x_t] = C_{T_C,st}[x_s, x_t]$ con $(T_C, st) \in \mathcal{S}$, podemos construir un programa de ICPDL sobre Q equivalente, llamado $\hat{\pi}_{T_C,st}$ de tamaño $|C|^{\mathcal{O}(1)}$ (esto es por Corolario 59 y por (2)). Entonces, $C[x_s, x_t]$ es equivalente a una unión exponencial de programas de ICPDL tipo $\hat{\pi}_{T_C,st}$ con $(T_C, st) \in \mathcal{S}$. La cota exponencial para $|\mathcal{S}|$ la probaremos más adelante (Proposición 80) después de definir ciertos parámetros de interés. Ahora, podemos aplicar a esta última expresión la traducción para ICPDL (la del artículo [103], referenciado antes como el caso (h)) obteniendo el TWAPTA $\mathcal{T}(C[x_s, x_t])$ y el NFA $\mathcal{A}(C[x_s, x_t])$ sobre $\mathcal{T}(C[x_s, x_t])$ tal que $[\![\mathcal{A}(C[x_s, x_t])]\!] = [\![C[x_s, x_t]]\!]$.

Hasta aquí queda descrita la traducción de expresiones en ICPDL$^+$ a TWAPTAs. Ahora verificaremos que esta reducción es exponencial en el tamaño de la expresión. Para ello volveremos a seguir el esquema de [103] y lo extenderemos a los casos que nos conciernen.

Tamaño del autómata El ancho de intersección $\mathrm{iw}(\pi)$ de un programa π de ICPDL se define en [103] como sigue:

$$\begin{aligned}&\mathrm{iw}(a) = \mathrm{iw}(\bar{a}) := 1 \text{ para todo } a \in \mathbb{A},\\ &\mathrm{iw}(\varphi?) := 1 \text{ para toda fórmula } \varphi \text{ de ICPDL},\\ &\mathrm{iw}(\pi_1 \cup \pi_2) = \mathrm{iw}(\pi_1 \circ \pi_2) := \max\{\mathrm{iw}(\pi_1), \mathrm{iw}(\pi_2)\},\\ &\mathrm{iw}(\pi^*) := \mathrm{iw}(\pi), \qquad \mathrm{iw}(\pi_1 \cap \pi_2) := \mathrm{iw}(\pi_1) + \mathrm{iw}(\pi_2).\end{aligned}$$

Esto se generaliza a cualquier expresión e definiendo $\mathrm{IW}(e)$ como el máximo de los anchos de intersección de los subprogramas de e (o 1 si no tiene subprogramas). El interés por el parámetro IW radica en el hecho de que para una expresión e de ICPDL, el tamaño de los autómatas construidos son solo exponenciales en $\mathrm{IW}(e)$:

Lema 77. [103, Lemma 3.6] *Para cada programa π de* ICPDL, $|\mathcal{A}(\pi)| \leq 2 \cdot |\pi|^{\mathrm{iw}(\pi)}$.

Lema 78. [103, Lemma 3.7] *Para cada expresión e de* ICPDL, $|\mathcal{T}(e)| \leq |e| + 8 \cdot |e|^{2\cdot \mathrm{IW}(e)+1}$ *y* $i(\mathcal{T}(e)) \leq |e|$. *Si e es una fórmula, entonces para la función de transición δ de $\mathcal{T}(e)$ ocurre que* $|\delta| \leq 2^{|e|} \cdot |e|^{\mathcal{O}(\mathrm{IW}(e))}$.

Por la construcción hecha en [103] (que hemos reproducido parcialmente en las páginas anteriores), la cota para el índice en el lema anterior puede mejorarse a $i(\mathcal{T}(e)) \leq \mathrm{nd}(e)$, donde $\mathrm{nd}(e)$ denota el número máximo de negaciones encajadas (*negation depth* en inglés) en la expresión e (más 1). Explícitamente, $\mathrm{nd}(e)$ se define como

$$\begin{aligned}
&\mathrm{nd}(p) = \mathrm{nd}(a) = \mathrm{nd}(\bar{a}) := 1 && p \in \mathbb{P} \text{ y } a \in \mathbb{A},\\
&\mathrm{nd}(\neg\varphi) := \mathrm{nd}(\varphi) + 1 &&\\
&\mathrm{nd}(\langle\pi\rangle) = \mathrm{nd}(\pi^*) := \mathrm{nd}(\pi) &&\\
&\mathrm{nd}(\varphi?) := \mathrm{nd}(\varphi) &&\\
&\mathrm{nd}(\pi_1 \star \pi_2) := \max\{\mathrm{nd}(\pi_1), \mathrm{nd}(\pi_2)\} && \star \in \{\cup, \cap, \circ\}
\end{aligned}$$

Extendemos esta definición a programas conjuntivos definiendo

$$\mathrm{nd}(C[x_s, x_t]) := \max_{\pi(x,y)\in C}\{\mathrm{nd}(\pi)\}.$$

En nuestra construcción, dada una expresión en CPDL$^+$ obtenemos un ancho de intersección polinomialmente acotado en función de la expresión, ya que en cada paso convertimos un programa conjuntivo en una unión de programas de ICPDL de tamaño polinomial sobre el autómata ya construido. Concretamente, el ancho de intersección de cada $\hat{\pi}_{T_C,st}$ es a lo sumo el número de átomos de $C_{T_C,st}$, por lo tanto, es cuadrático en el número de átomos de C. Más aún, esta construcción no cambia el número de negaciones encajadas. Para hacer explícitas estas cotas, utilizaremos un nuevo parámetro $\mathrm{cw}(e)$, denominado ancho de conjunción:

$$\begin{aligned}
&\mathrm{cw}(a) = \mathrm{cw}(\bar{a}) := 1 \text{ para todo } a \in \mathbb{A},\\
&\mathrm{cw}(\varphi?) := 1 \text{ para toda fórmula } \varphi \text{ de CPDL}^+,\\
&\mathrm{cw}(\pi_1 \cup \pi_2) = \mathrm{cw}(\pi_1 \circ \pi_2) := \max\{\mathrm{cw}(\pi_1), \mathrm{cw}(\pi_2)\},\\
&\mathrm{cw}(\pi^*) := \mathrm{cw}(\pi), \qquad \mathrm{cw}(C[x_s, x_t]) := \sum_{\pi(x,y)\in C} \mathrm{cw}(\pi).
\end{aligned} \tag{K}$$

Esta función se extiende fácilmente a ICPDL$^+$, ya que por medio de la Proposición 49 podemos establecer simplemente $\mathrm{cw}(\pi_1 \cap \pi_2) := \mathrm{cw}(C[x, y])$, donde $C = \{\pi_1(x, y), \pi_2(x, y)\}$ con $x \neq y$. Notemos que para todo programa conjuntivo $\pi = C[x_s, x_t]$, $|C| \leq \mathrm{cw}(\pi)$. Generalizamos $\mathrm{CW}(e)$ para cualquier expresión e como el máximo de los anchos de conjunción de los subprogramas de e (o 1 si no tiene subprogramas). El siguiente es un resultado que relaciona los parámetros ya definidos y que necesitamos para establecer cotas de complejidad para el problema de satisfacibilidad.

Proposición 79. *Sea $\pi = C[x_s, x_t]$ un programa conjuntivo de* ICPDL$^+$ *y $\pi' = \bigcup_{(T_C,st)\in\mathcal{S}} \hat{\pi}_{T_C,st}$ el equivalente programa de* ICPDL *sobre los autómatas asociados a π (construcción (i)). Entonces valen las siguientes propiedades:*

(1) El tamaño de π' está acotado por $\exp(|\pi|)$;

(2) $\mathrm{iw}(\pi') \leq \mathrm{cw}(\pi)^{\mathcal{O}(1)}$;

(3) $\mathrm{nd}(\pi') = \mathrm{nd}(\pi)$.

Demostración. Dado que cada $\hat{\pi}_{T_C,st}$ es de tamaño $\mathrm{cw}(\pi)^{\mathcal{O}(1)}$, como resultado de la traducción polinomial del Corolario 59, aplicado al programa conjuntivo $C_{T_C,st}[x_s, x_t]$ que es de tamaño cuadrático sobre $|C|$, el resultado (1) se obtiene acotando a $|\mathcal{S}|$, lo cual realizaremos en la demostración de la Proposición 80. El hecho que $\mathrm{iw}(\pi')$ sea polinomial en $\mathrm{cw}(\pi)$ se debe a que

$$\mathrm{iw}\left(\bigcup_{T_C,st} \hat{\pi}_{T_C,st}\right) = \max_{T_C,st}\{\mathrm{iw}(\hat{\pi}_{T_C,st})\}, \tag{L}$$

y nuevamente, a que cada $\hat{\pi}_{T_C,st}$ es de tamaño polinomial respecto a $\mathrm{cw}(\pi)$. Por último, (3) se cumple simplemente por construcción de π'. □

El siguiente es nuestro resultado análogo del Lema 77.

Proposición 80. *Para cada programa* π *de* ICPDL$^+$, $|\mathcal{A}(\pi)|$ *está acotado por* $\exp(|\pi|)$.

Demostración. *Caso base* Cualquier programa π sin subprogramas conjuntivos es simplemente un programa de CPDL. Entonces la cota en el tamaño del NFA equivalente del Lema 77 aplica, considerando además que en estos casos $\mathrm{iw}(\pi) = \mathrm{cw}(\pi)$.
Caso inductivo Solo analizaremos el caso de programas conjuntivos. Si $\pi = C[x_s, x_t]$, con $C = \{\pi_i(x_i, y_i) \mid 1 \leq i \leq n\}$, por hipótesis inductiva tenemos que $|\mathcal{A}(\pi_i)|$ está acotado por $\exp(|\pi_i|)$, para todo $1 \leq i \leq n$.

Consideremos el programa $\pi' = \bigcup_{(T_C,st)\in\mathcal{S}} \hat{\pi}_{T_C,st}$ construido en (i) a partir de π. Entonces

$$\begin{aligned}
|\mathcal{A}(\pi)| = |\mathcal{A}(\pi')| &\leq \sum_{(T_C,st)\in\mathcal{S}} |\mathcal{A}(\hat{\pi}_{T_C,st})| \\
&\leq \sum_{(T_C,st)\in\mathcal{S}} 2 \cdot |\hat{\pi}_{T_C,st}|^{\mathrm{iw}(\hat{\pi}_{T_C,st})} \\
&\leq 2 \cdot \sum_{(T_C,st)\in\mathcal{S}} \mathrm{cw}(\pi)^{\mathrm{cw}(\pi)^{\mathcal{O}(1)}} \\
&\leq 2 \cdot \exp(\mathrm{cw}(\pi)) \cdot |\mathcal{S}|.
\end{aligned}$$

La primera desigualdad es consecuencia del caso (g); la segunda debido a que cada $\hat{\pi}_{T_C,st}$ es un programa de ICPDL y por Lema 77; la tercera porque $|\hat{\pi}_{T_C,st}|$ es polinomial en $|C|$, porque $\mathrm{iw}(\hat{\pi}_{T_C,st}) \leq \mathrm{iw}(\pi')$ por (L) y por Proposición 79(2); y la cuarta es por reescritura.

Para acotar el cardinal de $\mathcal{S}$ usaremos la hipótesis inductiva y la definición de los pares (T_C, st) dada en (i). La cantidad de árboles T_C es exponencial en la cantidad de nodos,[26] que es un valor lineal en $\mathrm{cw}(\pi)$. Para acotar la cantidad de funciones st recordemos que para cada nodo y de T_C, el conjunto $st(y)$ contiene a lo sumo un estado de cada NFA $|\mathcal{A}(\pi_i)|$. De esta manera,

$$|\mathcal{S}| \leq \exp(\mathrm{cw}(\pi)) \cdot \left(\prod_{i=1}^{n} (|\mathcal{A}(\pi_i)| + 1)\right)^{\mathrm{cw}(\pi)}.$$

[26] Recordemos que T_C es un árbol binario respecto a ciertos nodos y las demás características de T_C como las hojas o los ejes etiquetados solo influyen en un factor $\exp(\mathrm{cw}(\pi))$.

Luego,

$$\begin{aligned}\prod_{i=1}^{n}(|\mathcal{A}(\pi_i)|+1) &\leq \prod_{i=1}^{n}\exp(|\pi_i|)\\ &\leq \prod_{i=1}^{n}\exp(|\pi|)\\ &\leq \exp(|\pi|)^{\mathrm{cw}(\pi)}.\end{aligned} \tag{M}$$

La primera desigualdad se da por hipótesis inductiva y las siguientes porque $|\pi_i| \leq |\pi|$ y reescritura de la productoria. Dado que las expresiones $\exp(|\pi|)$ y $\exp(|\pi|)^{\mathrm{cw}(\pi)}$ son del mismo orden, obtenemos el resultado requerido. □

Proposición 81. *Para cada expresión e de* ICPDL$^+$, $|\mathcal{T}(e)|$ *está acotado por* $\exp(|\pi|)$. *Si e es una fórmula, entonces para la función de transición δ de $\mathcal{T}(e)$ ocurre que* $|\delta| \leq 2^{|e|} \cdot |e|^{\mathrm{CW}(e)^{\mathcal{O}(1)}}$.

Demostración. *Caso base* Cualquier expresión e sin subprogramas conjuntivos es simplemente una expresión de ICPDL. Entonces la cota para el tamaño del TWAPTA equivalente del Lema 78 aplica, considerando además que en estos casos $\mathrm{IW}(e) = \mathrm{CW}(e)$.
Caso inductivo Solo analizaremos el caso de programas conjuntivos. Si $\pi = C[x_s, x_t]$, con $C = \{\pi_i(x_i, y_i) \mid 1 \leq i \leq n\}$, por hipótesis inductiva tenemos que $|\mathcal{A}(\pi_i)|$ está acotado por $\exp(|\pi_i|)$, para todo $1 \leq i \leq n$.

Consideremos el programa $\pi' = \bigcup_{(T_C,st)\in\mathcal{S}} \hat{\pi}_{T_C,st}$ construido en (i) a partir de π. Entonces

$$\begin{aligned}|\mathcal{T}(\pi)| = |\mathcal{T}(\pi')| &\leq \sum_{(T_C,st)\in\mathcal{S}} |\mathcal{T}(\hat{\pi}_{T_C,st})|\\ &\leq \sum_{(T_C,st)\in\mathcal{S}} 9 \cdot |\hat{\pi}_{T_C,st}|^{2\,\mathrm{iw}(\hat{\pi}_{T_C,st})+1}\\ &\leq 9 \cdot \sum_{(T_C,st)\in\mathcal{S}} \mathrm{cw}(\pi)^{\mathrm{cw}(\pi)^{\mathcal{O}(1)}}\\ &\leq 9 \cdot \exp(\mathrm{cw}(\pi)) \cdot |\mathcal{S}|.\end{aligned}$$

La primera desigualdad es consecuencia del caso (g); la segunda debido a que cada $\hat{\pi}_{T_C,st}$ es un programa de ICPDL y por Lema 78; la tercera porque $|\hat{\pi}_{T_C,st}|$ es polinomial en $|C|$, porque $\mathrm{iw}(\hat{\pi}_{T_C,st}) \leq \mathrm{iw}(\pi')$ por (L) y por Proposición 79(2); y la cuarta es por reescritura. Ya vimos en la demostración de la Proposición 80 que $|\mathcal{S}|$ está acotado por $\exp(|\pi|)$, de donde obtenemos nuestro resultado.

La cota para $|\delta|$ se obtiene de las desigualdades dadas en Lema 78 y en Proposición 79(2). □

Combinando estos últimos resultados con la complejidad del problema de *emptiness* para TWAPTAs dada por el Teorema 71 obtenemos lo siguiente:

Proposición 82. *Para un* TWAPTA $\mathcal{T}$ *con función de transición δ y para una fórmula φ de* CPDL$^+$, *podemos decidir en tiempo*

$$\exp(|\mathcal{T}| + i(\mathcal{T}) + \exp(|\varphi|)) \cdot |\delta|^{\mathcal{O}(1)}$$

si existe algún $T \in L(\mathcal{T})$ *tal que* $\varepsilon \in [\![\varphi]\!]_T$. *Por lo tanto,* ω-REGULAR TREE SATISFIABILITY *para* CPDL$^+$ *está en* 2ExpTime.

Demostración. Aplicamos la complejidad del problema *emptiness* dada por el Teorema 71, junto con las cotas obtenidas para el tamaño y el índice de un TWAPTA (Proposición 81). Dada una fórmula φ de CPDL$^+$, consideremos el TWAPTA $\mathcal{T}(\varphi)$ con función de transición δ. Se puede chequear en tiempo

$$\begin{aligned}\exp(|\mathcal{T}(\varphi)| + i(\mathcal{T}(\varphi))) \cdot |\delta|^{\mathcal{O}(1)} &\leq \exp(\exp(|\varphi|) + \mathrm{nd}(\varphi)) \cdot \exp(|\varphi|) \\ &\leq \exp(\exp(|\varphi|))\end{aligned}$$

si $L(\mathcal{T}(\varphi))$ es no vacío. Luego, dada la fórmula φ y un TWAPTA $\mathcal{T}$ con función de transición δ, se puede chequear en tiempo

$$\exp(|\mathcal{T}| + i(\mathcal{T}) + \exp(|\varphi|)) \cdot |\delta|^{\mathcal{O}(1)}$$

si $L(\mathcal{T}) \cap L(\mathcal{T}(\varphi))$ es no vacío. □

Dado que la reducción presentada en la Sección 3.6.1 es polinomial para fórmulas con treewidth acotado, pero exponencial en general, tenemos el siguiente resultado.

Teorema 83. *El problema de satisfacibilidad de* CPDL$^+$(TW$_k$) *es* 2ExpTime *y el problema de satisfacibilidad de* CPDL$^+$ *es* 3ExpTime.

3.7 Model Checking de CPDL$^+$

Consideremos el siguiente problema

MODEL CHECKING CPDL$^+$	
Entrada:	una estructura de Kripke K (finita), un mundo w de K y una fórmula φ de CPDL$^+$
Salida:	Decidir si $K, w \models \varphi$

Este problema se sabe que es PTime-completo para fórmulas en PDL, ICPDL, y muchas otras variantes de PDL [124]. En general MODEL CHECKING CPDL$^+$ está en P$^{\mathrm{NP}}$, donde el oráculo de NP se utiliza para adivinar una posible asignación de satisfacción para algún programa conjuntivo $C[x_s, x_t]$ que sea subexpresión de φ. Como cota inferior tenemos el siguiente resultado.

Lema 84. MODEL CHECKING CPDL$^+$ *es* DP*-hard.*

Demostración. Consideremos el problema de *graph homomorphism* que dado un par (G, H) de grafos simples y finitos, decide si hay un homomorfismo de G a H. Este problema es NP-completo incluso si G y H son conexos. El problema en el que dada una terna (G_1, G_2, H) de grafos simples, conexos y finitos, decide si hay un homomorfismo de G_1 a H y no hay un homomorfismo de G_2 a H es DP-completo, por lo que basta hacer una reducción de este problema a MODEL CHECKING CPDL$^+$.

Para ello usaremos dos programas atómicos r, s y definiremos una estructura de Kripke K que dependerá de H, y una fórmula φ de CPDL$^+$ que dependerá de G_1 y G_2. A K lo definimos tal que

- $W(K) = V(H)$;
- $X_p = \emptyset$ para todo $p \in \mathbb{P}$;
- $\rightarrow_r = W(K) \times W(K)$; y
- $\rightarrow_s = \{(u, v) \mid \{u, v\}$ es arista de $H\}$ (claramente, si $(u, v) \in \rightarrow_s$ entonces $(v, u) \in \rightarrow_s$).

Para definir φ, consideremos primero una función biyectiva $\mathbf{v}_i : V(G) \to \mathsf{Var}$, que se puede ver como un renombramiento de los nodos del grafo G_i a variables. Para cada arista $\{u, v\}$ de G_i definamos el átomo de programa $s(\mathbf{v}_i(u), \mathbf{v}_i(v))$, y sea C_i el conjunto de todos estos programas. Escojamos para cada i un par de variables distintas $x_i, y_i \in vars(C_i)$, y definamos φ como

$$\langle r^* \rangle \langle C_1[x_1, y_1] \rangle \wedge \neg \langle r^* \rangle \langle C_2[x_2, y_2] \rangle.$$

Para cualquier mundo u de K, es claro que si $K, u \models \varphi$, entonces la primera subfórmula de φ induce un homomorfismo de G_1 en H, y la segunda fórmula implica que no existe un homomorfismo de G_2 en H. Y un argumento similar sirve para el recíproco. □

Para fragmentos de CPDL$^+$ podemos obtener mejores resultados. Sea $\mathcal{G}$ una clase de grafos. Definimos el problema

MODEL CHECKING CPDL$^+(\mathcal{G})$	
Entrada:	una estructura de Kripke K (finita), un mundo w de K y una fórmula φ de CPDL$^+(\mathcal{G})$
Salida:	Decidir si $K, w \models \varphi$

Teorema 85. *Para cualquier clase $\mathcal{G}$ de grafos conexos y finitos:*

(1) si $\mathcal{G} \subseteq \mathrm{TW}_k$ para algún k, entonces MODEL CHECKING CPDL$^+(\mathcal{G})$ *es* PTime*-completo;*

(2) en otro caso, MODEL CHECKING CPDL$^+(\mathcal{G})$ *no es* PTime*, bajo la hipótesis de que $\mathcal{G}$ es recursivamente enumerable y* W[1] $\neq$ FPT*. Esto se mantiene incluso para el fragmento positivo de* CPDL$^+(\mathcal{G})$*.*

Demostración. (1) El procedimiento para mostrar que MODEL CHECKING CPDL$^+(\mathrm{TW}_k)$ está en PTime es un algoritmo clásico de programación dinámica. Dada una fórmula φ y una estructura de Kripke K, iterativamente etiquetamos los mundos de K con subexpresiones ψ de φ en base al etiquetado de subexpresiones de ψ. Usaremos una relación unaria $U_\psi \subseteq W(K)$ para cada fórmula $\psi \in \mathrm{sub}(\varphi)$ y una relación binaria $B_\pi \subseteq W(K) \times W(K)$ para cada programa $\pi \in \mathrm{sub}(\varphi)$, que se inicializan todos como conjuntos vacíos. Comenzamos procesando todas las proposiciones atómicas $p \in \mathrm{sub}(\varphi)$ y programas atómicos $a \in \mathrm{sub}(\varphi)$: fijamos $U_p = X_p$ y $B_a = \rightarrow_a$. Para los inversos de programas atómicos $\bar{a} \in \mathrm{sub}(\varphi)$ fijamos $B_{\bar{a}} = (\rightarrow_a)^{-1}$. Ahora, sea $\psi \in \mathrm{sub}(\varphi)$ tal que todas las subexpresiones en $\mathrm{sub}(\psi) \setminus \{\psi\}$ ya se procesaron. Procesamos ψ como sigue:

- Si $\psi = \psi_1 \wedge \psi_2$, fijamos $U_\psi = U_{\psi_1} \cap U_{\psi_2}$,
- si $\psi = \neg\psi'$, fijamos $U_\psi = W(K) \setminus U_{\psi'}$,
- si $\psi = \langle\pi\rangle$, fijamos $U_\psi = \{w \in W(K) \mid \exists w'.(w, w') \in B_\pi\}$.

Estas operaciones se realizan en tiempo polinomial. Para procesar programas, sea $\pi \in \mathrm{sub}(\varphi)$ tal que todas las subexpresiones en $\mathrm{sub}(\pi) \setminus \{\pi\}$ ya se procesaron. Procesamos π como sigue:

- Si $\pi = \pi_1 \star \pi_2$ con $\star \in \{\circ, \cup\}$, fijamos $B_\pi = B_{\pi_1} \star B_{\pi_2}$,
- si $\pi = (\pi')^*$, fijamos $B_\pi = B^*_{\pi'}$,
- si $\pi = C[x_s, x_t]$, evaluamos C como si fuera una CQ sobre las expresiones ya procesadas para definir B_π.

Los primeros dos ítems se pueden hacer en tiempo cuadrático. Para el caso de programas conjuntivos nos valemos del hecho de que la evaluación de CQs de treewidth $\leq k$ se puede hacer en tiempo polinomial [60, Theorem 3], realizando un proceso sobre la descomposición de árbol de la consulta de abajo hacia arriba. Esto proporciona un algoritmo que es lineal en el tamaño de φ y polinomial en K, donde el grado del polinomio es $k+1$ si $\varphi \in \mathrm{CPDL}^+(\mathrm{TW}_k)$.

La cota inferior surge porque *model checking* de la lógica modal es PTime-hard [125, Proposition 5].

(2) Esto se obtiene del hecho de que una afirmación similar se conoce para CQs: *model checking* de CQs Booleanas sobre relaciones binarias cuyos grafos subyacentes están en $\mathcal{G}$ no es PTime a menos que W[1] = FPT [109, Corollary 19]. El resultado se sigue de una reducción polinomial del problema de evaluación de CQs Booleanas al MODEL CHECKING $\mathrm{CPDL}^+(\mathcal{G})$: una CQ Booleana q se cumple en una estructura (finita) K si y solo si la fórmula $\langle C[x,x]\rangle$ se satisface en algunos mundos de K, donde C es el conjunto de átomos de q y x es cualquier variable de q. Esta reducción, que es polinomial, implica que MODEL CHECKING $\mathrm{CPDL}^+(\mathcal{G})$ no puede estar en PTime bajo la hipótesis de que W[1] $\neq$ FPT. □

Capítulo 4

Conclusiones

Para finalizar hablaremos de algunos temas de discusión recientes y de cómo nuestros resultados contribuyen o son de relevancia en estos planteamientos, comparando con otros trabajos surgidos en paralelo y que abordan problemas similares a los nuestros. También mencionaremos posibles direcciones nuevas de investigación que se desprenden de esta tesis.

Estandarización. En abril del año 2024 fue oficializado por la *International Organization for Standardization* (ISO) que GQL (siglas de *Graph Query Language*) sea el lenguaje de consulta estándar para el manejo y administración de bases de datos orientadas a grafos, en analogía al papel que cumple SQL respecto a las bases de datos relacionales. En concreto, GQL es un lenguaje que permite razonar sobre *property graphs*[1] [16, 69], que es una forma de representación de la información que prioriza el almacenamiento de datos y búsqueda óptima de respuestas a consultas, a diferencia de otros modelos como los RDF graphs (que es un W3C standard), más enfocado a la interoperabilidad de distintas entidades. Muchos de nuestros resultados y planteamientos se pueden entender en esta misma línea, ya sea desde los resultados de reparación y CQA (también llamado OMQA) vistos en los Capítulos 1 y 2, hasta los resultados de satisfacibilidad y *model checking* del Capítulo 3. En virtud del creciente interés que ha habido por GQL en los últimos años, se vuelve necesario explorar con rigurosidad teórica muchos aspectos formales del mismo, lo cual se encuentra en un estado inicial de desarrollo [98]. Algunos trabajos recientes ya apuntan en esta dirección valiéndose de los standards de W3C para la Web Semántica, como [17, 145], donde se establece la equivalencia de SPARQL (lenguaje de consulta para RDF graphs y que es también un W3C standard) con el álgebra relacional, [59], donde estudian el *containment problem* para varios fragmentos de SPARQL, o [10, 11], donde analizan los problemas de reparación y CQA para restricciones expresadas en el fragmento SHACL de SPARQL. Por otra parte, GQL generaliza a los CRPQ por medio de la implementación de *graph patterns* [69], siendo esta una característica que comparte con SPARQL. Si bien a lo largo del trabajo utilizamos lenguajes altamente expresivos como Reg-GXPath y CPDL$^+$ que son incomparables con GQL y SPARQL, puede resultar beneficioso analizar el impacto de nuestros resultados para la comunidad de GQL en base a las similitudes sintácticas y semánticas que todos estos lenguajes poseen entre sí.

[1] En particular, todos los tipos de modelos finitos que usamos en este trabajo (data-grafos, bases de conocimiento y estructuras de Kripke) se pueden entender como *property graphs*.

Opciones de lenguajes con fines ontológicos. Reg-GXPath se presentó originalmente como un lenguaje de consulta para grafos con valores de datos en los nodos [127]. La expresividad de Reg-GXPath en relación a otros lenguajes de consulta y lenguajes lógicos fue estudiado detalladamente por Libkin et al. incluyendo además algunos resultados de interés relacionados a *query containment* y *query equivalence*, que son relevantes para optimización de evaluación y análisis estático [169]. Otras opciones de lenguajes que consideramos para analizar los problemas del Capítulo 1 son RQM (*regular query with memory*) y RDPQ (*regular data path query*) [127, 128], pero ambos tienen una complejidad de evaluación de PSpace, mientras que la de Reg-GXPath es PTime en complejidad combinada, lo que influyó en la decisión de utilizar Reg-GXPath por encima de estas otras opciones. Otra razón se debe a ciertos aspectos sintácticos que permiten considerar a Reg-GXPath como un lenguaje de restricciones, ya que cuenta con la negación para expresiones de nodo y complementación para expresiones de camino, permitiendo así subsumir muchos tipos de restricciones que se expresan mediante implicación. La noción de consistencia usada en este capítulo (Definición 15) toma en cuenta este detalle al ser presentada bajo un criterio de satisfacción global. Conceptualmente, esto parece alejarse de la visión actual de la Web Semántica que ha planteado en los últimos años varias nociones de consistencia basadas en el uso de *graph patterns* [76, 77] con la idea de capturar clases tradicionales de restricciones como dependencias funcionales o reglas existenciales [81], pero esto último ciertamente se asemeja a nuestra intención de escoger un lenguaje que abarca distintos tipos de expresiones.

Paradigmas semánticos. En muchos casos, al estudiar problemas como reparación y CQA se establecen suposiciones lógicas como la semántica de mundo abierto, que es compatible con el concepto de subreparación usado en el Capítulo 1. Sin embargo, nuestro análisis no se limitó a esta condición y estudiamos distintas semánticas que lidian con inconsistencias, como las superreparaciones vistas también en el Capítulo 1 o las extensiones consistentes del Capítulo 2. Otra forma de lidiar con inconsistencias que es de sumo interés se basa en reparaciones conjuntistas respecto al operador de *diferencia simétrica*, que fue de hecho estudiada para bases de datos orientadas a grafos en [32] bajo restricciones en C2RPC, sin embargo, estos resultados no son fácilmente adaptables a nuestro esquema adoptado de [127] basado en data-grafos y requiere del planteamiento de ciertos formalismos que hemos dejado para trabajo futuro. El problema de CQA para restricciones y consultas en Reg-GXPath no fue estudiado aún, pero notamos que respecto a semántica de subreparación y con restricciones en Reg-GXPath^{pos} la complejidad de CQA es polinomial, en vista del Teorema 25, que establece que bajo estas condiciones un data-grafo inconsistente posee una única subreparación que se puede computar en tiempo polinomial. Por otra parte, CQA fue rigurosamente estudiado en el Capítulo 2 habiéndose obtenido varios resultados de decibilidad de este problema respecto a determinados fragmentos de UC2RPQ y GNFO caracterizados por la propiedad de controlabilidad finita, pero la decibilidad de CQA para el caso concreto de consultas en UC2RPQ bajo restricciones en GNFO sigue abierto. En [113] se obtiene una respuesta parcial y afirmativa a esta pregunta considerando consultas en UCRPQ y restricciones definidas en la lógica de descripción $\mathcal{ALC}$, obteniendo además la cota superior de 2ExpTime. Sin embargo, las técnicas utilizadas en el artículo de Gutiérrez-Basulto et al. difieren de las aplicadas en este trabajo. Creemos que algunos resultados similares a los de [113] se pueden obtener mediante reducciones como la del Lema 40, pero definiendo ahora una expresión en MSO en lugar de GNFO, lo cual

puede ser una alternativa distinta y quizás más intuitiva de lo hecho por Gutiérrez-Basulto et al. Por último, nuestros resultados fueron basados en el ampliamente conocido lenguaje UC2RPQ, pero todo lo realizado es fácilmente adaptable a generalizaciones de UC2RPQ que involucren el uso de constantes e incluso conjunción con relaciones de cualquier aridad.

Identificando inconsistencias bajo criterios probabilísticos. La alternativa de computar una reparación puede no ser siempre viable en la vida real y en muchos casos resulta necesario incorporar herramientas estadísticas como alternativa para reconocer las posibles *impurezas* de una base de datos y que pueden ser difíciles de detectar y eliminar siguiendo estrictos razonamientos lógicos. Inspirados por [68] donde introducen el concepto de *Probabilistic Unclean Database* (PUD), en [4] analizamos los problemas de *data cleaning* (inferir el data-grafo con mayor probabilidad dado un PUD) y *probabilistic query answering* (computar la probabilidad de una respuesta sobre un data-grafo inconsistente observado) con el mismo trasfondo teórico del Capítulo 1 y de [7]. Los resultados de [4] no forman parte de este trabajo por alejarse del enfoque principalmente lógico que subyace en el esquema general, pero destacamos que el mismo surgió como continuación de [7] tratando de seguir una línea de investigación de carácter más aplicado.

Más sobre CPDL$^+$. En relación al Capítulo 3 quedan varios planteamientos por investigar en torno al lenguaje CPDL$^+$. En los resultados de satisfacibilidad estudiados en la Sección 3.6 una fórmula se decide si es satisfacible o no sobre estructuras de Kripke posiblemente infinitas, pero desconocemos si el planteamiento análogo de satisfacibilidad finita es computable. Por los resultados obtenidos en la Sección 3.3 intuimos que esto puede ser difícil de responder, ya que generaliza el problema de decidir si una fórmula en loop-CPDL es satisfacible sobre estructuras de Kripke finitas, cuya decidibilidad es un problema abierto [103, §7]. Tampoco es claro si los resultados de satisfacibilidad obtenidos son adaptables al caso de programas generalizados. Un planteamiento análogo al de *model checking* para estructuras infinitas aparece en algunos trabajos con el nombre de *infinite state model checking* y ha sido estudiado para CPDL [102] e ICPDL [103], e intuimos que las cotas de complejidad conocidas para estos lenguajes pueden extenderse a CPDL$^+$. Creemos que nuestros resultados del Capítulo 3 también podrían valer considerando extensiones de CPDL$^+$ que admitan el uso de constantes, modelados como *nominales*, pero los detalles de esta conjetura se dejan como trabajo futuro. Por otra parte, los resultados de bisimulación vistos en la Sección 3.4 fueron adaptados recientemente para corresponderse con otra propiedad de grafos ampliamente estudiada denominada *path width*, y los resultados de ese desarrollo formaron parte de una tesis de grado en Computación dirigida por el autor de este documento [153].

Bibliografía

[1] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

[2] Samson Abramsky and Nihil Shah. Relating structure and power: Comonadic semantics for computational resources. *Journal of Logic and Computation*, 31(6):1390–1428, 08 2021.

[3] Sergio Abriola, Pablo Barceló, Diego Figueira, and Santiago Figueira. Bisimulations on Data Graphs. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 309–318, Cape Town, South Africa, April 2016. AAAI Press.

[4] Sergio Abriola, Santiago Cifuentes, Maria Vanina Martinez, Nina Pardal, and Edwin Pin. An epistemic approach to model uncertainty in data-graphs. *International Journal of Approximate Reasoning*, 160:108948, 2023.

[5] Sergio Abriola, María Emilia Descotte, Raul Fervari, and Santiago Figueira. Axiomatizations for downward XPath on data trees. *Journal of Computer and System Sciences*, 89:209–245, 2017.

[6] Sergio Abriola, María Emilia Descotte, and Santiago Figueira. Model theory of XPath on data trees. Part II: Binary bisimulation and definability. *Information and Computation*, 255:195–223, 2017. WoLLIC 2014.

[7] Sergio Abriola, María Vanina Martínez, Nina Pardal, Santiago Cifuentes, and Edwin Pin Baque. On the Complexity of Finding Set Repairs for Data-Graphs. *J. Artif. Int. Res.*, 76, mar 2023.

[8] Loredana Afanasiev, Patrick Blackburn, Ioanna Dimitriou, Bertrand Gaiffe, Evan Goris, Maarten Marx, and Maarten de Rijke. PDL for ordered trees. *Journal of Applied Non-Classical Logics*, 15(2):115–135, 2005.

[9] Foto N Afrati and Phokion G Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *Proceedings of the 12th International Conference on Database Theory*, pages 31–41, 2009.

[10] Shqiponja Ahmetaj, Robert David, Axel Polleres, and Mantas Šimkus. Repairing SHACL constraint violations using answer set programming. In Ulrike Sattler, Aidan Hogan, Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d'Amato, editors, *The Semantic Web – ISWC 2022*, pages 375–391, Cham, 2022. Springer International Publishing.

[11] Shqiponja Ahmetaj, Timo Camillo Merkl, and Reinhard Pichler. Consistent Query Answering over SHACL Constraints. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*, pages 2–13, 8 2024.

[12] Natasha Alechina, Stephane Demri, and M. de Rijke. A modal perspective on path constraints. *J. Log. Comput.*, 13:939–956, 2003.

[13] Giovanni Amendola, Nicola Leone, and Marco Manna. Finite controllability of conjunctive query answering with existential rules: Two steps forward. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5189–5193. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[14] Manish Kumar Anand, Shawn Bowers, and Bertram Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT*, volume 10, pages 287–298, 2010.

[15] Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[16] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. PG-Schema: Schemas for property graphs. *Proc. ACM Manag. Data*, 1(2), June 2023.

[17] Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008*, pages 114–129, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[18] Carlos Areces, Christof Monz, Hans Nivelle, and Maarten Rijke. The guarded fragment: Ins and outs. *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*, 08 2004.

[19] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, volume 99, pages 68–79. Citeseer, 1999.

[20] Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 305–316, 2011.

[21] Albert Atserias. Conjunctive query evaluation by search-tree revisited. *Theoretical Computer Science*, 371(3):155–168, 2007. Database Theory.

[22] Franz Baader, Ian Horrocks, and Ulrike Sattler. Chapter 3 description logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 135–179. Elsevier, 2008.

[23] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Êric Salvat. Extending decidable cases for rules with existential variables. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, page 677–682, San Francisco, CA, USA, 2009.

Morgan Kaufmann Publishers Inc.

[24] Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Michaël Thomazo. Answering conjunctive regular path queries over guarded existential rules. pages 793–799. ijcai.org, 2017.

[25] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the complexity lines for generalized guarded existential rules. pages 712–717. IJCAI/AAAI, 2011.

[26] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9):1620–1654, 2011.

[27] Peter Bailis and Ali Ghodsi. Eventual consistency today: Limitations, extensions, and beyond: How can applications be built on eventually consistent infrastructure given no guarantee of safety? *Queue*, 11(3):20–32, mar 2013.

[28] José Balcázar, Joaquim Gabarró, and Miklós Sántha. Deciding bisimilarity is P-complete. *Form. Asp. Comput.*, 4(Suppl 1):638–648, nov 1992.

[29] Vince Bárány and Mikołaj Bojańczyk. Finite satisfiability for guarded fixpoint logic. *Information Processing Letters*, 112(10):371–375, 2012.

[30] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *Journal of the ACM*, 62(3):22:1–22:26, 2015.

[31] Pablo Barceló, Diego Figueira, and Miguel Romero. Boundedness of conjunctive regular path queries. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 104:1–104:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[32] Pablo Barceló and Gaëlle Fontaine. On the data complexity of consistent query answering over graph databases. *Journal of Computer and System Sciences*, 88:164–194, 2017.

[33] Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying graph patterns. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, page 199–210, New York, NY, USA, 2011. Association for Computing Machinery.

[34] Pablo Barceló, Jorge Pérez, and Juan L Reutter. Relative expressiveness of nested regular expressions. *AMW*, 12:180–195, 2012.

[35] Pablo Barceló Baeza. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 175–188. ACM, 2013.

[36] Catriel Beeri and Moshe Vardi. Formal systems for tuple and equality generating dependencies. *SIAM J. Comput.*, 13:76–98, 02 1984.

[37] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*,

31(4):718–741, sep 1984.

[38] Luigi Bellomarini, Daniele Fakhoury, Georg Gottlob, and Emanuel Sallinger. Knowledge graphs and enterprise AI: The promise of an enabling technology. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 26–37, 2019.

[39] Mordechai Ben-Ari, Joseph Y. Halpern, and Amir Pnueli. Deterministic propositional dynamic logic: Finite models, complexity, and completeness. *Journal of Computer and System Sciences*, 25(3):402–417, 1982.

[40] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. pages 817–826, 2016.

[41] Leopoldo Bertossi. Database repairing and consistent query answering. *Synthesis Lectures on Data Management*, 3(5):1–121, 2011.

[42] Meghyn Bienvenu. Ontology-mediated query answering: harnessing knowledge to get more from data. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 4058–4061. AAAI Press, 2016.

[43] Meghyn Bienvenu. A short survey on inconsistency handling in ontology-mediated query answering. *KI - Künstliche Intelligenz*, 34, 07 2020.

[44] Meghyn Bienvenu, Camille Bourgaux, and François Goasdoué. Querying inconsistent description logic knowledge bases under preferred repair semantics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[45] Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In Wolfgang Faber and Adrian Paschke, editors, *Reasoning Web. Web Logic Rules - 11th International Summer School 2015*, volume 9203, pages 218–307, 2015.

[46] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2002.

[47] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 226–234, New York, NY, USA, 1993. Association for Computing Machinery.

[48] Mikoaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3), May 2009.

[49] Iovka Boneva, Sławek Staworko, and Jose Lozano. Consistency and certain answers in relational to RDF data exchange with shape constraints. In Jérôme Darmont, Boris Novikov, and Robert Wrembel, editors, *New Trends in Databases and Information Systems*, pages 97–107, Cham, 2020. Springer International Publishing.

[50] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. How to Best Nest Regular Path Queries. In *Informal Proceedings of the 27th International Workshop on Description Logics*,

Vienne, Austria, July 2014.

[51] Florian Bruse and Martin Lange. A Decidable Non-Regular Modal Fixpoint Logic. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[52] Olaf Burkart, Didier Caucal, Faron Moller, and Bernhard Steffen. Chapter 9 - verification on infinite structures. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 545–623. Elsevier Science, Amsterdam, 2001.

[53] Vince Bárány, Georg Gottlob, and Martin Otto. Querying the guarded fragment. *Logical Methods in Computer Science*, 10, 09 2013.

[54] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: query answering under expressive relational constraints. *J. Artif. Int. Res.*, 48(1):115–174, oct 2013.

[55] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog extensions for tractable query answering over ontologies. In Roberto De Virgilio, Fausto Giunchiglia, and Letizia Tanca, editors, *Semantic Web Information Management - A Model-Based Perspective*, pages 249–279. Springer, 2009.

[56] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Semant.*, 14:57–83, 2012.

[57] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 260–271. ACM, 2003.

[58] Steve Cassidy. Generalising XPath for directed graphs. In Tonya Gaylord and Kate Hamilton, editors, *Proceedings of Extreme Markup Languages 2003*. Mulberry Technologies, 2003. Extreme Markup Languages 2003 ; Conference date: 04-08-2003 Through 08-08-2003.

[59] Melisachew Chekol, Jérôme Euzenat, Pierre Genevès, and Nabil Layaïda. SPARQL query containment under schema. *Journal on Data Semantics*, 7, 09 2018.

[60] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. In Foto Afrati and Phokion Kolaitis, editors, *Database Theory — ICDT '97*, pages 56–70, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[61] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1):90–121, 2005.

[62] James Clark. XML path language (xpath). 2001.

[63] Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[64] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language

supporting recursion. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, SIGMOD '87, page 323–330, New York, NY, USA, 1987. Association for Computing Machinery.

[65] Ryszard Danecki. Nondeterministic propositional dynamic logic with intersection is decidable. In Andrzej Skowron, editor, *Computation Theory*, pages 34–53, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[66] Ryszard Danecki. Propositional dynamic logic with strong loop predicate. In *Proceedings of the Mathematical Foundations of Computer Science 1984*, page 573–581, Berlin, Heidelberg, 1984. Springer-Verlag.

[67] Maarten de Rijke. Description logics and modal logics. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Proceedings of the 1998 International Workshop on Description Logics (DL'98), IRST, Povo - Trento, Italy, June 6-8, 1998*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998.

[68] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. A formal framework for probabilistic unclean databases. In Pablo Barcelo and Marco Calautti, editors, *22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:18, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[69] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Hannes Voigt, Oskar van Rest, Domagoj Vrgoč, Mingxi Wu, and Fred Zemke. Graph Pattern Matching in GQL and SQL/PGQ. In *SIGMOD '22: International Conference on Management of Data*, Philadelphia, United States, June 2022.

[70] Alin Deutsch, Alan Nash, and Jeff Remmel. The chase revisited. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '08, page 149–158, New York, NY, USA, 2008. Association for Computing Machinery.

[71] AnHai Doan, Alon Halevy, and Zachary Ives. 2 - manipulating query expressions. In AnHai Doan, Alon Halevy, and Zachary Ives, editors, *Principles of Data Integration*, pages 21–63. Morgan Kaufmann, Boston, 2012.

[72] Christian Doczkal and Joachim Bard. Completeness and decidability of converse PDL in the constructive type theory of Coq. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, page 42–52, New York, NY, USA, 2018. Association for Computing Machinery.

[73] Christian Doczkal, Guillaume Combette, and Damien Pous. A formal proof of the minor-exclusion property for treewidth-two graphs. In Jeremy Avigad and Assia Mahboubi, editors, *Interactive Theorem Proving*, pages 178–195, Cham, 2018. Springer International Publishing.

[74] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, and Li Ma. Scalable semantic retrieval through summarization and refine-

ment. volume 1, pages 299–304, 01 2007.

[75] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Li Ma, Edith Schonberg, Kavitha Srinivas, and Xingzhi Sun. Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008*, pages 403–418, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[76] Dominique Duval, Rachid Echahed, and Frédéric Prost. An algebraic graph transformation approach for RDF and SPARQL. In *GCM@STAF*, 2020.

[77] Dominique Duval, Rachid Echahed, and Frederic Prost. On foundational aspects of RDF and SPARQL, 2020.

[78] Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49(2):129–141, 1961.

[79] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005. Database Theory.

[80] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21, 2012.

[81] Wenfei Fan. Dependencies for graphs: Challenges and opportunities. *Journal of Data and Information Quality (JDIQ)*, 11(2):1–12, 2019.

[82] Diego Figueira. Satisfiability of downward XPath with data equality tests. In *Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, page 197–206, New York, NY, USA, 2009. Association for Computing Machinery.

[83] Diego Figueira. *Reasoning on words and trees with data*. Theses, École normale supérieure de Cachan - ENS Cachan, December 2010.

[84] Diego Figueira, Santiago Figueira, and Carlos Areces. Model theory of XPath on data trees. Part I: bisimulation and characterization. *J. Artif. Int. Res.*, 53(1):271–314, May 2015.

[85] Diego Figueira, Santiago Figueira, and Edwin Pin. PDL on Steroids: on Expressive Extensions of PDL with Intersection and Converse. In *Annual Symposium on Logic in Computer Science (LICS)*, Proceedings of Annual Symposium on Logic in Computer Science (LICS), Boston, United States, June 2023.

[86] Diego Figueira, Santiago Figueira, and Edwin Pin Baque. Finite Controllability for Ontology-Mediated Query Answering of CRPQ. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 381–391, 9 2020.

[87] Diego Figueira, Artur Jez, and Anthony W. Lin. Data path queries over embedded graph databases. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '22, page 189–201, New York, NY, USA, 2022. Association

for Computing Machinery.

[88] Michael J. Fischer and Richard E. Ladner. Propositional modal logic of programs. STOC '77, page 286–294, New York, NY, USA, 1977. Association for Computing Machinery.

[89] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[90] S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Querying and repairing inconsistent XML data. In Anne H. H. Ngu, Masaru Kitsuregawa, Erich J. Neuhold, Jen-Yao Chung, and Quan Z. Sheng, editors, *Web Information Systems Engineering – WISE 2005*, pages 175–188, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[91] S. Flesca, F. Furfaro, and E. Masciari. On the minimization of XPath queries. *J. ACM*, 55(1), February 2008.

[92] M. Forti and F. Honsell. Set theory with free construction principles. *Annali Scuola Normale Superiore, Pisa*, X(3):493–522, 1983.

[93] Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, and Domagoj Vrgoč. A researcher's digest of GQL. In Floris Geerts and Brecht Vandevoort, editors, *26th International Conference on Database Theory (ICDT 2023)*, volume 255 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:22, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[94] Roland Fraïssé. Sur les classifications des systems de relations. *Publ. Sci. Univ. Alger I*, 1954.

[95] Nicolas Fröhlich, Arne Meier, Nina Pardal, and Jonni Virtema. A logic-based framework for database repairs, 2024.

[96] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *Journal of Computer and System Sciences*, 73(4):610–635, 2007. Special Issue: Database Theory 2005.

[97] Michael Gertz. *Diagnosis and Repair of Constraint Violations in Database Systems*, volume 19 of *DISDBIS*. Infix Verlag, St. Augustin, Germany, 1996.

[98] Amélie Gheerbrant, Leonid Libkin, Liat Peterfreund, and Alexandra Rogova. GQL and SQL/PGQ: Theoretical models and expressive power, 2024.

[99] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, AAAI'94, page 205–212. AAAI Press, 1994.

[100] Tomasz Gogacz, Víctor Gutiérrez-Basulto, Yazmin Ibáñez-García, Jean Christoph Jung, and Filip Murlak. On finite and unrestricted query entailment beyond SQ with number restrictions on transitive roles. volume 2373 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[101] Tomasz Gogacz, Yazmín Angélica Ibáñez-García, and Filip Murlak. Finite query answering in expressive description logics with transitive roles. *CoRR*, abs/1808.03130, 2018.

[102] Stefan Göller and Markus Lohrey. Infinite state model-checking of propositional dynamic logics. In Zoltán Ésik, editor, *Computer Science Logic*, pages 349–364, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[103] Stefan Göller, Markus Lohrey, and Carsten Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. 74(1):279–314, 2009.

[104] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, June 2005.

[105] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, May 2001.

[106] Erich Grädel. Description logics and guarded fragments of first order logic. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Proceedings of the 1998 International Workshop on Description Logics (DL'98), IRST, Povo - Trento, Italy, June 6-8, 1998*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998.

[107] Erich Grädel, Colin Hirsch, and Martin Otto. Back and forth between guarded and modal logics. *ACM Transactions on Computational Logic*, 3, 01 2001.

[108] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata logics, and infinite games: a guide to current research*. Springer-Verlag, Berlin, Heidelberg, 2002.

[109] Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, page 657–666, New York, NY, USA, 2001. Association for Computing Machinery.

[110] Erich Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999.

[111] Erich Grädel. Guarded fixed point logics and the monadic theory of countable trees. *Theoretical Computer Science*, 288(1):129–152, 2002. Complexity and Logic.

[112] Víctor Gutiérrez-Basulto, Yazmín Ibáñez García, Jean Christoph Jung, and Filip Murlak. Answering regular path queries mediated by unrestricted SQ ontologies. *Artif. Intell.*, 314(C), January 2023.

[113] Víctor Gutiérrez-Basulto, Albert Gutowski, Yazmín Ibáñez-García, and Filip Murlak. Finite entailment of UCRPQs over ALC ontologies (extended abstract). In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 6442–6446. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Sister Conferences Best Papers.

[114] David Harel, Amir Pnueli, and Jonathan Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[115] David Harel and Eli Singerman. More on nonregular PDL: Finite models and fibonacci-like

programs. *Information and Computation*, 128(2):109–118, 1996.

[116] Stijn Heymans, Li Ma, Darko Anicic, Zhilei Ma, Nathalie Steinmetz, Yue Pan, Jing Mei, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, Cristina Feier, Graham Hench, Branimir Wetzstein, and Uwe Keller. Ontology reasoning with large data repositories. In Martin Hepp, Pieter De Leenheer, Aldo de Moor, and York Sure, editors, *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications*, volume 7 of *Semantic Web and Beyond: Computing for Human Experience*, pages 89–128. Springer, 2008.

[117] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37, July 2021.

[118] Neil Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25(1):76–98, 1982.

[119] David S Johnson and Anthony Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and system Sciences*, 28(1):167–189, 1984.

[120] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[121] Phokion G. Kolaitis and Moshe Y. Vardi. On the expressive power of datalog: Tools and a case study. *J. Comput. Syst. Sci.*, 51(1):110–134, 1995.

[122] Dexter Kozen and Rohit Parikh. An elementary proof of the completeness of PDL. *Theoretical Computer Science*, 14(1):113–118, 1981.

[123] Saul Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24(1):1–14, 1959.

[124] Martin Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic*, 4(1):39–49, 2006.

[125] Martin Lange and Carsten Lutz. 2-ExpTime lower bounds for propositional dynamic logics with intersection. *Journal of Symbolic Logic*, 70, 12 2005.

[126] Domenico Lembo and Marco Ruzzi. Consistent query answering over description logic ontologies. In Massimo Marchiori, Jeff Z. Pan, and Christian de Sainte Marie, editors, *Web Reasoning and Rule Systems*, pages 194–208, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[127] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *Journal of the ACM (JACM)*, 63(2):1–53, 2016.

[128] Leonid Libkin and Domagoj Vrgoč. Regular path queries on graphs with data. In *Proceedings*

of the 15th International Conference on Database Theory, pages 74–85, 2012.

[129] Christof Löding and Olivier Serre. Propositional dynamic logic with recursive programs. In Luca Aceto and Anna Ingólfsdóttir, editors, *Foundations of Software Science and Computation Structures*, pages 292–306, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[130] Andrei Lopatenko and Leopoldo Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *In ICDT*, pages 179–193. Springer, 2007.

[131] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I Simari. From classical to consistent query answering under existential rules. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[132] Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I Simari. Complexity of inconsistency-tolerant query answering in Datalog+/–. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*, pages 488–500. Springer, 2013.

[133] C. Lutz. PDL with intersection and converse is decidable. LTCS-Report LTCS-05-05, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.

[134] Maarten Marx. XPath and modal logics of finite DAG's. In Marta Cialdea Mayer and Fiora Pirri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 150–164, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[135] Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, January 2004.

[136] Robin Milner. *A Calculus of Communicating Systems*, volume 92. 1980.

[137] Yoàv Montacute and Glynn Winskel. Concurrent games over relational structures: The origin of game comonads. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '24, New York, NY, USA, 2024. Association for Computing Machinery.

[138] Marie-Laure Mugnier and Michaël Thomazo. An introduction to ontology-based query answering with existential rules. In Manolis Koubarakis, Giorgos B. Stamou, Giorgos Stoilos, Ian Horrocks, Phokion G. Kolaitis, Georg Lausen, and Gerhard Weikum, editors, *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, volume 8714 of *Lecture Notes in Computer Science*, pages 245–278. Springer, 2014.

[139] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2):267–276, 1987.

[140] Adrian Onet. The Chase Procedure and its Applications in Data Exchange. In Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt, editors, *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 1–37. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2013.

[141] Magdalena Ortiz and Mantas Simkus. Reasoning and query answering in description logics. In Thomas Eiter and Thomas Krennwallner, editors, *Reasoning Web. Semantic Technologies for Advanced Query Answering - 8th International Summer School 2012*, volume 7487, pages 1–53, 2012.

[142] D. Park. Concurrency and automata on infinite sequences. In *Proc. of the 5th GI-Conference on Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183. Springer-Verlag, 1981.

[143] Sophie Pinchinat, Sasha Rubin, and François Schwarzentruber. Formula synthesis in propositional dynamic logic with shuffle. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 9902–9909. AAAI Press, 2022.

[144] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

[145] Axel Polleres. From SPARQL to rules (and back). In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 787–796, New York, NY, USA, 2007. Association for Computing Machinery.

[146] V. R. Pratt. Models of program logics. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 115–122, 1979.

[147] Gerard Renardel de Lavalette, Barteld Kooi, and Rineke Verbrugge. *A strongly complete proof system for propositional dynamic logic*, pages 377–393. University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science, 2002. Relation: http://www.rug.nl/informatica/organisatie/overorganisatie/iwi Rights: University of Groningen. Research Institute for Mathematics and Computing Science (IWI).

[148] Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. Regular queries on graph databases. *Theoretical Computer Science*, 61(1):31–83, 2017.

[149] Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. pages 356–365, 2006.

[150] Riccardo Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *Journal of Computer and System Sciences*, 77(3):572–594, 2011.

[151] Sebastian Rudolph. Undecidability results for database-inspired reasoning problems in very expressive description logics. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2016.

[152] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4), May 2009.

[153] Mauro Schiavinato. Fragmentos de CPDL$^+$ mediante propiedades de pathwidth. Tesis de grado, Departamento de Computación, Universidad de Buenos Aires, 2024.

[154] Klaus Schild. A correspondence theory for terminological logics: preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, page 466–471, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[155] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[156] Krister Segerberg. A completeness theorem in the modal logic of programs. *Notices AMS*, 24(6), 1977.

[157] Luc Segoufin. Static analysis of XML processing with data values. *SIGMOD Rec.*, 36(1):31–38, mar 2007.

[158] Luc Segoufin. A survey on guarded negation. *SIGLOG News*, 4(3):12–26, 2017.

[159] Amit Singhal. Introducing the knowledge graph: things, not strings. 2012.

[160] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1):121–141, 1982.

[161] Zijing Tan, Wei Wang, JianJun Xu, and Baile Shi. Repairing Inconsistent XML Documents. In Jérôme Lang, Fangzhen Lin, and Ju Wang, editors, *Knowledge Science, Engineering and Management*, pages 379–391, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[162] Balder ten Cate. The expressivity of XPath with transitive closure. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, page 328–337, New York, NY, USA, 2006. Association for Computing Machinery.

[163] Balder Ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. On the data complexity of consistent query answering. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 22–33, 2012.

[164] Balder ten Cate and Luc Segoufin. Unary negation. *Logical Methods in Computer Science*, Volume 9, Issue 3, September 2013.

[165] Johan van Benthem. Modal correspondence theory. 01 1976.

[166] H. P. Van Ditmarsch, W. Van Der Hoek, and B. P. Kooi. *Concurrent Dynamic Epistemic Logic*, pages 105–143. Springer Netherlands, Dordrecht, 2003.

[167] M. Vardi. A note on the reduction of two-way automata to one-way atuomata. *Inf. Process. Lett.*, 30(5):261–264, 1989.

[168] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.

[169] Domagoj Vrgoč. *Querying graphs with data.* PhD thesis, Ph.D. Dissertation. School of Informatics, University of Edinburgh., 2014.

[170] Aleksa Vukotic, Nicki Watt, Tareq Abedrabbo, Dominic Fox, and Jonas Partner. *Neo4j in Action*. Manning, 2015.

[171] Jef Wijsen. Condensed representation of database repairs for consistent query answering. In *Proceedings of the 9th International Conference on Database Theory*, ICDT '03, pages 378–393. Springer-Verlag, 2002.

[172] Jef Wijsen. Database repairing using updates. *ACM Transactions on Database Systems (TODS)*, 30(3):722–768, 2005.