Non-Standard Semantics for Graph Query Languages

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von
Stephan Mennicke
geboren am 29.10.1987
in Halle (Saale)

Eingereicht am:	25.10.2019
Disputation am:	11.03.2020
1. Referent:	Prof. Dr. Wolf-Tilo Balke
2. Referent:	Prof. Dr. Gerhard Weikum

Preliminary Online Version

Abstract

We address the fundamentals of graph queries, which is best characterized by the notion of *graph pattern matching*. In contrast to classical subgraph matching notions, like graph homomorphisms or graph isomorphisms, variants of *simulation* have been devised for numerous graph database management tasks.

In the late 1990s, simulations were introduced to graph database management as a tool for modeling languages for graph-structured data. Back then, graphs were almost always tree-like. Therefore, we devise an old theory for modern graph databases, which are not necessarily trees. Here, we observe an interesting interplay between the notion of simulation and the root nodes of tree-structured data. After we have reestablished the fundamental assumptions behind root nodes – without introducing them syntactically to general graphs – we derive a sound semantics for graph schemas. We even enhance the original schema model by so-called *mandatory attributes*, strongly relating to key attributes in the relational data model. Again we obtain a sound semantic foundations for this model of graph schemas. However, we show that this is only possible introducing syntactic restrictions on the graph schema model.

Simulations have a summarizing character in that they are capable of collapsing and expanding arbitrary graphs. This is impossible in the classical graph database perspective without an appropriate complementation of graph homomorphisms with query language operators. Therefore, we argue for simulation to have a higher pragmatic value over graph-homomorphic matching.

In the next step, we pull simulations out of their pure pattern matching method. We study the query language SPARQL under a simulation-based matching mechanism for *basic graph patterns*. As soon as interesting query operators (like joins) are added, the resulting language is not complete w. r. t. the original SPARQL semantics. In fact, SPARQL matches get lost under simulation. For some well-known SPARQL fragments, we prove completeness and even tractability. Several steps of approximation allow for the derivation of a complete semantics for full SPARQL. The semantics produces a single match that summarizes all original matches. From this semantics we develop a pruning method for SPARQL query processing. Therefore, we had to develop a novel algorithmic solution to the base simulation problem because well-known general solutions do not scale with the size of the data. Our solution exhibits the usual assumptions when dealing with (graph) database querying tasks. Beyond performance improvements, we evaluate our newly devised pruning semantics for SPARQL query processing.

Kurzfassung

In dieser Arbeit befassen wir uns mit den Grundlagen der Graphdatenbankanfragen, dem sogenannten *Graph Pattern Matching*. Als effiziente Alternative zur klassischen Subgraphisomorphie haben sich früh Varianten von Simulation für diverse Aufgaben in graphdatenbankverwandten Anwendungsgebieten etabliert.

Simulationen fanden in den späten 1990er Jahren Einzug in die Beschreibungssprachen von Graphdaten, damals noch in Form baumstrukturierter Daten. Dieser Theorie widmen wir uns zuallererst, da moderne Graphdatenbankmodelle nicht notwendigerweise baumstrukturiert sind. Hier stellt sich ein interessantes Zwischenspiel zwischen der vormals bekannten Halbordnung der Simulation und den Wurzelknoten baumstrukturierter Daten heraus. Nachdem die zugrundeliegenden Annahmen von Wurzeln wieder von uns etabliert worden sind, ohne dass die Wurzel als syntaktisches Datenelement auftaucht, gelingt es eine korrekte Semantik für Graphschemata abzuleiten. Zusätzlich erweitern wir das Modell um sogenannte *verpflichtende Attribute*, die beispielsweise beim klassischen relationalen Modell als Schlüsselattribute wiederauftauchen. Auch hierfür entwickeln wir eine korrekte Semantik, die sich leider aber nur durch starke Einschränkungen der modellierbaren Graphdaten aufrechterhalten lässt.

Um ein anderes Beispiel zu nennen, können Simulationen auch gut dazu verwendet werden, Kreise durch ein einziges endliches *Pattern* darzustellen und in einer entsprechenden Datenbasis aufzufinden. Dieses können homomorphismenbasierte Ansätze nicht leisten, solange sie nicht von einer Graphanfragesprache komplementiert werden. Wir argumentieren für Simulationen, im Speziellen für sogenannte *Duale Simulationen*, die einen hohen pragmatischen Wert gegenüber der Subgraphisomorphie aufweisen.

Im nächsten Schritt der Arbeit wollen wir duale Simulationen aus dem reinen *Graph Pattern Matching* herausholen und mit klassischen Operatoren der Anfragesprache SPARQL komplementieren. Leider stellt sich dieses als prinzipiell unlösbare Aufgabe heraus, sobald man interessante Verknüpfungsoperatoren der Sprache hinzufügen möchte. Die resultierenden Anfragesprachen sind weder korrekt noch vollständig bezüglich der Ursprungssemantik ist. Für Fragmente gelingt es, Vollständigkeit nachzuweisen. Sogar effiziente Lösbarkeit der klassischen Anfragesprachprobleme kann gezeigt werden. Über mehrere Approximierungsschritte gelingt es schließlich eine vollständige SPARQL Semantik auf Basis von dualer Simulation zu definieren. Die Semantik selbst hat die Eigenschaft, mit einem einzigen *Match* alle SPARQL-Resultate zu beschreiben bzw. zusammenzufassen. Daraus entwickeln wir eine algorithmische Lösung, die als Pruningschritt zur SPARQL-Anfrageverarbeitung verwendet werden kann. Auch hier gibt es zunächst Hindernisse. Etablierte Algorithmen, die das Simulationsproblem effizient lösen, skalieren alle gleich schlecht mit der Datenbankgröße. Da wir es aber mit enorm großen Datengraphen zutun haben, scheinen die allgemeineren Werkzeuge zu wenig auf die Annahmen in Graphdaten eingestellt zu sein. Wir analysieren solche Annahmen, entwickeln auf deren Basis einen Algorithmus und eine Anwendung, die im Vergleich zu den bestehenden Algorithmen deutlich performanter ist. Außerdem evaluieren wir mit dem entwickelten Werkzeug unsere Pruning-Semantik für SPARQL.

Contents

CHAPTER 1

Introduction

Not only do graphs span a wide variety of application contexts, but these contexts have also started producing enormous volumes of their data in graph-shape [23]. For instance, social networks store interconnections between its participants like *friendship* [26]. Other examples include *knowledge* [41] or *knowledge* paired with product information [42]. In many more application domains [118], vast amounts of graph data require organizational principles and operations to manipulate the data [10]. As *graph data* we understand entities, represented by nodes, and relationships between these entities, represented by (directed and labeled) edges. Organization and manipulation of *vast amounts of data* is the core competence of *database systems* but in contrast to well-structured relational data, graph data is usually unstructured, which massively complicates their analysis. Fortunately, early standardization efforts and foundational studies of graph data representations [8, 9, 13, 14, 62, 122] and the support of established and new database system vendors [47, 31] took place. The *Semantic Web* movement, institutionalized by the W3C, has been among the earliest such efforts. Also, several research prototypes concentrate on diverse graph database management tasks, e. g., from efficient storage [104], scalable join processing [16, 15], up to answering semantic queries [102]. Standardization and formalization by the W3C do not stop at graph data representation but also include languages for querying the stored data. SPARQL [114, 65], the W3C recommendation for querying data on the Web, is well-established by practitioners and researchers [14, 12, 71]. Alternatives like Cypher[1], the query language of the famous graph database management system Neo4j[2], have recently got the attention of researchers, who started formalizing the language's semantics [88, 58]. Such a formalization forms a solid ground for future research.

When I started focusing on graph databases some years ago, the field presented many familiar problems to a person with a scientific background in the *theory of programming language semantics*. Back then, my advisor gave me an initial pointer to *Exemplar Queries* [100].

A New and Old Way of Searching. Assume we are given a bibliographic database (already loaded into a database system), i. e., it stores information about published papers, preprints, authors, and the like. Unfortunately, the database system provides no interface we are familiar with, e. g., the query language and/or the database schema are entirely unknown to us. However, we do know something about an author and her scientific papers that should be stored somewhere in the database. We provide a list of keywords, e. g., the author's full name and some of her most important works, which may be the starting

[1] `https://neo4j.com/developer/cypher-query-language/`
[2] `https://neo4j.com/`

point of an *Exemplar Query process*. Exemplar Queries build on the premise that users are not generally trained in posing queries (in a formal query language) but know at least one match of interest. In the course of answering an exemplar query, a substructure from the database, matching the *keyword query* [33] we mentioned before, is retrieved. It is this substructure Mottin et al. call an *exemplar* [100]. In a second step, all matches *similar* to the exemplar are retrieved and presented to the user.

The measures Mottin et al. use in [100] are formally grounded in graph pattern matching. In particular, they devise their process for subgraph-isomorphic matching and matching up to *strong simulation*, a matching notion that has been introduced by Ma et al. [85]. Different forms of simulations are paradigmatic semantic models of concurrent processes [96, 107, 95, 69]. Their appearance in a database context came as a surprise at first but the connection is likewise, insightful and challenging: Processes can be represented as (possibly infinite) graphs (e. g., by structural operational semantics [95]). Two processes shall not be distinguished if the graphs match one another, e. g., in terms of executions (e. g., traces), deadlock behavior (e. g., failures), or general branching structure (e. g., bisimulations). In subgraph pattern matching (part 2 of exemplar queries), a subgraph from the database must be compared to the given graph pattern/query (i. e., exemplar) up to graph topology (e. g., graph isomorphisms). There are two distinguishing characteristics: (1) both input graphs, i. e., the pattern and the data graph, are finite; the pattern (i. e., exemplar) may also be assumed to be significantly smaller than the data graph. (2) a single match is often insufficient; in a graph query setting, we seek for all matches from a graph database for a given pattern. An immediate goal, guided by process-theoretic curiosity, was to find out whether other well-known semantic equivalences/preorders would provide meaning in a graph query setting.

By devising several examples, we found exciting uses for other matching notions drawn from the standard spectrum of semantic equivalences, the *linear-time branching-time spectrum* [133]. Therefore, we had a case to ask for the *meaning of graph patterns w. r. t. graph querying*, guided by the kind of matching mechanism that distinguishes matches from non-matches [90, 91].

1.1 Research Goals

Beyond exemplar queries, surveying the (graph) database literature for other use cases of simulation conducted in database research revealed graph schemas for semi-structured data [28, 29, 3, 103, 127, 30] and Offline indexing structures [98, 115, 75, 76, 35, 53, 123, 108, 125]. Simulations have also been proposed as viable alternatives to isomorphic subgraph matching [50, 85, 48, 55]. The main driver for the research, conducted for this thesis, circles around the following question:

How can we lift
tractable graph pattern matching
to a credible tool for
modern graph database systems?

In the course of answering this question, we take up on several challenges.

Scalability Beyond Tractability. By reviewing the motivations and experimental evaluations of the papers applying simulations in a graph pattern matching scenario, i. e., [50, 85, 48, 55], graphs associated with the label *real-world graphs* have remarkably few nodes and edges. In light of a recent survey [118], real-world graph data is, in fact, very large. If the devised pattern matching notions are tractable, why not letting them run on

real real-world graph data? Secondly, if existing algorithms cannot cope with these extents of data, are there other algorithms that can?

Graph Pattern Matching for Databases. A second peculiarity about the reported works is the choice of baseline notions and systems the authors compare themselves to. On the one hand, subgraph isomorphisms are too strict about coping with *emerging applications* [52, 50, 48]. On the other hand, subgraph isomorphisms and their associated tools are favored when it comes to evaluation. If the devised pattern matching notions are tractable and the application domain considers graph databases, why not comparing them to the runtime of a full-fledged graph database system?

Online Graph Processing. Indexing structures should be computed outside operational phases of database systems [45]. It is well-known that some pattern matching notions tend to be non-updatable, i. e., once the data graph changes, its (bi-)similarity classes must be recomputed. If we can find an algorithm that fulfills the first two goals, i. e., that algorithm processes real-world graph sizes and can keep up with database systems, is this algorithm useful for an Online task, such as graph query processing?

Credibility by Correctness. Regarding simulation as a method for schema instance matching, we first have to acknowledge the elegance through which graph schemas for semi-structured data have been devised. When considering semi-structured data, we assume a tree structure but observing modern graph databases, tree-structuredness is not necessarily given? Is there a way to renovate graph schemas towards modern graph database models? Can we derive a provably correct graph modeling methodology from it?

Semantics for Graphs Data. During our course through the literature, we saw simulations appearing at places where intractable matching notions like subgraph isomorphisms or graph homomorphisms shall be evaded. Often, a single graph pattern is insufficient to describe a user's information need up to graph homomorphism. Therefore, powerful query languages have been developed to complement the inabilities of basic matching. Can we combine tractable graph pattern matching with powerful query language operators without losing tractability or correctness?

1.2 Contributions

Throughout the last section, we briefly sketched five research goals. The findings we collect, describe, extend, and ultimately use to achieve our goals have partly been published in earlier works [90, 94, 91, 92, 93, 89]. At the beginning of each chapter, we substantiate the relationship between the chapter's contents and our earlier publications. Subsequently, find a brief description of each contribution, sorted by the order of appearance throughout this thesis.

The Semantics of (Modal) Graph Schemas. In Chapter 2, we are primarily concerned with describing the mathematical basis of graph data models, which will be used throughout the rest of this thesis. After that, we tackle our fourth research goal and devise graph schemas by Buneman et al. [29] for our graph data model. After having found the right preorder that relates graph schemas and their instances, also beyond a single graph schema via refinement, we take up on a requirement that was requested when the graph schema method was presented. In [3], a pragmatic solution was sketched. In this thesis, we aim for a more fundamental solution based on familiar principles from modal logics [81]. We contribute Buneman's graph schema model for modern graph databases and extend it

to so-called *modal graph schemas* [89]. In both cases, we intensively discuss under which conditions simulations provide us with a correct semantic foundation of the model.

The Semantics of Graph Patterns. Graph pattern matching is considered in Chapter 3. Thereby, we start with the premise that (sub-)graph isomorphisms are unnecessarily restrictive and devise scenarios that attest them a tendency to early incorporation of graph query language principles. Pattern matching up to different forms of similarity provides pragmatic power for single (finite) patterns. We contribute a comparative study of graph matching notions to arrive at dual simulations; the notion we will primarily study throughout the rest of this thesis. Furthermore, we contribute a failure theory for graph databases that allows us to express negation quite naturally by alphabet extensions of the pattern graph [90, 91].

From Graph Patterns to Graph Queries. What has been beneficial in the protected area of toy examples in Chapter 3, turns out to be a real barrier when trying to devise graph query operations for simulations. We contribute semantics for SPARQL that use dual simulations instead of graph homomorphisms. For several fragments and alternative semantic definitions, we provide proofs of (in-)correctness and tractability. Eventually, we arrive at the *maximal dual simulation semantics for* SPARQL, that culminates all the findings of the earlier semantics in a single correct pruning semantics.

Fast Online Graph Processing. We believe, the reason why former graph pattern matching algorithms have not been evaluated on real-world graph datasets and compared to established graph database systems is that these systems are incredibly optimized. The runtime reports of simulations in the mentioned papers are not overwhelming when regarding the runtime statistics of full-fledged graph database systems coping with even bigger graph data instances. Nevertheless, we have succeeded in finding a characterization of dual simulations that allows for flexible evaluation strategies [93, 92]. In Chapter 5, we contribute this new characterization of dual simulations in terms of *systems of inequalities* over bit-vector representations of node sets and families of bit-matrices for data graphs. Furthermore, our representation allows for a direct implementation of the maximal dual simulation semantics we developed in Chapter 4.

CHAPTER 2

Graph Data and Schema

Following the characterization of Angles and Gutierrez [9, 11], a *graph data model* has the following three characteristic components:

Data and Schema: Data, as well as its schema, are represented by graphs, incorporating entities or classes (as nodes) together with their properties, modeled as graph edges, in a concise and simple model. Presumably, the experienced simplicity of graph-based data stems from its direct visual representation. One of the core features of such models is to naturally capture unstructured data, as opposed to other data models like the *relational data model* [36]. If information about some entity is missing, we can omit it from the database instance. It comes from this unstructuredness that graph data is usually considered schema-less, making a clear-cut distinction between the data and its schema hard to establish, even if a schema was explicitly given.

Data Manipulation: Specialized graph transformation and query languages are established to operate on graph database instances. Often, such languages deal with graph-oriented operations like shortest paths, subgraphs, or graph patterns.

Integrity Constraints: Angles and Gutierrez count schema-instance consistency, referential integrity, as well as dependencies of a graph data model to this aspect [11].

In this chapter we concentrate on the first part which includes the basic graph structures and the means of modeling data and schema using graphs. Data manipulation, in terms of graph query languages, is the subject of Chapters 3 and 4. We only indirectly deal with constraints over data, e. g., by notions of graph schema conformance in Sections 2.3 and 2.4.

We will be concerned with one particular model of schemas for graph databases, that is not enforced in graph data, in that a graph database management system has to prevent the user from inserting inconsistent data. A graph schema is merely thought of as additional structural and semantic knowledge or *meta data*, supporting the user in query formulation or the query processor in the evaluation of a given user query. According to Abiteboul et al. [3], the two key questions we need to answer about the interrelation of a graph schema and their database instances are:

(1) When is a graph database an instance of a given schema?

(2) Which *classification* of instance objects is implied by a schema?

It is the answers to these questions that distinguish graph schemas from other approaches in the literature as we employ a non-standard, but formally well-founded, instance notion.

Goals. The main objective of this chapter is making the reader of this thesis familiar with our basic notions and notation. Beyond traditional topics of labeled and unlabeled graphs, e. g., graph morphisms, we strive *graph simulations*, which are non-standard relations for comparing graphs. Simulations appear in every subsequent chapter. Besides their definition, we also motivate, present, and prove theoretical results surrounding graph simulations. We do not intend to give a comprehensive classification or historical overview of graph data models. There are already plenty of them published in worthwhile surveys of the field [9, 62, 6, 11, 63].

Contribution. Throughout this chapter, we recover the notion of *graph schemas*, which was introduced by Buneman et al. [29] more than 20 years ago as a modeling tool for graph data. This notion has been grounded on a semistructured data model, which has been popular back then. The consequences of this data model make *graph schemas* not directly applicable to nowadays graph data models. Therefore, we discuss the particular assumptions and advantages of semistructured data towards *graph schemas*, in order to renovate them to a useful tool for today's more common graph data models. Beyond the recovery of its expressive power, we add to graph schemas the ability to flexibly requiring structure. For instance, we may want a book object to feature *title*, *author(s)*, and an *ISBN*. Buneman et al.'s graph schemas describe a graph database's allowed structure while missing out on so-called *key properties* for the objects to be classified. Parts of the findings of Sections 2.3 and 2.4 have been published as a full paper at the 38^{th} *International Conference on Conceptual Modeling* (ER 2019) [89].

Outline. In Section 2.1, we give a brief summary of basic notions and notations of (labeled) graphs from a mathematical perspective. Graphs as a representational instrument for data are discussed and formalized in Section 2.2. Therein, we begin with basic considerations for graph data models and present the *Resource Description Framework* [122], culminating in our notion of *graph databases*, as used throughout the rest of the thesis. While Section 2.3 solely cares for the notions of *graph schema* and *graph simulations*, Section 2.4 studies the modal extension of graph schemas. By Section 2.5, we close this chapter in a summarizing manner.

2.1 Graph Structure

Graph data models have been developed alongside diverse applications with quite different concerns of what aspect a graph shall model of data. These developments led to a variety of proposals that are summarized in worthwhile surveys and textbooks [3, 9, 6, 10]. The base concepts of all the different graph data models stem from the mathematical model of graphs. We summarize the core concepts and provide some data-independent examples for illustration purposes.

2.1.1 Basic Notions

The core notion of all the upcoming models is that of a *directed graph*,

$$G = (V, E), \tag{2.1}$$

which is a pair of a finite set of nodes V (sometimes also called *vertices*) and a directed edge relation $E \subseteq V \times V$. The elements of E, called *edges*, are ordered pairs of nodes. The first component of an edge $e = (v, w)$ is the *source node of* e, denoted $source(e) = v$, while the second component, here w, is the *target node of* e, denoted $target(e) = w$. We define $nodes(e)$ to represent the set of nodes of an edge e, i. e., $nodes(e) := \{source(e), target(e)\}$.

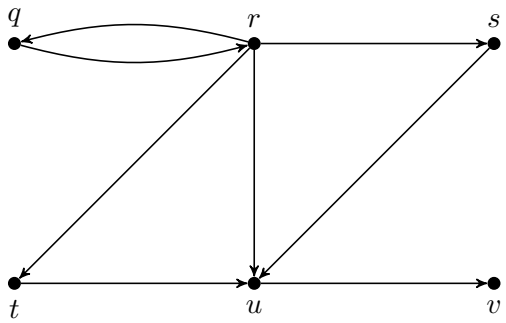


Figure 2.1: A Directed (Unlabeled) Graph

Instead of $(v, w) \in E$ we often use the infix notation $v\ E\ w$. If $v\ E\ w$, w is a *successor of* v *in* G while v is called a *predecessor of* w *in* G. The set of all successor nodes of $v \in V$ (in G) is defined by $vE := \{w \in V \mid (v, w) \in E\}$. Likewise, the set of all predecessor nodes of $v \in V$ (in G) is denoted by $Ev := \{u \in V \mid (u, v) \in E\}$.

Example 2.1 Consider the graph $G_{2.1} = (V, E)$ with

$$\begin{aligned} V &= \{q, r, s, t, u, v\} \text{ and} \\ E &= \{(q, r), (r, q), (r, s), (r, t), (r, u), (s, u), (t, u), (u, v)\}. \end{aligned}$$

A possible graphical notation of $G_{2.1}$ is depicted in Figure 2.1. The nodes of $G_{2.1}$ are drawn as black dots with their associated identities written next to them. In graphical representations, we represent edges as directed arrows between the nodes associated with the edge. The successor nodes of r are q, s, t, u, making up the elements of the set rE. Note that $vE = \emptyset$ since v has no outgoing edges. Likewise, the predecessor nodes of u are collected in $Eu = \{r, s, t\}$. ∎

Let $G = (V, E)$ be a directed graph. A *path in* G is a non-empty sequence of nodes $\pi = v_0 v_1 v_2 \ldots v_k \in V^+$, such that $v_{i-1}\ E\ v_i$ or $v_i\ E\ v_{i-1}$ $(0 < i \leq k)$. Note that a path is undirected, i. e., the direction of the path components does not matter. The first node of path π is denoted by $\mathsf{first}(\pi) = v_0$. The last node of path π is denoted by $\mathsf{last}(\pi) = v_k$. We denote the set of all paths of G by $\mathsf{Paths}(G)$. The *length of path* π, denoted $|\pi|$, is defined as the number of edges it traverses, i. e., for $\pi = v_0 v_1 \ldots v_k$, $|\pi| = k$. A path $\pi = v_0 v_1 \ldots v_k \in V^+$ is called a *directed path in* G iff $v_{i-1}\ E\ v_i$ $(0 < i \leq k)$. The set of all directed paths of G is denoted by $\mathsf{diPaths}(G)$. Let $v, w \in V$ be two nodes of a directed graph $G = (V, E)$. A (directed) path $\pi \in \mathsf{Paths}(G)$ ($\pi \in \mathsf{diPaths}(G)$, resp.) is a *(directed) path between* v *and* w iff $v = \mathsf{first}(\pi)$ and $w = \mathsf{last}(\pi)$. If π is a path between v and w, we say that w *is reachable from* v (via π). Thus, if w is reachable from v, then v is reachable from w. The set of all reachable nodes from v in G is denoted by $\mathcal{R}_G(v)$. The graph G is *connected* iff $\mathcal{R}_G(v) = V$ $(v \in V)$.

Example 2.2 Reconsidering our graph $G_{2.1}$ from Example 2.1, there are several paths to observe, e. g., $\pi_1 = q\,r\,u\,v$ is a directed path from q to v. Furthermore, $\pi_2 = v\,u\,t\,r\,q$ is a path from v to q, this time an undirected one. From any node but r and q, there are only undirected paths to nodes r and q. There is at least one directed path to node v from any other node. ∎

A node $r \in V$ of a directed graph $G = (V, E)$ is called a *root node of* G iff there is a directed path to any other node in the graph, i. e.,

$$\forall v \in V \setminus \{r\} : \exists \pi \in \mathsf{diPaths}(G) : |\pi| > 0 \wedge \mathsf{first}(\pi) = r \wedge \mathsf{last}(\pi) = v.$$

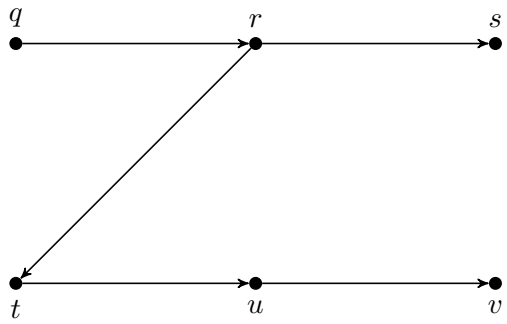


Figure 2.2: A Directed Tree

Thus, every node $v \in V \setminus \{r\}$ is reachable from r, even by a directed path. G is called a *rooted directed graph* iff it has a root node $r \in V$. If a root node r has been chosen, G is sometimes denoted as a triple (V, E, r). While rooted graphs allow for undirected cycles, *trees* do not. A rooted graph is a *tree* iff

$$\forall v \in V \setminus \{r\} : |\{\pi \in \mathsf{diPaths}(G) \mid \mathsf{first}(\pi) = r \wedge \mathsf{last}(\pi) = v\}| = 1,$$

i. e., the directed path from root r to a node $v \in V$ is unique. If G is a tree or has a root node, it is a connected graph.

Example 2.3 Regarding our running example, only nodes q and r qualify as root nodes of $G_{2.1}$. In order to obtain a tree from $G_{2.1}$ we could remove the edges (r, q), (r, u), and (s, u). These removals result in the tree $G_{2.2}$, depicted in Figure 2.2, with root node q. Note that r is no root node anymore because there is no directed path from r to q in $G_{2.2}$.■

Distinct graphs may relate to one another. In the rest of this section, we discuss two types of structural graph comparisons. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two directed graphs. *G is a subgraph of H*, denoted $G \subseteq H$, iff $V_G \subseteq V_H$ and $E_G \subseteq E_H$. The subgraph relationship depends on the identity of nodes, in that every node of G must also be a node of H (edges accordingly). Identity-independent, i. e., purely structural, subgraph relations may be found in the notions of homomorphisms for graphs. Instead of $\subseteq$, functions relate the nodes of G with those of H. A *graph homomorphism between G and H* is an edge-preserving function $\eta : V_G \to V_H$, i. e., if $v \; E_G \; w$, then $\eta(v) \; E_H \; \eta(w)$. For a graph homomorphism η between G and H, $\eta(G)$ defines a subgraph of H by

$$\eta(G) := (\{\eta(v) \mid v \in V_G\}, \{(\eta(v), \eta(w)) \mid (v, w) \in E_G\}), \tag{2.2}$$

the *η-induced subgraph of H*. An injective graph homomorphism is a *subgraph isomorphism*. A bijective graph homomorphism is called a *graph isomorphism*. Graph homomorphisms relate graphs with similar or even identical structures. While plain homomorphisms allow for mapping several nodes of G to one node of H, (subgraph) isomorphisms are injective, i. e., every two distinct nodes of G are mapped to distinct nodes of H. Graph isomorphisms provide the formal device for proving that the same graphical representation identifies two formally different graphs.

Example 2.4 The subgraph relationship is the most discriminating, compared to the other morphism-based graph relations, because nodes and edges must be identical. We exemplify four different graph structures in Figure 2.3, which we subsequently refer to by $G_{(a)}$, $G_{(b)}$, $G_{(c)}$, and $G_{(d)}$. It certainly holds that $G_{(a)} \subseteq G_{2.1}$. However, $G_{(b)}$ is not a subgraph of $G_{2.1}$, although it shows high structural similarity to $G_{(a)}$. Function $\eta_{(a)\mapsto(b)}$ $(y \mapsto r, x \mapsto s, w \mapsto u)$ witnesses this similarity in terms of a graph homomorphism between $G_{(b)}$ and $G_{(a)}$. In fact, $\eta_{(a)\mapsto(b)}$ qualifies as a graph isomorphism between $G_{(b)}$ and $G_{(a)}$.

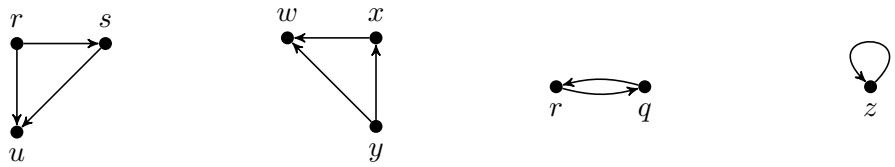


Figure 2.3: Related (Sub-)Graphs

Hence, it is also a subgraph isomorphism between $G_{(b)}$ and $G_{2.1}$. The subgraph $G_{(a)}$ is not the only subgraph of $G_{2.1}$, $G_{(b)}$ is isomorphic to. Function ι $(w \mapsto u, x \mapsto t, y \mapsto r)$ is another subgraph isomorphism between $G_{(b)}$ and $G_{2.1}$.

Consider now the graphs $G_{(c)}$ and $G_{(d)}$, which are neither isomorphic nor subgraph isomorphic to one another. They can, however, be related by graph homomorphism $\eta_{(c)\mapsto(d)}$ $(r \mapsto z, q \mapsto z)$. Thus, graph cycles can be reduced to simple loops via homomorphisms. Furthermore, it can be shown that every directed graph G is homomorphic to graph $G_{(d)}$. The necessary graph homomorphism maps every node of G to z. ■

So far, directed graphs allow us to model relationships of a single type because, beyond node identities, there is nothing that distinguishes an edge from another one. In order to permit the expression of several relationship types, usually labeled graphs are considered as more flexible data structures.

2.1.2 Labeled Graphs

Labels for graphs may be introduced for a plethora of purposes. One of the most important ones is *readability* since abstract nodes, such as the ones we used above $(v, v', w, v_1, v_2, \ldots)$, do not necessarily translate well to real-world objects to be modeled. For instance, a road network is recognized to model a geographical area only if the *points of interest* resemble the ones in the area, at least by their names, but also by the way they are interconnected. Another purpose is to overcome the limitations inherited from set theory partially, that is, the impossibility to include *the same object* twice, i. e., having two distinct nodes or edges modeling the same real-world object or relationship.

No matter what kind of labeling we pursue, a labeling alphabet is required. Let Σ be such an alphabet. Although not limited in what it may contain, Σ is usually assumed to be finite. The least invasive form of labeling a directed graph $G = (V, E)$ is to introduce a *node labeling function* $l : V \to \Sigma$, that assigns a label from Σ to every node in the graph, maintaining G's mathematical structure as introduced in Equation (2.1). Since the alphabet represents an integral part of labeled graphs, it is usually a component of the signature of graphs. A *directed node-labeled graph* is, thus, a quadruple $G = (V, \Sigma, E, l)$ where (V, E) is a directed graph and $l : V \to \Sigma$ a node labeling function over the finite alphabet Σ.

Many of the basic notions introduced in Section 2.1.1 directly apply to node-labeled graphs. The decision whether two node-labeled graphs are considered *equal*, usually depends on the application but is often based on a notion of graph isomorphism. Besides relating two such graphs, $G = (V_G, \Sigma, E_G, l_G)$ and $H = (V_H, \Sigma, E_H, l_H)$, on a mere structural basis, i. e., by isomorphisms between (V_G, E_G) and (V_H, E_H), we may also foster more elaborate notions of equality by integrating the labeling functions. The quasi-standard is to require label equality of isomorphic nodes. However, more general notions are conceptually available, for instance, *alignments over* Σ [110]. A binary relation over Σ, $\simeq \subseteq \Sigma \times \Sigma$, is called an *alignment*, which is a purpose-driven notion saying that some symbol $a \in \Sigma$ may be the same as another symbol $b \in \Sigma$, expressed by $(a, b) \in \simeq$. We write $a \simeq b$ for $(a, b) \in \simeq$.

As a relation, an alignment may be of any form, e. g., an (injective/bijective) function or an equivalence relation. Examples for $\simeq$ are *identity*, i. e., $\simeq = \mathsf{id}_\Sigma := \{(a, a) \mid a \in \Sigma\}$, or, less formally, *synonymity*, i. e., all symbols that may replace one another in any context, based on some linguistic model. Label equality, as sketched above, is the special case of choosing $\simeq = \mathsf{id}_\Sigma$. Incorporating a given alignment $\simeq$, we obtain a generalization of graph isomorphisms ι between node-labeled graphs G and H by

1. ι is a graph isomorphism between (V_G, E_G) and (V_H, E_H) and
2. ι is $\simeq$-preserving, i. e., for all $v \in V_G$, $l_G(v) \simeq l_H(\iota(v))$.

This alignment version of graph isomorphisms does indeed make the notion of graph isomorphisms more liberal as one node label may be aligned with several others. On the other hand, alignments have the power to be more restrictive since nodes, although structurally isomorphic, disqualify to be related as their labels cannot be aligned under $\simeq$.

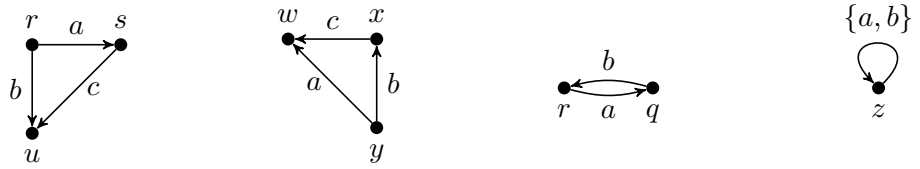
Example 2.5 Suppose we have a labeling alphabet $\Sigma = \{a, b, c\}$ and an alignment $\simeq$ with $a \simeq c$ and $b \simeq c$. Then we can find graph isomorphisms between G and H, respecting $\simeq$, only if G uses the labels a or b, while H may only use label c. As soon as G also uses label c or H uses one of the labels a or b, no isomorphism exists, that preserves $\simeq$. ■

Note that we assumed G and H to be labeled over the same alphabet Σ. This assumption may appear as limiting at first but actually is none. Assume, G is labeled over Σ_G and H over Σ_H. Then indeed, both graphs are labeled over $\Sigma_G \cup \Sigma_H$ without contradicting any of the previous definitions. Hence, graph alphabets can always be made the same without causing harm in the course of comparing two graphs that use them. Having both graphs labeled over the same alphabet does not necessarily mean that they are using all available labels. The alphabet is an upper bound for which we have to check label equality (or alignment).

In principle, the same procedure as for node labels may be followed when assigning labels to edges employing an edge labeling function $l : E \to \Sigma$. Thereby, we achieve that two distinct edges may represent different relationships, e. g., one may express *friendship*, and the other might mean *customer relationship*. Both of these types can be expressed in a single edge-labeled graph model. However, a concrete relationship (v, w) can only be assigned a single relationship type (also called *predicate*), although more than one relationship type associated with v and w could be desired. Therefore, the edge labeling is usually integrated into the edge structure of a directed graph, in that an edge e is considered to be a triple (v, a, w) of a source node v ($= \mathit{source}(e)$), a label a ($= \mathit{label}(e)$), and a target node w ($= \mathit{target}(e)$). The labeling function is left implicit, but the number of different relationship types between any two nodes is increased to the number of different labels in Σ. Since edge-labeled graphs are the core data structure we use throughout the rest of the thesis, we call them simply *labeled graphs*.

Definition 2.6 (Labeled Graph)
A *labeled graph* G is a triple (V, Σ, E), where V is a finite set of nodes, Σ a finite (label) alphabet, and $E \subseteq V \times \Sigma \times V$. ▲

All the notations introduced for directed graphs in Section 2.1.1 carry over to labeled graphs, naturally. Additionally, the labeling of edges allows for more fine-grained considerations w. r. t. neighborhood. Let $G = (V, \Sigma, E)$ be a labeled graph and $e = (v, a, w) \in E$. As an infix notation we use $v\ E^a\ w$, where the edge relation E is superscripted with the label $a \in \Sigma$, formally justified by $E^a := \{(v, w) \mid (v, a, w) \in E\}$. w is not only some successor of v (v a predecessor of w, resp.), but, more specifically, w is an *a-successor of v* (v is an *a-predecessor of w*, resp.) in G. Utilizing this notation, the sets of all a-successors and a-predecessors ($a \in \Sigma$) of $v \in V$ are naturally expressed by vE^a and E^av.



(a) (b) (c) (d)

Figure 2.4: Sample Graphs for Labeled Morphisms

Assessing equality of labeled graphs $G = (V_G, \Sigma, E_G)$ and $H = (V_H, \Sigma, E_H)$ is again based on graph isomorphisms. Additional incorporation of alignments is imaginable. However, the most prominent cases discussed and used in the literature are bijective alignments. Hence, the alignment itself does not play any rôle in the course of deciding label equality. The edge labels can easily be adapted so that G and H use precisely the same labels.

Example 2.7 We align the labeled graphs G and H, where G is labeled over Σ_G and H over Σ_H. Let $\simeq$ be a bijective function from Σ_G to Σ_H, i. e., for any pair of labels $a, b \in \Sigma_G$, if $a \simeq c$ and $b \simeq c$, then $a = b$, and for all $c \in \Sigma_H$ an $a \in \Sigma_G$ exists with $a \simeq c$. Define $H_{/\simeq}$ to have the same set of nodes as H, but for every edge (v, c, w) of H include the edge (v, a, w) instead, where $a \simeq c$. As a consequence, $H_{/\simeq}$ is labeled over Σ_G. ■

Thus, if not stated otherwise, we assume all graphs to be labeled over the same fixed alphabet Σ. Since the notions of graph homomorphisms and isomorphisms are needed often throughout the thesis, we define their labeled versions formally.

Definition 2.8 (Graph Morphisms)
Let $G = (V_G, \Sigma, E_G)$ and $H = (V_H, \Sigma, E_H)$ be labeled graphs. A function $\eta : V_G \to V_H$ is called a *graph homomorphism between G and H* iff $v\ E_G^a\ w$ implies $\eta(v)\ E_H^a\ \eta(w)$ $(a \in \Sigma)$. An injective graph homomorphism is called a *subgraph isomorphism*.

A *graph isomorphism between G and H* is a bijective function $\iota : V_G \to V_H$, such that $v\ E_G^a\ w$ iff $\iota(v)\ E_H^a\ \iota(w)$. ▲

Example 2.9 Let us first reconsider the unlabeled graphs of Example 2.4. They are essentially labeled graphs using a single letter from the alphabet, say $\tau \in \Sigma$, as each edge's label. Thus, all the homomorphisms exemplified there are valid homomorphisms for the labeled versions.

In contrast, if graph $G_{(a)}$ and $G_{(b)}$ involve a different labeling function, they may not be associated by any homomorphism, as shown by the graphs depicted in Figure 2.4 (a) and (b). The only candidate homomorphism is $\eta_{(a)\mapsto(b)}$ $(r \mapsto y, s \mapsto x, u \mapsto w)$ from Example 2.4 since it is only this morphism that respects the graph structure (independently of the labeling). But while u is the b-successor of r in $G_{(a)}$, the b-successor of $\eta_{(a)\mapsto(b)}(r) = y$ is x and $x \neq w = \eta_{(a)\mapsto(b)}(u)$. In fact, there is no graph homomorphism between the labeled graphs $G_{(a)}$ and $G_{(b)}$.

Regarding the graphs $G_{(c)}$ and $G_{(d)}$ in Figure 2.4 (c) and (d), there is a homomorphism between them, namely $\eta_{(c)\mapsto(d)}$ $(r \mapsto z, q \mapsto z)$. The semantics of the label $\{a, b\}$ associated with the edge from z to z in $G_{(d)}$ is that there is an a-labeled and a b-labeled edge. ■

Let $G = (V, \Sigma, E)$ be a labeled graph, $\Gamma \subseteq \Sigma$, and $v, w \in V$. If there is an edge $v\ E^a\ w$ for every $a \in \Gamma$, we usually summarize all these edges to a single edge labeled by Γ in drawings of G.

2.2 Graph Data

In order to make use of labeled graphs $G = (V, \Sigma, E)$ as a model for data, we have to be clear about what the nodes and edges shall mean. Thus, we specify what the objects that account for G represent.

Usually, graph databases capture entity-centric information, which are entities represented as nodes, their properties/attributes as edges to actual data values, and their relations to other database objects also represented as edges. Remaining in the realm of labeled graphs, we have at least two types of nodes, one representing entities and one for data items such as *string* or *number* objects. One of the most general and stable data models stems from the impressive standardization efforts of the *Semantic Web community* and the *World Wide Web Consortium* (W3C), trying to build "an infrastructure of machine-readable semantics for the data on the Web" [13].

2.2.1 The Resource Description Framework

The *Resource Description Framework*, RDF for short, provides a simple and extensible data model that comes with a formal semantics. It has been a W3C recommendation since 1999 and, from there on, sparked much attention from researchers and practitioners. The current recommendation provides RDF 1.1 [38, 67]. As the name suggests, RDF allows for expressing information about *resources*. A resource can be anything, from Web documents up to physical objects or actual people [122].

Modeling information in RDF means to formulate statements about resources, following the simple structure of

$$\textit{subject} \text{ — } \textit{predicate} \text{ — } \textit{object}.$$

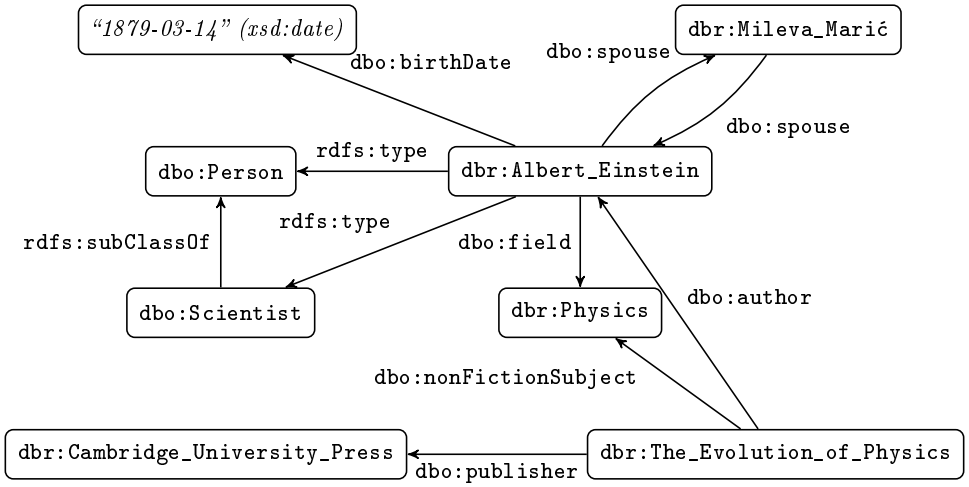
Subject and *object* are resources related by the *predicate*. Because RDF statements consist of three components, they are commonly referred to as *RDF triples*. A set of RDF triples makes up an *RDF graph*. Three different types of data may occur in RDF triples, namely IRIs, literals, and blank nodes [122].

Every resource is uniquely identified and implemented by *International Resource Identifiers* [44] (IRIs), a generalization of *Uniform Resource Identifiers* (URIs). IRIs may occur in the subject, object, as well as predicate position of an RDF triple. Technically speaking, predicates are resources, which makes sense as soon as we think of statements about relationship types. For example, we may want to express that *is child of* is the inverse relation of *is parent of*. IRIs are thought of as global identifiers, i. e., if two different people talk about the same IRI, they refer to the same object. URLs are an essential subset of IRIs, referencing Web locations.

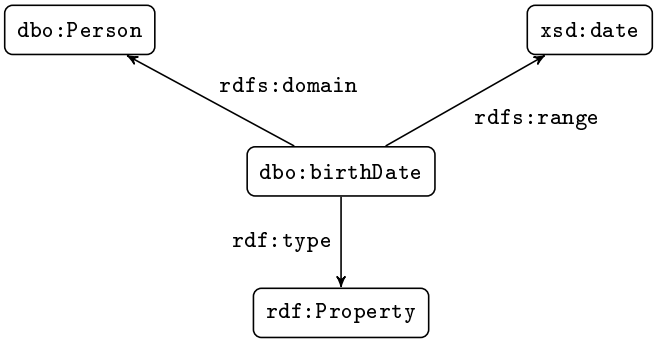
Literals are data values, not represented as IRIs. They come with a data type, such as `string`, `int`, or `date` (cf. [38] for a list of valid data types). Such data values are used to define attribute values of a resource, such as a *date of birth* or a person's *address*, or *title*, *author*, or *publication year* of a book. Therefore, literals solely occur in object position.

Finally, RDF provides us with the possibility of expressing anonymous resources, called *blank nodes*. According to [38], blank nodes have a local scope, i. e., they are not to be referenced outside an RDF graph. They can be used in subject and object positions and refer to some unnamed data objects.

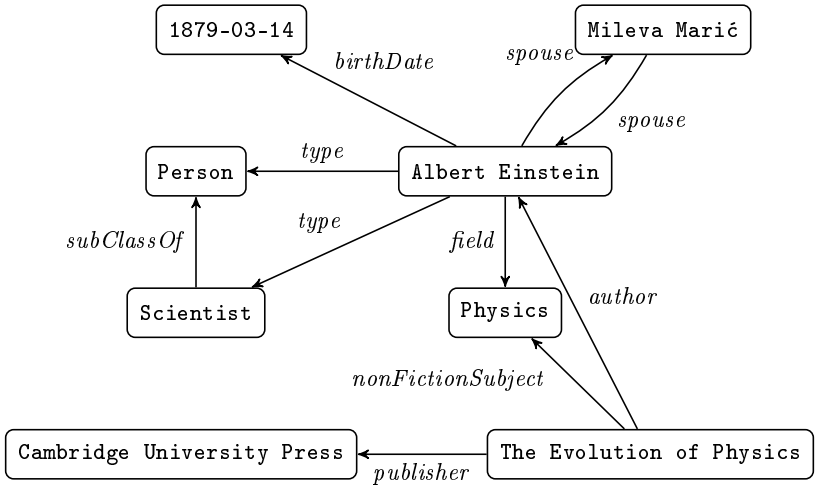
Let I, L, B be disjoint universes of IRIs, literals, and blank nodes. An *RDF triple* is a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$. A set of RDF triples G is an *RDF graph*. Throughout this thesis, we are considering so-called *ground RDF graphs* [62], which are subsets of $I \times I \times (I \cup L)$, i. e., the are free of blank nodes.



(a)



(b)



(c)

Figure 2.5: (a) Graph Representation of an Example RDF Graph from DBpedia [17] (b) An RDF Graph Describing the Predicate `dbo:birthDate` (c) A Graph Database Representation of Figure 2.5 (a)

Example 2.10 If an RDF graph $\mathsf{G} \subseteq (I \cup B) \times I \times (I \cup B \cup L)$ does not contain statements about predicates, it may be represented as a labeled graph $G(\mathsf{G})$, as defined in Definition 2.6. All subjects and objects, occurring in G, amount to the set of nodes of $G(\mathsf{G})$. All predicates form the labeling alphabet. The set of edges is the RDF graph G itself, i. e.,

$$G(\mathsf{G}) = (\{s, o \mid (s, p, o) \in \mathsf{G}\}, \{p \mid (s, p, o) \in \mathsf{G}\}, \mathsf{G}).$$

Thus, many RDF graphs can be graphically represented as labeled graphs. An example, manually extracted from DBpedia [17], is shown in Figure 2.5 (a). As the nodes' identities are essential for RDF, they fill in as node labels in the center of the respective nodes. From now on, we solely rely on this kind of graphical notation of data modeled by graphs.

This RDF graph contains information about `dbr:Albert_Einstein`, the resource to access information about the person *Albert Einstein*. DBpedia introduces prefixes to shorten IRIs, for representational purposes as well as to reduce the size of RDF dataset dump files. For instance, the prefix `dbr:` unwinds to the URL `http://dbpedia.org/resource/`. Hence, `dbr:Physics` actually represents `http://dbpedia.org/resource/Physics`, the URL linking to a DBpedia page with information about the scientific field of *physics*. We have one literal, being the date of birth of *Albert Einstein*. The string in brackets specifies the type of the literal, here `xsd:date`, an *XML Schema Definition* for data formats.

Also included in this excerpt of DBpedia is some schema information prefixed by `rdfs:`. They state that *Albert Einstein*, represented by the resource `Albert_Einstein`, is of the types *person* and *scientist*, represented by the *DBpedia ontology* classes `Person` and `Scientist`. Every object of type *scientist* is also a *person*, stated by the triple (`dbo:Scientist`, `rdfs:subClassOf`, `dbo:Person`).

As suggested by the font used for the predicates, also the edge labels are resources and may be, as such, part of RDF statements. For instance, predicate `dbo:birthDate` is itself described by an RDF graph, from which we draw an excerpt in Figure 2.5 (b). It specifies the domain and range of the predicate, which can be used as a constraint when inserting a concrete RDF triple with this predicate. In this example, only *persons* may have associated birth dates, which must be of type `xsd:date`. The graph in Figure 2.5 (a) conforms to these constraints. However, integrating both graphs into a single graphical representation leaves the realm of standard graphs [66] as not all information about `dbo:birthDate` is collected in a single place, that is the node labeled `dbo:birthDate`. ■

As already mentioned, and enforced by the W3C, an IRI can be anything, making RDF highly extensible towards so-called vocabularies that capture the semantics of resources and statements [122]. RDF supports the definition of such vocabularies by incorporating *RDF Schema* (RDFS), which deals with typing of entities, building hierarchies of classes, and putting restrictions on domains/ranges of predicates. To cope with these and other extensions, RDF comes with a model-theoretic semantics [67] that formally grasps all such features. However, our view on RDF shall be restricted to a basic representational level because our focus will be on querying explicit extensions of graph databases. We formally substantiate this representational level by the notion of *graph databases*, grounded in the principles of RDF graphs. We provide further information about the capabilities of RDF to express data schemas in Section 2.3.5.

2.2.2 Graph Databases

From an RDF perspective, we use the grounded model of graph data and ignore the entailment capabilities of RDFS vocabularies. We do acknowledge there are universes of objects $\mathcal{U}$, to be used as graph nodes, and predicates $\mathcal{P}$, used as edge labels. For ease of notation, $\mathcal{U}$ captures everything that can be in subject or object position, including predicates and literals. Note that this automatically implies non-disjoint universes $\mathcal{U}$ and

$\mathcal{P}$. Therefore, we work with a non-standard graph model $G = (V, \Sigma, E)$ with a set of objects V and a set of predicates Σ, but $V \cap \Sigma = \emptyset$ does not generally hold. Although a node's neighborhood does not exhaustively describe a single node [66], the following contents will not suffer from this inconvenience. Beyond Example 2.10, there will be no example that uses RDF (sub-)graphs dealing with predicates as resources explicitly.

As we are concerned with graph databases extensionally, there is also no need to include blank nodes. Even if we used RDFS vocabulary and blank nodes, Gutierrez et al. have shown that the maximal extension, called *closure* that can be derived from all the implicit information present in an RDF graph is unique [62]. Hence, we would always work with the closure of an RDF graph (cf. Theorem 3.6 [62]).

Definition 2.11 (Graph Database)
A *graph database* is a labeled graph $DB = (O_{DB}, \Sigma, E_{DB})$ where $O_{DB} \subsetneq \mathcal{U}$ and $\Sigma \subsetneq \mathcal{P}$. ▲

In divergence of alternative definitions, e. g., the one given by Hayes and Gutierrez [66], we omit auxiliary labeling functions of nodes and edges but assume database objects (O_{DB}) and predicates (Σ) to be identical with their respective labels.

Example 2.12 The graphs depicted in Figures 2.5 (a) and 2.5 (b) already are visualizations of graph databases. We will, however, make the notation easier. Every object will be represented as a box labeled by its identifier, written in `typewriter font`. We do not insist on using IRIs and make no distinction between resources and literals. Predicates will have an *italicized font*. Thus, a simplified graph database representation of our RDF graph sample on *Albert Einstein* (cf. Figure 2.5 (a)) is the one depicted in Figure 2.5 (c).■

Note that graph morphisms (cf. Definition 2.8 on Page 11) serve a purely structural comparison purpose, later excessively used for different querying tasks. Mapping different database objects to one another may account for structural similarity, but an object's identity carries information that gets lost by graph homomorphisms. Having reduced our graph database model by blank nodes and RDFS vocabulary, the decision of equality of two graph databases DB_1 and DB_2 boils down to actual equality of the database's objects and edges, i. e., $DB_1 \subseteq DB_2$ and $DB_2 \subseteq DB_1$.

2.3 Graph Schema

One of the key features of graph data is that a prior schema, describing all the possible entity and relationship types, is not needed. A restrictive schema that forces a graph database management system to disallow untypable data is even rated as undesirable [28, 9, 62, 91]. Nevertheless, semantic or structural information about the stored data may help in

(1) reducing and uncovering uncontrolled heterogeneity, e. g., heterogeneous representation of one entity type and absent information (incompleteness),

(2) managing query formulation – a schema describes what kinds of nodes exist and how they are related to one another in the graph – and

(3) integrating data from diverse data sources into one intermediate representation that may finally be transferred back to structured data.

Such a schema for graph data shall rather be descriptive than restrictive [28, 2]. To this end, we recover *graph schemas*, first introduced by Buneman et al. [29], which we analyze and extend w. r. t. our graph database model. This section appears, in parts, in [89]. Including this material allows us to formally introduce *graph simulations*, one of the non-standard

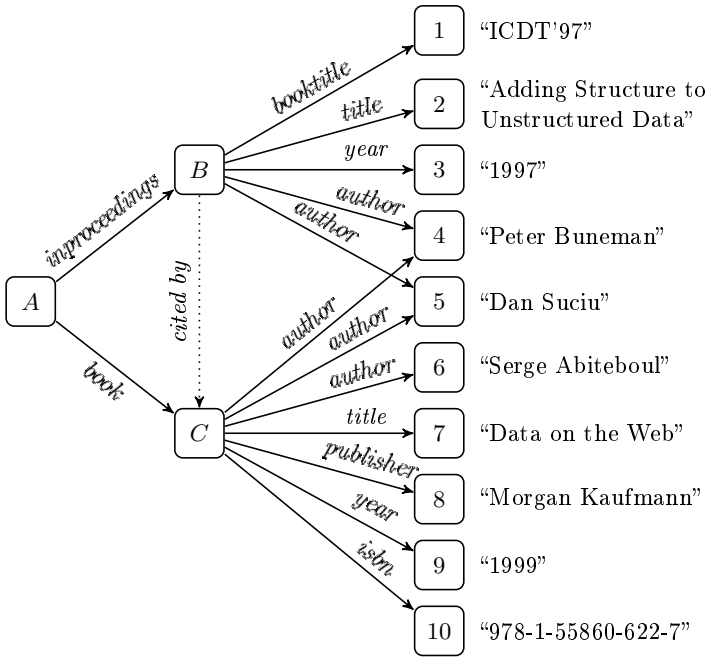


Figure 2.6: An Example Rooted Graph Database

pattern matching notions, that is going to be reoccurring in the remaining chapters of this thesis. General results about simulations are presented and proven.

Throughout the rest of this section, we develop a model of schemas for graph databases, a notion of schema instantiation, and the precise semantics of the model.

2.3.1 From Semistructured to Graph Data

Graph schemas have been developed in light of the upcoming tree-like graph data structures of the 1990's, called *semistructured data* [28, 2, 9]. Hence, there are design decisions that do not withstand more general graph databases. A graph schema forms an upper bound of its database instances [29]. This upper bound is achieved by characterizing the *instance of*-relation between graph schemas and graph databases by a proxy notion relating objects of the database and the types specified by the schema so that whenever a database object participates in a predicate, the corresponding schema type partakes in the same predicate. Buneman et al. [29] presumed the semistructured data model to be represented by rooted labeled graphs $DB = (O_{DB}, \Sigma, E_{DB}, r_{DB})$ (cf. Section 2.1.2).

Example 2.13 Figure 2.6 shows an excerpt of this chapter's bibliography file, worked up as a rooted labeled graph. First of all, it is incomplete. For instance, the paper with title *Adding Structure to Unstructured Data* has two more authors, namely *Susan Davidson* and *Mary Fernandez*. Second, actual data is only stored in the leaf nodes, depicted as labels next to the nodes, a representation of rooted graph data we adopted from Abiteboul et al. [3]. The node labeled *"1999"* represents a data node with the unique identifier 9 holding the data item *1999*. The string *1999* may be interpreted as a number or, as in this example, as a year. Alternatively, such data may be represented as labeled edges from the leaf nodes without an explicit target node [28]. The two representations are entirely equivalent.

Independent of data representation, internal nodes serve a mere structuring purpose. The graph is not a tree since, e. g., authors 4 (*Peter Buneman*) and 5 (*Dan Suciu*) share both predecessor nodes, B and C.

The root node represents the bibliography as a document, an entry point for every analysis and querying task. The second layer nodes are the bibliographic entries within

the document. The edge labels used between root and entries constitute each entry's type. All other nodes refer to attributes of at least one entry. Edges contradicting the pure tree structure may occur at any level, as exemplified by the *cited by*-edge between the two entries. This edge is dotted, as it does not qualify as classical bibliographic content of a BibTeX document, but will be used in subsequent examples. ■

It was Buneman et al.'s goal to have a schema model that resembles their graph data. Hence, a *graph schema* was defined as a rooted labeled graph $S = (V_S, \Pi, E_S, r_S)$, too, where Π is a particular alphabet for schemas. For Buneman et al., Π contains first-order sentences that, when used as edge labels, not only describe a single edge, but quite possibly infinitely many [29]. The only restriction is that each such label must stem from a decidable first-order theory, i. e., for a given predicate $p(\mathsf{x})$ (with free variable x) there is a decision procedure checking whether $p(a)$ holds for an input label $a \in \Sigma$.

Example 2.14 An example of such a predicate p is $p(\mathsf{x}) :\Leftrightarrow \mathsf{x} = a$. This predicate is only evaluated to true if the symbol a substitutes x. Thus, any p-labeled edge may be replaced by an a-labeled edge.

A more sophisticated example is $q(\mathsf{x}) :\Leftrightarrow \mathsf{x} \equiv 0 \bmod 2$, i. e., every symbol that qualifies as an even number may substitute x. The set of all possible substitutions is, in this case, infinite. Thus, the structure that Buneman et al. describe is a finite representation of an infinite graph object. Other examples that easily lead to infinite structures are those using first-order sentences over *regular expressions*. ■

Leveraging such decision procedures [77] is nowadays the realm of *Satisfiability Modulo Theories solvers* (SMT solvers), such as Z3 [40] or CVC4 [20].

First-order sentences can be easily reintegrated into the model we develop throughout this section, but they do not provide any fundamental insights towards the semantics of graph data or schemas. Therefore, we keep the alphabets for graph schemas simple, in that they acquire symbols from the same universe as the one graph databases draw their labels from (i. e., $\mathcal{P}$). As before, the graph databases' alphabets and the ones for graph schemas are aligned into a single alphabet (cf. Example 2.7).

One characteristic we would like to maintain from Buneman et al.'s rooted graph schemas is the essence of the root node. Recall that from the root node, every other node in the graph schema is reachable by a directed path. Hence, the resulting graph structure is, at least, a connected graph. We do not explicitly introduce root nodes to graph schemas (see Section 2.3.4 for a discussion of the consequences), but maintain the connectedness property.

Definition 2.15 (Graph Schema)
A *graph schema* S is a connected labeled graph (T_S, Σ, E_S) with a non-empty node set T_S, called the *types of* S. ▲

When describing all the structures of a graph database, possibly more than one graph schema is needed. A rooted graph schema is depicted in Figure 2.7. Examples for graph schemas without explicit root nodes are given in Figures 2.11 (b) and 2.11 (c) (Page 24). Note that we primarily follow the graphical conventions we established for graph databases. Since types are conceptually different from database objects, we represent their identifiers by Small Capitals. By design, a graph schema $S = (T_S, \Sigma, E_S)$ describes the permitted structure [29]. For instance, the schema in Figure 2.7 allows for capturing bibliographic contents, such as *inproceedings* and *books*. Conversely, if an edge is not present in a graph schema, it must not be used in any database instance, e. g., Figure 2.7 does not feature a *cited by*-relationship concerning Book and Inproceedings. Thus, the graph database in Figure 2.6 shall not be an instance of Figure 2.7 if the dotted edge is included.

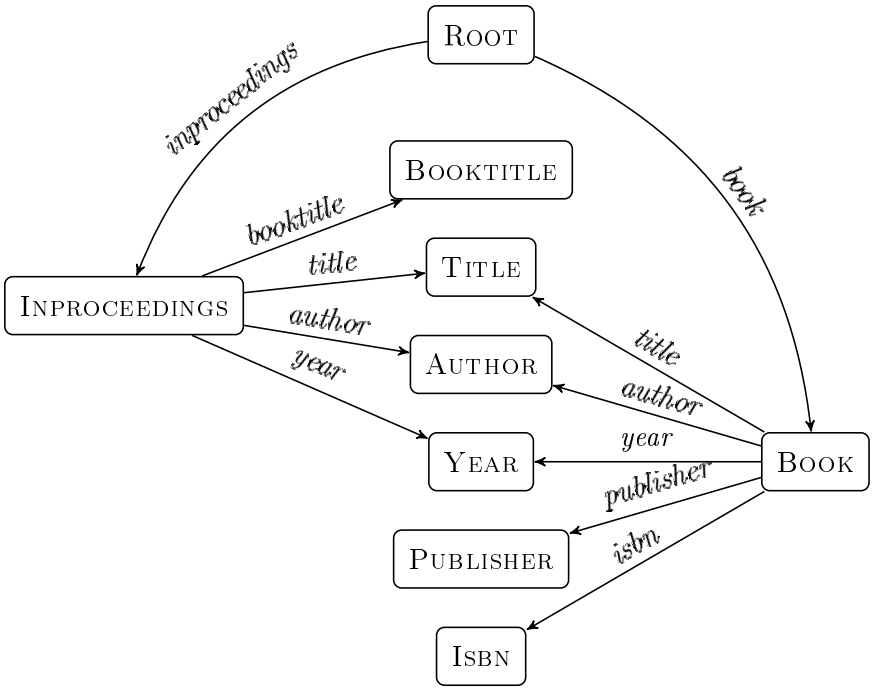


Figure 2.7: A Rooted Graph Schema for Bibliographic Data

Towards a formal treatment of *instances* of graph schemas, we need to answer the following questions [3]: Let $S = (T_S, \Sigma, E_S)$ be a graph schema.

(1) What makes a graph database DB an instance of S?

(2) Once we have an instance of S, what is the induced classification/typing of database objects?

Towards (1), it is the notion of *graph simulation* that captures the upper bound nature of schemas. Buneman et al. define that a graph database DB *conforms to graph schema* S, denoted $DB \preccurlyeq S$, iff there is a *rooted graph simulation between* DB *and* S [29]. Before we dive into this particular characterization of instances in the rooted graph data model, let us first clarify the notion of *simulation* for labeled graphs.

Back in 1971, it was Robin Milner who thought about an algebraic characterization of when one program simulates another [96]. He used so-called *program graphs* to abstract from hardware and other implementation details, only representing the states (e. g., program counters or valuations of program variables) as nodes of the graph. One step (an edge from one node to another) transforms the current state of the program. Program P_2 simulates P_1 if whatever computational step P_1 performs, P_2 is capable of doing the same. Intuitively speaking, P_2 mimics the computational behavior of P_1. If both programs start at the same configuration (i. e., initialization of variables), then for any configuration P_1 reaches, P_2 can reach the same configuration by simulating the steps of P_1. Two programs shall not be distinguished if they could simulate one another.

Definition 2.16 (Graph Simulation)
Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2)$ be two graphs. A binary relation $R \subseteq V_1 \times V_2$ is called a *graph simulation between* G_1 *and* G_2 iff for every pair $(v_1, v_2) \in R$, $v_1\ E_1^a\ w_1$ implies that $w_2 \in V_2$ exists with $v_2\ E_2^a\ w_2$ and $(w_1, w_2) \in R$.

G_2 *simulates* G_1, denoted $G_1 \sqsubseteq_{\mathsf{sim}} G_2$, if there is a non-empty graph simulation between G_1 and G_2. ▲

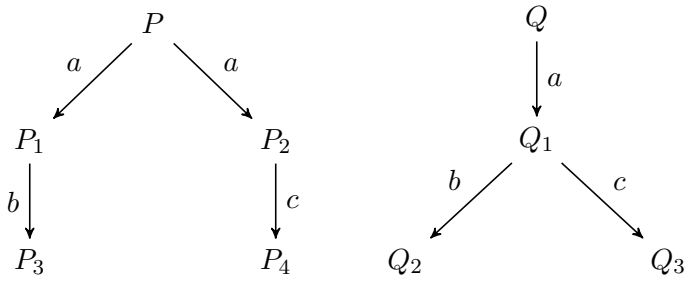


Figure 2.8: Two Program Graphs P and Q

Example 2.17 Before heading back to the realm of graph data, let us discuss simulation with an informal picture of program graphs: The programs P and Q, whose program graphs are depicted in Figure 2.8, execute abstract actions a, b, and c. They both terminate after they have either executed the sequence ab or the sequence ac. If we only compare programs P and Q up to these computational sequences (also known as *traces*), they are identical. Simulation, however, distinguishes between P and Q. While program Q simulates P, e. g., by the non-empty graph simulation

$$R = \{(P, Q), (P_1, Q_1), (P_2, Q_1), (P_3, Q_2), (P_4, Q_3)\},$$

program P cannot simulate Q. If Q performs the a-move to Q_1, P must decide which of its successor states to take. Q_1 outperforms P_1 by a c-move and P_2 by a b-move. ■

Returning to the notion of graph schema conformance, as of Buneman et al. [29], a *rooted graph simulation* between two rooted labeled graphs is a graph simulation that relates the roots of the graphs. In this way, a graph schema S indeed represents an upper bound to every graph database instance conforming to it.

Example 2.18 A rooted simulation between the graph database in Figure 2.6 (without the dotted edge) and the graph schema in Figure 2.7 is

$$\widehat{R} = \left\{ \begin{array}{l} (A, \textsc{Root}), (B, \textsc{Inproceedings}), (C, \textsc{Book}), (1, \textsc{Booktitle}), \\ (2, \textsc{Title}), (3, \textsc{Year}), (4, \textsc{Author}), (5, \textsc{Author}), (6, \textsc{Author}), \\ (7, \textsc{Title}), (8, \textsc{Publisher}), (9, \textsc{Year}), (10, \textsc{Isbn}) \end{array} \right\}.$$

$\widehat{R}$, in fact, represents the smallest non-empty but rooted graph simulation between the two graphs. ■

Crucial to this notion of conformance is the treatment of root nodes. Without the root node condition, graph schema conformance is rendered trivial.

Proposition 2.19 *There is a simulation R between any two graphs G_1 and G_2, but $G_1 \sqsubseteq_{\mathsf{sim}} G_2$ does not hold, in general.*

Proof: We choose the *empty simulation*, that is $R_\emptyset = \emptyset$, fulfilling the simulation property for all pairs of graphs G_1 and G_2.

Towards the second goal, we give $G_1 = (\{v_1\}, \{a\}, \{(v_1, a, v_1)\})$, depicted in Figure 2.9 (a), and $G_2 = (\{w_1, w_2\}, \{a\}, \{(w_1, a, w_2)\})$ in Figure 2.9 (b). Of course, $R_\emptyset$ is a graph simulation between G_1 and G_2, but there is no other one. Every other simulation (candidate) R needed to relate v_1 of G_1 with either w_1 or w_2 of G_2. $(v_1, w_2) \notin R$ because there is an edge (v_1, a, v_1) in G_1 but w_2 has no outgoing edge. Consequently, $(v_1, w_1) \notin R$ because, although the a-labeled step in G_1 can be simulated by w_1, it reaches node w_2, which is impossible to be related to v_1 by R (see argument above). Hence, there is no non-empty simulation between G_1 and G_2, such that $G_1 \sqsubseteq_{\mathsf{sim}} G_2$. Q. E. D.

Figure 2.9: Two Graphs (a) G_1 and (b) G_2 with $G_1 \not\sqsubseteq_{\mathsf{sim}} G_2$

The *empty simulation* $R_\emptyset$ would be a viable witness for conformance between any pair of graphs. As a consequence, if we applied the concepts of Buneman et al. to non-rooted graph databases and schemas, we would not obtain any meaningful conformance relation because every graph database would be an instance of every graph schema. There are several ways out of this issue, from which we briefly address the following three: (a) artificial root nodes on the database-side, (b) artificial root nodes on the schema-side, and (c) require non-emptiness of simulations R witnessing conformance.

Introducing artificial roots to graph databases is infeasible as, for instance, *update anomalies* are foreseeable: Whenever a graph database is subject to change, its root node's incidence must often be adjusted to meet the root node property still. Furthermore, choosing a particular root node would have a significant influence on conformance. In that case, conformance needed to be adjusted to instead ask for the existence of a root node positioning, such that conformance holds. This way, conformance boils down to finding minimal sets covering every other node in the graph database w. r. t. reachability, very likely to be an intractable problem.

Second, we might reintroduce the root node concept to the notion of graph schema. When designing a graph schema, one would be able to specify which of the types are necessary to be covered by a graph database instance. These *types of interest* would undoubtedly follow a *universe of discourse*-kind of argument. The more such types are specified, the more restrictive the graph schema becomes. Unfortunately, the implied instance notion misses out on an important property: There may be two graph schemas describing the same set of instances, which cannot be proven to be equivalent inside the model we develop. We provide an extended discussion about this issue in Section 2.3.4.

The last of the three possibilities encounters (only but effectively) the issue's symptoms by reestablishing what rooted graph simulations have been for Buneman et al.'s graph schemas, namely non-empty. Having the root nodes related guarantees non-empty simulations, and, since the database's root node initiates a path to every other node in the database, it is guaranteed that every database object is covered by at least one type. If we require non-empty graph simulations instead of any graph simulation, we will obtain a first non-trivial notion of graph schema instances. However, instances of graph schemas, solely relying on graph simulation, will face two peculiarities, (a) the problem of partial simulations and (b) the problem of *leaf node insensitiveness*.

Example 2.20 Let us review some non-empty graph simulations between the rooted graph database and schema in Figures 2.6 and 2.7. The graph database is denoted by $DB_{2.6}$, the graph schema by $S_{2.7}$. For the sake of this example, we read $DB_{2.6}$ as if the dotted edge is not present.

In $\widehat{R}$ from Example 2.18, all the nodes received their appropriate type, even the document, i. e., node A in $DB_{2.6}$, is associated with the root of $S_{2.7}$. What if we do not consider the root as part of the simulation? What if we only capture the book or only the conference paper? In both cases, the resulting relations between $DB_{2.6}$ and $S_{2.7}$ are, in fact, non-empty simulations. Take, for instance, the missing root node case:

$$R_1 = \left\{ \begin{array}{l} (B, \textsc{Inproceedings}), (C, \textsc{Book}), (1, \textsc{Booktitle}), (2, \textsc{Title}), \\ (3, \textsc{Year}), (4, \textsc{Author}), (5, \textsc{Author}), (6, \textsc{Author}), (7, \textsc{Title}), \\ (8, \textsc{Publisher}), (9, \textsc{Year}), (10, \textsc{Isbn}) \end{array} \right\}$$

Every pair in R_1 fulfills the graph simulation property. As root node A in $DB_{2.6}$ is not reachable by a directed path from any of the nodes covered by R_1, it is not required to include the pair (A, ROOT) in order for the relation to be a graph simulation. Of course, $\widehat{R} \supseteq R_1$ and, therefore, $\widehat{R}$ provides us with more information about the instance $DB_{2.6}$ w. r. t. $S_{2.7}$. However, from a graph simulation perspective, R_1 is as good as $\widehat{R}$ to confirm that $DB_{2.6}$ is an instance of $S_{2.7}$.

It is impossible, though, to reduce R_1 above to only capture B without covering all of its authors. Suppose we have a candidate graph simulation R_4 that includes $(B, \text{INPROCEEDINGS})$ but excludes $(4, \text{AUTHOR})$. Then R_4 cannot be a graph simulation between $DB_{2.6}$ and $S_{2.7}$ as there is an edge $B\ E^{author}_{DB_{2.6}}\ 4$, but the only successor of INPROCEEDINGS is node AUTHOR and $(4, \text{AUTHOR}) \notin R_4$. ■

This example shows that there are different non-empty graph simulations with different capabilities towards typing. Another important observation is that graph simulations are highly dependent on the reachability of nodes by directed paths. Pushing Example 2.20 towards the extreme case yields a graph simulation that only captures leaf nodes of the database.

Example 2.21 Again, we are concerned with $DB_{2.6}$ and $S_{2.7}$ from Figures 2.6 and 2.7. Node 10 is a leaf node of $DB_{2.6}$, which shall represent the *ISBN* of a book object in our bibliographic database. As there is no edge in $DB_{2.6}$, having node 10 as a source node, $R_2 = \{(10, \text{ISBN})\}$ is also a valid non-empty graph simulation between $DB_{2.6}$ and $S_{2.7}$. Compared to the simulations from Example 2.20, we now have $R_2 \subseteq R_1 \subseteq \widehat{R}$. For the same reason, also

$$R_0 = \{(10, \text{BOOK}), (10, \text{TITLE}), (10, \text{AUTHOR}), (10, \text{YEAR}), (10, \text{PUBLISHER})\}$$

is a graph simulation. Even node ROOT in $S_{2.7}$ is capable of simulating node 10. We could also unify each of the simulations mentioned above with R_0 without harming the graph simulation property because only outgoing edges of data nodes are considered when checking for graph simulation. As a consequence, node 10 can be of any type captured in $S_{2.7}$, while it is an ISBN. ■

The just described phenomenon of graph simulation is well-known in process theories. In process-theoretical terms, simulations are not *deadlock-sensitive* [95]. Node 10 is a classical deadlock for its incapability of performing any *action*, which would be displayed by an outgoing edge. Rephrased in our graph data scenario, we find that simulations are *leaf node insensitive*, which may lead to terrible classifications, as in Example 2.21. The following proposition provides a formal argument of this insensitiveness towards leaf nodes (or deadlocks, resp.).

Proposition 2.22 *Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2)$, such that there is a node $d \in V_1$ with $\bigcup_{a \in \Sigma} dE_1^a = \emptyset$. Then $R = \{(d, v) \mid v \in V_2\}$ is a non-empty simulation between G_1 and G_2 if $V_2 \neq \emptyset$.*

PROOF: There is at least one $(d, v) \in R$ because $V_2 \neq \emptyset$. Hence, $R \neq \emptyset$. Let $(d, v) \in R$. Since $dE_1^a = \emptyset$ for all $a \in \Sigma$, v canonically simulates d without further consideration. Hence, R is indeed a non-empty simulation. Q. E. D.

Even in general graph databases, simulations quickly trivialize the instance notion. For example, in light of literals in RDF graphs, there is a canonical simulation between any RDF graph and every graph schema. Of course, node 10 in $DB_{2.6}$ is not a book, or a title, or anything else but an ISBN. What makes it an ISBN, at least in this example, is that it is the target node of an edge labeled by *isbn*. Simulations neglect incoming

edges completely[1], leading to simulations like R_0 in Example 2.21. Capturing all kinds of relationships a database object participates in, including the ones expressed by backward edges, is crucial when it comes to graph schemas and the conformance of graph databases. The extension of simulations incorporating backward edges, in a graph database setting, was first sketched by Abiteboul et al. [3] and later coined to the notion of *dual simulation* by Ma et al. [85].

Definition 2.23 (Dual Simulation)
Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2)$ be two graphs. A binary relation $R \subseteq V_1 \times V_2$ is called a *dual simulation between G_1 and G_2* iff for every pair $(v_1, v_2) \in R$,

1. $v_1\ E_1^a\ w_1$ implies $\exists w_2 \in V_2$ with $v_2\ E_2^a\ w_2$ and $(w_1, w_2) \in R$, and
2. $u_1\ E_1^a\ v_1$ implies $\exists u_2 \in V_2$ with $u_2\ E_2^a\ v_2$ and $(u_1, u_2) \in R$.

G_2 *dual simulates* G_1, denoted $G_1 \sqsubseteq_{\mathsf{Dsim}} G_2$, if there is a non-empty dual simulation between G_1 and G_2. ▲

Only isolated nodes, i. e., nodes with neither incoming nor outgoing edges, may be simulated by any other node and thus by any other graph. Although we did not restrict our graph database model to obey all restrictions of RDF, it is still worthwhile noticing that RDF graphs cannot contain isolated nodes because every data object must occur in an RDF triple (cf. Section 2.2.1). Thus, every data object occurs at least as one subject or object, i. e., source or target of an edge in the graph database.

Example 2.24 We review the graph simulations $\widehat{R}$, R_1, R_2, and R_0 from Examples 2.20 and 2.21 between the graph database $DB_{2.6}$ and schema $S_{2.7}$. Graph simulation $\widehat{R}$ is a dual simulation. R_1, being $\widehat{R}$ without pair $(A, \textsc{Root})$, is not a dual simulation. Here, property 2 recognizes, for instance, edge $A\ E^{book}_{DB_{2.6}}\ C$. Since $(C, \textsc{Book}) \in R_1$, Definition 2.23 expects some node v in $S_{2.7}$ with $v\ E^{book}_{S_{2.7}}$ Book and $(A, v) \in R_1$. But the only node qualifying as v is Root and $(A, \textsc{Root}) \notin R_1$. Not even R_2 is a dual simulation: Although node 10 is a node representing an Isbn, the predecessor nodes of 10 are not covered by R_2. The same holds for R_0. In summary, while $\widehat{R}$ is a dual simulation, neither R_0, R_1, nor R_2 are. ■

We have finally reached a reasonable notion of conformance between a graph database DB and a graph schema S, namely non-empty dual simulations between DB and S.

Definition 2.25 (Graph Schema Conformance)
Let S be a graph schema. A graph database DB *conforms to* S, denoted $DB \preccurlyeq S$, iff $DB \sqsubseteq_{\mathsf{Dsim}} S$. We call a non-empty dual simulation R between DB and S a *conformance witness between DB and S*. If $DB \preccurlyeq S$, then DB is called an *instance of* S. ▲

An empty-structured graph database DB, i. e., $DB = (O_{DB}, \Sigma, \emptyset)$ with $O_{DB} \neq \emptyset$, trivially conforms to any graph schema (cf. Proposition 2.22). Conversely, every graph database conforms to the *unit graph schema*, which is a schema with a single node U and a Σ-self-loop, as depicted in Figure 2.10. Node U dual simulates every node of a given graph database, indicated by the self-loop labeled by Σ. Hence, the unit graph schema may be a good start when designing a schema for graph data from scratch.

In summary, we have developed a non-trivial schema for graph databases, capturing relationships in general, no matter whether incoming or outgoing edges model them. Compared to Buneman et al. [29], the only trait we have dropped is that the root node on the database-side guarantees to cover all other database nodes. In their case, either all database objects participate in a conformance witness, or none do. Hence, we may observe conformance witnesses concerning some part of a graph database while neglecting another.

[1] Neglecting backward edges in program graphs is feasible because programs/processes usually do not run backward.

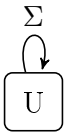


Figure 2.10: The Unit Graph Schema

Example 2.26 Consider the graph depicted in Figure 2.11 (a), denoted as $DB_{2.11\,(a)}$. As before with our rooted graphs, $DB_{2.11\,(a)}$ represents some bibliographic contents, here the book *Data on the Web* (`DotW`) and the survey article *Survey of Graph Database Models* (`SGDM`). $DB_{2.11\,(a)}$ consists of two instances of separate graph schemas. While `DotW` has type Book w. r. t. the graph schema in Figure 2.11 (b), `SGDM` has type Article w. r. t. the schema depicted in Figure 2.11 (c). Thus, $DB_{2.11\,(a)}$ is an instance of both graph schemas, but one of its connected subgraphs is untyped w. r. t. a single one. For instance, `SGDM` is not a Book, according to the graph schema in Figure 2.11 (b), as type Book has no outgoing edge labeled *journal*. ■

Thus, we may miss out on some subgraphs of a graph database, which is to be expected in such data models. Otherwise, we guarantee meaningful conformance witnesses by requiring graph schemas to be connected.

2.3.2 Object Classification

In the analysis of Abiteboul et al. [3], deriving a classification of graph database objects (in their case, of semistructured data objects) is characterized by three aspects, distinguishing it from other data models, such as relational or object-oriented databases:

(1) A type is *less precise*, i. e., one database object may belong to more than one type. This aspect is covered by the nature of dual simulations, being relations rather than functions.

(2) An object may belong to *no type at all*. We sketched this situation by Example 2.26. In semistructured data, modeled by rooted labeled graphs, it was impossible to leave a database object untyped because, once the database's root node participates in a simulation (which was required by Buneman et al. [29]), every database object is also covered by the respective conformance witness (as they are reachable via directed paths from the root).

(3) The induced typing is *approximate*, meaning some objects do not meet all structural aspects of a type. For instance, the bibliographic objects in $DB_{2.11\,(a)}$ do not carry any information about their authors, while the respective graph schemas include them. Nevertheless, $DB_{2.11\,(a)}$ is an instance of both graph schemas in Figures 2.11 (b) and 2.11 (c).

By *graph schema conformance*, we have an answer of how to assess whether a graph database is an instance of a given schema. We make use of the conformance witnesses to derive a classification of graph database objects w. r. t. a graph schema [3].

Definition 2.27 (Object Classification)
Let $S = (T_S, \Sigma, E_S)$ be a graph schema and DB a graph database with $DB \preccurlyeq S$. An object o of DB has *type* $t \in T_S$, denoted $o \vdash_S t$, iff there is a conformance witness R between DB and S with $(o, t) \in R$. ▲

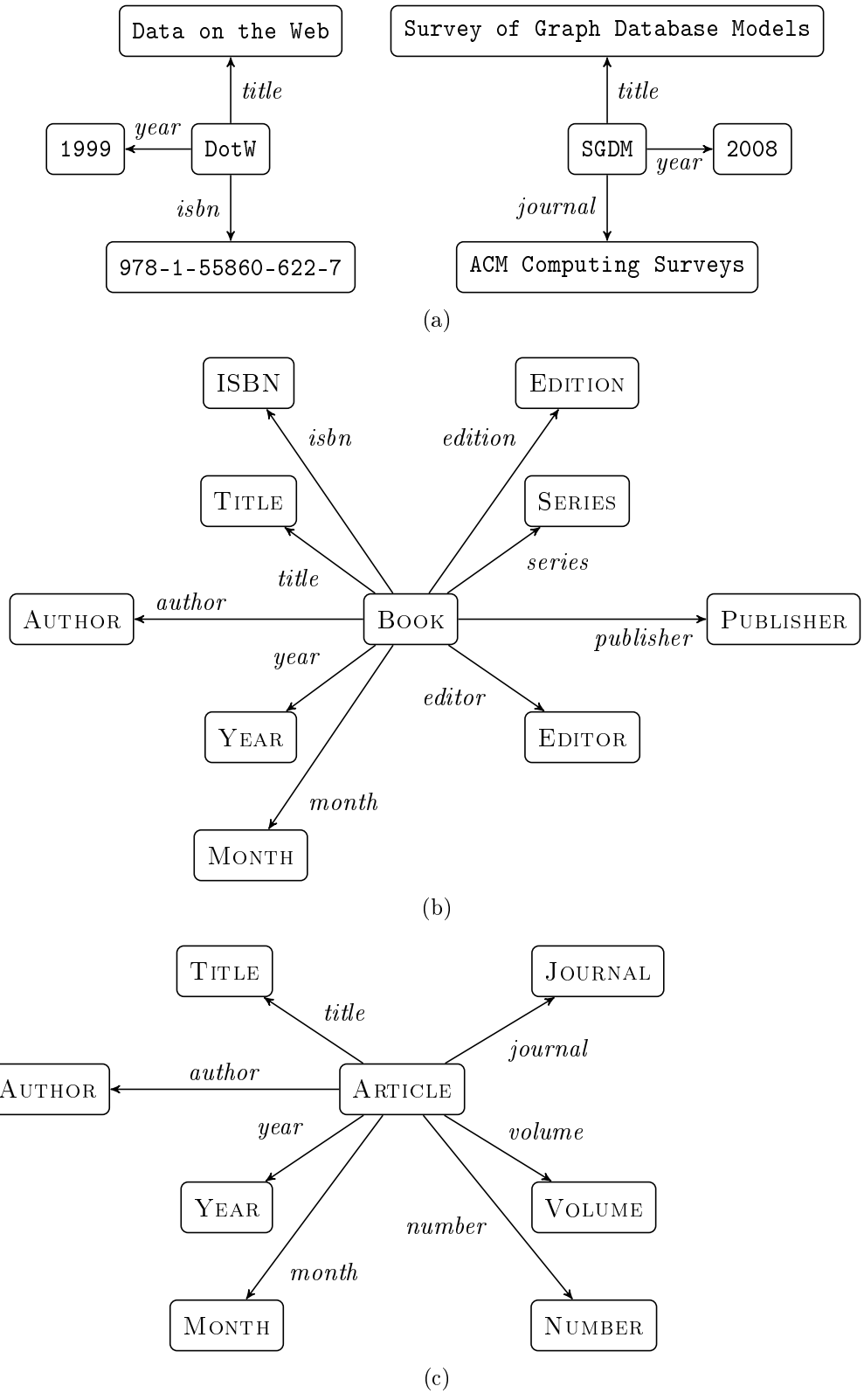


Figure 2.11: (a) A Disconnected Graph Database (b) A Graph Schema Describing Books (c) A Graph Schema Describing Articles

Example 2.28 Suppose we have given the graph schema $S_{2.11\,(b)+2.11\,(c)}$, being the *join* of the graphs depicted in Figures 2.11 (b) and 2.11 (c). Both graph schemas agree on the types TITLE and YEAR, making $S_{2.11\,(b)+2.11\,(c)}$ a connected graph, as required by Definition 2.15. As argued in Example 2.26, $DB_{2.11\,(a)}$ is an instance of both graph schemas in Figures 2.11 (b) and 2.11 (c). Consequently, $DB_{2.11\,(a)}$ is an instance of $S_{2.11\,(b)+2.11\,(c)}$. A conformance witness validating $\texttt{SGDM} \vdash_{S_{2.11\,(b)+2.11\,(c)}} \text{ARTICLE}$ may be given by

$$R_1 = \left\{ \begin{array}{l} (\texttt{SGDM}, \text{ARTICLE}), (\texttt{Survey of Graph Database Models}, \text{TITLE}), \\ (\texttt{2008}, \text{YEAR}), (\texttt{ACM Computing Surveys}, \text{JOURNAL}) \end{array} \right\}.$$

The same witness cannot be used to verify $\texttt{DotW} \vdash_{S_{2.11\,(b)+2.11\,(c)}} \text{BOOK}$. However,

$$R_2 = \left\{ \begin{array}{l} (\texttt{DotW}, \text{BOOK}), (\texttt{Date on the Web}, \text{TITLE}), \\ (\texttt{1999}, \text{YEAR}), (\texttt{978-1-55860-622-7}, \text{ISBN}) \end{array} \right\}$$

would do. ■

This approach to classification is not very practical since the number of conformance witnesses is in $\mathcal{O}(2^{|O_{DB}|\cdot|T_S|})$. Scanning through an exponential number of witnesses in order to verify a single typing, especially in a big data setting, is infeasible. Fortunately, (dual) simulations are union-closed [96, 86], which entails the *fundamental (dual) simulation property*: Uniqueness of the greatest (dual) simulation.

Theorem 2.29 *Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2)$ and $\mathcal{S}(G_1, G_2)$ the set of all dual simulations between G_1 and G_2. Then the following properties hold.*

(1) $\mathcal{S}(G_1, G_2)$ is union-closed*, i. e., for every $X \subseteq \mathcal{S}(G_1, G_2)$ with $|X| \geq 1$, $\bigcup X \in \mathcal{S}(G_1, G_2)$.*

(2) $\mathcal{S}(G_1, G_2)$ has a unique greatest element w. r. t. $\subseteq$.

PROOF: (2) follows directly from (1). Towards a contradiction, assume there are two distinct greatest elements R^1_ω and R^2_ω in $\mathcal{S}(G_1, G_2)$, i. e., for all $R' \in \mathcal{S}(G_1, G_2)$, $R^i_\omega \subseteq R'$ implies $R^i_\omega = R'$ $(i = 1, 2)$. Thus, $R^i_\omega \nsubseteq R^j_\omega$ $(i, j = 1, 2$ and $i \neq j)$. Consider now $R_\omega = R^1_\omega \cup R^2_\omega$. It holds that $R^i_\omega \subseteq R_\omega$ $(i = 1, 2)$ and because of (1), $R_\omega \in \mathcal{S}(G_1, G_2)$. Thus, $R^1_\omega = R_\omega$ and $R_\omega = R^2_\omega$, contradicting the assumption that R^i_ω $(i = 1, 2)$ are distinct elements from $\mathcal{S}(G_1, G_2)$. Therefore, (2) holds if (1) holds.

It remains to be shown that (1) holds, proven by induction on the size of $X \subseteq \mathcal{S}(G_1, G_2)$ for $|X| \geq 1$.

Base: If $|X| = 1$, it contains a single dual simulation $R \in \mathcal{S}(G_1, G_2)$, i. e., $X = \{R\}$. Since $\bigcup X = R$, $\bigcup X \in \mathcal{S}(G_1, G_2)$ by assumption.

Hypothesis: For all $X \subseteq \mathcal{S}(G_1, G_2)$ with $|X| < n$, it holds that $\bigcup X \in \mathcal{S}(G_1, G_2)$.

Step: Let $X \subseteq \mathcal{S}(G_1, G_2)$ with $|X| = n$. Then $X = X_{<n} \cup \{R^*\}$, such that $|X_{<n}| = n - 1$ and $R^* \in \mathcal{S}(G_1, G_2)$. Since $|X_{<n}| = n - 1$, it holds that $\bigcup X_{<n} \in \mathcal{S}(G_1, G_2)$, i. e., $\bigcup X_{<n}$ is a dual simulation between G_1 and G_2. It holds that $\bigcup X = R^* \cup \bigcup X_{<n}$, of which we need to show that it is a dual simulation between G_1 and G_2. Let $(v_1, v_2) \in \bigcup X$ and $v_1\ E^a_1\ w_1$.

1. If $(v_1, v_2) \in R^*$, there is a $w_2 \in V_2$ with $v_2\ E^a_2\ w_2$ and $(w_1, w_2) \in R^*$ because R^* is a dual simulation. Hence, $(w_1, w_2) \in \bigcup X$.
2. If $(v_1, v_2) \in \bigcup X_{<n}$, there is a $w_2 \in V_2$ with $v_2\ E^a_2\ w_2$ and $(w_1, w_2) \in \bigcup X_{<n}$ because $\bigcup X_{<n}$ is a dual simulation by induction hypothesis.

Hence, $(w_1, w_2) \in \bigcup X$ because $(w_1, w_2) \in R^* (\subseteq \bigcup X)$ or $(w_1, w_2) \in \bigcup X_{<n} (\subseteq \bigcup X)$. The case of $u_1 \; E_1^a \; v_1$ is completely analogous. Thus, $\bigcup X$ is a dual simulation between G_1 and G_2.

It holds that $\bigcup X \in \mathcal{S}(G_1, G_2)$ for every $X \subseteq \mathcal{S}(G_1, G_2)$. Q. E. D.

Between any two graphs, there is a unique greatest dual simulation that subsumes every other dual simulation between the two graphs. Since this particular dual simulation is of primary importance, also beyond this chapter, we define it formally as the *maximal dual simulation*, justified by Theorem 2.29.

Definition 2.30 (Maximal Dual Simulation)
Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2)$ be two graphs. The union of all dual simulations between G_1 and G_2 is called the *maximal dual simulation between G_1 and G_2*. ▲

Example 2.31 We apply the maximal dual simulation to conformance witnesses R_1 and R_2 from Example 2.26. $R_1 \cup R_2$ is the maximal dual simulation between $DB_{2.11\,(a)}$ and $S_{2.11\,(b)+2.11\,(c)}$. It verifies both, $\texttt{SGDM} \vdash_{S_{2.11\,(b)+2.11\,(c)}}$ ARTICLE and $\texttt{DotW} \vdash_{S_{2.11\,(b)+2.11\,(c)}}$ BOOK. ■

Hence, the typing relation $\vdash_S$ of database objects from DB and types from S ($DB \preccurlyeq S$) is the maximal dual simulation between DB and S. Beyond object classification, the maximal dual simulation helps us deciding whether there is a non-empty dual simulation between graphs G_1 and G_2, i. e., if $G_1 \sqsubseteq_{\mathsf{Dsim}} G_2$, being a direct consequence of Theorem 2.29.

Corollary 2.32 *Let G_1, G_2 be graphs. $G_1 \sqsubseteq_{\mathsf{Dsim}} G_2$ iff the maximal dual simulation between G_1 and G_2 is non-empty.*

2.3.3 Semantics of Graph Schemas

Dual simulations provide us with the means of graph schema instances (cf. Definition 2.25). In this section, we study the properties of dual simulations in a broader context to find out what they entail for graph databases and schemas. This way, we will obtain the precise *semantics of graph schemas*.

First note that a graph G always dual simulates itself by a non-trivial, i. e., non-empty, dual simulation R. The necessary dual simulation is the identity function on the set of nodes of G. Second, if we have dual simulations R_1 between graphs G_1 and G_2, and R_2 between G_2 and G_3, then

$$R_1 \circ R_2 := \{(u, w) \mid \exists v : (u, v) \in R_1 \wedge (v, w) \in R_2\} \tag{2.3}$$

is a dual simulation between G_1 and G_2.

Proposition 2.33 *Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2, 3)$ be graphs.*

(1) id_{V_1} is a dual simulation between G_1 and G_1.

(2) Let R_i $(i = 1, 2)$ be dual simulations between G_i and G_{i+1}. Then $R_1 \circ R_2$ is a dual simulation between G_1 and G_3.

PROOF: Note, according to Proposition 2.19, the empty simulation proves the existence of dual simulations in both cases. The claims here are stronger.

(1) Let $(v_1, v_2) \in \mathsf{id}_{V_1}$ and $v_1 \; E_1^a \; w_1$. Since $v_2 = \mathsf{id}_{V_1}(v_1) = v_1$, take $w_2 = w_1$. For sure, $v_2 = v_1 \; E_1^a \; w_1 = w_2$ and $w_2 = w_1 = \mathsf{id}_{V_1}(w_1)$, i. e., $(w_1, w_2) \in \mathsf{id}_{V_1}$. The case $u_1 \; E_1^a \; v_1$ is completely analogous. Thus, id_{V_1} is a dual simulation.

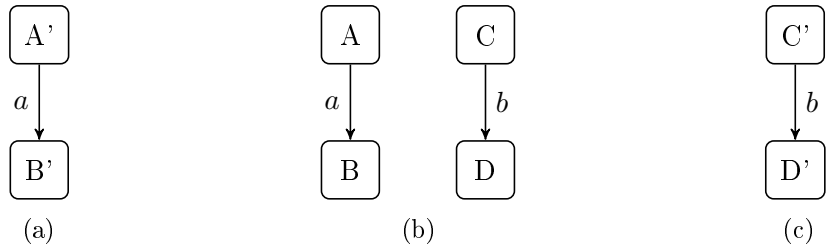


Figure 2.12: Non-Transitivity of $\preccurlyeq$

(2) Let $(v_1, v_3) \in R_1 \circ R_2$ and $v_1\ E_1^a\ w_1$. By construction of $R_1 \circ R_2$ (cf. Equation (2.3)), there is a $v_2 \in V_2$ with $(v_1, v_2) \in R_1$ and $(v_2, v_3) \in R_2$. Because R_1 is a dual simulation between G_1 and G_2, there is a $w_2 \in V_2$ with $v_2\ E_2^a\ w_2$ and $(w_1, w_2) \in R_1$. Because R_2 is a dual simulation between G_2 and G_3, there is a $w_3 \in V_3$ with $v_3\ E_3^a\ w_3$ and $(w_2, w_3) \in R_2$. Take $v_3\ E_3^a\ w_3$ to dual simulate $v_1\ E_1^a\ w_1$. By construction, $(w_1, w_3) \in R_1 \circ R_2$. Again, the case of $u_1\ E_1^a\ v_1$ is completely analogous. Q. E. D.

This proof is an important step towards the semantics of graph schemas. We already know that a graph schema S dual simulates itself by a non-empty dual simulation. Prolonging this fact to the instance level, we derive

$$DB \preccurlyeq S \sqsubseteq_{\mathsf{Dsim}} S. \tag{2.4}$$

Moreover, Proposition 2.33 tells us how to obtain a conformance witness between DB and the right-most graph schema S: The conformance witness R between DB and the inner S is concatenated with the identity on the nodes of S. As R is non-empty, its concatenation with the identity on the right yields R, thus, a non-empty dual simulation.

What if we replace the right-most occurrence of S in (2.4) by another graph schema T? Does $DB \preccurlyeq S \sqsubseteq_{\mathsf{Dsim}} T$ still guarantee that DB is an instance of T? While Proposition 2.33 has been shown for general graphs, it turns out that $\sqsubseteq_{\mathsf{Dsim}}$ is, in general, not a preorder since non-emptiness of dual simulations is not transitively guaranteed, even if the dual simulation components are non-empty.

Example 2.34 Consider the three graphs $G_{(a)}$, $G_{(b)}$, and $G_{(c)}$ in Figure 2.12 (a), (b), and (c). It holds that $G_{(a)} \sqsubseteq_{\mathsf{Dsim}} G_{(b)}$ and $G_{(b)} \sqsubseteq_{\mathsf{Dsim}} G_{(c)}$ by non-empty dual simulations $R_1 = \{(A', A), (B', B)\}$ and $R_2 = \{(C, C'), (D, D')\}$. By Proposition 2.33, $R_1 \circ R_2$ is also a dual simulation, but in this case $R_1 \circ R_2 = \emptyset$. The two witnesses R_1 and R_2 provide proofs of $\sqsubseteq_{\mathsf{Dsim}}$ in different subgraphs of $G_{(b)}$. There is no non-empty dual simulation between $G_{(a)}$ and $G_{(c)}$. ■

Fortunately, graph schemas are not general graphs but they are connected graphs (cf. Definition 2.15). The reason for this early design decision may finally be justified by the requirement to maintain the non-emptiness of dual simulations transitively. We call the thus obtained restriction of non-empty dual simulations between graph schemas *graph schema refinement*.

Definition 2.35 (Graph Schema Refinement)
Let S_1 and S_2 be two graph schemas. S_1 *refines* S_2, denoted $S_1 \preccurlyeq S_2$, iff there is a non-empty dual simulation between S_1 and S_2. Analogously to conformance witnesses (cf. Definition 2.25), we call a non-empty dual simulation R between S_1 and S_2 a *refinement witness between S_1 and S_2*. ▲

As the only difference to graph schema conformance is the type of graphs participating in $\preccurlyeq$, graph schema refinement extends graph schema conformance to arbitrary graph

schemas. Thus, the minimal elements of $\preccurlyeq$ are database instances that may conform to several less and less restrictive, graph schemas. Graph schema refinement is, in fact, a preorder.

Lemma 2.36 *Graph schema refinement $\preccurlyeq$ is a preorder.*

PROOF: Let $S_i = (T_i, \Sigma, E_i)$ $(i = 1, 2, 3)$ be graph schemas.

Reflexivity: $S_1 \preccurlyeq S_1$ is already proven in Proposition 2.33 because the identity on the nodes of S_1 is non-empty if there is at least one type in S_1, a requirement of Definition 2.15.

Transitivity: Let $S_1 \preccurlyeq S_2$ by witness R_1 and $S_2 \preccurlyeq S_3$ by witness R_2. Again by Proposition 2.33, $R_1 \circ R_2$ is a dual simulation between S_1 and S_3. If $R_1 \circ R_2$ is non-empty, the proof is complete.

Towards a contradiction, assume $R_1 \circ R_2$ is the empty dual simulation. Since R_1 and R_2 are refinement witnesses, and thereby non-empty, for every pair $(u, v) \in R_1$ there is no pair $(v, w) \in R_2$ (for some $w \in T_3$), i. e., node $v \in T_2$ cannot be simulated by any node of S_3. As $R_2 \neq \emptyset$, there is some pair $(v_0, w_0) \in R_2$. If we can show that for every node v_1 that can be reached by a path from v_0, there is some pair $(v_1, w_1) \in R_2$, $(v, w) \notin R_2$ only means that S_2 is a disconnected graph, contradicting the prerequisite that S_2 is a graph schema (cf. Definition 2.15). Hence, the assumption is wrong and $R_1 \circ R_2$ is a refinement witness between S_1 and S_3.

It remains to be shown that if π is a path between v_0 (recall that $(v_0, w_0) \in R_2$) and some nodes $v_1 \in T_2$, then there is a node $w_1 \in T_3$ with $(v_1, w_1) \in R_2$. By induction on the length of π.

Base: If $|\pi| = 0$, then $\pi = v_0$ and $(v_0, w_0) \in R_2$ by assumption. Hence, set $w_1 = w_0$.

Hypothesis: For paths π between v_0 and v_1 with $|\pi| < n$, there is a $w_1 \in T_3$ with $(v_1, w_1) \in R_2$.

Step: Let π be a path with $|\pi| = n$. Hence, $\pi = \pi_{<n} \cdot v_1$ with $|\pi_{<n}| = n - 1 < n$. Let $v_{<n} = \mathsf{last}(\pi_{<n})$. Thus, by induction hypothesis, there is a $w_{<n} \in T_3$ with $(v_{<n}, w_{<n}) \in R_2$. As π is a path, for some $a \in \Sigma$ either (i) $v_{<n}\ E_2^a\ v_1$ or (ii) $v_1\ E_2^a\ v_{<n}$. Since R_2 is a dual simulation, (i) ((ii), resp.) implies the existence of a $w_1 \in T_3$ with $w_{<n}\ E_3^a\ w_1$ ($w_1\ E_3^a\ w_{<n}$, resp.) and $(v_1, w_1) \in R_2$.

Thus, every node reachable from v_0 by a path π must also occur in R_2. Consequently, there must be a pair $(v, w) \in R_2$ and $R_1 \circ R_2$ is non-empty.

Thus, graph schema refinement is a preorder. Q. E. D.

In $S \preccurlyeq T$, graph schema S is thought of as more restrictive w. r. t. its database instances than graph schema T. A refinement witness automatically covers all objects of S but not necessarily all of T. Hence, a database instance of S may be typed by T as well.

Example 2.37 Consider $S_{2.13}$ in Figure 2.13, a graph schema for instances storing chapters of book publications. The only difference to the graph schema about books (cf. Figure 2.11 (b), referred to as $S_{2.11\,(b)}$) is that an INBOOK features a CHAPTER. All other properties are kept the same. Thus, an instance of $S_{2.13}$ describes every instance $S_{2.11\,(b)}$ describes. Additionally, instances of $S_{2.13}$ are allowed to have a CHAPTER associated with the bibliographic item.

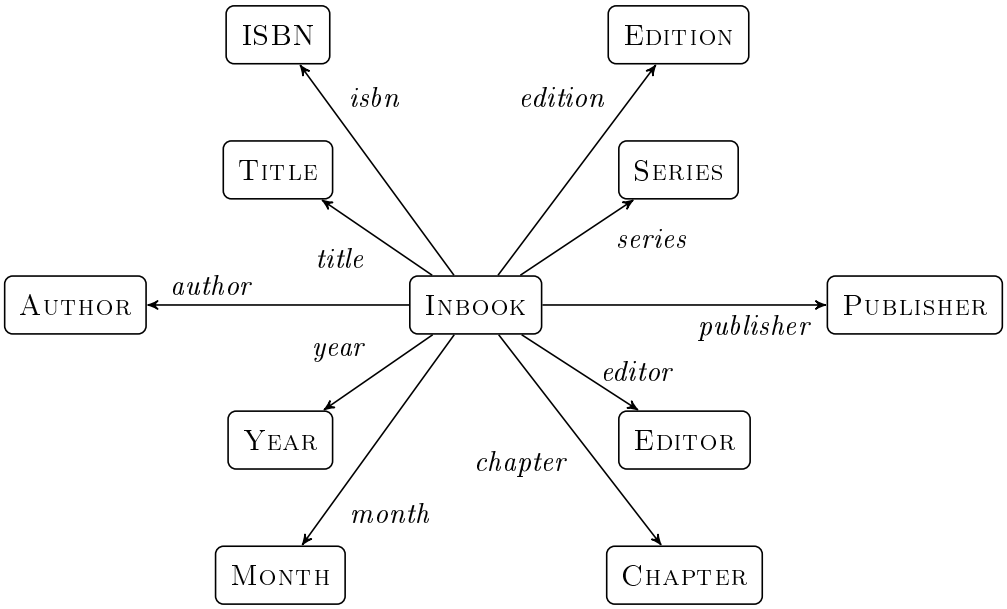


Figure 2.13: A Graph Schema Describing INBOOK

Every instance of type BOOK is an instance of INBOOK. Refinement allows us to make this relationship precise, in that $S_{2.11\,(b)} \preccurlyeq S_{2.13}$. The necessary refinement witness is

$$R = \left\{ \begin{array}{l} (\text{BOOK}, \text{INBOOK}), (\text{ISBN}, \text{ISBN}), (\text{TITLE}, \text{TITLE}), \\ (\text{YEAR}, \text{YEAR}), (\text{MONTH}, \text{MONTH}), (\text{AUTHOR}, \text{AUTHOR}), \\ (\text{EDITOR}, \text{EDITOR}), (\text{PUBLISHER}, \text{PUBLISHER}), \\ (\text{EDITION}, \text{EDITION}), (\text{SERIES}, \text{SERIES}) \end{array} \right\}.$$

CHAPTER does not occur in R at all. Hence, there are instances of $S_{2.13}$ that are not instances of $S_{2.11\,(b)}$. ■

In order to show transitivity of $\preccurlyeq$, the proof of Lemma 2.36 relied on non-emptiness of R_1 and the fact that R_2 is a refinement witness between S_2 and S_3, where connectedness of S_2 was exploited. Especially S_1 does not necessarily have to be a graph schema. We make use of this observation in showing that an instance of S is an instance of T, given that $S \preccurlyeq T$.

Lemma 2.38 (Soundness of $\preccurlyeq$) *Let S_1, S_2 be graph schemas. If $S_1 \preccurlyeq S_2$, then every instance of S_1 is an instance of S_2, i. e., $DB \preccurlyeq S_1$ implies $DB \preccurlyeq S_2$.*

PROOF: Let DB be an instance of S_1, i. e., $DB \preccurlyeq S_1$ by a non-empty dual simulation (conformance witness, resp.) R_1. Since $S_1 \preccurlyeq S_2$, there is a non-empty dual simulation (refinement witness, resp.) R_2 between S_1 and S_2. By the proof of Lemma 2.36, $R_1 \circ R_2$ is a non-empty dual simulation between DB and S_2 because S_1 is a connected graph. Hence, $R_1 \circ R_2$ is a conformance witness between DB and S_2, i. e., $DB \preccurlyeq S_2$. Q. E. D.

Therefore, an iterative design process for graph schemas is justified due to refinement. Start with some initial graph schema S_0 and refine it to S_1, S_2, ..., S_k ($k \in \mathbb{N}$) with $S_i \preccurlyeq S_{i-1}$ ($1 \leq i \leq k$). Lemma 2.38 guarantees that every instance of S_i is an instance of S_j ($1 \leq i < j \leq k$). Is this preservation of instances under refinement a coincidence, or is there a general pattern to be recognized? If, for graph schemas S and T, every instance of S is also an instance of T, refinement between S and T detects this for us.

Lemma 2.39 (Completeness of $\preccurlyeq$) *Let S_1 and S_2 be graph schemas. If all instances of S_1 are instances of S_2, then $S_1 \preccurlyeq S_2$.*

PROOF: Towards a contradiction, assume $S_1 \not\preccurlyeq S_2$, i. e., there is no non-empty dual simulation between S_1 and S_2. Then there is a largest subgraph $S_{\max} \subseteq S_1$ such that $S_{\max} \preccurlyeq S_2$ because, in the worst case, we remove every edge from S_1, leaving all its nodes isolated. Recall that graph schemas have at least one type (cf. Definition 2.15). By a similar argument as for simulations (cf. Proposition 2.22), a disconnected node (no incoming or outgoing edges) is dual simulated by every graph node. Now we add a single edge from S_1 that is not yet present in $S_{\max}$, let us refer to the resulting graph as $DB_{\not\downarrow}$. Since $DB_{\not\downarrow} \subseteq S_1$, there is a non-empty dual simulation between $DB_{\not\downarrow}$ and S_1 (cf. Theorem 3.24 on Page 58). Therefore, $DB_{\not\downarrow} \preccurlyeq S_1$. But $DB_{\not\downarrow} \not\preccurlyeq S_2$ because $S_{\max}$ was chosen to be a largest subgraph of S_1 with $S_{\max} \preccurlyeq S_2$, i. e., every additional structure from S_1 leads to non-conformance to S_2. $DB_{\not\downarrow} \not\preccurlyeq S_2$ is a contradiction to the assumption that every instance of S_1 is an instance of S_2. Thus, $S_1 \preccurlyeq S_2$. Q. E. D.

Graph schema refinement is a sound and complete characterization of the inclusion of instances of graph schemas. The following theorem is a direct consequence of Lemmas 2.38 and 2.39.

Theorem 2.40 *Let S_1, S_2 be two graph schemas. An instance of S_1 is an instance of S_2 iff $S_1 \preccurlyeq S_2$.*

Thus, we have reason to formulate the semantics of graph schemas as their sets of instances. As $\preccurlyeq$ relates graph schemas only if their sets of instances are related, it holds that two graph schemas S and T have the same instances iff they are equivalent up to $\preccurlyeq$, i. e., $S \preccurlyeq T$ and $T \preccurlyeq S$.

Definition 2.41 (Graph Schema Semantics)
The *semantics of a graph schema S up to $\preccurlyeq$* is defined by

$$[\![S]\!]_{\preccurlyeq} := \{DB \mid DB \preccurlyeq S\}.$$ ▲

Note that the semantic function $[\![\cdot]\!]_{\preccurlyeq}$ maps from a virtual universe of all graph schemas to a subset of another virtual universe, the set of all graph databases. Applying Theorem 2.40 to the semantics of graph schemas S and T, we obtain the following result as a corollary.

Corollary 2.42 *Let S_1, S_2 be two graph schemas. $[\![S_1]\!]_{\preccurlyeq} \subseteq [\![S_2]\!]_{\preccurlyeq}$ iff $S_1 \preccurlyeq S_2$. $[\![S_1]\!]_{\preccurlyeq} = [\![S_2]\!]_{\preccurlyeq}$ iff $S_1 \preccurlyeq S_2$ and $S_2 \preccurlyeq S_1$.*

Buneman et al. have given the same definition of graph schema semantics up to graph simulation [29]. We extended their results to dual simulations. What distinguishes our presentation from theirs is that Buneman et al. presumed the semantics to be defined as in Definition 2.41 and derived a sound and complete procedure for deciding instance subsumption and equivalence. We started from the properties of dual simulation, leading to graph schema refinement being a preorder, from which we were able to prove that instances are preserved (Lemma 2.38). We then showed that, if instances are preserved between any two graph schemas, then this is because the graph schemas refine one another (Lemma 2.39). These two results allowed us to give a formally justified definition of the graph schema semantics.

There is a subtle difference in the results obtained by Buneman et al., found in the proof of Lemma 2.39. Buneman et al.'s proof is based on the observation that a graph schema S is a least upper bound (w. r. t. $\preccurlyeq$) of the set $[\![S]\!]_{\preccurlyeq}$ (Theorem 7 in [29]). In their work, $[\![S]\!]_{\preccurlyeq}$ is partially ordered by $\preccurlyeq$, i. e., also database instances can be related by $\preccurlyeq$. $\preccurlyeq$ is a preorder when defined as rooted graph simulations on rooted labeled graphs. This is a property we could not rely on (cf. Example 2.34). From $[\![S]\!]_{\preccurlyeq} \subseteq [\![T]\!]_{\preccurlyeq}$, Buneman et al. deduce for every least upper bound S' of $[\![S]\!]_{\preccurlyeq}$, there must be a least upper bound T' of

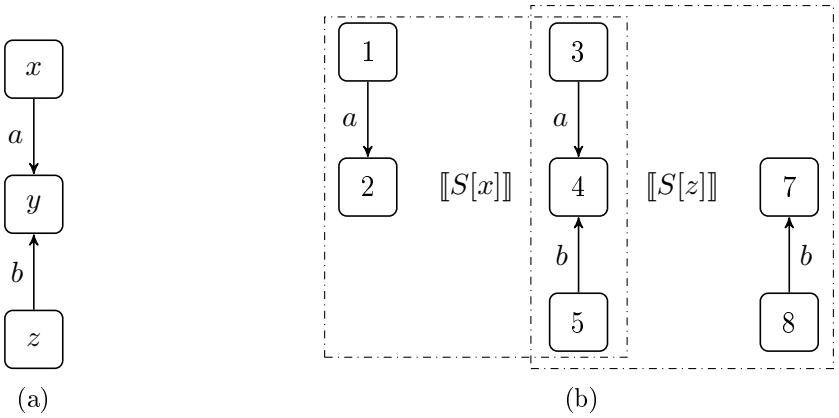


Figure 2.14: (a) A Schema Graph S and (b) Different Instances of S

$[\![T]\!]_{\preccurlyeq}$, such that $S' \preccurlyeq T'$. Hence, let T' be a least upper bound of $[\![T]\!]_{\preccurlyeq}$ with $S \preccurlyeq T'$. From Theorem 7 in [29], Buneman et al. deduce that T is a least upper bound of $[\![T]\!]_{\preccurlyeq}$. Thus, $T' \preccurlyeq T$. By transitivity of $\preccurlyeq$, $S \preccurlyeq T$ (Corollary 8 in [29]). The central idea of Buneman et al.'s proof is to exploit that $\preccurlyeq$ is a preorder, also on the level of their graph data.

2.3.4 On Types of Interest

In an earlier version of graph schemas, we have been using so-called *types of interest* instead of relying on non-emptiness of dual simulations for the notion of conformance witnesses. Each graph schema $S = (T_S, \Sigma, E_S)$ was equipped with a designated subset of its types, $X \subseteq T_S$, representing the types the schema designer is interested in modeling. We denoted such graph schemas by $S[X] = (T_S, \Sigma, E_S, X)$. A dual simulation would qualify as a conformance witness between a graph database DB and S if it covered all types of interest $x \in X$. An advantage of this kind of modeling tool is that $S[X]$ describes databases that guarantee to have objects of type x for all $x \in X$. Furthermore, an appropriate refinement between graph schemas turned out to have almost all beneficial properties, e. g., it was a preorder. The particular formulation we used is:

> Graph schema $S[X]$ *refines* graph schema $T[Y]$, denoted $S[X] \preccurlyeq T[Y]$, iff there is a dual simulation R between S and T, such that for all $y \in Y$ there is an $x \in X$ with $(x, y) \in R$.

Note that, by design, $X, Y \neq \emptyset$ for every graph schema, guaranteeing non-empty dual simulations as refinement witnesses. The soundness theorem (cf. Lemma 2.38) was reproducible for graph schemas with types of interest. However, when it comes to completeness (cf. Lemma 2.39), the set of all instances of schema $S[X]$ fails to capture what $S[X]$ is actually about.

Example 2.43 Consider the graph schema S depicted in Figure 2.14 (a). Depending on whether we choose x, y, or z as a type of interest, we describe different sets of instances. For example, focusing on x will yield the first two (from left to right) types of database instances in Figure 2.14 (b). The instance depicted on the right-hand-side cannot be typed by $S[\{x\}]$ because it is impossible to find an object whose incidence is reflected by node x. Node *8* has an outgoing b-labeled edge and node *7* an incoming one, while x has only an outgoing a-labeled edge. The same argument holds for $S[\{z\}]$ when trying to obtain an instance with nodes *1* and *2*. Thus, the sets of instances of $S[\{x\}]$ and $S[\{z\}]$ are incomparable, which is also confirmed by refinement since neither $S[\{x\}] \preccurlyeq S[\{z\}]$ nor $S[\{z\}] \preccurlyeq S[\{x\}]$.

When it comes to y, $S[\{y\}]$ describes all three instances depicted in Figure 2.14 (b). In fact, $[\![S[\{x\}]]\!] \subseteq [\![S[\{y\}]]\!]$ and $[\![S[\{z\}]]\!] \subseteq [\![S[\{y\}]]\!]$. However, $S[\{x\}] \not\preccurlyeq S[\{y\}]$ (in the above-mentioned sense) because y does not dual simulate x in S. Likewise, $S[\{z\}] \not\preccurlyeq S[\{y\}]$. In this situation, although one graph schema ($S[\{x\}]$ or $S[\{z\}]$) describes a subset of the instances of another ($S[\{y\}]$), $\preccurlyeq$ cannot confirm this circumstance. ■

Hence, graph schema refinement would not reliably identify the subsumption of graph schema instances, meaning, $[\![S[X]]\!]_{\preccurlyeq}$ is not only a set of database instances of $S[X]$, but must also include a distinguishing characteristic for the instances derived from the graph schemas in Example 2.43. One possible road to overcome this issue is taken in Section 2.4. As far as this section is concerned, we consider the characterization of graph schemas and their semantics completed.

2.3.5 Graph schemas in RDF

Our graph schema model is an extension of the one that Buneman et al. [29] and Abiteboul et al. [3] propose for semistructured data. Alternatively, RDF comes itself with a notion of *instances*, substantiated in graph homomorphisms between RDF graphs [62]. Because of blank nodes and RDFS vocabulary, RDF is a language describing schemas and instances that build on the same principles, namely that of RDF graphs. Let G and H be RDF graphs. H *is an instance of* G if there is a *map between* G *and* H. A function $\mu : (I \cup B \cup L) \to (I \cup B \cup L)$ is called a *map between* G *and* H iff

(1) $\mu(n) = n$ for all IRIs and literals $n \in I \cup L$, and

(2) $(\mu(s), \mu(p), \mu(o)) \in \mathsf{H}$ for all RDF triples $(s, p, o) \in \mathsf{G}$.

Note that maps are graph homomorphisms with the additional feature that also the edge labels are mapped by μ, mainly because predicates are resources also occurring as nodes [66]. Since predicates are IRIs, condition (1) requires μ to map each predicate to itself, being one specific form of alphabet alignment, namely identity (cf. Section 2.1.1). Due to (2), H is an instance of G if there is a graph homomorphism between G and H [62], when viewed from a graph-theoretic perspective.

Maps identifying instances of RDF graphs represent the first stage of what is called *entailment* in RDF terminology [62, 67]. Although not customary, we call G above an *RDF schema graph*. Compared to the method of graph schemas, every edge of the schema graph must be matched by its instances. Thus, RDF schema graphs comprise lower bounds of the instances. As a modeling tool for knowledge, RDF's instantiation highly relies on logical devices. A statement, once present, must not disappear in the course of instantiation. There is, however, the freedom (a) to join or instantiate one or more blank nodes and (b) to add arbitrarily many facts, not covered by the RDF schema graph. Regarding (b), our notion of graph schema conformance entails the same effect. If some subgraph of the database has nothing to do with the schema, it is not respected by conformance. However, RDF's instantiation is not robust w. r. t. incomplete information. If an RDF schema graph states a book to contain information about the title, author, and ISBN, an object, missing only a single attribute, disqualifies as an object of type Book. In this respect, graph schemas are more liberal instead, meeting the requirements of typing unstructured data (cf. Section 2.3.2). Towards (a), it is blank nodes that may be used to implement graph schema types, but instances may grow over their type in an uncontrolled fashion. Graph schemas provide, thus, more information about the possible structures of an entity type. Providing arbitrary, even nonsensical, information for a book instance, e. g., a *death date* or *place of living*, is allowed. There are other correctives to encounter such arbitrary additions. For instance, a *death date* is restricted to objects of type person [17], being outside the

scope of this thesis. Furthermore, in order to obtain a typing of instance nodes in RDF, we had to enumerate all maps between schema and instance, for sure, an intractable task [68]. Here, graph schemas also promise a tractable alternative as (maximal) dual simulations are computed in PTIME [69, 85, 93].

Further entailment regimes, incorporating RDF Schema (RDFS) [24], require a model-theoretic semantics [67, 62]. For sure, such capabilities make RDF more expressive, compared to graph schemas. Any extension to the map-based entailment regime leaves the scope of this thesis. We deal with required structures, the core competence of RDF maps, in the next section.

2.4 Modal Graph Schema

Graph schemas represent upper bounds to their database instances. Presume some graph database DB and graph schema S. A database object o qualifies as type t, $o \vdash_S t$, if o does not exceed the structure t allows. For instance, a single object that participates in relationships labeled *title* and *author* may qualify as a book chapter (INBOOK in Figure 2.13), a journal article (ARTICLE in Figure 2.11 (c)), as well as a book (BOOK in Figure 2.11 (b)). All three graph schemas feature a title and an author. However, does any object with a title and an author qualify as each of the types above? As far as graph schemas are concerned, we have no chance to disqualify one of the types. In order to better recognize a bibliographic item as a journal article, we must additionally include a journal (JOURNAL) and a year of publication (YEAR). For all entry types, BIBTEX has a list of required fields[2]. We wish to express such required properties as part of our graph schema model.

Calvanese et al. extend graph schemas to overcome some limitations, including a form of requirement expressions [32]. They use description logics to complement a single graph schema, but maintain tree-shaped graph data. Furthermore, they extend the labeling alphabet of schemas for more expressiveness. As Abiteboul et al. highlight, a logical system, like Datalog or one of the description logics, always serves as a highly expressive alternative. However, general Datalog programs may capture many more constraints, not necessarily classifying as pure schema information [3]. In order to study schemas for graph data, independent of further constraints, such as keys or dependencies [49, 54, 32], we pursue a model staying inside the graphical formalism we adopted for graph schemas. To this end, Abiteboul et al. sketch *dual graph schemas* [3], that capture required properties, i. e., a core structure to be exhibited by any graph database instance. Since dual graph schemas are again rooted graphs, the expression of required properties is restricted to those reachable from the root nodes.

Example 2.44 An example dual graph schema is depicted in Figure 2.15. It is dual to the rooted graph schema from the beginning of this chapter (cf. Figure 2.7 on Page 18). An instance obeying the schema in Figure 2.7 and its dual schema in Figure 2.15 must not exceed the structure of the graph schema but is required to include objects of both types, *books* as well as *inproceedings*. It is impossible to only conditionally require *title*, *author(s)*, and *ISBN* if a book is present. The same holds for the *inproceedings* case. ∎

We aim for a more flexible model that allows us to formulate graph requirements altogether with allowed structures in an integrated fashion. We achieve the desired flexibility by employing the theory of *Modal Specifications*, initially introduced by Larsen and Thomsen [78]. It has been their goal to have a system specification language capturing the behavior of a variety of implementations [81]. This goal is quite comparable to ours since the graph schemas can be seen as specifications and instances as their implementations.

[2] See the BIBTEX documentation available at `https://ctan.org/pkg/bibtex`.

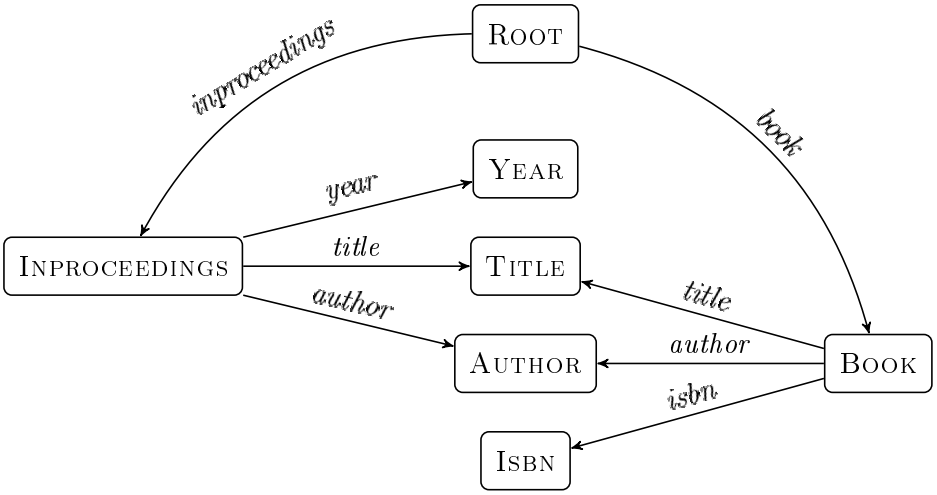


Figure 2.15: A Dual Graph Schema for Bibliographic Contents

Larsen and Thomsen employ a may/must dichotomy, translating to *allowed* and *required* system behavior. We use the same dichotomy in our extended graph schema model, the *modal graph schema*, to capture allowed and required graph structures.

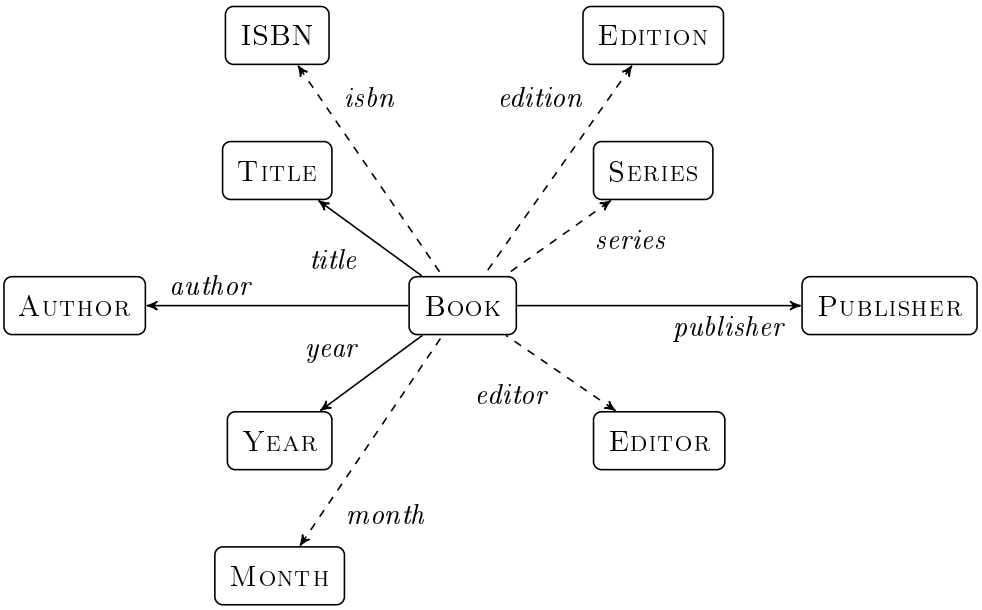
Definition 2.45 (Modal Graph Schema)
$S = (T_S, \Sigma, E_S^{\Diamond}, E_S^{\Box})$ is a *modal graph schema* iff $\mathit{skeleton}(S) := (T_S, \Sigma, E_S^{\Diamond})$ is a graph schema, called the *skeleton of* S, and $E_S^{\Box} \subseteq E_S^{\Diamond}$ (syntactic consistency). ▲

A modal graph schema $S = (T_S, \Sigma, E_S^{\Diamond}, E_S^{\Box})$ comes with two types of edges. First, the *may edges* $E_S^{\Diamond}$ are those describing the allowed structure. As the instances will optionally match them, they are graphically represented by dashed edges, as e. g., in Figure 2.16. The *must edges* $E_S^{\Box}$ are graphically drawn as solid directed edges. The may modality ($\Diamond$) is thought of as expressing the allowed structures in the same way as graph schemas do (Definitions 2.15 and 2.25). The must modality ($\Box$) encodes the required edges. In fact, every graph schema $S = (T_S, \Sigma, E_S)$ is the skeleton of a modal graph schema $(T_S, \Sigma, E_S, \emptyset)$. Syntactic consistency is necessary because the structure can only be required if it is allowed. Other than this mild syntactic requirement, may and must edges are free to be used throughout the whole graph, which is the main syntactic distinction to dual graph schemas (cf. Example 2.44).

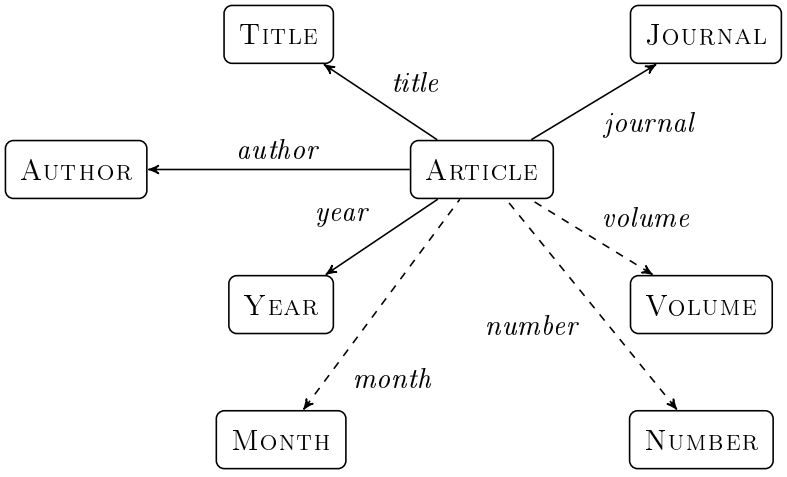
We use $v\ E_S^{\Diamond}(a)\ w$ as the infix notation for $(v, a, w) \in E_S^{\Diamond}$. Accordingly, $v\ E_S^{\Box}(a)\ w$ denotes $(v, a, w) \in E_S^{\Box}$. Conformance between databases and modal graph schemas have to obey the meaning of may and must edges.

Example 2.46 Let us reconsider graph database $DB_{2.11\,(a)}$ in Figure 2.11 (a) on Page 24. According to the non-modal graph schemas in Figures 2.11 (b) and 2.11 (c), objects `SGDM` and `DotW` can be typed, the former as an ARTICLE while the latter is of type BOOK. With respect to modal graph schemas for ARTICLE and BOOK in Figures 2.16 (a) and 2.16 (b), $DB_{2.11\,(a)}$ cannot be typed anymore. In order to get `DotW` typed as a BOOK, its schema requires at least one author and one publisher. Likewise, `SGDM` also misses out on any authors. Thus, modal graph schema conformance shall not qualify $DB_{2.11\,(a)}$ as an instance of either modal schema. ■

While for may edges we follow the same lines as for graph schema conformance of Section 2.3, the must edges of a modal graph schema are simulated by a graph database instance, resulting in an alternating-style of dual simulation.



(a)



(b)

Figure 2.16: (a) Modal Graph Schema for Type Book (b) Modal Graph Schema for Type Article

Definition 2.47 (Modal Graph Schema Conformance)
For modal graph schema $S = (T_S, \Sigma, E_S^{\Diamond}, E_S^{\Box})$ and graph database $DB = (O_{DB}, \Sigma, E_{DB})$, we say that *DB conforms to S*, denoted $DB \preccurlyeq_m S$, iff $DB \preccurlyeq \mathit{skeleton}(S)$ by conformance witness R, such that for $(v_1, v_2) \in R$,

1. $v_2\ E_S^{\Box}(a)\ w_2$ implies $\exists w_1 \in O_{DB} : v_1\ E_{DB}^{a}\ w_1 \wedge (w_1, w_2) \in R$ and
2. $u_2\ E_S^{\Box}(a)\ v_2$ implies $\exists u_1 \in O_{DB} : u_1\ E_{DB}^{a}\ v_1 \wedge (u_1, u_2) \in R$.

Similarly to Definition 2.25, we call R a *modal conformance witness between DB and S.* ▲

By modal graph schema conformance, a schema simulates an instance and an instance simulates the required (must) edges of its schema. The built-in may/must dichotomy implies three different aspects, now covered in a single concise model: allowed, required, as well as forbidden structure. In accordance with non-modal graph schemas, database object classification is performed under the same assumptions as given in Section 2.3.2, i. e., we employ the modal conformance witnesses. Modal conformance witnesses are closed under set-union, making the greatest modal conformance witness between a database and a modal graph schema unique.

Proposition 2.48 *Let DB be a graph database and S a modal graph schema with* $DB \preccurlyeq_m S$. *The greatest modal conformance witness* $\widehat{R}$ *between DB and S, subsuming every other witness, exists. We call* $\widehat{R}$ *the* maximal modal conformance witness between *DB* and *S*.

Proof: Let R_1, R_2 be two modal conformance witnesses between DB and S. We subsequently show that $R_1 \cup R_2$ is a modal conformance witness between DB and S. As R_1 and R_2 are modal conformance witnesses, they are conformance witnesses between DB and $\mathit{skeleton}(S)$, i. e., non-empty dual simulations. Their union is again non-empty. Furthermore, by Theorem 2.29, $R_1 \cup R_2$ is a dual simulation, hence, a conformance witness between DB and $\mathit{skeleton}(S)$. It remains to be shown that the must requirements of Definition 2.47 are met. These are automatically met since $(v_1, v_2) \in R_1 \cup R_2$ implies either $(v_1, v_2) \in R_1$ or $(v_1, v_2) \in R_2$. In both cases, each must edge initiating/terminating in v_1 is reflected by an appropriate edge initiating/terminating in v_2 because R_1 and R_2 are modal conformance witnesses.

Towards a contradiction, suppose there are two distinct greatest modal conformance witnesses $\widehat{R_1}$ and $\widehat{R_2}$, i. e., there is no modal conformance witness R with $\widehat{R_i} \subsetneq R$ $(i = 1, 2)$. As $\widehat{R_1} \cup \widehat{R_2}$ is again a modal conformance witness, $\widehat{R_i} \subseteq \widehat{R_j}$ $(i = 1, 2$ and $i \neq j)$, contradicting the assumption that $\widehat{R_1}$ and $\widehat{R_2}$ are distinct largest witnesses. Q. E. D.

Thus, once again, it is the maximal modal conformance witness that presents to us the typing of database objects w. r. t. a graph schema.

Definition 2.49 (Database Object Classification)
Let $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database and $S = (T_S, \Sigma, E_S^{\Diamond}, E_S^{\Box})$ a modal graph schema. An object $o \in O_{DB}$ *is of type* $t \in T_S$ *(w. r. t. S)* iff $o \vdash_S t$, where $\vdash_S$ is the maximal modal conformance witness between DB and S (cf. Proposition 2.48). ▲

2.4.1 Expressive Power

In analogy to non-modal graph schemas, the *semantics of a modal graph schema* might be defined as the set of its instances. We formulate this definition as a claim, rather than an actual definition because we have not yet studied the properties of our conformance relation $\preccurlyeq_m$.

Claim *The* semantics of a modal graph schema S *is the set of instances of* S*, i. e.,*

$$[\![S]\!]_m := \{DB \mid DB \preccurlyeq_m S\}.$$ ♠

Subsequently, we study the consequences of this semantics of modal graph schemas w. r. t. expressiveness. Recall that the properties of $\preccurlyeq$ led to the definition of the semantics of (non-modal) graph schemas S. Here, we study whether the claimed semantics of modal graph schemas S, $[\![S]\!]_m$, withstands its notions.

First, observe that modal graph schemas are conservative extensions of graph schemas without modalities, in that the skeleton of a modal graph schema is a graph schema. Hence, the set of instances of a modal graph schema is a subset of the instances of its skeleton.

Proposition 2.50 *Let* $S = (T_S, \Sigma, E_S^{\Diamond}, E_S^{\Box})$ *be a modal graph schema.*

(I) $\mathit{skeleton}(S) \in [\![S]\!]_m$ *and*

(II) $[\![S]\!]_m \subseteq [\![\mathit{skeleton}(S)]\!]_{\preccurlyeq}$.

Proof: We show that id_{T_S} is a modal conformance witness for $\mathit{skeleton}(S)$ and S. Of course, id_{T_S} is a non-empty dual simulation between $\mathit{skeleton}(S)$ and $\mathit{skeleton}(S)$ (Proposition 2.33) and, as such, is a conformance witness between $\mathit{skeleton}(S)$ and $\mathit{skeleton}(S)$. It remains to be shown that Definition 2.47 is satisfied. Let $(v_1, v_2) \in \mathsf{id}_{T_S}$, i. e., $\mathsf{id}_{T_S}(v_1) = v_2 = v_1$ with $v_2\ E_S^{\Box}(a)\ w_2$ ($w_2\ E_S^{\Box}(a)\ v_2$, resp.). As $v_2 = v_1$ (by id_{T_S}), there is of course a w_1 with $v_1\ E_S^{\Box}(a)\ w_1$ ($w_1\ E_S^{\Box}(a)\ v_1$, resp.) and $(w_1, w_2) \in \mathsf{id}_{T_S}$ (as $w_1 = w_2$). Hence, id_{T_S} is indeed a modal conformance witness for $\mathit{skeleton}(S)$ and S, which justifies (I).

Towards (II), we show that every instance DB of S, i. e., $DB \preccurlyeq_m S$, is an instance of $\mathit{skeleton}(S)$, i. e., $DB \preccurlyeq \mathit{skeleton}(S)$. There is, by Definition 2.47, a non-empty dual simulation R between DB and $\mathit{skeleton}(S)$. Thus, by Definition 2.25, R is a conformance witness for DB and $\mathit{skeleton}(S)$, implying $DB \preccurlyeq \mathit{skeleton}(S)$. Q. E. D.

In Section 2.3, $\preccurlyeq$ extended to graph schemas nurtured the desired semantics of graph schemas. A modal version of graph schema refinement provides us with a similar extension.

Definition 2.51 (Modal Graph Schema Refinement)
Let $S_1 = (T_1, \Sigma, E_1^{\Diamond}, E_1^{\Box})$ and $S_2 = (T_2, \Sigma, E_2^{\Diamond}, E_2^{\Box})$ be two modal schema graphs. S_1 *refines* S_2, denoted $S_1 \preccurlyeq_m S_2$, iff $\mathit{skeleton}(S_1) \preccurlyeq \mathit{skeleton}(S_1)$ by refinement witness R, such that for all $(v_1, v_2) \in R$,

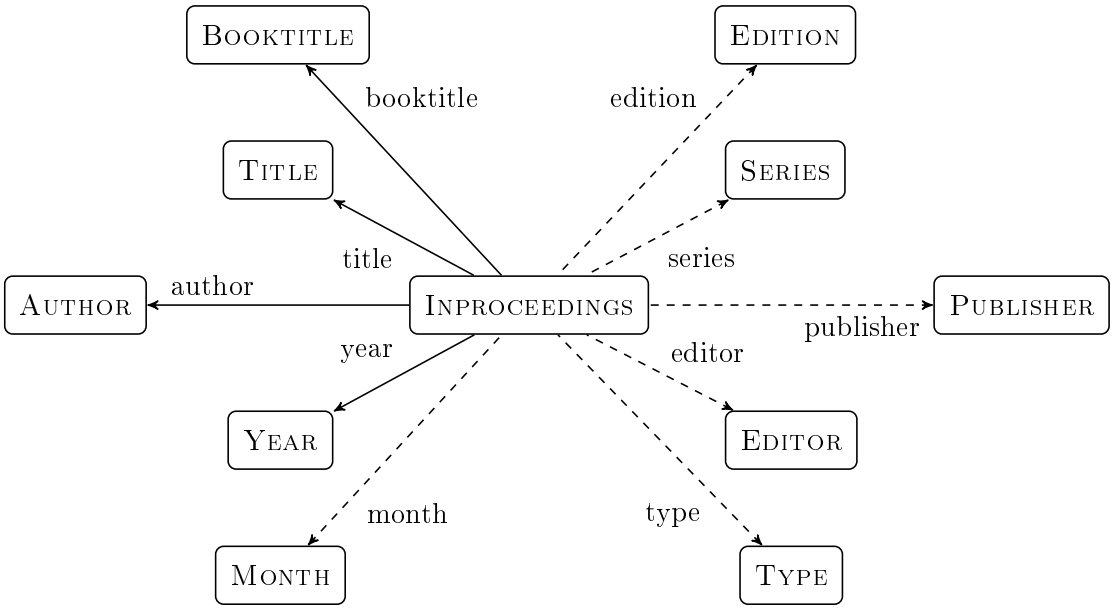
1. if $v_2\ E_2^{\Box}(a)\ w_2$, then there is a $w_1 \in T_1$ with $v_1\ E_1^{\Box}(a)\ w_1$ and $(w_1, w_2) \in R$, and
2. if $u_2\ E_2^{\Box}(a)\ v_2$, then there is a $u_1 \in T_1$ with $u_1\ E_1^{\Box}(a)\ v_1$ and $(u_1, u_2) \in R$.

Analogously to Definition 2.35, we call R a *modal refinement witness.* ▲

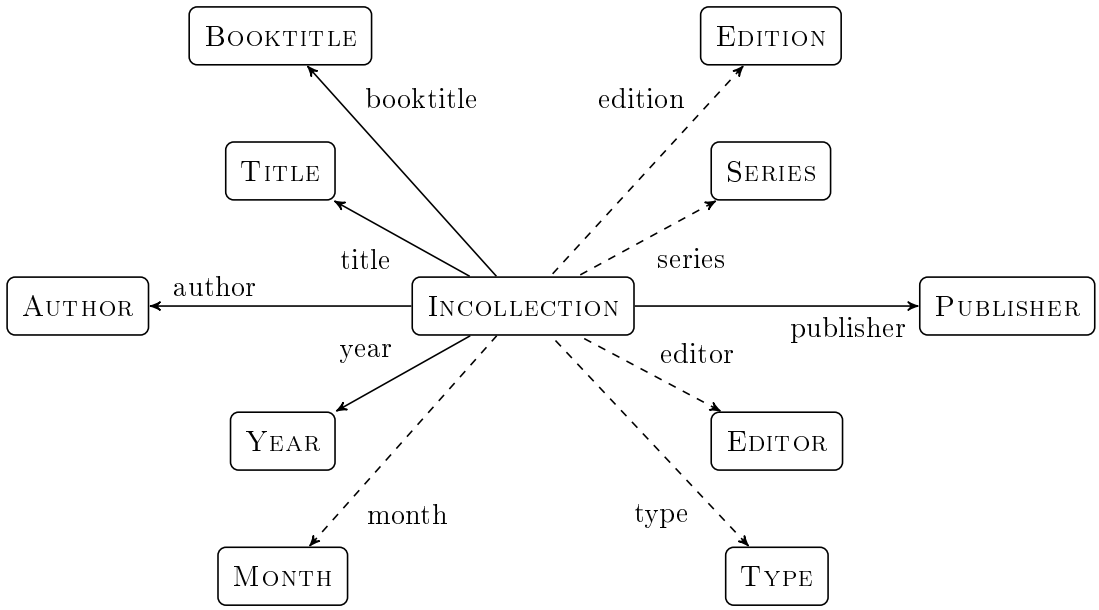
Example 2.52 Consider the two graph schemas $S_{2.17\,(a)}$ and $S_{2.17\,(b)}$ in Figures 2.17 (a) and 2.17 (b). It holds that $S_{2.17\,(b)} \preccurlyeq_m S_{2.17\,(a)}$, but not vice versa. While $S_{2.17\,(a)}$ only allows to mention the Publisher, $S_{2.17\,(b)}$ requires it. Hence, an instance of type Incollection certainly is also an instance of Inproceedings, but not vice versa. ■

The informal outcome of Example 2.52 is grounded in the following soundness result for $\preccurlyeq_m$, analogously to Lemma 2.38.

Lemma 2.53 (Soundness of $\preccurlyeq_m$) *Let* $S_i = (T_i, \Sigma, E_i^{\Diamond}, E_i^{\Box})$ *($i = 1, 2$) be modal graph schemas. If* $S_1 \preccurlyeq_m S_2$*, then every instance* DB *of* S_1 *is an instance of* S_2.



(a)



(b)

Figure 2.17: (a) Modal Graph Schema of Inproceedings (b) Modal Graph Schema of Incollection

PROOF: The proof follows the same principles as performed for Lemma 2.38. We will observe that $\preccurlyeq_m$ is a preorder that may be prolonged to modal instances on the left.

Let $DB \preccurlyeq_m S_1$ by modal conformance witness R_1. Furthermore, $S_1 \preccurlyeq_m S_2$ is witnessed by R_2. Since R_1 and R_2 are dual simulations (cf. Definitions 2.47 and 2.51), $R_1 \circ R_2$ is a dual simulation by Proposition 2.33. We show that

(i) $R_1 \circ R_2$ is non-empty, analogously to Lemma 2.36, and

(ii) $R_1 \circ R_2$ is a modal conformance witness.

Towards a contradicition of (i), assume $R_1 \circ R_2$ is the empty dual simulation. As R_1 is a modal conformance witness and R_2 a modal refinement witness, both dual simulations are non-empty. Hence, $R_1 \circ R_2 = \emptyset$ only if for every pair $(u, v) \in R_1$, there is no matching pair $(v, w) \in R_2$. Let $(u, v) \in R_1$, i. e., there is no $w \in T_2$ with $(v, w) \in R_2$. Since $R_2 \neq \emptyset$, there is at least one pair, say $(v_0, w_0) \in R_2$. Analogously to the proof of Lemma 2.36, we show that every node v_1, reachable from v_0, has a partner w_1 such that $(v_1, w_1) \in R_2$. Thus, $(v, w) \notin R_2$ implies that S_1 is disconnected, contradicting the assumption that S_1 is a modal graph schema.

By induction, we show that for every path π of S_1 with $v_0 = \mathsf{first}(\pi)$ and $v_1 = \mathsf{last}(\pi)$, there is a node $w_1 \in T_2$ with $(v_1, w_1) \in R_2$.

Base: For $|\pi| = 0$, we have $\pi = v_0$ and $(v_0, w_0) \in R_2$ (by assumption).

Hypothesis: Assume for nodes $v_1 \in T_1$, for which a path π from v_0 with $v_1 = \mathsf{last}(\pi)$ and $|\pi| < n$ ($n \in \mathbb{N}$) exists, there is a $w_1 \in T_2$, such that $(v_1, w_1) \in R_2$.

Step: Let π be a path from v_0 to v_1 with $|\pi| = n$. Then $\pi = \pi_{<n} \cdot v_1$ with $|\pi_{<n}| = n-1 < n$. By induction hypothesis, there is a $w' \in T_2$ for $v' = \mathsf{last}(\pi_{<n})$ such that $(v', w') \in R_2$. As π is a path, either (i) $v'\ E_1^{\Diamond}(a)\ v_1$ or (ii) $v_1\ E_1^{\Diamond}(a)\ v'$. As R_2 is a modal refinement witness, there is a $w_1 \in T_2$ with $w'\ E_2^{\Diamond}(a)\ w_1$ ($w_1\ E_2^{\Diamond}(a)\ w'$, resp.) and $(v_1, w_1) \in R_2$.

Hence, there is a $w \in T_2$ with $(v, w) \in R_2$, implying $R_1 \circ R_2 \neq \emptyset$.

Let us now tackle (ii). Since $R_1 \circ R_2$ is a non-empty dual simulation, we show that every pair $(u, w) \in R_1 \circ R_2$ meets the requirements of Definition 2.51, i. e., for an edge $w\ E_{S_2}^{\Box}(a)\ w'$ ($w'\ E_{S_2}^{\Box}(a)\ w$, resp.), we give a $u' \in O_{DB}$ with $u\ E_{DB}^{a}\ u'$ ($u'\ E_{DB}^{a}\ u$, resp.) and $(u', w') \in R_1 \circ R_2$. By construction of $R_1 \circ R_2$, there is a node $v \in T_1$ with $(u, v) \in R_1$ and $(v, w) \in R_2$. As R_2 is a modal refinement witness, there must be a node $v' \in T_1$ with $v\ E_{S_1}^{\Box}(a)\ v'$ ($v'\ E_{S_1}^{\Box}(a)\ v$, resp.) with $(v', w') \in R_2$. As R_1 is a modal conformance witness, there is a node $u' \in O_{DB}$ with $u\ E_{DB}^{a}\ u'$ ($u'\ E_{DB}^{a}\ u$, resp.) with $(u', v') \in R_1$. Thus, from $(u', v') \in R_1$ and $(v', w') \in R_2$, $(u', w') \in R_1 \circ R_2$, which completes the proof. Q. E. D.

Thus, there is also a sound design process for modal graph schemas, similar to the one described for (non-modal) graph schemas (after Lemma 2.38 on Page 29).

Unfortunately, there are modal graph schemas S and T with $[\![S]\!]_m \subseteq [\![T]\!]_m$ but $S \not\preccurlyeq_m T$. This is a well-known problem of *modal refinement* and every refinement notion fixing this problem yields at least a coNP-hard refinement problem [80], as opposed to the PTIME complexity of checking graph schema conformance and refinement [69, 86, 93].

Example 2.54 The modal graph schemas depicted in Fig. 2.18 are adapted versions of the counterexample given by Larsen et al. [80] since the original counterexample cannot cope with the dual simulation aspect of our modal refinement. Let $S_{(a)}$ be a modal schema according to Figure 2.18 (a) and $S_{(b)}$ to Figure 2.18 (b). Of course $S_{(a)} \not\preccurlyeq_m S_{(b)}$ since in any modal refinement witness R, $(n, m) \in R$. But $mE_{S_{(b)}}^{\Box}(b) \neq \emptyset$, whereas n has only a b-labeled may edge, i. e., m requires b whereas n only allows it. Nevertheless, it holds

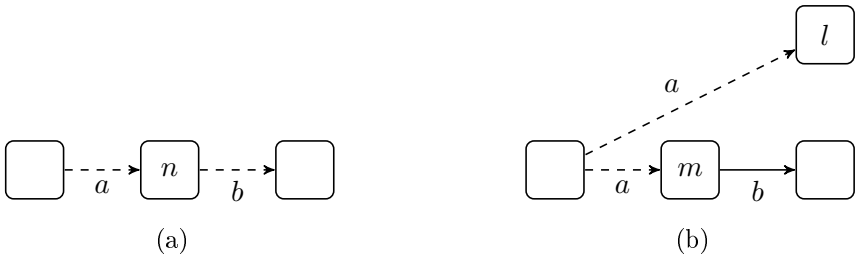


Figure 2.18: Counterexample Completeness of Modal Refinement, adapted from [80]

that $[\![S_{(a)}]\!]_m \subseteq [\![S_{(b)}]\!]_m$ because for databases DB conforming to $S_{(a)}$ that do not include a b-labeled edge, DB also conforms to $S_{(b)}$ by exploiting the non-determinism of the edges labeled by a. Node l perfectly simulates every node with an incoming a-labeled edge but a missing outgoing b-labeled edge. ■

Thus, if $S \not\preccurlyeq_m T$ for modal graph schemas S and T, this does not necessarily imply $[\![S]\!]_m \not\subseteq [\![T]\!]_m$. In other words, $[\![S]\!]_m$ for modal graph schemas S is not the set of all instances of S, at least not up to $\preccurlyeq_m$.

2.4.2 Sources of Nondeterminism

The counterexample given in Example 2.54 exploits nondeterminism of $S_{(b)}$, as it contains two distinct paths accounting for graph database instances with an a-labeled edge. The one ending in l does not need to be completed by a b-labeled edge. The one ending in m simulates a-b-sequences. Even if we excluded this example, we would still miss an important source of nondeterministic structure.

Example 2.55 Consider the two modal graph schemas $S_{(a)}$ (Figure 2.19 (a)) and $S_{(b)}$ (Figure 2.19 (b)). Of course, $S_{(a)} \not\preccurlyeq_m S_{(b)}$ because every modal refinement witness R would have to include (v, y) (and (w, z)), but y requires an outgoing b-labeled edge, which is only allowed by v. However, every instance of $S_{(a)}$ has either no edges (only isolated database objects) or at least one b-edge. Both types of instances are also supported by $S_{(b)}$ because isolated nodes are typed by x and b-edges are covered by nodes y and z. Additionally, $S_{(b)}$ allows for a-edges followed by b-edges. Nevertheless, $[\![S_{(a)}]\!]_m \subseteq [\![S_{(b)}]\!]_m$ but $S_{(a)} \not\preccurlyeq_m S_{(b)}$. ■

The fundamental problem in this example is the freedom of choosing the types of $S_{(b)}$, that participate in the modal conformance witnesses. In a database with isolated nodes but no b-edges, type x is the preferred type, while in databases with at least one b-edge, type y fills in. If we find a way to uncover this exchange of rôles in-between types of graph schemas, we will obtain the semantics of modal graph schemas.

2.4.3 Semantics of Deterministic Modal Graph Schemas

In Example 2.55, if we fixed y as mandatory type in $S_{(b)}$, every instance of $S_{(b)}$ would have contained at least a b-labeled edge. In this setting, y plays the rôle of a *root node*. Unfortunately, as discussed in Section 2.3.4, reintegrating root nodes on the schema-side does not entail a sound and complete theory of graph schemas. However, if we only observe

Figure 2.19: Deterministic Counterexample for Completeness of Modal Refinement

for y the set of database objects it types by $\vdash_{S_{(b)}}$, we evade Example 2.43 on Page 31 as well as Example 2.55.

Example 2.56 Let us reconsider $S_{(a)}$ and $S_{(b)}$, as in Example 2.55, but, this time, let us fix type v in $S_{(a)}$ and type y in $S_{(b)}$. An empty-structured graph database $DB = (O_{DB}, \Sigma, \emptyset)$ with $O_{DB} \neq \emptyset$ is an instance of $S_{(a)}$. The maximal modal conformance witness between DB and $S_{(a)}$ assigns to type v all the database objects. In contrast, although DB is an instance of $S_{(b)}$, there is not a single database object conforming to type y. As there is no modal refinement witness between $S_{(a)}$ and $S_{(b)}$, that relates types v and y, we will again find that $S_{(a)} \not\preccurlyeq_m S_{(b)}$. When comparing the instances of $S_{(a)}$ and $S_{(b)}$, we will now consider the induced typing of v and y, which distinguishes the sets of instances in the desired manner. It is exactly this distinction we subsequently exploit to derive a sound and complete characterization of the semantics of modal graph schemas. ■

Fixing a single type (of interest) per modal graph schema will ensure the absence of the first kind of nondeterminism (cf. Example 2.55). In order to avoid the second kind (cf. Example 2.54), a *deterministic modal graph schema* additionally exploits a *deterministic edge relation* $E_S^{\Diamond}$. This means, for every two edges $(v, a, w_1), (v, a, w_2) \in E_S^{\Diamond}$ (or, $(w_1, a, v), (w_2, a, v) \in E_S^{\Diamond}$, resp.), it holds that $w_1 = w_2$.

Definition 2.57 (Deterministic Modal Graph Schema)
A quintuple $S[x] = (T_S, \Sigma, E_S^{\Diamond}, E_S^{\Box}, x)$ is a *deterministic modal graph schema* iff $S = (T_S, \Sigma, E_S^{\Diamond}, E_S^{\Box})$ is a modal graph schema with $E_S^{\Diamond}$ is deterministic and $x \in T_S$. A graph database DB is an *instance of* $S[x]$ iff $DB \preccurlyeq_m S$. The set of all objects $o \in O_{DB}$ with $o \vdash_S x$ is denoted by $S[x](DB)$.

Let $S'[y]$ be another deterministic modal graph schema. $S[x]$ *modally refines* $S'[y]$, denoted $S[x] \preccurlyeq_m S'[y]$, iff $S \preccurlyeq_m S'$ by modal refinement witness R with $(x, y) \in R$. ▲

Example 2.58 Besides the rooted graph schema in Figure 2.7 (on Page 18), all graph schemas, with and without modality, are deterministic. As exemplified by the rooted graph schema, whenever we join two or more of the other schemas (Figures 2.11 (b), 2.11 (c), 2.13, 2.16 (a), 2.16 (b), 2.17 (a) and 2.17 (b)), the resulting (modal) graph schemas are connected but nondeterministic as, for instance, all of them feature types TITLE, AUTHOR, and YEAR. The joined schemas exhibit a nondeterministic structure in these types, as there is more than one (backward) edge, labeled *title*, *author*, and *year*, leading to different types within the schema. ■

As we merely restricted our scope of modal graph schemas to the deterministic case and kept the notion of instances invariant, the database instances $DB \preccurlyeq_m S[x]$ are instances of $T[y]$, given that $S[x] \preccurlyeq_m T[y]$. Furthermore, even the induced typing of x and y is guaranteed to subsume one another.

Lemma 2.59 (Soundness of $\preccurlyeq_m$ – The Deterministic Case)
Let $S_1[x]$ and $S_2[y]$ be deterministic modal graph schemas. If $S_1[x] \preccurlyeq_m S_2[y]$, then $DB \preccurlyeq_m S_1[x]$ implies $DB \preccurlyeq_m S_2[y]$ and $S_1[x](DB) \subseteq S_2[y](DB)$.

PROOF: Since $S_1[x] \preccurlyeq_m S_2[y]$ reduces to $S_1 \preccurlyeq_m S_2$ (cf. Definition 2.57), every instance of $S_1[x]$ is an instance of $S_2[y]$ by Lemma 2.53. It remains to be shown that for every instance DB of $S_1[x]$, $S_1[x](DB) \subseteq S_2[y](DB)$. Let $DB \preccurlyeq_m S_1[x]$ and $o \vdash_{S_1[x]} x$. We need to show that $o \vdash_{S_2[y]} y$. Recall that $\vdash_{S_1[x]}$ is the maximal conformance witness between DB and $S_1[x]$ (cf. Proposition 2.48), i.e., $\vdash_{S_1[x]}$ is a non-empty dual simulation. As $S_1[x] \preccurlyeq_m S_2[y]$, there is a refinement witness R between $S_1[x]$ and $S_2[y]$, such that $(x, y) \in R$ (cf. Definition 2.57). Hence, following the proof of Lemma 2.53, $\vdash_{S_1[x]} \circ R$ is a non-empty dual simulation between DB and $S_2[y]$. Thus, $\vdash_{S_1[x]} \circ R$ is contained in the maximal conformance witness $\vdash_{S_2[y]}$ between DB and $S_2[y]$ and, therefore, $o \vdash_{S_2[y]} y$. Q. E. D.

Additionally to the set of instances per schema, observing the induced typing of the nodes of interest resolves the issue presented in Section 2.3.4.

Lemma 2.60 (Completeness of $\preccurlyeq_m$ — The Deterministic Case)
Let $S_1[x]$ and $S_2[y]$ be deterministic modal graph schemas. If for every instance $DB \preccurlyeq_m S_1[x]$, it holds that $DB \preccurlyeq_m S_2[y]$ and $S_1[x](DB) \subseteq S_2[y](DB)$, then $S_1[x] \preccurlyeq_m S_2[y]$.

PROOF: By Proposition 2.50, $skeleton(S_1) \preccurlyeq_m S_1$. Therefore, $skeleton(S_1) \preccurlyeq_m S_1[x]$ with $x \in S_1[x](skeleton(S_1))$ (the proof of Proposition 2.50 uses $\mathsf{id}_{T_{S_1}}$ as conformance witness). As every instance of $S_1[x]$ is an instance of $S_2[y]$,

$$skeleton(S_1) \preccurlyeq_m S_2[y] \text{ and } x \in S_2[y](skeleton(S_1)). \tag{2.5}$$

The second part of (2.5) is guided by the assumption that $S_1[x](DB) \subseteq S_2[y](DB)$ for every instance DB of $S_1[x]$.

Towards a contradiction, suppose $S_1[x] \not\preccurlyeq_m S_2[y]$, i. e., there is no modal refinement witness R between S_1 and S_2 with $(x, y) \in R$. By this assumption and (2.5), there are paths (of minimal length) $\pi_1 = t_0 t_1 t_2 \dots t_k$ of S_1 and $\pi_2 = u_0 u_1 u_2 \dots u_k$ of S_2 ($k \geq 0$), such that

(i) $t_0 = x$ and $u_0 = y$,

(ii) $t_{i-1}\ E_2^{\Diamond}(a_i)\ t_i$ and $u_{i-1}\ E_2^{\Diamond}(a_i)\ u_i$ $(0 < i \leq k)$, but

(iii) either $u_k\ E_2^{\Box}(a)\ u$ and $t_k E_1^{\Box}(a) = \emptyset$, or $u\ E_2^{\Box}(a)\ u_k$ and $E_1^{\Box}(a) t_k = \emptyset$.

Construct $DB_{\lightning}$ from π_1 as the smallest instance of $S_1[x]$ that contains all the edges (t_{i-1}, a_i, t_i) used in (ii). It holds that $DB_{\lightning}$ contains the path π_1 with all its nodes, from x to t_k, with $x \in S_1[x](DB_{\lightning})$, but neither an incoming a-edge to nor an outgoing a-edge from t_k. Note, $DB_{\lightning}$ may contain more edges than the ones in (ii) to guarantee edges to be preserved as they are must edges, incident to any of the t_i $(0 \leq i \leq k)$.

Because $S_1[x](DB_{\lightning}) \subseteq S_2[y](DB_{\lightning})$, the maximal modal conformance witness $\widehat{R}$ between $DB_{\lightning}$ and $S_2[y]$ has the pair (x, y) as an element. Because $S_2[y]$ is deterministic, the only path of S_2, dual simulating π_1, is π_2. But $(t_k, u_k) \notin \widehat{R}$ because u_k requires an incoming/outgoing a-edge not present in $DB_{\lightning}$. Each prefix of π_1 cannot be dual simulated by a prefix of π_2. Hence, $(x, y) \notin \widehat{R}$ which contradicts the assumption that $S_1[x](DB) \subseteq S_2[y](DB)$ for all $DB \preccurlyeq_m S_1[x]$.

Hence, $S_1[x] \preccurlyeq_m S_2[y]$, given that all instances DB of $S_1[x]$ are instances of $S_2[y]$ and $S_1[x](DB) \subseteq S_2[y](DB)$. Q. E. D.

The proof only uses the fact that $S_2[y]$ is deterministic in $S_1[x] \preccurlyeq_m S_2[y]$. Although this fact allows for more general subsumptions, the equivalence of two modal schemas requires both schemas to be deterministic.

Consequently, the semantics of a deterministic modal graph schema $S[x]$ is not only its set of instances but also the induced typing of x. We, thereby, withdraw the claim at the beginning of Section 2.4.1 (Page 37) and conclude the semantics of deterministic modal graph schemas to be defined as follows.

Definition 2.61 (Semantics of Deterministic Modal Graph Schema)
Let $S[x], T[y]$ be a deterministic modal graph schema. The *semantics of* $S[x]$ is defined by $[\![S[x]]\!]_m := \{(DB, S[x](DB)) \mid DB \preccurlyeq_m S[x]\}$.

$[\![S[x]]\!]_m$ is subsumed by $[\![T[y]]\!]_m$, denoted $[\![S[x]]\!]_m \sqsubseteq [\![T[y]]\!]_m$, iff for all $(DB, X) \in [\![S[x]]\!]_m$, a $(DB, Y) \in [\![T[y]]\!]_m$ exists with $X \subseteq Y$.

$[\![S[x]]\!]_m$ is equivalent to $[\![T[y]]\!]_m$, denoted $[\![S[x]]\!]_m \equiv [\![T[y]]\!]_m$, iff $[\![S[x]]\!]_m \sqsubseteq [\![T[y]]\!]_m$ and $[\![T[y]]\!]_m \sqsubseteq [\![S[x]]\!]_m$. ▲

As promised at the end of Section 2.3.4, we now obtained a feasible solution for the issues discovered with nodes of interest, at least for the deterministic case. Contrarily to the refinement notion sketched there, we do not enforce a typing of the variables of interest, but use them as an additional information source to derive the semantics of modal graph schemas. With regard to Example 2.43 (Page 31), it holds that $[\![S[x]]\!]_m \not\sqsubseteq [\![S[y]]\!]_m$ because, although the sketched instance types of $S[x]$ are also instance types of $S[y]$, x and y fulfill completely different rôles as verified by the absence of a non-empty dual simulation relating x and y.

Theorem 2.62 *Let $S_1[x]$ and $S_2[y]$ be deterministic modal graph schemas. $[\![S_1[x]]\!]_m \sqsubseteq [\![S_2[y]]\!]_m$ iff $S_1[x] \preccurlyeq_m S_2[y]$. Consequently, $[\![S_1[x]]\!]_m \equiv [\![S_2[y]]\!]_m$ iff $S_1[x] \preccurlyeq_m S_2[y]$ and $S_2[y] \preccurlyeq_m S_1[x]$.*

PROOF: The characterization of subsumption ($\sqsubseteq$) follows directly from Lemmas 2.59 and 2.60. Equivalence ($\equiv$) is defined by two subsumptions (cf. Definition 2.61). Q. E. D.

2.5 Summary

Throughout this chapter, we have learned about basic notions of graph structures, such as unlabeled and labeled graphs, as well as some elementary techniques relating graphs. These notions included graph homomorphisms, (sub-)graph isomorphisms, and (dual) simulations. To bring data into graphs, we roamed through the structural components of RDF and derived a useful model for *graph databases*. Although graph data is usually schemaless, some insights into the stored structure may have advantages for query formulation and query processing [28, 29, 57, 3, 21, 127]. We defined graph schemas and a sensible instance notion that even allowed us to put an order on graph schemas. This study culminated in the semantics of graph schemas, given as the set of instances they describe.

Our graph data model is a labeled graph, relating data objects from a universe of data objects $\mathcal{U}$ with other data objects. The relationships expressed in this model stem from a universe of predicate symbols $\mathcal{P}$. Data objects are the nodes of a graph database, while the edges represent the properties associated with the objects. We may generally model all kinds of relationships, but they must be explicitly given. We exclude entailment regimes and other deductive devices from our graph data model but stay within the algebraic framework of labeled graphs.

A graph schema is a connected labeled graph with at least one node. The nodes of a graph schema represent types, later assigned to the database objects of the instances of the schema. The relationships of a graph schema form an upper bound of the relations occurring in an instance. Thereby, we meet the three characteristics of *typing graph data* (cf. Section 2.3.2), being (a) lack of precision (types do not partition the set of database objects), (b) incompleteness (some objects do not belong to any type), and (c) approximation (not all structural aspects of a type must be met). The notion of *conformance witnesses*, given as non-empty dual simulations, allowed us to derive a concise means of object classification. Furthermore, we provide a formal semantics of graph schemas up to dual simulations. It holds that two graph schemas have the same set of instances iff they (non-trivially) dual simulate one another (cf. Theorem 2.40).

The notable restriction of our model is the connectedness of graph schemas, which is a necessary design decision that allowed us to prove that graph schema refinement ($\preccurlyeq$) is a preorder, a prerequisite for a sound and complete method to decide graph schema equivalence. Hence, when we join the example graph schemas that appeared in Section 2.3 to one single graph, we have to ensure connectedness. We cannot simply combine the schemas of books and journal articles (Figures 2.11 (b) and 2.11 (c)) separating the types AUTHOR, TITLE, YEAR, and MONTH. This way, we would reach a schema that describes

books and articles with their respective authors. An author who wrote a book and an article must occur as two separate objects in an instance of such a schema, one for each rôle, e. g., as *book author* and as *journal author*. In most cases, though, such a schema is undesirable, as it does not capture any interrelations of books and journal articles. A graph database conforming to such a disconnected schema may very well be separated into two, as there is no answer to a query that asks for some relation between books and articles. Although we cannot completely exclude a use case for disconnected graph schemas, this example at least questions the usefulness of such a model of graph schemas. If a graph database is an instance of more than one graph schema, conformance can be checked schema-wise.

Algorithms verifying conformance between a graph database and a graph schema are available and described in Chapter 5. The core problem to be solved is the *non-emptiness problem of dual simulations*, asking for the existence of a non-empty dual simulation. This problem is PTIME-solvable by HHK by Henzinger et al. [69], GPP by Gentilini et al. [60] (later corrected by van Glabbeek and Ploeger [132]), or our own solution [93]. Especially the development of fast decision procedures, such as HHK running in cubic time (in the size of the input graphs), paved the way for simulation concepts to be used in the context of early graph databases [28, 3]. Each of the algorithms mentioned above primarily compute the maximal (dual) simulation between two given graphs. Thus, there is a non-empty (dual) simulation if the computed maximal (dual) simulation is non-empty (Corollary 2.32). This solution to the dual simulation problem decides graph schema conformance and directly returns the typing relation $\vdash_S$ between the objects of a database DB and the types in graph schema S.

The way we derived the semantics of graph schemas up to graph schema refinement ($\preccurlyeq$) in Section 2.3.3 may be adopted to other notions relating graph schemas. Given such a relation $\sqsubseteq$ between graph schemas, we would have to answer, whether the instances of a graph schema S are preserved in T if $S \sqsubseteq T$. Moreover, we would be asked if this information about the instances is sufficient, in the sense of completeness. Finally, a sensible *semantics up to* $\sqsubseteq$ *of a graph schema* S, $[\![S]\!]_\sqsubseteq$, would have to be obtained. This semantics would feature the property we have proven for $\preccurlyeq$ in Theorem 2.40, i. e., $[\![S]\!]_\sqsubseteq \subseteq [\![T]\!]_\sqsubseteq$ iff $S \sqsubseteq T$. Adjustments to the models of graph schema and conformance are imaginable. We have seen one such study (beyond graph schema conformance) in Section 2.4 when we studied the modal extension of graph schemas.

Schema graphs, here and 20 years ago, provide a means of allowed structures, but it is sometimes necessary to also require structure by a schema [3]. To address this incapability of graph schemas, we proposed a flexible incorporation of required structures employing modal specifications [78, 81]. The may/must dichotomy, employed by Larsen and Thomsen [78], translates to *allowed* and *required* system behavior, which we reinterpret as allowed and required graph structure in *modal graph schemas*. We presented modal versions of conformance as well as refinement, being conservative extensions of graph schema conformance and refinement. Unfortunately, the additional requirements, emerging from the must modality, have severe consequences for refinement as well as for the implied algorithmic complexity. It was only the restriction to deterministic modal graph schemas allowing for a sound and complete characterization of graph schema subsumption.

The extension of graph schemas to graph databases, as well as by modalities, has been published as a full paper in the proceedings of the 38^{th} International Conference on Conceptual Modeling (ER 2019) [89]. Sections 2.3 and 2.4 provide more explanation, full proofs, and more detailed examples than the conference version could contain, given the limited space.

CHAPTER 3

Graph Patterns

Throughout the last chapter, we have cared for the representation and the description of graph data. After basic notions, we rediscovered a model describing the graph data's schematic structure. Hypothetically, a system that builds on the principles of Chapter 2 is a system that allows for describing, storing, and viewing graph data. By the semantics we developed for graph schemas, which is based on graph (dual) simulations, the system automatically (and efficiently) identifies typing information for database objects stored in our database instance. Any further manipulation takes place by posing graph patterns and finding those subgraphs of the database that *match the pattern* [59, 50].

Interest in graph patterns for graph querying tasks has been raised by requirements of application domains, like the Semantic Web [8, 14], social network analysis [26, 48], or modern Web-based techniques [41]. The volume of such data sources vastly increased over the last decade, e. g., Wikidata has grown from more than 45 Million entities in 2018 [87] to 64,055,544 items, as of today[1]. Graph homomorphisms or graph isomorphisms form the quasi-standard for the *graph pattern matching problem* [50]. Alternatives are only rarely spotted, e. g., [26, 51, 85]. According to Gallagher [59], the *graph pattern matching problem* gets a *data graph* $G = (V, \Sigma, E)$ and a *pattern graph* $Q = (V_Q, \Sigma, E_Q)$ as input. Matching requires finding the set $[\![Q]\!]_G$ of all subgraphs of G that are *matches for* Q. Instantiated for graph isomorphisms, a subgraph $A \subseteq G$ is an *isomorphic match for* Q *in* G iff there is a graph isomorphism between Q and A.

Matching up to graph isomorphisms often suffers from a bad reputation regarding its complexity. The infamous *subgraph isomorphism problem* is among Cook's NP-complete problems, for which he provides a reduction from 3SAT [37]. The situation is no better with subgraph homomorphisms as the decision problem behind it remains NP-complete [68]. According to Fan et al., (bijective) functions are often too strict [51], by which they mean to say that in real-world graph data, a single graph pattern does not suffice to satisfy the information need of a user or application. Graph isomorphisms do not cope well with incompleteness of data, i. e., the exactness required by isomorphisms clashes with heterogeneity in graph databases. Another reason is that graph isomorphisms do not allow for iterations.

Example 3.1 Assume, we want to identify *citation rings*[2] in our bibliographic database. The presence of a directed cycle characterizes such a ring, exemplified for two, three, and four research works in Figure 3.1. If we knew the number of participants k in every citation ring in the graph database in advance, we would just pick one of the respective

[1] statistics retrieved on Oct 22, 2019, from `https://www.wikidata.org/wiki/Wikidata:Statistics`

[2] This example is adopted and adjusted from the examples used by Fan et al. [51] and Ma et al. [85].

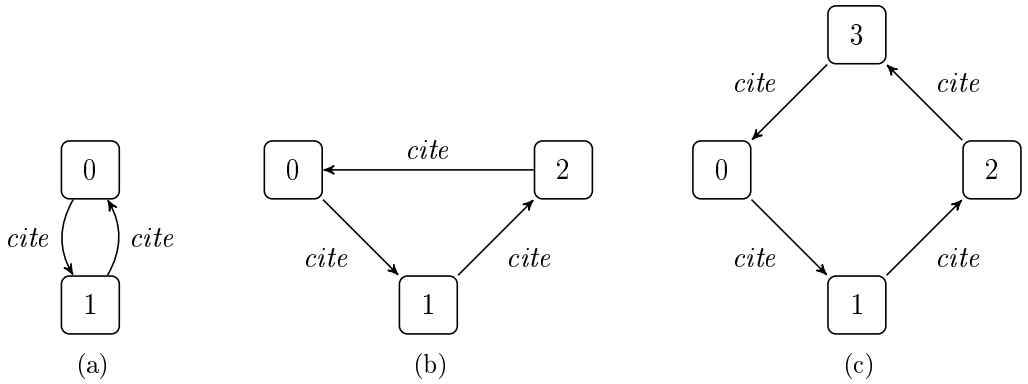


Figure 3.1: Three Non-Isomorphic Citation Ring Patterns

ring patterns, e. g., Figure 3.1 (b) for $k = 3$. Usually, more than one such k exists, which is why we need to pose several ring patterns and collect all their results. It is impossible for graph-isomorphic matching to find cycles of some finite length $k \in \mathbb{N}$. Even if this maximal number is known, a possible query, formulated in SPARQL, is rather complex:

$$\underbrace{((0, \mathit{cite}, 1)\ \texttt{AND}\ (1, \mathit{cite}, 0))}_{\text{Figure 3.1 (a)}}\ \texttt{UNION}\ \underbrace{((0, \mathit{cite}, 1)\ \texttt{AND}\ (1, \mathit{cite}, 2)\ \texttt{AND}\ (2, \mathit{cite}, 0))}_{\text{Figure 3.1 (b)}}$$
$$\texttt{UNION}\ \underbrace{((0, \mathit{cite}, 1)\ \texttt{AND}\ (1, \mathit{cite}, 2)\ \texttt{AND}\ (2, \mathit{cite}, 3)\ \texttt{AND}\ (3, \mathit{cite}, 0))}_{\text{Figure 3.1 (c)}}$$

■

For sure, iterating through data graphs may be done by *path queries*, which are built into many modern graph query languages [5, 83], but the general computational problems behind answering path queries often exceed NP-completeness of graph-isomorphic pattern matching [19]. What if a single graph pattern already describes the essence of a citation ring?

Another issue of graph isomorphisms reveals itself by considering graph schema conformance and object classification from Section 2.3.2. If we had chosen graph isomorphisms instead of (dual) simulations, the conformance check would have immediately turned NP-complete. Furthermore, in order to obtain object classifications, there is no more elegant way than enumerating all graph isomorphisms between (subgraphs) of the database and the graph schema. There is no union-closedness result for graph isomorphisms. The just described issue also has a graph querying-related correspondence.

Example 3.2 Assume we want to find the papers written by *Turing Award* winners. Up to any relevant notion, a single pattern as the one depicted in Figure 3.2 suffices. However, depending on the number of adjacent paper nodes, a single Turing Award winner

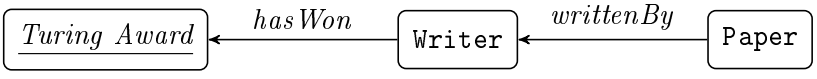


Figure 3.2: A Graph Pattern Asking for the Paper(s) Written by Turing Award Winners

is represented in as many matches as there are papers written by her, when considering matching up to graph isomorphisms. For instance, this yields at least 373 matches[3] having a node `Michael Stonebraker` as `Writer`. Would not a single match, collecting all the written papers in a bag, suffice? ■

[3]On October 24, 2019, `https://dblp.uni-trier.de/pers/hd/s/Stonebraker:Michael` lists 373 records.

Summarizing, the most prominent issue of graph homomorphisms in the literature is intractability. Also, Sahu et al. certify general scalability of graph database techniques to be the most pressing issue, likewise by researchers and practitioners [118].

Goals. Beyond intractability, we argue for a low *pragmatic value* of graph homomorphisms: A single pattern is often insufficient. Hence, full-fledged graph query languages are developed around graph homomorphisms. Loosely speaking, graph homomorphisms are early adopters of external operations like joins or unions. Simulations, we became familiar with during Chapter 2, resolve some of the issues of graph homomorphisms mentioned above. Since simulations are union-closed relations between the nodes of the pattern and those of the data graph (cf. Theorem 2.29), they allow for summarization of single matches. Of course, they have been proposed as tractable alternatives to otherwise intractable matching tasks based on graph homomorphisms [51, 85, 53]. However, there are some cases where the pragmatic value of simulations is at least mentioned [26, 51]. Uniqueness of the maximal dual simulation already aided the methods of graph schema conformance and object classification (cf. Sections 2.3.1 and 2.3.2). Furthermore, the citation network task requires only a single pattern, e. g., the one depicted in Figure 3.1 (a), to cope with citation rings with arbitrarily many participants. The reason is that all three graph patterns in Figure 3.1 are equivalent up to dual similarity. Remarkably, one finite pattern allows for expressing a class of infinitely many graph patterns by employing graph similarity principles in the pattern matching task.

Contribution. We contribute an exercise in comparative semantics for graph pattern matching in graph databases. In a series of pattern matching mechanisms, such as graph homomorphisms and several forms of graph simulation, we outline their landscape and elaborate on their uses in concrete scenarios. Therefore, we first study graph equivalences ($\simeq$, $\leftrightarrows_{\mathsf{D}}$, and $\leftrightarrow_{\mathsf{D}}$) to estimate the degrees of freedom we have in the matching process. Knowing about what graphs are indistinguishable by a matching mechanism reflects on the matches we can expect. Of course, we lose precision w. r. t. graph isomorphisms in the pattern matching task but, at the same time, gain tractability (cf. Chapter 5) and the ability to describe infinitely many patterns finitely and without the machinery of path queries and the like. Furthermore, simulations show all the peculiarities already discussed in Chapter 2, e. g.,*leaf node insensitiveness* (cf. Example 2.21), leading to an early adoption of dual simulation principles. From a formal perspective, bisimilarity ($\leftrightarrows$) and similarity ($\leftrightarrow$) lose their status as equivalences in the general graph setting. Only the simultaneous consideration of backward and forward edges reestablishes rightful equivalence notions for (connected) graphs via dual bisimilarity ($\leftrightarrows_{\mathsf{D}}$) and dual similarity ($\leftrightarrow_{\mathsf{D}}$).

We briefly discuss instances of subgraph matching problems, based on the just established equivalence notions. Unfortunately, whenever we ask for subgraphs as solutions, the associated matching problem is NP-complete. Loosening the relationship between graph pattern and data graph will guide us from intractable graph-homomorphic matching to dual simulation pattern matching, which inherits all properties we studied for dual similarity and overcomes intractability of subgraph dual similarity matching.

Besides all the advertised characteristics of dual simulations, they are at least as bad at recognizing incompleteness of data as graph isomorphisms. To partially overcome this issue, we borrow ideas from another semantic equivalence, that has been used to characterize (partial) deadlock situations in processes. We pick the idea of process failures [25, 133] and devise them for graph pattern matching incorporating a light-weight means of negation. Therefore, we contribute a complementary *failures theory* and exemplify it as an extension to subgraph matching up to dual simulations ($\sqsubseteq_{\mathsf{Dsim}}$).

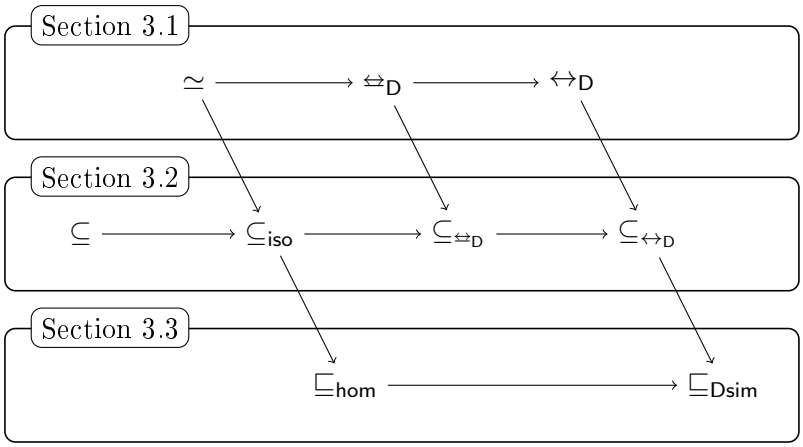


Figure 3.3: The Graph Matching Notions and Their Interrelation

We published an earlier comparative study of graph matching notions in the proceedings of the 36th International Conference of Conceptual Modeling [90]. Back then, we included trace-based equivalences and studied the incomparability of simulations and failures equivalence. We made the failures theory more concise in a special interest article that appeared in the Datenbankspektrum [91]. In this chapter, we collect and combine our earlier findings with related work to obtain a discussion of the pragmatic value of different graph pattern matching mechanisms. Compared to [90, 91], we also include bisimilarity and argue for the early adoption of dual simulation principles.

Outline. The study of graph equivalences in Section 3.1, comprising graph isomorphisms, similarity, and bisimilarity, enables us to discuss what a single graph pattern expresses. Intuitively, two equivalent graph patterns return the same matches. Based on these equivalences, we discuss the particular subgraph matching problems in Section 3.2. In Section 3.3, our focus will be on matching graph patterns without the need of a subgraph mechanism. The *failures theory for graph databases* is developed and applied in Section 3.4. Section 3.5 summarizes the material of this section by justifying Figure 3.3.

3.1 Cornerstones

In this section, we consider different relations between graphs. Hence, we usually pose *connected* graphs $G_1 = (V_1, \Sigma, E_1)$ and $G_2 = (V_2, \Sigma, E_2)$ and ask whether or not they are related. Some of these relations are equivalences. Requiring connectedness, at least for the graph pattern (later G_1), is a usual assumption in graph pattern matching [59, 48]. The most discriminating form of equivalence is *identity*. This kind of equality, i. e., $G_1 = G_2$, only holds if $V_1 = V_2$ and $E_1 = E_2$. By *most discriminating* we mean that any other relation/equivalence of this section is coarser than identity, i. e., two identical graphs will always be related/equivalent up to some other notion in this section. The relations are binary relations over a virtual universe of graphs. Recall, an equivalence $\equiv$ over graphs is reflexive (i. e., $G \equiv G$ for all graphs G), symmetric (i. e., $G \equiv H$ implies $H \equiv G$ for all graphs G and H), and transitive (i. e., $G_1 \equiv G_2$ and $G_2 \equiv G_3$ imply $G_1 \equiv G_3$ for all graphs G_1, G_2, G_3).

3.1.1 Graph Isomorphisms

The first relation we discuss is an equivalence relation, namely *equivalence up to graph isomorphism*. We already introduced basic graph morphisms formally in Definition 2.8,

including *graph isomorphisms*, that were bijective homomorphisms with a 1-to-1 correspondence on the graph edges. If there is a graph isomorphism between graphs G_1 and G_2, we say that G_1 and G_2 are *equivalent up to graph isomorphism*, denoted by $G_1 \simeq G_2$. Alternatively, we say that G_1 and G_2 are *(graph-)isomorphic*. As the name suggests it, $\simeq$ is an equivalence relation.

Proposition 3.3 *$\simeq$ is an equivalence for graphs.*

PROOF: Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2, 3)$ be graphs.

Reflexivity: $G_1 \simeq G_1$ by canonical isomorphism id_{V_1}.

Symmetry: Let $G_1 \simeq G_2$ by graph isomorphism $\iota : V_1 \to V_2$. ι is a bijective function and, therefore, ι^{-1} is also a bijective function. It remains to be shown that ι^{-1} is a graph homomorphism between G_2 and G_1. From $v\ E_2^a\ w$, we know that $\iota^{-1}(v)\ E_1^a\ \iota^{-1}(w)$ because if $\iota(\iota^{-1}(v)) = v$ and $\iota(\iota^{-1}(w)) = w$ and ι is a graph isomorphism (cf. 2.8).

Transitivity: Let $G_1 \simeq G_2$ by ι_1 and $G_2 \simeq G_3$ by ι_2. As ι_1 and ι_2 are bijective, $\iota_1 \circ \iota_2$ is bijective. We show that $\iota_1 \circ \iota_2$ is a graph isomorphism between G_1 and G_3. Thus, $G_1 \simeq G_3$ follows: $v\ E_1^a\ w$ iff $\iota_1(v)\ E_2^a\ \iota_1(w)$ iff $\iota_2(\iota_1(v))\ E_3^a\ \iota_2(\iota_1(w))$. Q. E. D.

There are sufficient conditions that apply to graph-isomorphic graphs G_1 and G_2. First, the sizes of the node sets are identical, i. e., $|V_1| = |V_2|$, being a consequence of bijectivity of graph isomorphisms. Furthermore, if we obtain a sorted list of node degrees (number of neighbors) from G_1 and G_2, they will turn out identical. Graph isomorphisms are mere renaming functions between the nodes of G_1 and those of G_2. Loosely speaking, matching up to graph isomorphisms follows a *what-you-ask-is-what-you-get* manner, which is highly precise on one hand, but also inflexible on the other hand.

Throughout this section, we will use the abstract but simple graphs, depicted in Figure 3.4, to illustrate the differences between the graph equivalence notions. These are more or less standard graphs, that may also be found in collections of comparative semantics of process theories, e. g., in van Glabbeek's *Linear-Time Branching-Time Sepctrum* [133]. The interpretation towards graph pattern matching for graph databases is our contribution.

Example 3.4 Consider first $G_{(a)}$ and $G_{(b)}$ in Figure 3.4. They are isomorphic by ι $(A \mapsto A, B_1 \mapsto B_2', B_2 \mapsto B_1', C_1 \mapsto C_2', C_2 \mapsto C_1')$. Neither $G_{(a)}$ nor $G_{(b)}$ are graph-isomorphic to $G_{(c)}$ because there is no graph isomorphism between these graphs. Observe the different number of nodes (5 in $G_{(a)}/G_{(b)}$ and only 4 in $G_{(c)}$). Similarly, $G_{(d)}$ is not isomorphic to $G_{(a)}$, $G_{(b)}$, or $G_{(c)}$. However, $G_{(d)}$ has two subgraphs that are isomorphic to $G_{(a)}$ and $G_{(b)}$, i. e., $G_{(d)}$ basically considers the same kinds of entities as $G_{(a)}$ and $G_{(a)}$. There would be no harm in identifying these two graphs. ■

3.1.2 Similarity

As already mentioned throughout Section 2.3, graph isomorphisms have been found as too strict in diverse domains. Robin Milner presented the algebraic notion of graph similarity [96] with the identical intend: comparing program graphs by graph isomorphisms is too restrictive to account for the *computational behavior of processes*. Graphs, or process graphs, shall not be distinguished if they can simulate one another (cf. Example 2.17). *Graph similarity* between two graphs G_1 and G_2, denoted $G_1 \leftrightarrow G_2$, requires $G_1 \sqsubseteq_{\mathsf{sim}} G_2$ and $G_2 \sqsubseteq_{\mathsf{sim}} G_1$, i. e., we need two non-empty simulations, one between G_1 and G_2 and another one between G_2 and G_1. If $G_1 \leftrightarrow G_2$ holds, then G_1 and G_2 are *similar* or *related up to similarity*.

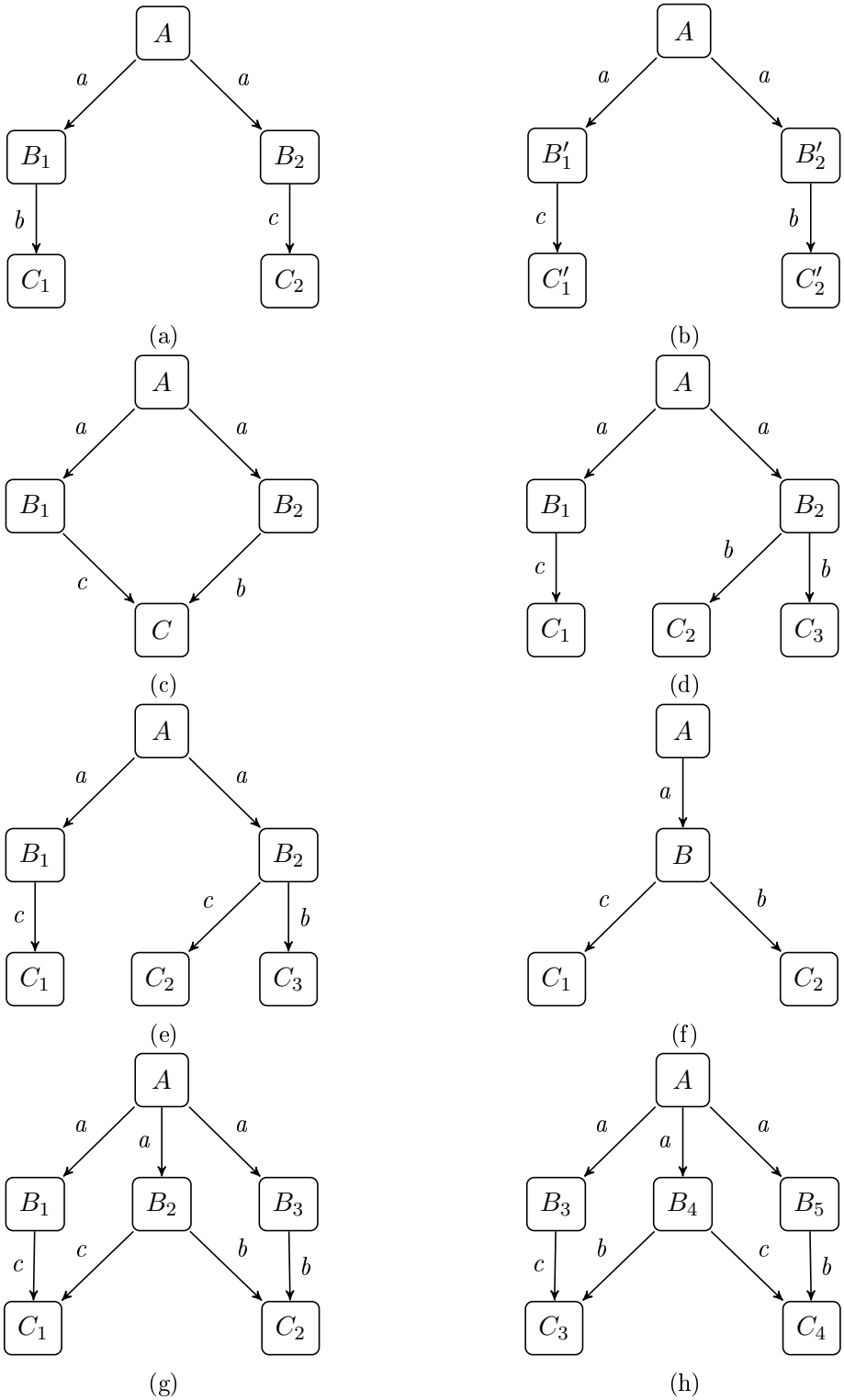


Figure 3.4: Example Graphs that Distinguish Graph Equivalences of Section 3.1

Example 3.5 (Example 3.4 continued) First, every graph-isomorphic pair of graphs is also similar (cf. Theorem 3.24), e. g., $G_{(a)} \leftrightarrow G_{(b)}$. Second, $G_{(a)}$ and $G_{(b)}$ are similar to $G_{(c)}$ and $G_{(d)}$. For instance,

$$S = \{(A, A), (B_1, B_2), (B_2, B_1), (C_1, C), (C_2, C)\}$$

is a simulation between $G_{(a)}$ and $G_{(c)}$. Its inverse S^{-1}, i. e.,

$$S^{-1} = \{(A, A), (B_2, B_1), (B_1, B_2), (C, C_1), (C, C_2)\}$$

is the required simulation between $G_{(c)}$ and $G_{(a)}$. ■

Hence, similarity capable of splitting and joining nodes with similar incidences. The two decoupled simulations allow for even worse topological destructions between two similar graphs.

Example 3.6 Consider the graphs $G_{(e)}$ and $G_{(f)}$ in Figure 3.4. $G_{(e)}$ has a subgraph that is isomorphic to $G_{(f)}$. Simulation

$$S_{(e)\to(f)} = \{(A, A), (B_1, B), (B_2, B), (C_1, C_1), (C_2, C_1), (C_3, C_2)\}$$

justifies $G_{(e)} \sqsubseteq_{\mathsf{sim}} G_{(f)}$. This, time $S^{-1}_{(e)\to(f)}$ is not a simulation because $(B, B_1) \in S^{-1}_{(e)\to(f)}$ and $(B, c, C_1) \in E_{(f)}$, but there is no node reachable via a c-edge from B_1. However, $G_{(e)}$ simulates $G_{(f)}$ by only taking the right-hand side a-branch:

$$S_{(f)\to(e)} = \{(A, A), (B, B_2), (C_1, C_2), (C_2, C_3)\}.$$

Thus, $G_{(e)} \leftrightarrow G_{(f)}$. ■

Evidently, graph simulations do not hold node relationships as tight as graph isomorphisms do which is a reason why similarity is not a general equivalence for graphs. An extreme case, showing that non-empty simulations are not transitively preserved, has been discussed by Example 2.34, meaning, similar graph patterns may yield drastically different matches in a graph pattern matching scenario (cf. Section 3.2). Just in case we have a simulation S between G_1 and G_2, so that S^{-1} is a simulation between G_2 and G_1, verifies a stronger correspondence between the graph patterns. In that case, we call S a *graph bisimulation*, which contributes to the notion of *graph bisimilarity*.

Note, similarity as it appears in process theories [96, 95, 133] is an equivalence relation. The reason is that every process has an *initial state*, i. e., one node in the graph, from which all computations begin. In graph database settings, such an initial state is comparable with the notion of a root node (cf. Section 2.1.1). As argued throughout Section 2.3.1, modern graph databases do not necessarily possess such a root node. Hence, we also lose the status of equivalence of graph similarity, which will be regained in Section 3.1.4.

3.1.3 Bisimilarity

Bisimilarity has been introduced to computer science as a proof method for algebraic fixpoint theories over concurrent processes by David Park [107], but has earlier appeared in the philosophical context of modal logics [131, 119] and has now become the quasi-standard of equivalence notions for concurrent processes [95, 97, 126]. Also, the field of databases keeps in touch with the developments on bisimilarity [116, 109, 123, 30, 4]. Bisimulations ask for tightly connected simulations between the graph subjects.

Definition 3.7 (Bisimulation)
Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2)$ be graphs. A simulation R between G_1 and G_2 is a *bisimulation between G_1 and G_2* iff R^{-1} is a simulation between G_2 and G_1. If a non-empty bisimulation between G_1 and G_2 exists, G_1 and G_2 are *bisimilar*, denoted $G_1 \leftrightarrows G_2$. ▲

As an alternative characterization, consider bisimulations between G_1 and G_2 as relations $R \subseteq V_1 \times V_2$, so that whenever $(v_1, v_2) \in R$,

(1) $v_1\ E_1^a\ v_1'$ implies a $v_2' \in V_2$ with $v_2\ E_2^a\ v_2'$ and $(v_1', v_2') \in R$ and

(2) $v_2\ E_2^a\ v_2'$ implies a $v_1' \in V_1$ with $v_1\ E_1^a\ v_1'$ and $(v_1', v_2') \in R$.

Example 3.8 We already had bisimulations, namely S and S^{-1} in Example 3.5. Hence, $G_{(a)} \leftrightarrows G_{(c)}$. In fact, all four graphs, $G_{(a)}$ to $G_{(d)}$, are bisimilar. Note, if S is a bisimulation, then S^{-1} is bisimulation. Regarding Example 3.6, there is no bisimulation R between graphs $G_{(e)}$ and $G_{(f)}$ with $(A, A) \in R$. As soon as $(A, A) \in R$, also $(B_1, B) \in R$, but $B\ E_{(f)}^c\ C_1$ and $B_1\ E_{(e)}^c = \emptyset$. ■

Similarly to the case of graph similarity, we do not speak of *bisimulation equivalence* because, for the same reasons as discussed for similarity, bisimilarity fails to persist transitively. What would be needed is complete coverage of the left-hand side graph. The complementation of (bi-)similarity of the next section provides relief in this respect.

3.1.4 On Graph Topology and (Bi-)Simulation Equivalence

The issue of partial matching we exemplified for graph similarity in Example 3.6 is only one dimension of a bigger issue of graph similarity and bisimilarity, absent in graph isomorphisms and homomorphisms. Because homomorphisms are functions from one node set to the other, it is guaranteed that all nodes of the first graph are captured. In contrast, (bi-)simulations are relations with no such condition. This yields different decisions of bisimilarity than we suggested earlier.

Example 3.9 (Example 3.8 continued) Although $G_{(e)}$ and $G_{(f)}$ are not bisimilar by a relation R with $(A, A) \in R$, they are bisimilar, e. g., by

$$R_{(e)\to(f)} = \{(B_2, B), (C_2, C_1), (C_3, C_2)\}.$$ ■

This is an issue we already discussed for simulations as it eventually culminates in *leaf node insensitiveness* (cf. Examples 2.20 and 2.21, or Proposition 2.22). Beyond insensitiveness towards leaf nodes, there is also a more pragmatic reason why we look for something different in graph database pattern matching.

Example 3.10 Consider the graphs $G_{(g)}$ and $G_{(h)}$ in Figure 3.4. They are bisimilar, hence also similar, by the bisimulation relation

$$R^* = \{(A, A), (B_1, B_3), (B_2, B_4), (B_3, B_5), (C_1, C_3), (C_1, C_4), (C_2, C_3), (C_2, C_4)\}.$$

Note, (C_1, C_4) and (C_2, C_3) are related by R^*. Hence, a pattern matching process based on (bi-)similarity treats both nodes as equivalent, although C_1 is the target node of c-labeled edges, only. In contrast, C_4 participates in c- and b-relationships. Putting it in a more realistic context, consider the label b to represent an *author* relation and c as an *editor* relation. Hence, C_1 represents an editor of some journals/proceedings and C_2 an author of articles. R^* suggests that also C_3 is an author but disregards it as an editor in the relationship $(C_2, C_3) \in R^*$, which makes a difference in the quality of the results obtained from $G_{(g)}$ and those from $G_{(h)}$. Bisimilarity does not distinguish them at all. ■

Fortunately, the same formal method as for simulations in Chapter 2, namely the simultaneous consideration of forward and backward edges, resolves the issue. Following the

naming convention of Ma et al. [85, 86], we obtain *dual similarity* ($\leftrightarrow_{\mathsf{D}}$) and *dual bisimilarity* ($\leftrightarrows_{\mathsf{D}}$) based on the notion of dual simulations[4] (cf. Definition 2.23).

Definition 3.11 (Dual (Bi-)Similarity)
Two graphs G_1 and G_2 are *dual similar* iff $G_1 \sqsubseteq_{\mathsf{Dsim}} G_2$ and $G_2 \sqsubseteq_{\mathsf{Dsim}} G_1$. They are *dual bisimilar* iff $G_1 \sqsubseteq_{\mathsf{Dsim}} G_2$ by dual simulation R, such that R^{-1} is a dual simulation between G_2 and G_1. ▲

Example 3.12 (Example 3.10 continued) Relation R^* is a not a dual bisimulation because $(C_2, C_3) \in R^*$ and $B_4\ E^c_{(h)}\ C_3$ but $E^c_{(g)}C_2 = \emptyset$. Furthermore, $R_{(e)\to(f)}$ from Example 3.9 is not a dual bisimilarity between $G_{(e)}$ and $G_{(f)}$ because $(B_2, B) \in R_{(e)\to(f)}$ and $A\ E^a_{(e)}\ B_2$, but $(A, A) \notin R_{(e)\to(f)}$. $G_{(e)}$ and $G_{(f)}$ are not dual bisimilar because every dual bisimulation must contain (A, A) but, as argued in Example 3.8, a bisimulation containing (A, A) does not exist. All other (bi-)similar examples are dual (bi-)similar. ■

As already mentioned, the idea of incorporating dual simulations already appeared in 1999 [3]. In the process-theoretic context, a similar notion appeared as invariants of a proof calculus ten years earlier [74]. Lynch and Vaandrager [84] established some of its properties regarding trace-based proof methods.

The resulting notions of dual (bi-)similarity are equivalence relations, given that we assumed connected graphs in this section. Dual similarity is an equivalence because of the proof steps we performed in Lemma 2.36. To be self-contained, we provide the equivalence result for dual bisimilarity.

Proposition 3.13 $\leftrightarrows_{\mathsf{D}}$ *is an equivalence relation over connected graphs.*

Proof: Let $G_i = (V_i, \Sigma, E_i)$ $(i = 1, 2, 3)$ be connected graphs.

Reflexivity: $G_1 \leftrightarrows_{\mathsf{D}} G_1$ by canonical bisimulation id_{V_1}. In fact, every graph isomorphism is a dual bisimulation (cf. Theorem 3.24).

Symmetry: If $G_1 \leftrightarrows_{\mathsf{D}} G_2$, then there is a non-empty dual bisimulation $R \subseteq V_1 \times V_2$. We show that $R^{-1} := \{(w, v) \mid (v, w) \in R\}$ is a dual bisimulation between G_2 and G_1. Therefore, consider $(v_2, v_1) \in R^{-1}$ and $v_2\ E^a_2\ w_2$. It holds that $(v_1, v_2) \in R$ and $v_1\ E^a_2\ w_1$ with $(w_1, w_2) \in R$ because R is a dual bisimulation. Thus, $(w_2, w_1) \in R^{-1}$. The case of $u_2\ E^b_2\ v_2$ follows a similar line of arguments. Furthermore, the cases $v_1\ E^a_1\ w_1$ and $u_1\ E^b_1\ v_1$ are handled analogously.

Transitivity: Let $G_1 \leftrightarrows_{\mathsf{D}} G_2$ by R_1 and $G_2 \leftrightarrows_{\mathsf{D}} G_3$ by R_2. We show that $R_1 \circ R_2$ is a dual bisimulation between G_1 and G_3. Let $(u_1, w_1) \in R_1 \circ R_2$. Then there is some $v_1 \in V_2$ with $(u_1, v_1) \in R_1$ and $(v_1, w_1) \in R_2$. For some edge involving u_1 (w_1, resp.), i. e., $u_1\ E^a_1\ u_2$ or $u_0\ E^b_1\ u_1$, we need to show a $w_2/w_0 \in V_3$, so that $w_1\ E^a_3\ w_2$ or $w_0\ E^a_3\ w_1$ and $(u_2, w_2) \in R_1 \circ R_2$ or $(u_0, w_0) \in R_1 \circ R_2$. As the other cases are completely analogous, we only show the case of $u_1\ E^a_1\ u_2$. Since R_1 is a dual bisimulation, $v_1\ E^a_2\ v_2$ with $(u_2, v_2) \in R_1$. Since R_2 is a dual bisimulation, $w_1\ E^a_3\ w_2$ with $(v_2, w_2) \in R_2$. Thus, $(u_2, w_2) \in R_1 \circ R_2$ by construction.

The remainder, namely that $R_1 \circ R_2 \neq \emptyset$, follows identical principles as we performed for showing that graph schema refinement ($\preccurlyeq$) is a preorder. The crucial argument is connectedness of G_2. Thus, $G_1 \leftrightarrows_{\mathsf{D}} G_3$. Q. E. D.

[4] In contrast to what the name suggests, dual simulation is not at all the dual concept of simulation, in a mathematical sense. Dual simulations are rather *back-and-forth simulations* or *transposition-invariant simulations*. Admittedly, the notion Ma et al. chose is way more catchy.

The characteristic missing in dual (bi-)similarity is locality [85, 53]: The matched subgraphs will only be bounded by characteristics of the database, not by those of the graph pattern, especially in the presence of cycles.

Example 3.14 Consider our three cyclic graphs from Figure 3.1. They are dual bisimilar, and for every number $k > 2$, a cycle with $k - 1$ nodes, following the shape of those in the figure, is bisimilar to Figure 3.1 (a). Thus, the size of the induced match graphs up to dual bisimilarity will not be determined by characteristics of the pattern graph. ■

Ma et al. even show that any matching mechanism that fixed this so-called *bounded cycle problem* directly entails NP-completeness of the respective pattern matching problem [86].

Instead of stigmatizing unboundedly cyclic matching of (dual) (bi-)similarity, we lift it as a feature of the mechanisms. Recall our Example 3.1, where we postulated a single pattern to represent an unbounded number of graph patterns. Thus, every citation ring will be discovered in a given graph database *DB* by posing Figure 3.1 (a) (as the minimal candidate) as graph pattern. It is not necessary to collect any statistics from the graph database instance *DB* because all the graphs in Figure 3.1 are dual bisimilar.

3.2 Graph Pattern Matching

So far, we have studied different graph relations that may be used to compare graph patterns with one another. We subsequently integrate the graph equivalence notions ($\simeq$, $\leftrightarrows_{\mathsf{D}}$, and $\leftrightarrow_{\mathsf{D}}$) of the last section into a general pattern matching framework. Collected from diverse areas of computer science and non-computer science, Gallagher [59] identifies the following task as *graph pattern matching*.

Problem (Graph Pattern Matching (up to $\equiv$))
Input: Labeled graphs $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$, where Q is connected [59].
Output: The set $[\![Q]\!]^{\equiv}_G$ of all subgraphs $M \subseteq G$ with $Q \equiv G$.

Q is generally called the *pattern graph* while G is called the *data graph* [59, 48]. There is a graph pattern matching problem for graph isomorphisms ($\subseteq_{\mathsf{iso}}$), one for dual bisimilarity ($\subseteq_{\leftrightarrows_{\mathsf{D}}}$), and another one for dual similarity ($\subseteq_{\leftrightarrow_{\mathsf{D}}}$). All three problems are related as indicated in Figure 3.4, being proven in Theorem 3.24 in Section 3.5. By using equivalences on (connected) graphs, we directly obtain an equivalence result for graph pattern matches.

Proposition 3.15 *Let Q_1 and Q_2 be two graph patterns and let $\equiv$ be an equivalence relation on connected graphs. If $Q_1 \equiv Q_2$, then $[\![Q_1]\!]^{\equiv}_G = [\![Q_2]\!]^{\equiv}_G$ for all data graphs G.*

Proof: $M \in [\![Q_1]\!]_G$ iff $Q_1 \equiv M$. Furthermore, by the symmetry of $\equiv$, we have $Q_2 \equiv Q_1$. Thus, $Q_2 \equiv M$ (by transitivity of $\equiv$) which implies $M \in [\![Q_2]\!]_G$. The case of $M \in [\![Q_2]\!]_G$ uses the reversed argumentation. Q. E. D.

The converse direction also holds for our equivalences ($\simeq$,$\leftrightarrows_{\mathsf{D}}$, and $\leftrightarrow_{\mathsf{D}}$). A potential proof would construct a counterexample database $\widehat{G}$, for which $Q_1 \not\equiv Q_2$ implies $[\![Q_1]\!]_{\widehat{G}} \neq [\![Q_2]\!]_{\widehat{G}}$, a contradiction to the assumption. The proof strategy would be the same as the one we used for proving completeness of graph schema refinement in Lemma 2.60.

All the implied graph pattern matching mechanisms face an inevitable difficulty regarding practical implementations: they all have an NP-complete *non-emptiness problem* (cf. Chapter 4 for more details in a graph query context).

Proposition 3.16 ([37, 43, 101]) *The following problem is NP-complete for all $\equiv \in \{\simeq, \leftrightarrows_{\mathsf{D}}, \leftrightarrow_{\mathsf{D}}\}$: For graph pattern Q and data graph G, decide whether $[\![Q]\!]_G \neq \emptyset$.*

The problem instance for graph isomorphisms is the standard *subgraph isomorphism problem* [37, 68]. Through subgraph (bi-)similarity, we gain the ability to describe infinitely many graph patterns by a single finite one but remain intractable in answering the subgraph matching problem. A different angle on graph pattern matching partially resolves the issue. Thereby, we finally reach our notion of dual simulations in the graph pattern matching context.

3.3 Graph Pattern Matches

Graph pattern matching, as described in the last section, has two subtasks: (1) obtaining a subgraph M of G and (2) comparing Q and M for equivalence ($\equiv$). (1) is an immediate candidate for dropping from the formulation of the graph pattern matching problem to potentially gain tractability. One of the computational difficulties of the discussed subgraph matching problems might stem from the intuition, that we always have to guess a subgraph M of G, for which equivalence with Q can be established. However, the following discussion reveals that NP-completeness is decoupled from this intuition, at least for graph isomorphisms (and generally, homomorphisms).

The *subgraph isomorphism problem* may be turned into the following equivalent problem: Given Q and G, collect in $[\![Q]\!]_G^{\equiv}$ all *witnesses for* $\equiv$, e. g., subgraph isomorphisms for $\equiv=\simeq$. Instead of collecting all subgraphs of G isomorphic to Q, we capture the subgraph isomorphisms themselves. However, each of these subgraph isomorphisms induces a subgraph that would be contained in the set $[\![Q]\!]_G^{\simeq}$. By this reduction sketch, we derive that capturing the subgraph isomorphisms instead of isomorphic subgraphs of G also poses an NP-complete non-emptiness problem and subgraph isomorphism problem.

Recall that subgraph isomorphisms are surjective graph homomorphisms. Pattern matching up to graph homomorphisms ($\sqsubseteq_{\mathsf{hom}}$) collects all graph homomorphisms between Q and G. Once again, the associated subgraph homomorphism problem is implied by using the resulting homomorphisms to induce subgraphs of G. Graph homomorphisms form the ground for many graph query languages [5]. However, they inherit some of the inflexibility of its stricter siblings (graph isomorphisms). As we already discussed at the end of Section 2.2.2, graph homomorphisms may collapse similar nodes, e. g., there are graph homomorphisms between the graph $G_{(a)}$ ($G_{(b)}$, resp.) and $G_{(c)}$, but these graphs are not equivalent up to graph homomorphisms. Graph homomorphic-matching represents a more liberal matching mechanism than subgraph isomorphic matching, but it is by no means tractable. Obtaining any graph homomorphism is as hard as obtaining any subgraph isomorphism [68]. Hence, it is the tight connections between the node sets due to functions that make the problem so restrictively hard.

Resolution has been found in the notion of simulations [51, 48], and in particular dual simulations [85, 86].

Definition 3.17 (Dual Simulation Pattern Matching)
Let Q be a connected graph pattern and G be a data graph. A non-empty dual simulation is called a *dual simulation match for Q in G*. The set of all dual simulation matches for Q in G is denoted by $[\![Q]\!]_G^{\sqsubseteq_{\mathsf{Dsim}}}$. ▲

Dual simulations are preorders between connected graph patterns (cf. Lemma 2.36). Hence, $Q_1 \sqsubseteq_{\mathsf{Dsim}} Q_2$ implies $[\![Q_1]\!]_G^{\sqsubseteq_{\mathsf{Dsim}}} \subseteq [\![Q_2]\!]_G^{\sqsubseteq_{\mathsf{Dsim}}}$ for all graphs G. Furthermore, we also know that $[\![Q]\!]_G^{\sqsubseteq_{\mathsf{Dsim}}}$ is union-closed and, if $[\![Q]\!]_G^{\sqsubseteq_{\mathsf{Dsim}}} \neq \emptyset$, it has a unique greatest dual simulation match (cf. Theorem 2.29 and Definition 2.30). Especially the unique greatest dual simulation match will be handy when considering dual simulations in the context of graph query language operators (cf. Chapter 4).

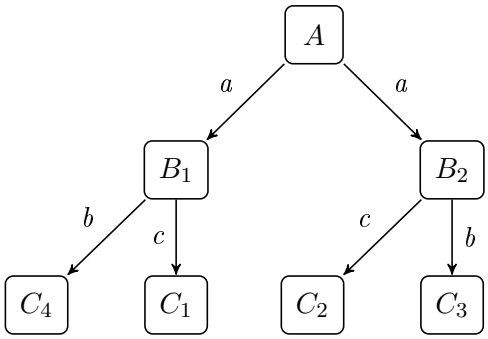


Figure 3.5: Heterogeneity in Subgraph Similar Matches

By dual simulation pattern matching, we have found a tractable [85] alternative to homomorphism-based pattern matching. As discussed in Section 3.1.4, the lack of locality, that dual simulation matches inherit from dual similarity (cf. Section 3.1.4), is a feature that allows for obtaining matches, even if the structures are not identical. Thereby, a single pattern graph easily represents an infinite family of pattern graphs (cf. Figure 3.1) without prior knowledge of the structures that are captured in a graph database instance. Next, we combine these features with a notion of negation, that we also borrow from process theories.

3.4 Failures Theory for Graph Patterns

Let us recapitulate one of our earlier examples, namely $G_{(e)}$ and $G_{(f)}$ of Figure 3.4, now in terms of graph pattern matching up to dual simulations/dual similarity. Recall from Example 3.6, that $G_{(e)} \leftrightarrow G_{(f)}$. The simulations we gave in the example are dual simulations.

Example 3.18 Assume $G_{(f)}$ is given as the pattern graph and the graph G as given in Figure 3.5. It holds that $G_{(e)}$ is a subgraph of G. Hence, $G_{(e)}$ is a subgraph match up to dual similarity because $G_{(e)}$ and $G_{(f)}$ are equivalent up to dual similarity. If we present $G_{(e)}$ alone as a match for $G_{(f)}$, two different reasons for this match can be: (1) There is no b-labeled edge from node B_1 or (2) we do not know whether B_1 has an outgoing b-labeled edge, based on the given match. The second reason at least does not impose the wrong facts about the database. However, can we know that there is an outgoing b-labeled edge? Alternatively, can we exclude the match $G_{(e)}$ to the pattern $G_{(f)}$ in G? ■

After a subgraph has been drawn from the data graph, the cut-out adjacencies are lost. Even without the tool of obtaining subgraphs, dual simulations conceal wanted heterogeneity.

Example 3.19 Once more, we look at $G_{(e)}$ and $G_{(f)}$. This time, $G_{(e)}$ shall be the pattern graph, and $G_{(f)}$ is the data graph. As both graphs are dual similar, there is a non-empty dual simulation between $G_{(e)}$ and $G_{(f)}$, namely $S_{(e)\to(f)}$ (cf. Example 3.6). Thus, $G_{(e)} \sqsubseteq_{\mathsf{Dsim}} G_{(f)}$ and $S_{(e)\to(f)}$ is a dual simulation match for $G_{(e)}$ in $G_{(f)}$. Observe that $(B_1, B) \in S_{(e)\to(f)}$ but B_1 has no outgoing b-labeled edge. As far as dual simulations are concerned, this is fine because it implements a *minimum support* for the adjacent edges of the pattern graph. Reviewing the other simulation given in Example 3.6 revealed the partial relation of B_1 and B, but then the intractable subgraph dual similarity must be employed. Is there a way to impose a restriction on the pattern that rules such (possibly) unwanted matches out? ■

In the Example 3.18, it is sufficient to observe that node B_1 in $G_{(e)}$ has no outgoing edge labeled by b while B in $G_{(f)}$ has one. Pattern matching is then restricted to those matches with the same (in-)capabilities as the pattern node they match via simulations. Regarding Example 3.19, it is again sufficient to observe that B_1 in $G_{(e)}$ does not have an outgoing b-labeled edge, this time it is thought of as a property of the pattern graph. For the second example, we have introduced a failure theory for graph databases in [91]. Here, we extend and analyze this notion to also cope with Example 3.18.

3.4.1 Failures

As in the case of graph simulations, the original notion of *failures* appeared as part of the analysis of abstract programs/program graphs [25]. Let $G = (V, \Sigma, E)$. In a first attempt, a graph node v *fails a relation symbol* $a \in \Sigma$ if $vE^a = \emptyset$, i. e., v has no outgoing edge labeled by a. As discussed for graph (bi-)similarity in Section 3.1.4, the direction of relationships, here embodied by the notion of failure, plays a vital rôle when assessing the matches to a pattern. We, therefore, observe each relation symbol $a \in \Sigma$ in two different shapes. First, we have a failure a of v, meaning that there is no outgoing a-labeled edge from v, as before. Second, we also encode failures due to incoming edges by $\bar{a}$, i. e., v fails at $\bar{a}$ iff $E^a v = \emptyset$.

This is the only time, throughout the whole thesis, that we diverge from our assumption that graphs that are compared by some matching notion share their alphabets. In the failure setting, a graph pattern will provide its own alphabet to specify its relationships/predicates of interest. The notion of failures is, thus, parameterized by some alphabet $\Gamma \subseteq \Sigma \subsetneq \mathcal{P}$.

Definition 3.20 (Γ-Failure)
Let $G = (V, \Sigma, E)$ be a graph, $v \in V$, $\Gamma \subseteq \Sigma$ any finite alphabet alphabet. Define the directed alphabet over Γ as $\bar{\Gamma} := \{a, \bar{a} \mid a \in \Gamma\}$. $\alpha \in \Gamma$ is a Γ-*failure of* v iff $\alpha = a$ implies $vE^a = \emptyset$ and $\alpha = \bar{a}$ implies $E^a v = \emptyset$. $\mathcal{F}_G^\Gamma(v)$ denotes the set of all Γ-failures of v. ▲

Example 3.21 (Example 3.18 and Example 3.19 continued) Let $\Gamma = \{a, b, c\}$. According to Definition 3.20 $\bar{\Gamma} = \{a, \bar{a}, b, \bar{b}, c, \bar{c}\}$. In $G_{(e)}$, $b, \bar{b} \in \mathcal{F}_G^\gamma(B_1)$. In contrast, while $\bar{b}$ is a Γ-failure of B_2 in $G_{(e)}$, b is none. ■

3.4.2 Failure Simulation in Two Examples

A formal account for the definition of *failure simulation* may be found in our previous work [91]. Here, we want to resolve the issues of the examples from the beginning of this section, informally.

The way we generally assess failures in a dual simulation pattern matching scenario is that we first establish dual simulations S between the node sets of the graphs. Towards failure incorporation, the pattern nodes carry information about their failures. Let $Q = (V_Q, \Gamma, E_Q)$ be a graph pattern and let $G = (V, \Sigma, E)$ be a data graph. Based on the alphabet of Q, there is a set of Γ-failures for every pattern node $v \in V_Q$. If we establish a dual simulation between Q and G, i. e., a relation $S \subseteq V_Q \times V$, we may additionally require that $\mathcal{F}_Q^\Gamma(v) = \mathcal{F}_G^\Gamma(w)$ for all $(v, w) \in S$.

Example 3.22 (Example 3.19 continued) Let $\Gamma = \{a, b, c\}$ be the alphabet of $G_{(e)}$. We had $G_{(e)} \sqsubseteq_{\mathsf{Dsim}} G_{(f)}$ by $S_{(e)\to(f)}$. But $(B_1, B) \in S_{(e)\to(f)}$ and $\mathcal{F}_Q^\Gamma(B_1) \supseteq \{b\} \not\subseteq \mathcal{F}_G^\Gamma(B)$. Hence, $S_{(e)\to(f)}$ is ruled out by regarding the failure of b in B_1. ■

The other scenario we exemplified in Example 3.18 may be resolved by the same means.

Example 3.23 (Example 3.18 continued) Recall, we had $G_{(f)}$ given as pattern graph and G (depicted in Figure 3.5) as data graph. By subgraph similarity, we obtained $G_{(e)}$

as a subgraph of G, for which $G_{(e)} \leftrightarrow_{\mathsf{D}} G_{(f)}$ by $S_{(e)\to(f)}$ and $S_{(f)\to(e)}$. We get that $(B_1, B) \in S_{(e)\to(f)}$ and $\mathcal{F}_{G_{(f)}}^{\{a,b,c\}} \neq \mathcal{F}_{G_{(e)}}^{\{a,b,c\}}$ by the same reason as before. ∎

Hence, failures additionally implement negation of node properties, being missing information in a general sense. Of course, graph query languages like SPARQL provide an operator to express negation [65] (the minus operator [12]). While this operator is associated with high complexity bounds, dual simulations appreciate failures in a more lightweight sense. Recall that the first step we sketched was establishing a dual simulation between the pattern and the data graph. The next step requires a polynomial iteration over all pattern nodes and their matches. Failures can be indexed so that the actual computational time is spent on computing dual simulations, a task performed in PTIME (cf. Chapter 5).

3.5 Summary

All the matching notions are interrelated, as summarized in Figure 3.3. We read the figure by picking any two relations r_1 and r_2 connected by a directed path, e. g., $r_1 = \leftrightarrows_{\mathsf{D}}$ and $r_2 = \sqsubseteq_{\mathsf{sim}}$ but not $r_1 = \subseteq_{\mathsf{iso}}$ and $r_2 = \leftrightarrows$. The general idea of a path is that whenever two graphs are related by r_1, then they are also related by r_2. We even prove a stronger result, regarding so-called witnesses of r_1 and r_2. A *witness for* $\simeq$, $\subseteq_{\mathsf{iso}}$, *or* $\sqsubseteq_{\mathsf{hom}}$ is a graph isomorphism, a subgraph isomorphism, or a graph homomorphism. Likewise, witnesses for the similarity notions $\leftrightarrows_{\mathsf{D}}$, $\subseteq_{\leftrightarrows_{\mathsf{D}}}$, $\leftrightarrow_{\mathsf{D}}$, $\subseteq_{\leftrightarrow_{\mathsf{D}}}$, and $\sqsubseteq_{\mathsf{Dsim}}$ are dual bisimulations and dual simulations. Ordering graph matching notions in the way of Figure 3.3 is a useful tool in comparative semantics [133] and helps in estimating how strongly or weakly two graphs are related. Furthermore, the hierarchy itself (Figure 3.3) provides a proof method. If we know already that two graphs are related by r_2, then they cannot be related by any r_1 with a directed path to r_2. Conversely, proving relatedness by r_1 implies that the graphs are related by r_2.

Theorem 3.24 *Let* $\Psi = \{\simeq, \subseteq_{\mathsf{iso}}, \sqsubseteq_{\mathsf{hom}}, \leftrightarrows_{\mathsf{D}}, \subseteq_{\leftrightarrows_{\mathsf{D}}}, \leftrightarrow_{\mathsf{D}}, \subseteq_{\leftrightarrow_{\mathsf{D}}}, \sqsubseteq_{\mathsf{Dsim}}\}$ *and* $r_1, r_2 \in \Psi$, *such that* $r_1 \longrightarrow^* r_2$. *Every witness for* r_1 *is a witness for* r_2.

PROOF: Let $G_1 = (V_1, \Sigma, E_1)$ and $G_2 = (V_2, \Sigma, E_2)$ be two graphs. The case of $r_1 = r_2$ is trivial, e. g., a witness ι for $G_1 \simeq G_2$ is a graph isomorphism. We distinguish the cases of the direct neighbors in Figure 3.3 and then show that $\longrightarrow$ is transitive.

$\simeq \longrightarrow \leftrightarrows_{\mathsf{D}}$: Let ι be a graph isomorphism between G_1 and G_2 and let $(v_1, v_2) \in \iota$, i. e., $\iota(v_1) = v_2$. If $v_1\ E_1^a\ w_1$, then there is a $w_2 = \iota(w_1)$ and $v_2\ E_2^a\ w_2$, because ι is a homomorphism. For the same reason, if $u_1\ E_1^b\ v_1$, then there is an $u_2 = \iota(u_1)$ and $u_2\ E_2^b\ v_2$. Conversely, if $v_2\ E_2^a\ w_2$, then there is a $w_1 = \iota^{-1}(w_2)$ with $v_1\ E_1^a\ w_1$, because ι is a surjective homomorphism. Again for the same reason, if $u_2\ E_2^b\ v_2$, then $\iota^{-1}(u_2) = u_1$ exists with $u_1\ E_1^b\ v_1$.

$\leftrightarrows_{\mathsf{D}} \longrightarrow \leftrightarrow_{\mathsf{D}}$: By Definition 3.11, there is a dual simulation R between G_1 and G_2.

$\simeq \longrightarrow \subseteq_{\mathsf{iso}}$: Every graph isomorphism ι is a bijective homomorphism (cf. Definition 2.8), i. e., it is surjective and injective. Thus, ι is a subgraph isomorphism. As subgraph of G_2 choose G_2 itself.

$\leftrightarrows_{\mathsf{D}} \longrightarrow \subseteq_{\leftrightarrows_{\mathsf{D}}}$: We need to give a subgraph of $M \subseteq G_2$, so that $G_1 \leftrightarrows_{\mathsf{D}} M$. Choose $M = G_2$.

$\leftrightarrow_{\mathsf{D}} \longrightarrow \subseteq_{\leftrightarrow_{\mathsf{D}}}$: By the same argumentation, $G_1 \leftrightarrow_{\mathsf{D}} G_2$ and $G_2 \subseteq G_2$ justifies the claim.

$\subseteq \longrightarrow \subseteq_{\mathsf{iso}}$: Assume $G_1 \subseteq G_2$, i. e., $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. It holds that id_{V_1} is an injective graph homomorphism between G_1 and G_2 as (i) $\mathsf{id}_{V_1}(v) = \mathsf{id}_{V_1}(v')$ implies $v = v'$ and

(ii) if $v\ E_1^a\ w$, then $\mathsf{id}_{V_1}(v)\ E_2^a\ \mathsf{id}_{V_1}(w)$. (ii) is justified by $\mathsf{id}_{V_1}(v) = v$ and $\mathsf{id}_{V_1}(w) = w$, $v, w \in V_1 \subseteq V_2$, i. e., $v, w \in V_2$, and $(v, a, w) \in E_1 \subseteq E_2$, i. e., $(v, a, w) \in E_2$.

$\subseteq_{\mathsf{iso}} \longrightarrow \subseteq_{\leftrightarrows_{\mathsf{D}}}$: Let M be a subgraph of G_2 with $G_1 \simeq M$. $G_1 \leftrightarrows_{\mathsf{D}} M$ since $\simeq \longrightarrow \leftrightarrows_{\mathsf{D}}$. Hence, $G_1 \subseteq_{\leftrightarrows_{\mathsf{D}}} G_2$.

$\subseteq_{\leftrightarrows_{\mathsf{D}}} \longrightarrow \subseteq_{\leftrightarrow_{\mathsf{D}}}$: Take subgraph M of G_2 with $G_1 \leftrightarrows_{\mathsf{D}} M$. By $\leftrightarrows_{\mathsf{D}} \longrightarrow \leftrightarrow_{\mathsf{D}}$, $G_1 \leftrightarrow_{\mathsf{D}} M$. Thus, $G_1 \subseteq_{\leftrightarrow_{\mathsf{D}}} G_2$.

$\subseteq_{\mathsf{iso}} \longrightarrow \sqsubseteq_{\mathsf{hom}}$: Every subgraph isomorphism η is a homomorphism (cf. Definition 2.8).

$\subseteq_{\leftrightarrow_{\mathsf{D}}} \longrightarrow \sqsubseteq_{\mathsf{Dsim}}$: Take $M \subseteq G_2$ with $G_1 \leftrightarrow_{\mathsf{D}} M$. Hence, there is a non-empty dual simulation R between G_1 and M. As M is included in G_2, R is a dual simulation between G_1 and G_2. Thus, $G_1 \sqsubseteq_{\mathsf{Dsim}} G_2$.

$\sqsubseteq_{\mathsf{hom}} \longrightarrow \sqsubseteq_{\mathsf{Dsim}}$: Let η be a graph homomorphism between G_1 and G_2 with $\eta(v_1) = v_2$, i. e., $(v_1, v_2) \in \eta$, and $v_1\ E_1^a\ w_1$. Since η is a graph homomorphism, $\eta(v_1)\ E_2^a\ \eta(w_1)$ and, thus, there is a $w_2 = \eta(w_1)$ with $v_2\ E_2^a\ w_2$ and $(w_1, w_2) \in \eta$. Furthermore, if $u_1\ E_1^a\ v_1$, then there is a $u_2 = \eta(u_1)$ with $\eta(u_1)\ E_2^a\ \eta(v_1)$ because η is a graph homomorphism. Thus, η is a dual simulation between G_1 and G_2.

It remains to be shown that $\longrightarrow$ can be iterated transitively. Let therefore $r_1, r_2, r_3 \in \{\simeq, \subseteq_{\mathsf{iso}}, \sqsubseteq_{\mathsf{hom}}, \leftrightarrows_{\mathsf{D}}, \sqsubseteq_{\mathsf{Dsim}}, \leftrightarrows, \sqsubseteq_{\mathsf{sim}}\}$, such that $r_1 \longrightarrow r_2$ and $r_2 \longrightarrow r_3$, i. e., every witness of r_1 is a witness of r_2 and every witness of r_2 is a witness of r_3. Let us denote the witnesses by $W_{r_i}(G_1, G_2)$ $(i = 1, 2, 3)$. By assumption, $W_{r_1}(G_1, G_2) \subseteq W_{r_2}(G_1, G_2)$ and $W_{r_2}(G_1, G_2) \subseteq W_{r_3}(G_1, G_2)$. The claim follows by transitivity of $\subseteq$ and an inductive argument over $\longrightarrow^*$. Q. E. D.

Through small examples, we have argued for lack of *pragmatic value* of matching up to graph isomorphisms. Therefore, we first looked at equivalences up to graph isomorphisms and compared it to bisimilarity and similarity. These two notions are not equivalences over connected graphs, in contrast to their process-algebraic counterparts [95, 133]. Only the complementation due to dual simulations reestablished the equivalences, now called dual (bi-)similarity. We provided examples of how far these notions enhance the pragmatic value of graph isomorphisms. First of all, dual simulations are relations and, therefore, have the potential to represent a single match where otherwise multiple matches must be inspected. This is one aspect of simulations also found positive in user studies regarding *Exemplar Queries* [100]. Furthermore, every notion we presented is capable of unifying infinitely many graph patterns in a single finite graph pattern. Therefore, graph simulations may be applied where otherwise, path queries are necessary.

Our interpretations were not driven by computational complexity. However, we nevertheless provided proof that subgraph matching problems are NP-complete. Dual bisimilarity and dual similarity, on the other hand, can be computed in PTIME. Thereby, bisimilarity is a special case as it is among the hardest problems solvable in PTIME [18]. Incorporation of graph query language principles [5] would even turn the respective querying problems PSPACE-complete.

Beyond all the useful properties of simulations, graph pattern matching based on dual simulations also has its drawbacks, regarding awareness of incomplete data. We have given two examples, in which dual simulation matches are hard to justify. To overcome the issues, we introduced failures as a light-weight addition to express a minimal amount of negation. We applied the failures theory to subgraph dual similarity and dual simulation pattern matching.

For future work, an in-depth analysis of the failures theory in more application-driven contexts, e. g., as conformance notion for graph schemas, must be carried out. Beyond that,

there are certainly more scenarios, in which failures and versions of them help in establishing more accurate results without falling back to the weaknesses of graph-isomorphic matching. Beyond graph homomorphisms and simulations, there is a plethora of other matching notions to be explored, e. g., [133]. So far, the coverage of the spectrum by van Glabbeek is quite limited and might even stay invariant over the next few years because the matching notions cannot conservatively be transferred. We believe, particular adaptations or even wholly new mechanisms need to be developed. For instance, Fan et al. provide *p-homomorphisms* and *bounded simulations* [50, 52] that incorporate a means of counting-quantification over the traversed edges of a match. Thereby, a single graph pattern may describe not necessarily infinitely many graph patterns that are simultaneously matched, but at least more than the single graph pattern describes up to graph isomorphisms. Thus, the pragmatic value of these notions is already enhanced upon graph homomorphisms. Furthermore, the principles of [52, 50] went into the development of a key paradigm for graph databases [49]. What we learn from these and other examples is that a concrete application or information need drives new developments in graph pattern matching. (Sub-)graph isomorphisms are not always the best choice. Sometimes a way coarser matching notion is even more powerful w. r. t. what a single pattern expresses.

Having learned about pattern matching that does not need the machinery of graph query languages, we are curious about what happens if we complement one of the matching notions with operators from graph query languages. During the next chapter, we will devise several SPARQL versions whose semantic foundation is redirected to dual simulations. Will the query results have anything to do with those of the original language? Regarding the complexity game, do we win upon the original semantics? What practical impact does such a SPARQL version possess?

CHAPTER 4

Graph Queries

So far, we assumed graph patterns to be given by some virtual entity, e. g., some application user. As soon as graph database management systems are involved, the creation of graph patterns is encapsulated in a more general process, namely the formulation of an expression of a *graph query language*. A graph query language allows for combining and restricting graph patterns, matched in a graph database instance. Loosely following Vardi [134], we consider a (graph) query language $\mathcal{L}$ to be a pair $(\mathbb{L}, [\![\cdot]\!]\_)$ where $\mathbb{L}$ is the set of *query expressions* in $\mathcal{L}$ and $[\![\cdot]\!]\_$ is a function mapping expressions $\mathcal{Q} \in \mathbb{L}$ and (graph) databases DB to the result (set) of $\mathcal{Q}$ in DB, denoted by $[\![\mathcal{Q}]\!]_{DB}$. We call a query expression $\mathcal{Q} \in \mathbb{L}$ a *query in/from* $\mathcal{L}$, or just $\mathcal{L}$*-query*. *Matches of a query* stem from some abstract domain $\mathbf{D}$. Thus, fixing a (graph) database DB, we have $[\![\cdot]\!]_{DB} : \mathbb{L} \to \mathbf{2^D}$. For instance, for tuple relational queries $\mathcal{Q}$, $\mathbf{D}$ would be chosen to be the set of all tuples over an active domain [1].

For a query language $\mathcal{L}$, the complexity of evaluating queries of the language, according to its semantics, is a fundamental issue [34, 1, 106]. It is customary to consider the decision-version of the *evaluation problem of* $\mathcal{L}$ [134].

Problem (Evaluation($\mathbb{L}, [\![\cdot]\!]\_$))
Input: Query $\mathcal{Q} \in \mathbb{L}$, database DB, and candidate $\mu \in \mathbf{D}$.
Output: True iff $\mu \in [\![\mathcal{Q}]\!]_{DB}$.

In the evaluation problem we are not only given a query and a database, but also a candidate match μ, as input. The complexity of Evaluation($\mathbb{L}, [\![\cdot]\!]\_$) accounts for how hard it is to verify whether μ is a match for a given query $\mathcal{Q} \in \mathbb{L}$ in the database DB, rather than enumerating all the matches in $[\![\mathcal{Q}]\!]_{DB}$. Following the input structure of the evaluation problem, we consider both, the database DB and the query $\mathcal{Q}$ as input. The implied complexity measure is called *combined complexity*. According to Vardi [134], there are two more measures, namely *data complexity* and *expression complexity*. While data complexity assumes the query to be fixed, i. e., it measures the complexity of the problem only in the size of the database, expression complexity fixes the database to be queried. Often, combined complexity and expression complexity are very close, if not even identical [134]. Data complexity is a *measure of expressiveness of* $\mathcal{L}$ [134]. As we want to assess complexity for every possible query in $\mathcal{L}$, we will mainly give combined complexity results here. From combined complexity, we directly derive the data complexity by assuming all query-based factors to be constants.

$\mathcal{L}$-queries may not only return matches carrying information about the data asked for by some users. Often, query languages offer modifiers altering the result form of a query

$\mathcal{Q}$, e. g., asking whether there is any match μ to be found for $\mathcal{Q}$ in a given database DB. The associated problem is the *non-emptiness problem of* $\mathcal{L}$.

Problem (NonEmpty($\mathbb{L}, [\![\cdot]\!]\_$))
Input: Query $\mathcal{Q} \in \mathbb{L}$ and database DB.
Output: True iff $[\![\mathcal{Q}]\!]_{DB} \neq \emptyset$.

The same complexity measures as for the evaluation problem apply. For graph pattern matching (cf. Chapter 3), the non-emptiness problem is considered the standard problem: The *subgraph isomorphism problem* asks whether there is a subgraph isomorphism, rather than enumerating all of them. Since the subgraph isomorphism problem is NP-complete [37], it follows that its respective evaluation problem is in P.

Graph patterns, as discussed throughout Chapter 3, form a fragment[1] of so-called *conjunctive (graph) queries* [1]. The use of the most primitive language construct, which in most graph languages are edge patterns and conjunctions thereof, characterize such queries. For this class, we already analyzed the meaning of the queries according to different matching mechanisms, from homomorphism-based to failures- and simulation-based. A full-fledged *graph query language* combines graph patterns by further operators and modifiers. To this end, we cover SPARQL in this thesis, which refers to edge patterns as *triple patterns* and conjunctive queries as *basic graph patterns*. Compared to other query languages, SPARQL's W3C recommendation [65] is highly accepted by researchers and practitioners [62], its semantics has been formalized [112, 113], and its associated querying problems are well-studied [113, 121, 72].

Goals. SPARQL is the name of the query language $\mathcal{S} = (\mathbb{S}, [\![\cdot]\!]^{\mathbb{S}}\_)$, provided with recommended syntax and semantics by the W3C [114, 65]. Its semantics is based on homomorphisms, directly rendering the non-emptiness problem for basic graph patterns NP-complete [68]. So-called *optional patterns* drastically increase upon evaluation complexity [113, 121]. Based on the fact that dual simulations between graphs can be verified and found in PTIME [69, 86, 93] (cf. Chapter 5 for more details), we desire a different semantic function associated with SPARQL exhibiting dual simulations to achieve **(I) correctness and (II) tractability.**

Since dual simulations provide a weaker mechanism than matching up to homomorphisms (cf. Chapter 3), there will be dual simulation matches that do not relate to any SPARQL match, already in the case of basic graph patterns. Recall that this matching behavior is a feature of dual simulations that is missing in graph isomorphisms. Nevertheless, in need of a gold standard, we require *completeness* to accomplish (I), letting the new dual simulation semantics preserve all of SPARQL's original matches[2]. Losing soundness in the course of redefining the matching semantics of SPARQL queries must not come at the price of complexity. Notably, we solely aim for *tractable semantics*, i. e., those semantics entailing PTIME non-emptiness and evaluation. Correctness and tractability as attributes of query languages $\mathcal{L}$ will guide us through the following sections. A correct and tractable query language is a valuable aid for SPARQL query evaluation, e. g., as a querying preprocessor.

Contribution. We are the first studying interactions between dual simulations and the query language SPARQL w. r. t. correctness and tractability. Initially, we define dual simulations as matches for basic graph patterns and analyze the consequences upon the remaining concepts of SPARQL. While for basic graph patterns, we directly obtain a correct semantics that is tractable, SPARQL's join operators (i. e., inner and left outer join) render this canonical dual simulation semantics incorrect for full SPARQL. We identify a non-trivial

[1] We miss *projections* in graph patterns.
[2] A formal definition of *preservation of* SPARQL *matches* will be given by Definition 4.31 on Page 76.

fragment of SPARQL for which the canonical dual simulation semantics is correct, called *well-designed* SPARQL. This fragment exhibits a coNP-complete evaluation problem [113]. We show evaluation as well as non-emptiness to be tractable querying problems of this fragment under a dual simulation semantics.

Up to this point, the correctness of dual simulations is only achievable for well-designed SPARQL, missing out on many queries asked for by users [71]. To obtain a correct semantics for full SPARQL, we use an approximation principle entailing a unique greatest dual simulation between any SPARQL query and graph database, the *maximal dual simulation semantics for* SPARQL. The correctness of this semantics entails that every SPARQL match will be found in the maximal dual simulation. Conversely, if the maximal dual simulation does not cover a candidate, then it is not a match for the query. Thus, if the greatest dual simulation between a query and a database is empty, the result set according to SPARQL is guaranteed to be empty as well. We do not only formally justify correctness and tractability of this final semantics, but also evaluate its effectiveness as a pruning process for SPARQL on large-scale synthetic and real-world datasets. The evaluation of its efficiency is postponed to Chapter 5, where we discuss an implementation of the maximal dual simulation semantics of SPARQL in more depth.

Summarizing, we present two different semantics for SPARQL based on dual simulations. The first semantics is correct and tractable only for the well-designed fragment of SPARQL. The other semantics approximates the matches of SPARQL in such a way that a single match includes all original SPARQL matches.

Outline. We introduce abstract syntax and semantics of SPARQL in Section 4.1, including the formal statements about the complexity of SPARQL from the literature. As a next step (Section 4.2), we take dual simulations to substitute homomorphisms in SPARQL's semantics of basic graph patterns and study the consequences of this change w. r. t. correctness. In Section 4.3, we prove correctness and tractability of the dual simulation semantics for well-designed SPARQL. We develop and evaluate a correct and tractable dual simulation semantics for full SPARQL in Section 4.4. A final discussion and summary in Section 4.5 concludes this chapter.

4.1 Landmark

Since its first release by the W3C as a public working draft in 2004, SPARQL has been rapidly adopted as the standard query language for data on the Semantic Web [13]. Its name "SPARQL" is a recursive acronym and reads as SPARQL *Protocol and RDF Query Language*. Already in 2008, it became a W3C recommendation [114]. In 2013, SPARQL 1.1 has been released [65]. The query language SPARQL, to be denoted by $\mathbb{S}$, contains *triple patterns* and combinations by joins, union, and built-in filter conditions. Throughout the thesis, we will concentrate on these pattern matching capabilities. Our formalization of SPARQL is inspired by the seminal work of Pérez et al. [113], which we adapt to fit in with our notations, as well as further assumptions to be explained throughout the subsequent sections. Following syntax and semantics of SPARQL, we state the basic results concerning the complexity of SPARQL's evaluation and non-emptiness problems from the literature (cf. Section 4.1.2). Consequently, we will be able to rate differ fragments of SPARQL by their tractability.

4.1.1 Graph Pattern Expressions

As for RDF, triples are first-class citizens of SPARQL, now called *triple patterns*. Every RDF triple is a valid triple pattern [65]. Besides the universes of objects $\mathcal{U}$ and properties/predicates $\mathcal{P}$, SPARQL employs a universe of variables $\mathcal{V} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}, \ldots\}$. During the

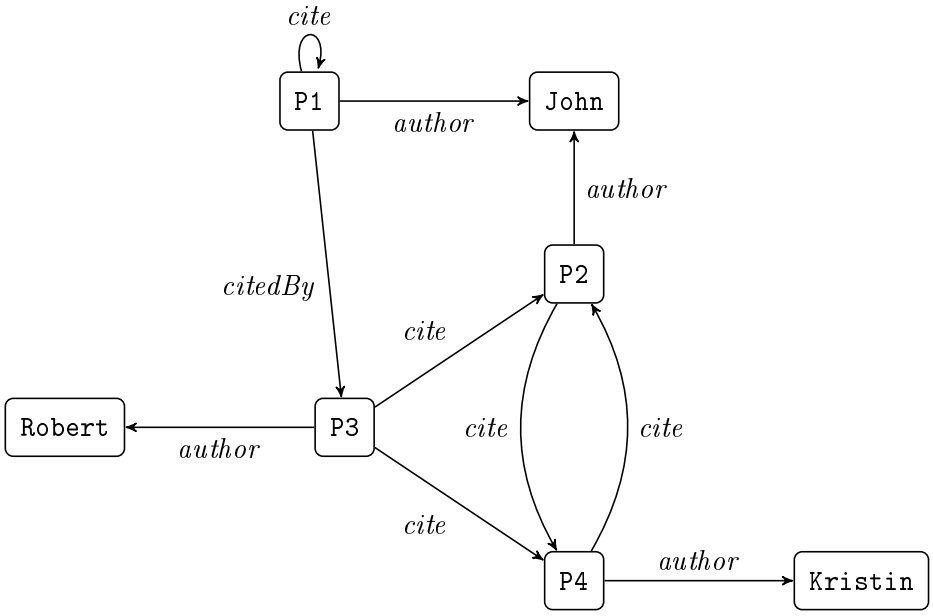


Figure 4.1: An Author-Citation Network

evaluation of SPARQL queries, matches map the variables occurring in a query to actual (database) objects in $\mathcal{U}$. Every component of a triple pattern $t = (s, p, o)$ may be a variable or a constant (i. e., $s, o \in \mathcal{U} \cup \mathcal{V}$ and $p \in \mathcal{P} \cup \mathcal{V}$). Thus, SPARQL's triple patterns follow the general shape of $(\mathcal{U} \cup \mathcal{V}) \times (\mathcal{P} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V})$.

Throughout the next few examples, we consider the graph database $DB_{4.1}$, depicted in Figure 4.1. It represents an author-citation network where nodes `John`, `Robert`, `Kristin` are meant to be authors and `P1`,...,`P4` are the papers written by those authors in an *author*-relationship with one of the author nodes. Furthermore, the *cite*-relation associates papers by the source node's reference list, e. g., triple (`P3`, *cite*, `P4`) means that `P4` occurs in the reference list of `P3`.

Example 4.1 Valid triple patterns are all edges in $DB_{4.1}$, e. g., (`P3`, *author*, `Robert`) or (`P2`, *cite*, `P4`). These two example triple patterns match in $DB_{4.1}$. (`P3`, *cite*, `P1`), however, does not match in $DB_{4.1}$ because there is no such edge.

Let $\mathtt{x}, \mathtt{y}, \mathtt{z} \in \mathcal{V}$ be SPARQL variables. Then by query (`x`, *author*, `John`) we intend to find *all papers `John` has (co-)authored.* In this particular case, two matches would be returned, mapping `x` to `P1` or `P2`. We may also ask for relationships between two nodes, e. g., the match for (`P1`, `x`, `P3`) in $DB_{4.1}$ assigns the predicate *citedBy* to variable `x`. Since variables may occur in every position, we may also ask *What did `P2` do with whom?* by (`P2`, `x`, `y`). The result reflects on the authorship of `John` and the reference to `P4`. Finally, (`x`, `y`, `z`) would simply return all edges from $DB_{4.1}$. ■

Before we go on with more complex constructs, we make a simplifying assumption about the shape of triple patterns: Subjects and objects stem from $\mathcal{V}$ while predicates stem from $\mathcal{P}$, i. e., every triple pattern t is an element of $\mathcal{V} \times \mathcal{P} \times \mathcal{V}$. The semantics we develop applies to more general shapes of triple patterns (cf. Section 4.5), but including all of them unnecessarily expands our proof obligations to more cases that are handled almost identically to the case of $t \in \mathcal{V} \times \mathcal{P} \times \mathcal{V}$. Hence, they only add a practical value that pays out when writing down complex queries. Furthermore, matching simplified triple patterns $t \in \mathcal{V} \times \mathcal{P} \times \mathcal{V}$ better resembles the pattern matching scenario we established in Chapter 3.

Syntactically, SPARQL variables $\mathtt{v} \in \mathcal{V}$ are introduced by a leading question mark, i. e., `?v`. Constants, here predicates from $\mathcal{P}$, are encoded by IRIs, i. e., enclosed in angle

brackets. Triple patterns, or general SPARQL queries, are embedded in modifiers, specifying the result form of the query.

Example 4.2 The query

```
SELECT * WHERE { ?paper <author> ?researcher . }
```

asks for all matches to the triple pattern (paper, *author*, researcher). A match will be a pair identifying concrete substitutions for paper and researcher being in an *author* relationship. The `SELECT`-modifier here takes a list of projection variables, or `*` to denote all variables occurring in the pattern, to instruct the query processor which variable matches to return. Hence, the query

```
SELECT ?paper,?researcher WHERE { ?paper <author> ?researcher . }
```

is equivalent to the one above. The result set of

```
SELECT ?researcher WHERE { ?paper <author> ?researcher . }
```

projects the matches of the first query to the variable researcher. The projection operator from relational algebra has the same effect [1].

The `ASK`-modifier instructs the query processor not to compute and enumerate all the results but to return TRUE if there is any match for the pattern and FALSE otherwise, e. g., query

```
ASK * WHERE { ?paper0 <cites> ?paper . }
```

evaluates to FALSE in $DB_{4.1}$. Since the assumed predicate *cites* is not used in $DB_{4.1}$, triple pattern (paper0, *cites*, paper) cannot be matched. ■

In formal notation, we drop the syntactic conventions on variables as well as the query modifiers. We recognize the modifiers by studying the particular decision problems, i. e., `SELECT *` through the evaluation problem and `ASK *` using the non-emptiness problem. We do not consider projection lists different from `*` in order to keep the focus of our study on pure pattern matching problems for SPARQL. The W3C recommendation lists two more modifiers, the `DESCRIBE`- and the `CONSTRUCT`-modifier, being out of the scope of this thesis because they do not relate to pattern matching capabilities of SPARQL.

Triple patterns, or SPARQL queries in general, may be combined by two join operators: conjunction (inner join) and optional patterns (left outer join). If $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are SPARQL queries, query $\mathcal{Q} = \mathcal{Q}_1$ `AND` $\mathcal{Q}_2$ represents their conjunction. Such queries are meant to return those matches constructed (by $\cup$) from matches to both query parts.

Example 4.3 With regard to $DB_{4.1}$, a query asking for researchers (r) of papers (p) and the papers they cited (s for *source*) may be expressed by the conjunction

$$Q_a = (\mathsf{p}, \mathit{author}, \mathsf{r})\ \texttt{AND}\ (\mathsf{p}, \mathit{cite}, \mathsf{s}).$$

The query's results (w. r. t. $DB_{4.1}$) are summarized in the following table:

matches of Q_a:

p	r	s	
P1	John	P1	} match of Q_b
P2	John	P4	
P3	Robert	P2	
P3	Robert	P4	
P4	Kristin	P2	

Each row entry represents a match of Q_a in $DB_{4.1}$. Note that the first row assigns P1 to both variables, p and s[3]. We observe, SPARQL matches are homomorphisms rather than isomorphisms. The query

$$Q_b = (\mathsf{p}, \mathit{author}, \mathsf{r})\ \texttt{AND}\ (\mathsf{p}, \mathit{cite}, \mathsf{p})$$

returns a single match, as indicated in the table above. The actual match is obtained by ignoring the s-column. ■

Optional patterns $\mathcal{Q} = \mathcal{Q}_1\ \texttt{OPT}\ \mathcal{Q}_2$, on the other hand, are SPARQL's way of handling missing data. If a match for $\mathcal{Q}_1$ can be conjunctively combined with a match of $\mathcal{Q}_2$, the combination is returned. Therefore, every match of $\mathcal{Q}_1\ \texttt{AND}\ \mathcal{Q}_2$ is included in the result set of $\mathcal{Q}$. If a match for $\mathcal{Q}_1$ cannot be combined with any match for $\mathcal{Q}_2$, it is considered a match for $\mathcal{Q}$. This operator naturally handles the Semantic Web assumption, that every application/RDF database has only partial knowledge about its resources [12].

Example 4.4 For instance, finding all authors and their papers and, possibly, papers which cite the current piece may be expressed by query

$$Q_c = (\mathsf{p}, \mathit{author}, \mathsf{r})\ \texttt{OPT}\ (\mathsf{s}, \mathit{cite}, \mathsf{p}).$$

The following table summarizes its results w. r. t. $DB_{4.1}$.

p	r	s
P1	John	P1
P2	John	P3
P2	John	P4
P3	Robert	—
P4	Kristin	P3
P4	Kristin	P4

Note the match of Robert with the non-existent (or NULL) entry for s. This is because P3 is not in any of the other papers' reference lists, but then the query only asks for the citations optionally.

Employing optional patterns, we may also express a form of *preference*. Exhibiting the interpretation of *cite*-self-loops from Example 4.3, we may ask for papers citing papers or their same-titled technical reports by

$$\mathcal{Q}_{\mathsf{pref}} = \underbrace{(\mathsf{p}, \mathit{author}, \mathsf{r})}_{M}\ \texttt{OPT}\ \underbrace{((\mathsf{p}, \mathit{cite}, \mathsf{s})\ \texttt{AND}\ (\mathsf{s}, \mathit{cite}, \mathsf{s}))}_{O_1}\ \texttt{OPT}\ \underbrace{((\mathsf{s}, \mathit{citedBy}, \mathsf{p})\ \texttt{AND}\ (\mathsf{s}, \mathit{cite}, \mathsf{s}))}_{O_2}.$$

This query exhibits the left-associativity of the optional operator [113], trying to evaluate O_1 first together with M. Only if O_1 does not match, O_2 is evaluated. This mutually exclusive matching behavior is reached (w. r. t. $DB_{4.1}$) by using the same variables in both optional parts. Note that between any two connected nodes from {P1, P2, P3, P4}, either *cite* or *citedBy* is used as an edge label. The result to establish for the query is the following.

p	r	s
P1	John	P1
P3	Robert	P1
P4	Kristin	—

[3] Although this example appears a little artificial, it might be explained by an extraction error or the circumstance that conference versions of research papers and their preliminary/extended versions (e. g., technical reports) often share their title.

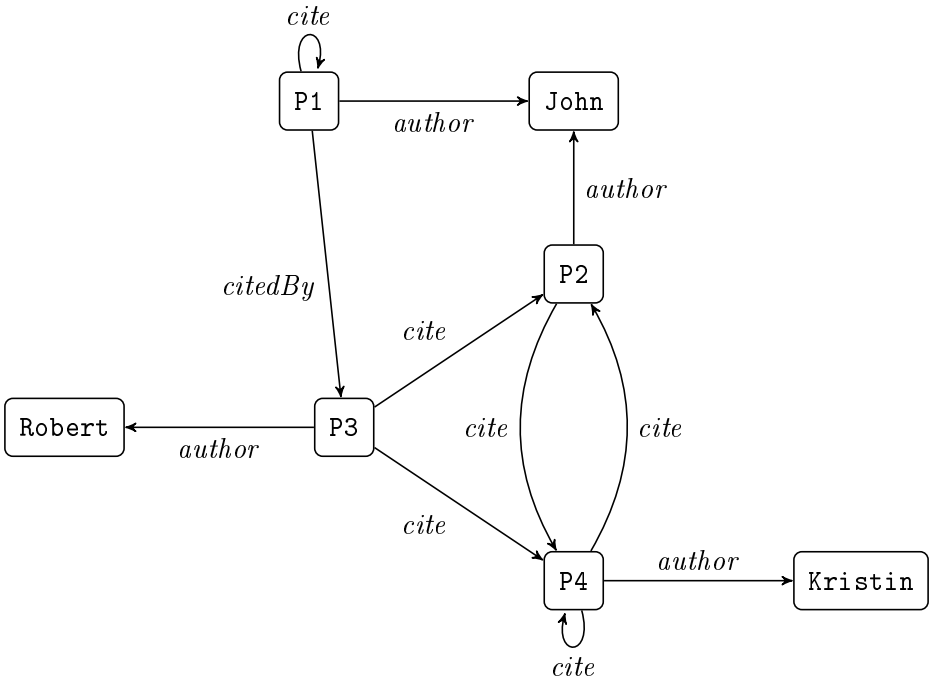


Figure 4.2: An Extended Version of Figure 4.1

Note that if there was a *cite*-self-loop incorporating `P4`, as in Figure 4.2, `P1` would not occur as a match for `s` with `Robert`. Such preference queries represent instances of the missing *monotonicity* of SPARQL, which we make more precise after having introduced SPARQL's semantics formally. ■

Beyond SPARQL's join operators, queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ may be further combined by union to $\mathcal{Q}_u = \mathcal{Q}_1$ `UNION` $\mathcal{Q}_2$, or filtered by *built-in filter conditions* R via $\mathcal{Q}_f = \mathcal{Q}_1$ `FILTER` R. While $\mathcal{Q}_u$ returns the matches from $\mathcal{Q}_1$ and $\mathcal{Q}_2$, $\mathcal{Q}_f$ includes only those matches for $\mathcal{Q}_1$ that satisfy condition R. Such built-in filter conditions may compare the values of variables to each other ($\mathsf{x} = \mathsf{y}$) or to constants ($\mathsf{x} = o$). Furthermore, a filter condition may check whether a match binds a variable, i. e., if a node is assigned to the variable ($\mathsf{bound}(\mathsf{x})$). At last, Boolean connectives ($\wedge$, $\vee$, $\neg$) may be used to combine one or more filter condition.

Example 4.5 Suppose, we look for the papers authored by `Robert`, and their list of references, i. e., those papers that the one written by `Robert` cites. As we know from previous examples, there are two predicates expressing citation relationships, *cite* and *citedBy*. To obtain a complete list, we have to query for both, e. g.,

$$Q_d = (\mathsf{p}, \mathit{author}, \mathsf{r})\ \texttt{AND}\ ((\mathsf{p}, \mathit{cite}, \mathsf{s})\ \texttt{UNION}\ (\mathsf{s}, \mathit{citedBy}, \mathsf{p}))\,.$$

The result to this query does not only contain matches assigning `Robert` to `r`, but every other author in $DB_{4.1}$:

matches of Q_d:

p	r	s	
P3	Robert	P1	matches of Q_e
P3	Robert	P2	
P3	Robert	P4	
P1	John	P1	
P2	John	P4	
P4	Kristin	P2	

A condition, restricting r to Robert, filters those results assigning only Robert to r, as in

$$Q_e = ((\mathsf{p}, \mathit{author}, \mathsf{r})\ \texttt{FILTER}\ \mathsf{r}{=}\texttt{Robert})\ \texttt{AND}\ ((\mathsf{p}, \mathit{cite}, \mathsf{s})\ \texttt{UNION}\ (\mathsf{s}, \mathit{citedBy}, \mathsf{p}))\,.$$

The result in $DB_{4.1}$ obtained from Q_e is indicated in the result table above. Contradicting our syntactic convention for triple patterns, using Robert as a constant instead of r yields the same result. When using a constant in subject and/or object position, i. e., $t = (c_1, a, c_2)$, the triple pattern may easily be rewritten to $(\mathsf{x}_{c_1}, a, \mathsf{x}_{c_2})\ \texttt{FILTER}\ \mathsf{x}_{c_1}{=}c_1 \wedge \mathsf{x}_{c_2}{=}c_2$, a query obeying our syntactic restriction, where $\mathsf{x}_{c_1}, \mathsf{x}_{c_2}$ are globally fresh variables. Hence, t may be seen as a shorthand for the unfolded version using a built-in filter condition. ■

Definition 4.6 (SPARQL Syntax)
The language $\mathbb{S}$ of all SPARQL queries is defined by the following grammar,

$$\mathcal{Q} \quad ::= \quad t \mid \mathcal{Q}\ \texttt{AND}\ \mathcal{Q} \mid \mathcal{Q}\ \texttt{OPT}\ \mathcal{Q} \mid \mathcal{Q}\ \texttt{UNION}\ \mathcal{Q} \mid \mathcal{Q}\ \texttt{FILTER}\ R,$$

where for $\mathsf{x}, \mathsf{y} \in \mathcal{V}$ and $a \in \mathcal{P}$, $t = (\mathsf{x}, a, \mathsf{y})$ is a triple pattern and R a *built-in filter condition*, drawn from the inductively defined set of all built-in fulter conditions:

(1) If $\mathsf{x}, \mathsf{y} \in \mathcal{V}$ and $o \in \mathcal{U}$, then $\mathsf{bound}(\mathsf{x})$, $\mathsf{x}{=}o$, and $\mathsf{x}{=}\mathsf{y}$ are built-in filter conditions.

(2) If R and S are built-in filter conditions, then $\neg R$, $R \vee S$, and $R \wedge S$ are built-in filter conditions.

Operator OPT is left-associative, i. e., $\mathcal{Q}_1\ \texttt{OPT}\ \mathcal{Q}_2\ \texttt{OPT}\ \mathcal{Q}_3 = (\mathcal{Q}_1\ \texttt{OPT}\ \mathcal{Q}_2)\ \texttt{OPT}\ \mathcal{Q}_3$. ▲

In $\mathcal{Q} = \mathcal{Q}_1 \theta \mathcal{Q}_2$ ($\theta \in \{\texttt{AND}, \texttt{OPT}, \texttt{UNION}\}$), queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are called *clauses of* $\mathcal{Q}$. In the special case of optional patterns $\mathcal{Q} = \mathcal{Q}_1\ \texttt{OPT}\ \mathcal{Q}_2$, $\mathcal{Q}_1$ is further differentiated as the *mandatory clause of* $\mathcal{Q}$ while $\mathcal{Q}_2$ is the *optional clause of* $\mathcal{Q}$.

To formally justify the indicated matches in Examples 4.1 to 4.5 we need to give a definition of what a querying process shall return for a query $\mathcal{Q} \in \mathbb{S}$ w. r. t. a graph database $DB = (O_{DB}, \Sigma, E_{DB})$. *Candidate matches*, or *candidates* for short, for SPARQL queries are partial functions $\mu : \mathcal{V} \hookrightarrow \mathcal{U}$, i. e., candidates assign objects to variables. $\mathit{dom}(\mu)$ denotes the set of all variables $\mathsf{x} \in \mathcal{V}$ for which $\mu(\mathsf{x})$ is defined. To match a triple pattern $t = (\mathsf{x}, a, \mathsf{y})$, we need to find substitutes $o_x, o_y \in O_{DB}$ for x, y, such that $(o_x, a, o_y) \in E_{DB}$, justifying the *match* $\mu = \{(\mathsf{x}, o_x), (\mathsf{y}, o_y)\}$. Formally, by $\mathit{vars}(t)$ we denote the set of variables occurring in triple pattern t, i. e., $\mathit{vars}(t) = \{\mathsf{x}, \mathsf{y}\}$ for $t = (\mathsf{x}, a, \mathsf{y})$. A candidate μ is a *match for triple pattern* t *in* DB iff $\mathit{dom}(\mu) = \mathit{vars}(t)$ and, assuming $t = (\mathsf{x}, a, \mathsf{y})$, $(\mu(\mathsf{x}), a, \mu(\mathsf{y})) \in E_{DB}$, abbreviated by $\mu(t) \in DB$.

Example 4.7 Reconsider the triple pattern $t = (\mathsf{p}, \mathit{author}, \mathsf{r})$ from Example 4.2. The partial functions $\mu_1, \mu_2, \mu_3, \mu_4$ are all matches for t in $DB_{4.1}$:

$$\begin{array}{lllllll} \mu_1 & = & \{(\mathsf{r}, \texttt{John}), (\mathsf{p}, \texttt{P1})\} & & \mu_2 & = & \{(\mathsf{r}, \texttt{John}), (\mathsf{p}, \texttt{P2})\} \\ \mu_3 & = & \{(\mathsf{r}, \texttt{Robert}), (\mathsf{p}, \texttt{P3})\} & & \mu_4 & = & \{(\mathsf{r}, \texttt{Kristin}), (\mathsf{p}, \texttt{P4})\} \end{array}$$

For all $i \in \{1, 2, 3, 4\}$, $\mathit{dom}(\mu_i) = \{\mathsf{r}, \mathsf{p}\} = \mathit{vars}(t)$. Each μ_i ($i \in \{1, 2, 3, 4\}$) represents a row of the indicated result table.

$\nu_1 = \mu_1 \cup \{(\mathsf{s}, \texttt{P3})\}$ is not a match for t as $\mathit{dom}(\nu_1)$ and $\mathit{vars}(t)$ are not equal, i. e., $\mathit{dom}(\nu_1) = \{\mathsf{r}, \mathsf{p}, \mathsf{s}\} \neq \mathit{vars}(t)$. $\nu_2 = \{(\mathsf{r}, \texttt{John}), (\mathsf{p}, \texttt{P3})\}$ is not a match for t because $(\texttt{P3}, \mathit{author}, \texttt{John})$ is not an edge of $DB_{4.1}$. ■

As a notational convention, we may specify partial functions by in-line lists of assignments, e. g., μ_1 from Example 4.7 by $(\mathsf{r} \mapsto \texttt{John}, \mathsf{p} \mapsto \texttt{P1})$.

The conjunction $\mathcal{Q}_1\ \texttt{AND}\ \mathcal{Q}_2$ joins pairwise *compatible matches* of $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Two candidates $\mu_1, \mu_2 : \mathcal{V} \hookrightarrow \mathcal{U}$ are *compatible*, denoted by $\mu_1 \rightleftarrows \mu_2$, iff for all variables v shared

between μ_1 and μ_2, i. e., $v \in dom(\mu_1) \cap dom(\mu_2)$, μ_1 and μ_2 agree on their assignment, i. e., $\mu_1(v) = \mu_2(v)$. Let μ_1 be a match for $\mathcal{Q}_1$ and μ_2 a match for $\mathcal{Q}_2$ in some database DB. Then $\mu_1 \cup \mu_2$ is a match for $\mathcal{Q}_1$ AND $\mathcal{Q}_2$ in DB iff μ_1 and μ_2 are compatible. As indicated in Example 4.4, optional patterns $\mathcal{Q}_1$ OPT $\mathcal{Q}_2$ are evaluated as if they were conjunctions $\mathcal{Q}_1$ AND $\mathcal{Q}_2$. Additionally, those matches μ for $\mathcal{Q}_1$ are returned, which cannot be compatibly extended by matches for $\mathcal{Q}_2$.

Example 4.8 To obtain a result for query Q_a from Example 4.3,

$$Q_a = \underbrace{(\mathsf{p}, \mathit{author}, \mathsf{r})}_{t_1} \texttt{ AND } \underbrace{(\mathsf{p}, \mathit{cite}, \mathsf{s})}_{t_2},$$

we take a look at the matches for its two triple patterns, e. g., μ_1 ($\mathsf{p} \mapsto \texttt{P1}, \mathsf{r} \mapsto \texttt{John}$) for t_1 in $DB_{4.1}$ and μ_2 ($\mathsf{p} \mapsto \texttt{P1}, \mathsf{s} \mapsto \texttt{P1}$) for t_2. Matches μ_1 and μ_2 are compatible since $dom(\mu_1) \cap dom(\mu_2) = \{\mathsf{p}\}$ and $\mu_1(\mathsf{p}) = \texttt{P1} = \mu_2(\mathsf{p})$. μ_2' ($\mathsf{p} \mapsto \texttt{P3}, \mathsf{s} \mapsto \texttt{P2}$) is incompatible with μ_1 (as $\mu_1(\mathsf{p}) = \texttt{P1} \neq \texttt{P3} = \mu_2'(\mathsf{p})$). Hence, $\mu_1 \cup \mu_2$ is a match for the conjunction above, while $\mu_1 \cup \mu_2'$ is not a match.

For the optional pattern

$$Q_c = (\mathsf{p}, \mathit{author}, \mathsf{r}) \texttt{ OPT } (\mathsf{s}, \mathit{cite}, \mathsf{p})$$

from Example 4.4, not only compatible matches to the triple patterns make up the result, but also partial matches only considering the left side are included. For instance, μ ($\mathsf{r} \mapsto \texttt{Robert}, \mathsf{p} \mapsto \texttt{P3}$) matches the mandatory clause of Q_c in $DB_{4.1}$. There is, however, no compatible match for the respective optional clause. Thus, μ is a match of Q_c in $DB_{4.1}$ (cf. Example 4.4).

As a last example, let us reconsider our preference query,

$$\mathcal{Q}_{\mathsf{pref}} = \underbrace{(\mathsf{p}, \mathit{author}, \mathsf{r})}_{M} \texttt{ OPT } \underbrace{((\mathsf{p}, \mathit{cite}, \mathsf{s}) \texttt{ AND } (\mathsf{s}, \mathit{cite}, \mathsf{s}))}_{O_1} \texttt{ OPT } \underbrace{((\mathsf{s}, \mathit{citedBy}, \mathsf{p}) \texttt{ AND } (\mathsf{s}, \mathit{cite}, \mathsf{s}))}_{O_2}.$$

Consider now $DB_{4.2}$, depicted in Figure 4.2. From the mandatory clause, we obtain the match μ ($\mathsf{r} \mapsto \texttt{Robert}, \mathsf{p} \mapsto \texttt{P3}$) and both optional clauses yield matches that are compatible with μ, i. e., μ_1 ($\mathsf{p} \mapsto \texttt{P3}, \mathsf{s} \mapsto \texttt{P4}$) for O_1 and μ_2 ($\mathsf{p} \mapsto \texttt{P3}, \mathsf{s} \mapsto \texttt{P1}$) for O_2. However, only $\mu \cup \mu_1$ is considered a match. Recall that optional patterns are left-associative, i. e., in order to include μ_2 in a match with μ, it must be compatible with $\mu \cup \mu_1$. But $(\mu \cup \mu_1)(\mathsf{s}) = \texttt{P4} \neq \texttt{P1} = \mu_2(\mathsf{s})$. Thus, $\mu \cup \mu_1 \not\rightleftarrows \mu_2$. ■

Unions, i. e., $\mathcal{Q} = \mathcal{Q}_1$ UNION $\mathcal{Q}_2$ simply consider all matches for $\mathcal{Q}_1$ and those for $\mathcal{Q}_2$ as matches for $\mathcal{Q}$. Hence, if μ is a match for $\mathcal{Q}_i$ ($i \in \{1, 2\}$), then μ is a match for $\mathcal{Q}$. The validity of built-in filter conditions is evaluated match-wise, i. e., if μ is a match for $\mathcal{Q}$, then μ is a match for $\mathcal{Q}$ FILTER R iff μ satisfies R, denoted $\mu \models R$. The evaluation of $\mu \models R$ is defined in the subsequent semantics of SPARQL.

Definition 4.9 (SPARQL Semantics)
Let $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database. The SPARQL *semantics of a query* $\mathcal{Q} \in \mathbb{S}$ *w. r. t.* DB, denoted $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$, is defined inductively on the structure of $\mathcal{Q}$:

$$\begin{aligned}
[\![t]\!]^{\mathbb{S}}_{DB} &:= \{\mu : \mathcal{V} \hookrightarrow \mathcal{U} \mid dom(\mu) = vars(t) \wedge \mu(t) \in DB\} \\
[\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB} &:= \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB} \wedge \mu_2 \in [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB} \wedge \mu_1 \rightleftarrows \mu_2\} \\
[\![\mathcal{Q}_1 \texttt{ OPT } \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB} &:= [\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB} \cup \\
&\quad \{\mu_1 \in [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB} \mid \forall \mu_2 \in [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB} : \mu_1 \not\rightleftarrows \mu_2\} \\
[\![\mathcal{Q}_1 \texttt{ UNION } \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB} &:= [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB} \cup [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}
\end{aligned}$$

Let R be a built-in filter condition and $\mu : \mathcal{V} \hookrightarrow \mathcal{U}$ a candidate. $\mu \models R$ iff

- $R = \mathsf{bound}(\mathsf{x})$ implies $\mathsf{x} \in dom(\mu)$,
- $R = \mathsf{x}{=}o$ implies $\mathsf{x} \in dom(\mu)$ and $\mu(\mathsf{x}) = o$,
- $R = \mathsf{x}{=}\mathsf{y}$ implies $\mathsf{x}, \mathsf{y} \in dom(\mu)$ and $\mu(\mathsf{x}) = \mu(\mathsf{y})$,
- $R = \neg R_1$ implies $\mu \not\models \psi$,
- $R = R_1 \vee R_2$ implies $\mu \models R_1$ or $\mu \models R_2$, and
- $R = R_1 \wedge R_2$ implies $\mu \models R_1$ and $\mu \models R_2$.

Then $[\![\mathcal{Q}\ \texttt{FILTER}\ R]\!]^{\mathbb{S}}_{DB} := \{\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \mid \mu \models R\}$. ▲

Based on the semantics $[\![\cdot]\!]^{\mathbb{S}}_{\_}$, operators `AND` and `UNION` inherit commutativity, associativity, and distributivity from their logical counterparts ($\wedge$ and $\vee$).

Proposition 4.10 (Lemma 2.5 [113]) *Operators* `AND` *and* `UNION` *are commutative and associative. Furthermore,* `AND` *distributes over* `UNION`.

Commutativity and associativity of `AND` justify the notion of *basic graph patterns*. A basic graph pattern is a set of triple patterns $\mathbb{G} \subseteq \mathcal{V} \times \mathcal{P} \times \mathcal{V}$ and is interpreted as shorthand for the conjunction of all $t \in \mathbb{G}$. Function *vars* extends to BGPs by

$$vars(\mathbb{G}) := \bigcup_{t \in \mathbb{G}} vars(t). \tag{4.1}$$

Intuitively, μ is a match of $\mathbb{G}$ iff μ is a match of all triple patterns $t \in \mathbb{G}$. More formally, let $V \subseteq \mathcal{V}$. Then $\mu{\restriction_V}\colon V \hookrightarrow \mathcal{U}$ is the partial function μ restricted to input variables $v \in V$, i. e.,

$$\mu{\restriction_V}(w) := \begin{cases} \mu(w) & \text{if } w \in V \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

Thus, μ is a match for BGP $\mathbb{G}$ iff for all $t \in \mathbb{G}$, $\mu{\restriction_{vars(t)}}$ is a match for t.

Every BGP $\mathbb{G}$ can be interpreted as a graph (pattern) $G(\mathbb{G}) := (V_{\mathbb{G}}, \Sigma, \mathbb{G})$ by taking the set of variables occurring in $\mathbb{G}$ as set of nodes, i. e., $V_{\mathbb{G}} = \{\mathsf{v}, \mathsf{w} \mid (\mathsf{v}, a, \mathsf{w}) \in \mathbb{G}\}$. Based on this interpretation, we easily see that matches for basic graph patterns $\mathbb{G}$ in DB embed graph homomorphisms between $G(\mathbb{G})$ and DB.

Proposition 4.11 *Let DB be a graph database, $\mathbb{G}$ a basic graph pattern, and $\mu \in [\![\mathbb{G}]\!]^{\mathbb{S}}_{DB}$. $\mu{\restriction_{vars(\mathbb{G})}}$ is a graph homomorphism between $G(\mathbb{G})$ and DB.*

Proof: First note that $\mu{\restriction_{vars(\mathbb{G})}}$ is a function from $vars(\mathbb{G})$ to $\mathcal{U}$. Let $\mathsf{v} \in vars(\mathbb{G})$. Then there is a triple pattern $t \in \mathbb{G}$ with $\mathsf{v} \in vars(t)$. Since μ is a match for $\mathbb{G}$ in DB, $\mu{\restriction_{vars(t)}}$ is a match for t in DB. Hence, $vars(t) = dom(\mu)$ and $\mu(\mathsf{v})$ is defined.

It remains to be shown that $\mu{\restriction_{vars(\mathbb{G})}}$ is a homomorphism between $G(\mathbb{G})$ and DB. Therefore, suppose there is an edge (v, a, w) in $G(\mathbb{G})$. As the edge relation of $G(\mathbb{G})$ is the set of triple patterns $\mathbb{G}$, $(v, a, w) \in \mathbb{G}$. Following the arguments above, $\mu{\restriction_{\{v,w\}}}$ is a match for (v, a, w). Thus, there is an edge $(\mu(v), a, \mu(w))$ in DB. As this argument carries over to all edges of $G(\mathbb{G})$, $\mu{\restriction_{vars(\mathbb{G})}}$ is a graph homomorphism between $G(\mathbb{G})$ and DB. Q. E. D.

As BGPs are solely constructed by conjunctions of triple patterns, the Sparql fragment of basic graph patterns is denoted by $\mathbb{S}_{\mathsf{A}}$. In general, the Sparql fragments using `AND`, `UNION`, `OPT`, or `FILTER` are indicated by subscripts A, U, O, or F of the language symbol $\mathbb{S}$. Each of these syntactic fragments of $\mathbb{S}$ forms its own query language fragment., e. g., $(\mathbb{S}_{\mathsf{A}}, [\![\cdot]\!]^{\mathbb{S}}_{\_})$ or $(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathbb{S}}_{\_})$.

Before diving into the complexity of SPARQL, let us discuss two *normal forms* of SPARQL. The first tackles the issue of non-monotonicity of SPARQL's optional patterns. A query $\mathcal{Q} \in \mathbb{S}$ is *monotone* iff for all pairs of graph databases DB, DB' with $DB \subseteq DB'$, we do not lose information inbetween $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$ and $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB'}$. *Strong monotonicity* requires $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \subseteq [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB'}$. Requiring subsumption ($\sqsubseteq$) between $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$ and $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB'}$ yields the notion of *weak monotonicity* [14]:

$$[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \sqsubseteq [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB'} :\Leftrightarrow \forall \mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \exists \mu' \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB'} : \mu \subseteq \mu'.$$

Note that strong monotonicity implies weak monotonicity. Optional patterns are not even weakly monotone [14].

Example 4.12 An example query is $\mathcal{Q}_{\mathsf{pref}}$ (cf. Example 4.8). While in $DB_{4.1}$, $\mathcal{Q}_{\mathsf{pref}}$ contains the result μ_0 ($\mathsf{r} \mapsto \mathtt{Robert}, \mathsf{p} \mapsto \mathtt{P3}, \mathsf{s} \mapsto \mathtt{P1}$), there is no such match in $DB_{4.2}$. ■

A class of queries syntactically excluding non-monotonicity is *well-designed* SPARQL [113, 14], denoted $\mathbb{S}_{\mathsf{wd}}$. In a well-designed query $\mathcal{Q}$, for every syntactic occurrence of an optional pattern $\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$ in $\mathcal{Q}$ and every variable $v \in \mathit{vars}(\mathcal{Q}_2) \setminus \mathit{vars}(\mathcal{Q}_1)$ (i. e., a variable solely belonging to the optional clause of $\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$), it holds that v does not occur outside of $\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$. Formally, a query $P \in \mathbb{S}$ is a *subpattern of query* $\mathcal{Q} \in \mathbb{S}$ iff $\mathcal{Q} = P$ or

- $\mathcal{Q} = \mathcal{Q}_1 \,\theta\, \mathcal{Q}_2$ ($\theta \in \{\mathtt{AND}, \mathtt{OPT}, \mathtt{UNION}\}$) implies P is a subpattern of $\mathcal{Q}_1$ or $\mathcal{Q}_2$, or
- $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{FILTER}\, R$ implies P is a subpattern of $\mathcal{Q}_1$.

Example 4.13 All queries so far, except for $\mathcal{Q}_{\mathsf{pref}}$, are well-designed queries. The subpatterns of $\mathcal{Q}_{\mathsf{pref}}$ are indicated in Example 4.8 as M, O_1, O_2. $\mathcal{Q}_{\mathsf{pref}}$ is not well-designed because $\mathsf{s} \in \mathit{vars}(O_1) \setminus \mathit{vars}(M)$ but s also belongs to $\mathit{vars}(O_2)$. ■

Well-designed SPARQL includes union-free SPARQL queries [113]. Although built-in filter conditions may be integrated [113, 72], they are not needed as part of well-designed SPARQL throughout the rest of this thesis. The following definition summarizes the characteristics mentioned above of well-designed SPARQL.

Definition 4.14 (Well-Designed SPARQL)
The language $\mathbb{S}_{\mathsf{wd}}$ of *well-designed* SPARQL includes all queries $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ with the property that for every subpattern $P = P_1 \,\mathtt{OPT}\, P_2$ of $\mathcal{Q}$ and all variables $v \in \mathit{vars}(P_2) \setminus \mathit{vars}(P_1)$, every subpattern P' of $\mathcal{Q}$ with $v \in \mathit{vars}(P')$ is a subpattern of P_2. ▲

Proposition 4.15 (Theorem 4.3 [14]) *Every well-designed query $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}$ is weakly monotone.*

Beyond well-designed queries, we can also syntactically rule out unions. Therefore, recall that `AND` is distributive over `UNION` from Proposition 4.10.

Example 4.16 Reconsider query

$$Q_d = (\mathsf{p}, \mathit{author}, \mathsf{r}) \,\mathtt{AND}\, ((\mathsf{p}, \mathit{cite}, \mathsf{s}) \,\mathtt{UNION}\, (\mathsf{s}, \mathit{citedBy}, \mathsf{p}))$$

from Example 4.5. Exploiting Proposition 4.10 yields the equivalent query

$$\widehat{Q_d} = ((\mathsf{p}, \mathit{author}, \mathsf{r}) \,\mathtt{AND}\, (\mathsf{p}, \mathit{cite}, \mathsf{s})) \,\mathtt{UNION}\, ((\mathsf{p}, \mathit{author}, \mathsf{r}) \,\mathtt{AND}\, (\mathsf{s}, \mathit{citedBy}, \mathsf{p}))\,.$$

Note that the final query is a union of union-free subpatterns. ■

Pérez et al. collected similar rules for the other operators of SPARQL, culminating in the following normal form.

Proposition 4.17 ([113]) *Let $\mathcal{Q} \in \mathbb{S}$. Then there is a query $\widehat{\mathcal{Q}} = \mathcal{Q}_1$ UNION $\mathcal{Q}_2$ UNION ... UNION $\mathcal{Q}_k$, where each $\mathcal{Q}_i$ ($1 \leq i \leq k$) is union-free, i. e., $\mathcal{Q}_i \in \mathbb{S}_{\mathsf{AOF}}$, and $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} = [\![\widehat{\mathcal{Q}}]\!]^{\mathbb{S}}_{DB}$ for every graph database DB.*

The construction of query $\widehat{\mathcal{Q}}$ is reminiscent of the one aiming for the *disjunctive normal form* of Boolean logic [124]. Thus, compared to the input query $\mathcal{Q}$, $\widehat{\mathcal{Q}}$ quite easily is of exponential size in the length of $\mathcal{Q}$. The resulting query may, however, be used to parallelize query evaluation. Since all of the k query parts, $\mathcal{Q}_1$ to $\mathcal{Q}_k$, are independent, they may be evaluated independently.

4.1.2 Complexity of SPARQL

For SPARQL, i. e., the query language $(\mathbb{S}, [\![\cdot]\!]^{\mathbb{S}}_{\_})$, the domain of matches $\mathbf{D}$ (cf. beginning of this chapter) is the set of all partial functions $\mu : \mathcal{V} \hookrightarrow \mathcal{U}$. A reduction from the problem of *satisfiability of quantified Boolean formulas* attests SPARQL an intractable evaluation problem.

Proposition 4.18 ([113, 121]) EVALUATION$(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathbb{S}}_{_})$ *is co*NP*-complete. For all* X *with* $\{\mathsf{O}\} \subseteq \mathsf{X} \subseteq \{\mathsf{A}, \mathsf{O}, \mathsf{U}, \mathsf{F}\}$, EVALUATION$(\mathbb{S}_{\mathsf{X}}, [\![\cdot]\!]^{\mathbb{S}}_{_})$ *is* PSPACE*-complete.*

Thus, verifying candidate matches of optional patterns or queries containing optional patterns is as hard as deciding inclusion of regular languages [124]. Evaluation remains intractable, even if we restrict our queries to the well-designed case. Verification of candidates for basic graph patterns (including built-in filter conditions) may be done in PTIME. SPARQL fragments containing the UNION operator but no optional patterns still have an NP-complete evaluation problem. For instance, specific queries in $\mathbb{S}_{\mathsf{AUF}}$ may be used to let a query processor solve the *satisfiability problem of Boolean formulas* [113].

Proposition 4.19 ([113, 121]) EVALUATION$(\mathbb{S}_{\mathsf{AF}}, [\![\cdot]\!]^{\mathbb{S}}_{_})$ *is in* PTIME. *For all* X *with* $\{\mathsf{U}\} \subseteq \mathsf{X} \subseteq \{\mathsf{A}, \mathsf{U}, \mathsf{F}\}$, EVALUATION$(\mathbb{S}_{\mathsf{X}}, [\![\cdot]\!]^{\mathbb{S}}_{_})$ *is* NP*-complete.*

Although Proposition 4.17 allows us to get rid of the union operator for query evaluation, the price for the necessary construction reflects on the NP-completeness of SPARQL queries with UNION (cf. end of last section).

At least w. r. t. the evaluation problem we consider $\mathbb{S}_{\mathsf{AO}}$ an interesting fragment of SPARQL since its evaluation problem is already PSPACE-complete. Furthermore, $\mathbb{S}_{\mathsf{wd}}$ is defined in terms of $\mathbb{S}_{\mathsf{AO}}$ (cf. Definition 4.14), a focus on this SPARQL fragment will make our results comparable. During the rest of this chapter, we primarily concentrate on $\mathbb{S}_{\mathsf{AO}}$ and postpone unions and built-in filter conditions to Chapter 5 (in particular, Section 5.3.3), where we implement a dual simulation method for full SPARQL.

Due to the interpretation of basic graph patterns $\mathcal{Q} \in \mathbb{S}_{\mathsf{A}}$ and the matches of $\mathcal{Q}$ in Proposition 4.11, we directly obtain NP-completeness of the non-emptiness problem due to NP-completeness of the subgraph homomorphism problem [68].

Proposition 4.20 NONEMPTY$(\mathbb{S}_{\mathsf{A}}, [\![\cdot]\!]^{\mathbb{S}}_{_})$ *is* NP*-complete.*

PROOF: Let DB be a graph database and $\mathbb{G} \in \mathbb{S}_{\mathsf{A}}$. By Proposition 4.11, matches $\mu \in [\![\mathbb{G}]\!]^{\mathbb{S}}_{DB}$ are essentially homomorphisms, by $\mu \restriction_{vars(\mathbb{G})}$, between $G(\mathbb{G})$ and DB. Thus, $[\![\mathbb{G}]\!]^{\mathbb{S}}_{DB} \neq \emptyset$ iff there is a homomorphism between $G(\mathbb{G})$ and DB. The problem of non-emptiness of homomorphisms between two graphs is NP-complete [68]. Q. E. D.

Thus, non-emptiness of the full query language of SPARQL is at least NP-complete. We conjecture that the problem has an even worse complexity in case of $\mathbb{S}_{\mathsf{O}}$ for the following reason: We would use an *oracle* guessing a candidate μ and then compute in PSPACE

(cf. Proposition 4.18) whether μ is an actual match. By *Savitch's Theorem* [120], nondeterministic PSPACE is equal to deterministic PSPACE. Thus, the non-emptiness problem of SPARQL has a PSPACE upper bound. A lower bound needs to be established.

4.2 The Dual Simulation Semantics of SPARQL

We conclude that $(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathbb{S}}_{\_})$ is intractable (cf. Propositions 4.18 and 4.20). At least w. r. t. the non-emptiness problem of basic graph patterns we have mentioned a tractable alternative multiple times, namely *dual simulations*. We subsequently apply dual simulations (cf. Definition 2.23) to SPARQL queries and analyze how they interact with SPARQL's compatibility notion in Sections 4.2.1 and 4.2.2. We obtain two results from this study. First, we justify and formally define the notion of *correctness* of an alternative semantics for SPARQL. Second, we conclude that the dual simulation semantics for $\mathbb{S}_{\mathsf{AO}}$ is incorrect. In particular, matches, according to SPARQL's semantics, get lost under dual simulation. However, restricted to well-designed SPARQL, the dual simulation semantics is correct and tractable (Section 4.3). A correct and tractable resolution for full SPARQL is developed and evaluated in Section 4.4.

4.2.1 Basic Graph Patterns

As we have seen in Proposition 4.11, every basic graph pattern $\mathbb{G}$ describes a graph pattern $G(\mathbb{G})$ and matches up to $[\![\cdot]\!]^{\mathbb{S}}_{DB}$ essentially are graph homomorphisms between $G(\mathbb{G})$ and DB. The *dual simulation semantics of* SPARQL, $[\![\cdot]\!]^{\mathsf{DS}}_{DB}$, shall take a query $\mathcal{Q} \in \mathbb{S}_{\mathsf{A}}$ and return the set of all dual simulations between $\mathcal{Q}$ and DB. Thus, canonically applying the principles of Definition 2.23 yields a dual simulation interpretation of basic graph patterns. Dual simulation candidates and matches will follow the generalized shape of SPARQL's candidates, i. e., $S \subseteq \mathcal{V} \times \mathcal{U}$. Therefore, function *dom* naturally extends to dual simulation candidates by $dom(S) := \{v \in \mathcal{V} \mid (v, o) \in S\}$.

Definition 4.21 (Dual Simulations of BGPs)
Let $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database. A *dual simulation between a triple pattern* $t = (\mathsf{x}, a, \mathsf{y})$ *and* DB is a relation $S \subseteq \mathcal{V} \times \mathcal{U}$, such that $dom(S) = \{\mathsf{x}, \mathsf{y}\}$ and

(1) for all $(\mathsf{x}, o_1) \in S$ exists an $o_2 \in O_{DB}$ with $o_1 \; E^a_{DB} \; o_2$ and $(\mathsf{y}, o_2) \in S$, and

(2) for all $(\mathsf{y}, o_1) \in S$ exists an $o_2 \in O_{DB}$ with $o_2 \; E^a_{DB} \; o_1$ and $(\mathsf{x}, o_2) \in S$.

The set of all dual simulations between t and DB is denoted by $[\![t]\!]^{\mathsf{DS}}_{DB}$.

A *dual simulation between* $\mathbb{G} \in \mathbb{S}_{\mathsf{A}}$ *and* DB is a relation $S \subseteq \mathcal{V} \times \mathcal{U}$, such that $dom(S) = vars(\mathbb{G})$ and for every $t \in \mathbb{G}$, $S \cap (vars(t) \times \mathcal{U}) \in [\![t]\!]^{\mathsf{DS}}_{DB}$. The set of all dual simulations between $\mathbb{G}$ and DB is denoted by $[\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$. ▲

Dual simulations between triple/basic graph patterns and graph databases are called *dual simulation matches*. By definition, dual simulation matches are non-empty. Furthermore, the dual simulation semantics of BGPs inherits every characteristic of dual simulations we addressed for graph patterns in Chapters 2 and 3. In fact, an equally tight connection between dual simulation matches for basic graph patterns and dual simulations for graph patterns as between SPARQL matches and graph homomorphisms (cf. Proposition 4.11) exists.

Proposition 4.22 *Let DB be a graph database and $\mathbb{G}$ a basic graph pattern. $S \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$ iff S is a non-empty dual simulation between $G(\mathbb{G})$ and DB.*

PROOF: Let $DB = (O_{DB}, \Sigma, E_{DB})$ and $G(\mathbb{G}) = (V, \Sigma, E)$. A dual simulation match $S \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$ is non-empty by Definition 4.21 and $S{\restriction_{vars(t)}} \in [\![t]\!]^{\mathsf{DS}}_{DB}$ for every $t \in \mathbb{G}$. We need to show that for every edge $(v, a, w) \in E$ and every pair $(v, o) \in S$, it holds that there is an $o' \in O_{DB}$ with $o\ E^a_{DB}\ o'$ and $(w, o') \in S$. Recall that $E = \mathbb{G}$, i. e., (v, a, w) is a triple pattern t of $\mathbb{G}$. As $S{\restriction_{vars(t)}} \in [\![t]\!]^{\mathsf{DS}}_{DB}$, $(v, o) \in S$ implies an $o_2 \in O_{DB}$ with $o\ E^a_{DB}\ o_2$ and $(w, o_2) \in S$ by Definition 4.21. Hence, if we set o' to such an o_2, we complete the proof for this case The case $(u, a, v) \in E$ and $(v, o) \in S$ uses (2) of Definition 4.21, but otherwise, exactly the same arguments. Thus, S is in fact a non-empty dual simulation between $G(\mathbb{G})$ and DB.

The converse direction is shown analogously. Here, we use the fact that any triple pattern $(\mathsf{x}, a, \mathsf{y}) \in \mathbb{G}$ is an edge of $G(\mathbb{G})$ by construction. Thus, when considering $(\mathsf{x}, a, \mathsf{y}) \in \mathbb{G}$ and $(\mathsf{x}, o_1) \in S$, the existence of an $o_2 \in O_{DB}$ with $o_1\ E^a_{DB}\ o_2$ and $(\mathsf{y}, o_2) \in S$ follows from the fact that S is a dual simulation between $G(\mathbb{G})$ and DB. The same arguments hold for the case of $(\mathsf{y}, o_1) \in S$. Q. E. D.

Example 4.23 We reconsider $DB_{4.1}$ from Figure 4.1. For triple pattern $t = (\mathsf{p}, \mathit{cite}, \mathsf{s})$, all the SPARQL matches for t are dual simulation matches, e. g., $S_1 = \{(\mathsf{p}, \mathtt{P3}), (\mathsf{s}, \mathtt{P2})\}$ or $S_2 = \{(\mathsf{p}, \mathtt{P3}), (\mathsf{s}, \mathtt{P4})\}$, but also unions thereof, e. g., $S_1 \cup S_2$. In general, we find for triple patterns and basic graph patterns that unions of dual simulation matches constitute matches up to dual simulation.

Next, consider the BGP $\mathbb{G} = \{(\mathsf{p}, \mathit{cite}, \mathsf{p}), (\mathsf{p}, \mathit{author}, \mathsf{r})\}$. While the SPARQL semantics has only a single match in $DB_{4.1}$, namely $\mu\ (\mathsf{p} \mapsto \mathtt{P1}, \mathsf{r} \mapsto \mathtt{John})$, dual simulations may expand onto the self-loop including

$$S = \{(\mathsf{p}, \mathtt{P4}), (\mathsf{p}, \mathtt{P2}), (\mathsf{r}, \mathtt{Kristin}), (\mathsf{r}, \mathtt{John})\}$$

as a dual simulation match for $\mathbb{G}$. There is no dual simulation match S' with $(\mathsf{p}, \mathtt{P3}) \in S'$ since there is no *cite*-labeled edge to P3 in $DB_{4.1}$, required for dual simulating triple pattern $(\mathsf{p}, \mathit{cite}, \mathsf{p})$. ■

Although $(\mathbb{S}_\mathsf{A}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$ clearly changes the matching quality, due to the *bounded cycle problem* (cf. Section 3.1.4), it is a conservative alternative w. r. t. $[\![\cdot]\!]^{\mathbb{S}}_{\_}$.

Lemma 4.24 *For BGP $\mathbb{G} \in \mathbb{S}_\mathsf{A}$ and graph database DB, $[\![\mathbb{G}]\!]^{\mathbb{S}}_{DB} \subseteq [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$.*

PROOF: This result is a direct consequence of Proposition 4.11 and Theorem 3.24. Nevertheless, we give a direct proof since basic graph patterns (and SPARQL queries in general) allow for a more syntactic proof strategy.

Let $\mu \in [\![\mathbb{G}]\!]^{\mathbb{S}}_{DB}$ and $t \in \mathbb{G}$. It holds that $t = (\mathsf{x}, a, \mathsf{y})$ ($\mathsf{x}, \mathsf{y} \in \mathcal{V}$ and $a \in \mathcal{P}$). We need to show that $\nu = \mu{\restriction_{\{\mathsf{x},\mathsf{y}\}}}$ ($\mu \cap (\{\mathsf{x}, \mathsf{y}\} \times \mathcal{U})$, resp.) is a dual simulation match of t in DB. Since $\mu \in [\![\mathbb{G}]\!]^{\mathbb{S}}_{DB}$, $(\mu(\mathsf{x}), a, \mu(\mathsf{y})) \in E_{DB}$. It follows that $(\nu(\mathsf{x}), a, \nu(\mathsf{y})) \in E_{DB}$. Consider now the pair $(\mathsf{x}, \nu(\mathsf{x})) \in \nu$. According to t we need to give some o, such that $(\nu(\mathsf{x}), a, o)$ and $(\mathsf{y}, o) \in \nu$. $o = \nu(\mathsf{y})$ provides this property. The case $(\mathsf{y}, \nu(\mathsf{y})) \in \nu$ is analogous. Since we did not further restrict the choice of the triple pattern t, the argument may be repeated for all triple patterns in $\mathbb{G}$. Hence, $\mu \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$. Q. E. D.

Based on the existing PTIME algorithms that solve the non-emptiness problem of dual simulation between graphs, which we explain in detail in Chapter 5, we arrive at the conclusion that $(\mathbb{S}_\mathsf{A}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$ is a tractable query language.

Lemma 4.25 EVALUATION$(\mathbb{S}_\mathsf{A}, [\![\cdot]\!]^{\mathsf{DS}}_{_})$ *and* NONEMPTY$(\mathbb{S}_\mathsf{A}, [\![\cdot]\!]^{\mathsf{DS}}_{_})$ *are both in* PTIME.

The proof of this lemma is postponed to Section 5.3.1. Note that the improvement upon SPARQL is found in the non-emptiness problem. Evaluation of basic graph patterns under SPARQL's semantics has already been in PTIME (cf. Proposition 4.19).

Analogously to dual simulations between graphs (cf. Definition 2.30), the notion of *the maximal dual simulation between a basic graph pattern* $\mathcal{Q} \in \mathbb{S}_{\mathsf{A}}$ *and a graph database* DB is well-defined. Once again, this is because the union of two dual simulation matches is again a dual simulation match.

Lemma 4.26 *Let* $DB = (O_{DB}, \Sigma, E_{DB})$ *be a graph database,* $\mathbb{G} \in \mathbb{S}_{\mathsf{A}}$*, and* $S_1, S_2 \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$*. Then* $S_1 \cup S_2 \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$*.*

PROOF: Let $(\mathsf{v}, o) \in (S_1 \cup S_2)$ and $(\mathsf{v}, a, \mathsf{w})$ a triple pattern in $\mathbb{G}$. Then $(\mathsf{v}, o) \in S_1$ or $(\mathsf{v}, o) \in S_2$, and since S_1 (S_2, resp.) is a dual simulation match for $\mathbb{G}$ in DB, there is an o' with $(o, a, o') \in E_{DB}$ and $(\mathsf{w}, o') \in S_1$ ($(\mathsf{w}, o') \in S_2$, resp.). Hence, $(\mathsf{w}, o') \in (S_1 \cup S_2)$. The case of a triple pattern $(\mathsf{u}, a, \mathsf{v}) \in \mathbb{G}$ is completely analogous. Q. E. D.

Proposition 4.27 *The* maximal dual simulation match between basic graph pattern $\mathbb{G} \in \mathbb{S}_{\mathsf{A}}$ and graph database DB *is unique and determined by* $\widehat{S} = \bigcup_{S \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}} S$.

PROOF: Suppose there is another maximal dual simulation match $\widehat{S}' \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$, i. e., for all $S \in \mathbb{G}$, $\widehat{S}' \subseteq S$ implies $\widehat{S}' = S$. By Lemma 4.26, $\widehat{S}' \cup \widehat{S} \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$. By construction $\widehat{S}' \subseteq \widehat{S}' \cup \widehat{S}$, which implies $\widehat{S}' = \widehat{S}' \cup \widehat{S}$. As $\widehat{S}$ is maximal, $\widehat{S} \subseteq \widehat{S}' \cup \widehat{S}$ implies $\widehat{S} = \widehat{S}' \cup \widehat{S}$ and $\widehat{S} = \widehat{S}'$. Hence, there is only a single maximal dual simulation match for $\mathbb{G}$ in DB.Q. E. D.

Besides its uniqueness, we compute the maximal dual simulation match in polynomial time. In fact, every published algorithm for computer dual simulations between graphs first computes the maximal dual simulation [69, 86, 93].

Lemma 4.28 *Computing the maximal dual simulation match of basic graph pattern* $\mathbb{G} \in \mathbb{S}_{\mathsf{A}}$ *and graph database* DB *is in* PTIME.

Once again, we postpone the proof of this lemma to Section 5.3.1.

4.2.2 Complex Patterns and Compatibility

Replacing homomorphisms by dual simulations yields a tractable alternative to the SPARQL semantics for basic graph patterns. Here, we study the interactions between dual simulations and the join operators (`AND` and `OPT`) to see whether the success for BGPs can be expanded to more complex SPARQL queries. Therefore, we canonically extend the notion of compatibility to dual simulations and study the consequences for the resulting semantics of $\mathbb{S}_{\mathsf{AO}}$. Unfortunately, as we will demonstrate in a few examples, this new semantics does not fulfill the requirement of *correctness*, whose formal definition will also be given in the course of this section. Dual simulations, exhibiting unbounded cycles, harmfully interact with optional clauses.

Compatibility of dual simulation candidates $S_1, S_2 \subseteq \mathcal{V} \times \mathcal{U}$ is conservatively defined as follows: $S_1 \rightleftarrows S_2$ iff for all $v \in dom(S_1) \cap dom(S_2)$, $vS_1 = vS_2$. This notion of compatibility allows us to directly define the dual simulation semantics of conjunctions as well as optional patterns.

Definition 4.29 (SPARQL's Dual Simulation Semantics)
Let $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database. The *dual simulation semantics of a query* $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ *w. r. t.* DB is defined inductively over the structure of $\mathcal{Q}$: If $\mathcal{Q}$ is a triple pattern t, then $[\![t]\!]^{\mathsf{DS}}_{DB}$ is defined according to Definition 4.21. For $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{AO}}$,

$$\begin{aligned} [\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB} &:= \{S_1 \cup S_2 \mid S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB} \wedge S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB} \wedge S_1 \rightleftarrows S_2\} \\ [\![\mathcal{Q}_1 \texttt{ OPT } \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB} &:= [\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB} \cup \{S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB} \mid \forall S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB} : S_1 \not\rightleftarrows S_2\} \end{aligned}$$

A dual simulation $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ is called a *dual simulation match for $\mathcal{Q}$ in DB*. ▲

Subsequently, we give characteristic examples for the interactions of dual simulations and $\mathbb{S}_{\mathsf{AO}}$-queries. If not stated otherwise, they all refer to the graph database $DB_{4.1}$ from Figure 4.1.

Example 4.30 The query

$$\mathcal{Q}_1 = (\mathsf{p}, \mathit{cite}, \mathsf{p})\ \texttt{AND}\ (\mathsf{p}, \mathit{author}, \mathsf{r})$$

is the unfolded version of the basic graph pattern $\mathbb{G}$ discussed in Example 4.23. While SPARQL's only match is $\mu\ (\mathsf{p} \mapsto \texttt{P1}, \mathsf{r} \mapsto \texttt{John})$, our dual simulation semantics additionally yields the dual simulation S from Example 4.23. The base dual simulations to the left and the right triple pattern are

$$S_1 = \{(\mathsf{p}, \texttt{P1}), (\mathsf{p}, \texttt{P2}), (\mathsf{p}, \texttt{P4})\} \qquad \text{and}$$
$$S_2 = S_1 \cup \{(\mathsf{r}, \texttt{John}), (\mathsf{r}, \texttt{Kristin})\}.$$

$S_1 \rightleftarrows S_2$ because $\mathit{dom}(S_1) \cap \mathit{dom}(S_2) = \{\mathsf{p}\}$ and $\mathsf{p}S_1 = \{\texttt{P1}, \texttt{P2}, \texttt{P4}\} = \mathsf{p}S_2$.

Such extensions of cycles may easily lead to more informative results than possibly returned under SPARQL's original semantics. Let us therefore consider the query

$$\mathcal{Q}_2 = (\mathsf{p}, \mathit{cite}, \mathsf{s})\ \texttt{OPT}\ (\mathsf{s}, \mathit{cite}, \mathsf{s})$$

being structurally quite similar to $\mathcal{Q}_1$. Only this time, we use `OPT` instead of `AND`. The SPARQL semantics exhibits one match for the conjunctive part[4] in $DB_{4.1}$, namely $\mu_1\ (\mathsf{p} \mapsto \texttt{P1}, \mathsf{s} \mapsto \texttt{P1})$. Note that there are four more matches to be found in $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB_{4.1}}$, only matching the left-hand side of $\mathcal{Q}_2$. Under the dual simulation semantics,

$$S_3 = \{(\mathsf{p}, \texttt{P3}), (\mathsf{s}, \texttt{P2}), (\mathsf{s}, \texttt{P4})\}$$

is a dual simulation between $(\mathsf{p}, \mathit{cite}, \mathsf{s})$ and $DB_{4.1}$. Furthermore,

$$S_4 = \{(\mathsf{s}, \texttt{P2}), (\mathsf{s}, \texttt{P4})\}$$

is a dual simulation match for $(\mathsf{s}, \mathit{cite}, \mathsf{s})$ in $DB_{4.1}$. S_3 and S_4 are compatible, meaning that $S_3 \cup S_4$ is a dual simulation match for $\mathcal{Q}_2$. Hence, although `P3` does not occur as a match for the conjunctive part of $\mathcal{Q}_2$ under SPARQL's semantics, it does so in the dual simulation interpretation. ■

This example is not yet harmful to the correctness of the resulting semantics. However, it already shows that not all SPARQL matches are preserved by the dual simulation semantics since there is a query $\mathcal{Q}$ (e. g., $\mathcal{Q}_2$ from Example 4.30) and a graph database DB (e. g., $DB_{4.1}$) for which $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \not\subseteq [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$. Therefore, the degree of correctness still achievable must be weaker. As with monotonicity, we bypass the strong requirement enforced by $\subseteq$ through subsumption ($\sqsubseteq$), i. e., every match $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$ must be included in some match $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$. Although *correctness* of a semantics $[\![\cdot]\!]\_$ for SPARQL queries clearly is a relative notion to some other semantics for SPARQL, we pursue it only w. r. t. SPARQL's well-accepted semantics $[\![\cdot]\!]^{\mathbb{S}}\_$.

Definition 4.31 (Correctness of $(\mathbb{L}, [\![\cdot]\!]\_)$)
Let $\mathbb{L} \subseteq \mathbb{S}$. A semantics $[\![\cdot]\!]\_$ is *correct for* $\mathbb{L}$ iff for all queries $\mathcal{Q} \in \mathbb{L}$ and all graph databases, $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \sqsubseteq [\![\mathcal{Q}]\!]_{DB}$. A query language $(\mathbb{L}, [\![\cdot]\!]\_)$ is *correct* iff $[\![\cdot]\!]\_$ is correct for $\mathbb{L}$. ▲

[4] Since the evaluation of $\mathcal{Q}_1$ `OPT` $\mathcal{Q}_2$ includes all matches to the conjunction $\mathcal{Q}_1$ `AND` $\mathcal{Q}_2$, these matches belong to the conjunctive part of $\mathcal{Q}_1$ `OPT` $\mathcal{Q}_2$.

As an application of the notion and as a consequence of Lemma 4.24, we obtain the following result about BGPs and their dual simulation interpretation.

Theorem 4.32 $(\mathbb{S}_{\mathsf{A}}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$ *is correct.*

PROOF: Let $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$. Then there is an $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$, namely $S = \mu$ by Lemma 4.24, with $\mu \subseteq S$. Q. E. D.

The subsequent example provides a counterexample disproving correctness of the dual simulation semantics for $\mathbb{S}_{\mathsf{AO}}$. The reason is that, due to unbounded cycles, dual simulations may match expanded cycles in optional clauses where SPARQL does not find a respective match.

Example 4.33 Consider $DB_{4.3\,(a)}$ (Figure 4.3 (a)), which extends $DB_{4.1}$ by node `P0`. Node `P0`, like `P1`, participates in a *citedBy*-relationship with `P3` and exhibits a *cite*-self-loop. Reconsider $\mathcal{Q}_{\mathsf{pref}}$ from Examples 4.4 and 4.8, i. e.,

$$\underbrace{(\mathsf{p}, \mathit{author}, \mathsf{r})}_{M}\ \mathtt{OPT}\ \underbrace{((\mathsf{p}, \mathit{cite}, \mathsf{s})\ \mathtt{AND}\ (\mathsf{s}, \mathit{cite}, \mathsf{s}))}_{O_1}\ \mathtt{OPT}\ \underbrace{((\mathsf{s}, \mathit{citedBy}, \mathsf{p})\ \mathtt{AND}\ (\mathsf{s}, \mathit{cite}, \mathsf{s}))}_{O_2}.$$

The first match of interest is

$$\mu\ (\mathsf{p} \mapsto \mathtt{P3}, \mathsf{r} \mapsto \mathtt{Robert}, \mathsf{s} \mapsto \mathtt{P1}),$$

being the result of the conjunction of M and O_2. There is no match for O_1 compatible to μ_1 $(\mathsf{p} \mapsto \mathtt{P3}, \mathsf{r} \mapsto \mathtt{Robert})$ under the original semantics of SPARQL. Under dual simulations, however, μ_1 (easily interpreted as a dual simulation between M and $DB_{4.3\,(a)}$ by Theorem 4.32) may be compatibly extended by a match for O_1, namely

$$S_2 = \{(\mathsf{p}, \mathtt{P3}), (\mathsf{s}, \mathtt{P2}), (\mathsf{s}, \mathtt{P4})\}.$$

It holds that $\mu_1 \rightleftarrows S_2$ since $\mathsf{p}\mu_1 = \{\mu_1(\mathsf{p})\} = \{\mathtt{P3}\} = \mathsf{p}S_2$. Thus, $\mu_1 \cup S_2 \in [\![\mathcal{Q}_{\mathsf{pref}}]\!]^{\mathsf{DS}}_{DB_{4.3\,(a)}}$, letting incompatibility of μ_1 to all matches of O_1 disappear. Only circumstantially, the dual simulation semantics preserves μ by the dual simulation match $S \in [\![\mathcal{Q}_{\mathsf{pref}}]\!]^{\mathsf{DS}}_{DB}$ with

$$S = \{(\mathsf{r}, \mathtt{Robert}), (\mathsf{p}, \mathtt{P3}), (\mathsf{r}, \mathtt{John}), (\mathsf{p}, \mathtt{P1})\} \cup \{(\mathsf{p}, \mathtt{P1}), (\mathsf{s}, \mathtt{P1}), (\mathsf{p}, \mathtt{P3}), (\mathsf{s}, \mathtt{P2}), (\mathsf{s}, \mathtt{P4})\}.$$

The counterexample SPARQL match

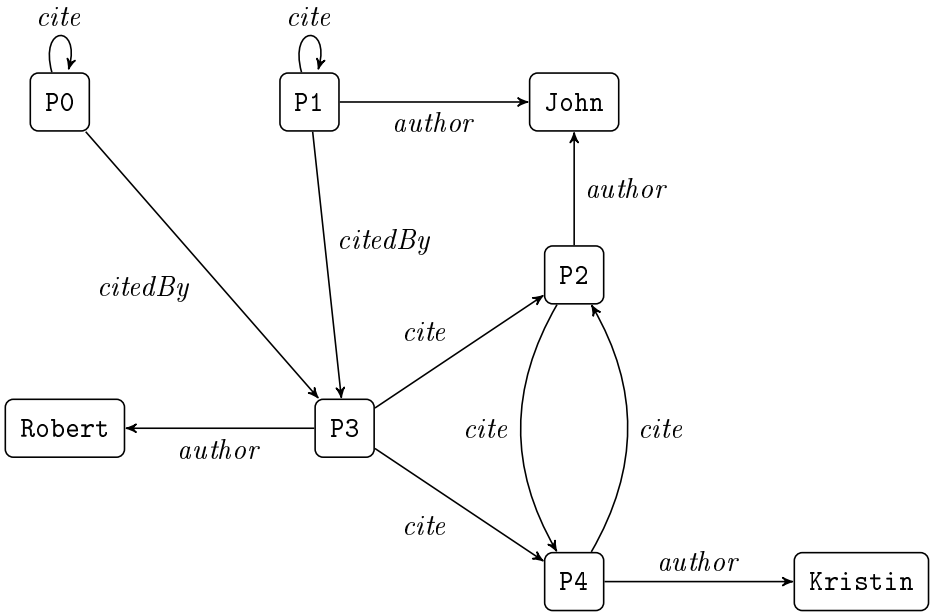
$$\mu'\ (\mathsf{p} \mapsto \mathtt{P3}, \mathsf{r} \mapsto \mathtt{Robert}, \mathsf{s} \mapsto \mathtt{P0})$$

once more stems from a conjunction of M and O_2. This time, however, μ' does not reappear in any dual simulation match for $\mathcal{Q}_{\mathsf{pref}}$. ■

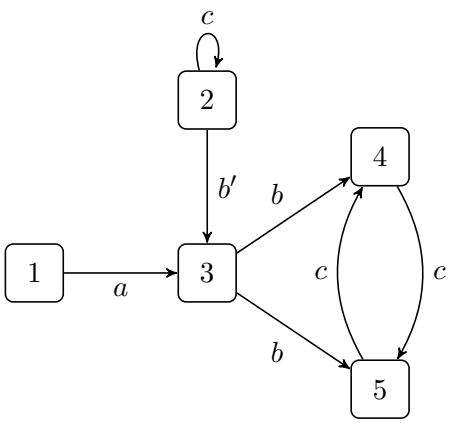
Proposition 4.34 *For some* $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ *and graph database* DB, $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} \not\sqsubseteq [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$. *Thus,* $(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$ *is not correct.*

PROOF: By query $\mathcal{Q}_{\mathsf{pref}}$ and graph database $DB_{4.3\,(a)}$, we already have witnesses for the claim. In order to reduce this proof's complexity, we simplify the counterexample. Consider

$$\mathcal{Q} = \underbrace{(\mathsf{x}, a, \mathsf{y})}_{M}\ \mathtt{OPT}\ \underbrace{((\mathsf{y}, b, \mathsf{z})\ \mathtt{AND}\ (\mathsf{z}, c, \mathsf{z}))}_{O_1}\ \mathtt{OPT}\ \underbrace{\big((\mathsf{z}, b', \mathsf{y})\ \mathtt{AND}\ (\mathsf{z}, c, \mathsf{z})\big)}_{O_2}$$



(a)



(b)

Figure 4.3: (a) Another Adaptation of the Network in Figure 4.1 (b) A Simplified Version of Figure 4.3 (a)

as the counterexample query and $DB_{4.3\,(b)}$ depicted in Figure 4.3 (b). $[\![M]\!]^{\mathsf{DS}}_{DB_{4.3\,(b)}} = \{S_m\}$, $[\![O_1]\!]^{\mathsf{DS}}_{DB_{4.3\,(b)}} = \{S_1\}$, and $[\![O_2]\!]^{\mathsf{DS}}_{DB_{4.3\,(b)}} = \{S_2\}$ with

$$\begin{array}{rcl} S_m & = & \{(\mathsf{x},1),(\mathsf{y},3)\}, \\ S_1 & = & \{(\mathsf{y},3),(\mathsf{z},4),(\mathsf{z},5)\}, \text{ and} \\ S_2 & = & \{(\mathsf{y},3),(\mathsf{z},2)\}. \end{array}$$

Since $S_m \rightleftarrows S_1$, $S_m \cup S_1 \in [\![M \texttt{ OPT } O_1]\!]^{\mathsf{DS}}_{DB_{4.3\,(b)}}$. $(S_m \cup S_1) \not\rightleftarrows S_2$ because $\mathsf{z} \in dom(S_m \cup S_1) \cap dom(S_2)$ but $\mathsf{z}(S_m \cup S_1) = \mathsf{z}S_1 = \{4,5\} \neq \{2\} = \mathsf{z}S_2$. Thus, $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB_{4.3\,(b)}} = \{S_m \cup S_1\}$.

$[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB_{4.3\,(b)}}$ also contains a single match, namely $\mu = \{(\mathsf{x},1),(\mathsf{y},3),(\mathsf{z},2)\}$. But $\mu \notin S_m \cup S_1$ which implies that $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB_{4.3\,(b)}} \sqsubseteq [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB_{4.3\,(b)}}$ does not hold. Q. E. D.

Thus, $(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathsf{DS}})$ cannot be shown correct as not all matches due to the original semantics of SPARQL are preserved for every SPARQL query.

Before going on to the rescue of $[\![\cdot]\!]^{\mathsf{DS}}$, at least for a fragment of $\mathbb{S}_{\mathsf{AO}}$, we provide two more technical results about the semantics. They are going to be useful in subsequent proofs. The first one is concerned with pairwise compatible dual simulation candidates. For every three dual simulation candidates, that are pairwise compatible, the union of every two of them is compatible with the third.

Proposition 4.35 *Let $S_i \subseteq \mathcal{V} \times \mathcal{U}$ $(i = 1,2,3)$ be dual simulation candidates. If $S_i \rightleftarrows S_j$ for all $i,j \in \{1,2,3\}$, then for $l,m,n \in \{1,2,3\}$ with $\{l,m,n\} = \{1,2,3\}$, $S_l \cup S_m \rightleftarrows S_n$.*

PROOF: We need to show that for all $v \in dom(S_l \cup S_m) \cap dom(S_n)$, $v(S_l \cup S_m) = vS_n$. Let $v \in dom(S_l \cup S_m) \cap dom(S_n)$. Then either (i) $v \in dom(S_l) \setminus dom(S_m)$, (ii) $v \in dom(S_m) \setminus dom(S_l)$, or (iii) $v \in dom(S_l) \cap dom(S_m)$. In case (i), $v(S_l \cup S_m) = vS_l$. Since $S_l \rightleftarrows S_n$ and $v \in dom(S_n)$, $vS_l = vS_n$. Thus, $v(S_l \cup S_m) = vS_n$. The case (ii) is entirely analogous. In case (iii), $v(S_l \cup S_m) = vS_l = vS_m$ because $S_l \rightleftarrows S_m$. Since $S_l \rightleftarrows S_n$ and $v \in dom(S_n)$, $v(S_l \cup S_m) = vS_n$. Q. E. D.

As a first application of Proposition 4.35, we subsequently prove that AND is commutative and associative for the dual simulation semantics. Since matches are no longer partial functions, the proof requires a slightly different argument than necessary for Proposition 4.10.

Proposition 4.36 *Let DB be a graph database and $\mathcal{Q}_i \in \mathbb{S}$ $(i = 1,2,3)$. It holds that (I) $[\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB} = [\![\mathcal{Q}_2 \texttt{ AND } \mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ and (II) $[\![(\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2) \texttt{ AND } \mathcal{Q}_3]\!]^{\mathsf{DS}}_{DB} = [\![\mathcal{Q}_1 \texttt{ AND } (\mathcal{Q}_2 \texttt{ AND } \mathcal{Q}_3)]\!]^{\mathsf{DS}}_{DB}$.*

PROOF: Let $S, S' \subseteq \mathcal{V} \times \mathcal{U}$. Then $S \rightleftarrows S'$ iff $S' \rightleftarrows S$, i. e., $\rightleftarrows$ is a symmetric relation. The reason is that $dom(S) \cap dom(S') = dom(S') \cap dom(S)$ and for every $v \in dom(S) \cap dom(S')$, $vS = vS'$ iff $vS' = vS$ because equality $(=)$ is symmetric for sets. We may now proceed with the proof.

(I) First, let $S \in [\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$. Then there are $S_i \in [\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB}$ $(i = 1,2)$ with $S_1 \rightleftarrows S_2$. Since $\rightleftarrows$ is symmetric, it holds that $S_2 \rightleftarrows S_1$. Thus, $S_2 \cup S_1 = S \in [\![\mathcal{Q}_2 \texttt{ AND } \mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$. The other direction follows the same line of arguments, only backwards.

(II) Let $S \in [\![(\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2) \texttt{ AND } \mathcal{Q}_3]\!]^{\mathsf{DS}}_{DB}$. Then there are $S_i \in [\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB}$ $(i = 1,2,3)$ with $S_1 \rightleftarrows S_2$ and $S_1 \cup S_2 \rightleftarrows S_3$. We need to show that (i) $S_2 \rightleftarrows S_3$ and (ii) $S_1 \rightleftarrows (S_2 \cup S_3)$. Towards (i), let $v \in dom(S_2) \cap dom(S_3)$. If $v \notin dom(S_1)$, $vS_2 = vS_3$ because $S_1 \cup S_2 \rightleftarrows S_3$. If $v \in dom(S_1)$, $vS_2 = vS_3$ because $S_1 \rightleftarrows S_2$ and $S_1 \cup S_2 \rightleftarrows S_3$.

The same arguments apply for showing that $S_1 \rightleftarrows S_3$. Then (ii) follows from Proposition 4.35. Q. E. D.

4.3 Dual Simulations for Well-Designed SPARQL

We showed that replacing SPARQL's homomorphisms by dual simulations entails an incorrect querying semantics (cf. Proposition 4.34). The counterexample we used is a SPARQL query that is not monotone. Although we did not explicitly extend the database, the shift from $[\![\cdot]\!]^{\mathbb{S}}_{\_}$ to $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$ poses an implicit database extension. As we have learned by Proposition 4.15, well-designed SPARQL solely contains weakly monotone queries and in fact, $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$ is a correct query language. To prove correctness, we give an even stronger result, certifying a strong relation between the matches up to $[\![\cdot]\!]^{\mathbb{S}}_{\_}$ and those produced by $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$ through the notion of compatibility.

Theorem 4.37 *For all $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}$, databases DB, and matches $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$, there is a dual simulation match $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ with $\mu \subseteq S$ and $\mu \rightleftarrows S$.*

PROOF: The proof proceeds by induction on the structure of $\mathcal{Q}$.

Base: If $\mathcal{Q}$ is a triple pattern, $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \subseteq [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ by Lemma 4.24. Therefore, for every $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$, $\mu \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ holds and, trivially, $\mu \subseteq \mu$ and $\mu \rightleftarrows \mu$.

Hypothesis: For $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{wd}}$ and every $\mu_i \in [\![\mathcal{Q}_i]\!]^{\mathbb{S}}_{DB}$ $(i = 1, 2)$, there are $S_i \in [\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB}$ with $\mu_i \subseteq S_i$ and $\mu_i \rightleftarrows S_i$.

Step: It remains to be shown that the claim holds for queries

$$\text{(i) } \mathcal{Q} = \mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2 \text{ and (ii) } \mathcal{Q} = \mathcal{Q}_1 \texttt{ OPT } \mathcal{Q}_2.$$

(i) If $\mu \in [\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$, then there are $\mu_i \in [\![\mathcal{Q}_i]\!]^{\mathbb{S}}_{DB}$ $(i = 1, 2)$ with $\mu = \mu_1 \cup \mu_2$ and $\mu_1 \rightleftarrows \mu_2$. By induction hypothesis, there are $S_i \in [\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB}$ $(i = 1, 2)$ with $\mu_i \subseteq S_i$ and $\mu_i \rightleftarrows S_i$. Note that $\mu_1 \cup \mu_2 \subseteq S_1 \cup S_2$. We will show that (a) $S_1 \rightleftarrows S_2$ and (b) $(\mu_1 \cup \mu_2) \rightleftarrows (S_1 \cup S_2)$. By (a), $S_1 \cup S_2 \in [\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$.

(a) Let $v \in dom(S_1) \cap dom(S_2)$. Thus, $v \in vars(\mathcal{Q}_1) \cap vars(\mathcal{Q}_2)$. We will prove that $v \in dom(\mu_1) \cap dom(\mu_2)$, such that $vS_1 = vS_2$ follows from $vS_1 = v\mu_1 = v\mu_2 = vS_2$.
Towards a contradiction, assume $v \notin dom(\mu_1) \cap dom(\mu_2)$. Then for some $i \in \{1, 2\}$, $v \notin dom(\mu_i)$. Hence, v can only occur in an optional clause in $\mathcal{Q}_i$. More precisely, $\mathcal{Q}_i$ contains an optional pattern $P_1 \texttt{ OPT } P_2$ with $v \in vars(P_2) \setminus vars(P_1)$. As $v \in dom(S_j)$ $(j = 1, 2,\ j \neq i)$ implies that $v \in vars(\mathcal{Q}_j)$, $\mathcal{Q}$ is not a well-designed query, contradicting the theorem's assumption. Hence, the assumption $v \notin dom(\mu_1) \cap dom(\mu_2)$ is false.

(b) For $v \in dom(\mu_1 \cup \mu_2) \cap dom(S_1 \cup S_2)$, we need to show that $v(\mu_1 \cup \mu_2) = v(S_1 \cup S_2)$. If $v \in dom(\mu_1)$, then $v \in dom(S_1)$ or $v \in dom(S_2)$. Whenever the former case holds, $v\mu_1 = vS_1$ by induction hypothesis and as $\mu_1 \rightleftarrows \mu_2$ and $S_1 \rightleftarrows S_2$ (cf. (a)), $v(\mu_1 \cup \mu_2) = v(S_1 \cup S_2)$. Regarding the latter case, it remains to be shown that the claim holds, even if $v \in dom(S_2) \setminus dom(S_1)$. If $v \in dom(\mu_2)$, the claim holds since $\mu_1 \rightleftarrows \mu_2$ and $\mu_2 \rightleftarrows S_2$. If $v \notin dom(\mu_2)$, there is an optional pattern $P_1 \texttt{ OPT } P_2$ in $\mathcal{Q}_2$ with $v \in vars(P_2) \setminus vars(P_1)$. In that case $v \in dom(\mu_1)$ contradicts the assumption that $\mathcal{Q}$ is a well-designed query (cf. (a)). The case of $v \in dom(\mu_2)$ is completely analogous.

Thus, $S_1 \cup S_2 \in [\![Q]\!]^{\mathsf{DS}}_{DB}$, $\mu = \mu_1 \cup \mu_2 \subseteq S_1 \cup S_2$, and $\mu \rightleftarrows S_1 \cup S_2$.

(ii) If $\mu \in [\![\mathcal{Q}_1 \texttt{ OPT } \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$, then (a) $\mu \in [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB}$ and there is no $\mu' \in [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$ with $\mu \rightleftarrows \mu'$ or (b) $\mu \in [\![\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$. Case (b) follows the same arguments as we described for case (i). In case (a), we have some $S \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ with $\mu \subseteq S$ and

$\mu \rightleftarrows S$ by induction hypothesis. Either there is an $S' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ with $S \rightleftarrows S'$ or there is no such S'. In both cases, μ is a subset of the resulting match, either $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ or $S \cup S' \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$. $\mu \rightleftarrows S \cup S'$ follows from the fact that $\mu \subseteq S$ and $S \rightleftarrows S'$. Q. E. D.

Corollary 4.38 $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$ *is correct.*

The rest of this section is devoted to proving tractability of the dual simulation semantics for well-designed SPARQL. First, we consider the non-emptiness problem that has a neat PTIME solution due to a normal form obtained for well-designed SPARQL. This normal form is later also exploited for giving a PTIME upper bound for the evaluation problem of $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$.

4.3.1 Tractability of Non-Emptiness

Let us consider any query $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}$. By definition, $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. In the base case, $\mathcal{Q}$ is a basic graph pattern, i. e., Lemma 4.25 provides polynomial-time non-emptiness of $\mathcal{Q}$. Deciding non-emptiness of optional patterns $\mathcal{Q} = \mathcal{Q}_1 \,\texttt{OPT}\, \mathcal{Q}_2$ in DB boils down to deciding non-emptiness of $\mathcal{Q}_1$ in DB. Whether there is a compatible match for $\mathcal{Q}_2$ in DB or not is negligible information. Non-emptiness of conjunctions $\mathcal{Q} = \mathcal{Q}_1 \,\texttt{AND}\, \mathcal{Q}_2$ in DB, on the other hand, requires us to not only derive non-emptiness of both, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ but also to find out whether there is a pair of compatible matches from the respective result sets. As this problem quickly degenerates into a combinatorial problem, we have to take up on a more structured solution. Fortunately, we may transform every well-designed query $\mathcal{Q}$ into an *equivalent* well-designed query that is an optional pattern. Informally, two queries $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}$ are *equivalent* iff for all graph databases, they return the same matches.

Proposition 4.39 (Proposition 4.10 [113]) *Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}$ and consider the following two rewriting rules,*

$$((P_1 \,\texttt{OPT}\, P_2) \,\texttt{AND}\, P_3) \quad \longrightarrow \quad ((P_1 \,\texttt{AND}\, P_3) \,\texttt{OPT}\, P_2) \tag{4.2}$$

and

$$(P_1 \,\texttt{AND}\, (P_2 \,\texttt{OPT}\, P_3)) \quad \longrightarrow \quad ((P_1 \,\texttt{AND}\, P_2) \,\texttt{OPT}\, P_3). \tag{4.3}$$

Every application of (4.2) or (4.3) to query $\mathcal{Q}$[5] yields a query $\widehat{\mathcal{Q}} \in \mathbb{S}_{\mathsf{wd}}$ with the property that $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} = [\![\widehat{\mathcal{Q}}]\!]^{\mathbb{S}}_{DB}$ for all graph databases DB.

Thus, applications of (4.2) and (4.3) yield equivalent queries in $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathbb{S}}_{\_})$. The idea of the rewriting rules, and the proof of Proposition 4.39, is to exhibit well-designedness of $\mathcal{Q}$. Intuitively, pushing P_3 to the mandatory side of the optional pattern (cf. (4.2)) does not change the overall matches of the pattern because well-designedness ensures P_3 not to interfere with partial matches for P_2. If we iteratively apply the rules (4.2) and (4.3) to well-designed SPARQL queries until they cannot be applied anymore, we obtain a normal form, the so-called OPT *normal form* [113]. As Proposition 4.39 has only been established for the original query language $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathbb{S}}_{\_})$, we can exploit this normal form only after ensuring the rewriting rules to also yield equivalent results up to $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$.

Proposition 4.40 *Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}$. Applying rule (4.2) or (4.3) to $\mathcal{Q}$ yields a query $\widehat{Q} \in \mathbb{S}_{\mathsf{wd}}$ with $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} = [\![\widehat{\mathcal{Q}}]\!]^{\mathsf{DS}}_{DB}$ for all graph databases DB.*

[5] An application of a rule to $\mathcal{Q}$ means finding a subpattern of $\mathcal{Q}$ that is structurally identical to the left-hand side of the rule and replace it according to its right-hand side.

PROOF: Let DB be a graph database. We give the proof for (4.2), only, because the case of (4.3) follows from (4.2) by commutativity of AND (cf. Proposition 4.36). Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}$ containing $((P_1 \texttt{ OPT } P_2) \texttt{ AND } P_3)$ as a subquery. The proof proceeds by induction on the structure of $\mathcal{Q}$.

Base: Consider $\mathcal{Q} = (P_1 \texttt{ OPT } P_2) \texttt{ AND } P_3$. We need to show that $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} = [\![\widehat{\mathcal{Q}}]\!]^{\mathsf{DS}}_{DB}$, where $\widehat{\mathcal{Q}} = (P_1 \texttt{ AND } P_3) \texttt{ OPT } P_2$.

$\subseteq$: Let $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \in [\![P_1]\!]^{\mathsf{DS}}_{DB}$, $S_3 \in [\![P_3]\!]^{\mathsf{DS}}_{DB}$, and $S_1 \cup S_3 \subseteq S$. If any match in $[\![P_2]\!]^{\mathsf{DS}}_{DB}$ exists that is compatible with S_1, then there is such an $S_2 \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S = S_1 \cup S_2 \cup S_3$ ($S_1 \rightleftarrows S_2$ and $S_1 \cup S_2 \rightleftarrows S_3$). Otherwise, $S = S_1 \cup S_3$ ($S_1 \rightleftarrows S_3$) and there is no $S_2 \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2$. In either case, we need to show that $S \in [\![\widehat{\mathcal{Q}}]\!]^{\mathsf{DS}}_{DB}$.

If $S = S_1 \cup S_2 \cup S_3$, we need to show that (i) $S_1 \rightleftarrows S_3$ and (ii) $S_1 \cup S_3 \rightleftarrows S_2$. From $S_1 \cup S_2 \rightleftarrows S_3$ and $S_1 \rightleftarrows S_2$, it follows that $S_1 \rightleftarrows S_3$ and $S_2 \rightleftarrows S_3$: Similarly to the proof of Proposition 4.36, let $v \in dom(S_1) \cap dom(S_3)$. If $v \in dom(S_2)$, $vS_1 = vS_2 = v(S_1 \cup S_2) = vS_3$. Otherwise, $vS_2 = \emptyset$ and $v(S_1 \cup S_2) = vS_1$. Thus, $vS_1 = vS_3$ is implied by $v(S_1 \cup S_2) = vS_3$. Almost analogously, the case S_2 and S_3 is proven. Here, a well-designedness argument will be necessary. (ii) follows from Proposition 4.35.

If $S = S_1 \cup S_3$, we need to show that there is still no $S_2 \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \cup S_3 \rightleftarrows S_2$. If there was any $S_2 \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \cup S_3 \rightleftarrows S_2$, $S_1 \rightleftarrows S_3$ implies that $S_1 \rightleftarrows S_2$. But there is no $S_2 \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2$ by the choice of S.

$\supseteq$: Let $S \in [\![\widehat{\mathcal{Q}}]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \in [\![P_1]\!]^{\mathsf{DS}}_{DB}$ and $S_3 \in [\![P_3]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_3$. An $S_2 \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \cup S_3 \rightleftarrows S_2$ implies that $S_1 \rightleftarrows S_2$ and $S_3 \rightleftarrows S_2$. Thus, $S_1 \cup S_2 \rightleftarrows S_3$ by Proposition 4.35. Therefore $S_1 \cup S_3 \cup S_2 \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$.

It remains to be shown that for $S = S_1 \cup S_3$ ($S_i \in [\![P_i]\!]^{\mathsf{DS}}_{DB}$ for $i = 1, 3$) and there is no $S_2 \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \cup S_3 \rightleftarrows S_2$, there is no $S_2' \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2'$. Towards a contradiction, let $S_2' \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ such that $S_1 \rightleftarrows S_2'$. Consequently, $S_1 \cup S_3 \not\rightleftarrows S_2'$ means there is a variable $v \in dom(S_3) \setminus dom(S_1)$ with $v \in dom(S_2')$ and $vS_3 \neq vS_2'$. Hence, $v \in vars(P_3) \cap vars(P_2)$. As $\mathcal{Q}$ is well-designed, $v \in vars(P_1)$. Since $v \notin dom(S_1)$, there must be an optional subpattern $R_1 \texttt{ OPT } R_2$ of P_1 with $v \in dom(R_2) \setminus dom(R_1)$. However, P_1 is a subpattern of $\mathcal{Q}$ contradicting the assumption that $\mathcal{Q}$ is a well-designed query. Thus, the assumption that $S_2' \in [\![P_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2'$ is false and $S_1 \cup S_3 \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ is true.

Hypothesis: Suppose for $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{wd}}$ we have shown that the application of (4.2) to either of them yields queries $\widehat{\mathcal{Q}_i} \in \mathbb{S}_{\mathsf{wd}}$ ($i \in \{1, 2\}$) with $[\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB} = [\![\widehat{\mathcal{Q}_i}]\!]^{\mathsf{DS}}_{DB}$.

Step: We need to verify the claim for queries

$$\text{(i) } \mathcal{Q} = \mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2 \text{ and (ii) } \mathcal{Q} = \mathcal{Q}_1 \texttt{ OPT } \mathcal{Q}_2.$$

(i) W. l. o. g. assume $\widehat{\mathcal{Q}} = \widehat{\mathcal{Q}_1} \texttt{ AND } \mathcal{Q}_2$. The symmetric case is handled by commutativity of AND (cf. Proposition 4.36). Let $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$. Then there are $S_i \in [\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB}$ ($i = 1, 2$) with $S_1 \rightleftarrows S_2$ and $S = S_1 \cup S_2$. By induction hypothesis, $S_1 \in [\![\widehat{\mathcal{Q}_1}]\!]^{\mathsf{DS}}_{DB}$. Thus, $S_1 \cup S_2 \in [\![\widehat{\mathcal{Q}}]\!]^{\mathsf{DS}}_{DB}$. The reverse direction holds by analogous arguments.

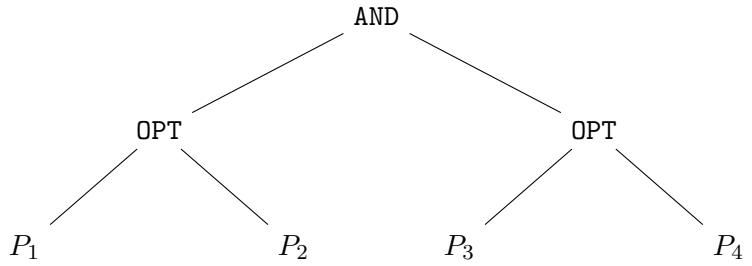
(ii) We distinguish two more cases here, i. e., $\widehat{\mathcal{Q}} = \widehat{\mathcal{Q}_1}\,\texttt{OPT}\,\mathcal{Q}_2$ and $\widehat{\mathcal{Q}} = \mathcal{Q}_1\,\texttt{OPT}\,\widehat{\mathcal{Q}_2}$. In both cases, we have some $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$. $S = S_1 \cup S_2$ with compatible $S_i \in [\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB}$ $(i = 1, 2)$ is already handled by (i). If $S = S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ and there is no $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2$, then the induction hypothesis applies. Thus, $S_1 \in [\![\widehat{\mathcal{Q}_1}]\!]^{\mathsf{DS}}_{DB}$ or there is no $S_2 \in [\![\widehat{\mathcal{Q}_2}]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2$. In both cases, $S \in [\![\widehat{\mathcal{Q}}]\!]^{\mathsf{DS}}_{DB}$. Q. E. D.

If we imagine the syntax tree of a well-designed query $\mathcal{Q}$, every application of (4.2) or (4.3) pushes a single conjunction deeper in the tree. Hence, pushing all conjunctions down the tree leaves us with a syntax tree that has basic graph patterns as leaf nodes and only OPT operators as inner nodes.

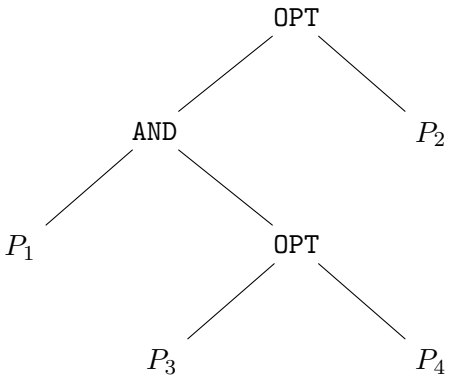
Example 4.41 Let $P_i \in \mathbb{S}_{\mathsf{A}}$ $(i = 1, 2, 3, 4)$ such that

$$\mathcal{Q} = (P_1\ \texttt{OPT}\ P_2)\ \texttt{AND}\ (P_3\ \texttt{OPT}\ P_4)$$

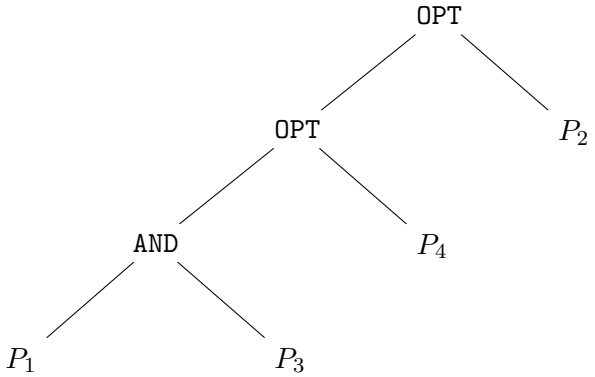
is a well-designed query. Without formally introducing it, a graphical representation of the syntax tree of $\mathcal{Q}$ may look as follows:



Rule (4.2) is applicable, altering the query $\mathcal{Q}$ to $(P_1\ \texttt{AND}\ (P_3\ \texttt{OPT}\ P_4))\ \texttt{OPT}\ P_2$ with syntax tree:



Applying rule (4.3) yields the final form, i. e., $((P_1\ \texttt{AND}\ P_3)\ \texttt{OPT}\ P_4)\ \texttt{OPT}\ P_2$:



Since $P_1, P_3 \in \mathbb{S}_\mathsf{A}$, P_1 AND P_3 is a basic graph pattern. Hence, the resulting syntax tree has OPT operators as inner nodes and BGPs at its leafs. ■

Proposition 4.40 tells us that the OPT normal form for well-designed SPARQL is also a normal form up to $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$.

Definition 4.42 (OPT Normal Form [113])
A well-designed query $\mathcal{Q} \in \mathbb{S}_\mathsf{wd}$ is in OPT *normal form* iff (a) $\mathcal{Q}$ is a basic graph pattern or (b) $\mathcal{Q} = \mathcal{Q}_1$ OPT $\mathcal{Q}_2$ and $\mathcal{Q}_1, \mathcal{Q}_2$ are in OPT normal form. Every well-designed query $\mathcal{Q}$ may be turned into OPT normal form in $\mathcal{O}(|\mathcal{Q}|^2)$ applications of rules (4.2) and (4.3) (cf. Theorem 4.11 [113]). ▲

We exploit this normal form in the proof showing tractability of the non-emptiness problem of $(\mathbb{S}_\mathsf{wd}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$. Since the transformation process of Pérez et al. [113] does not exceed PTIME, we assume it as a preprocessing step and henceforth work with the language of all well-designed queries in OPT normal form, denoted $\mathbb{S}^{ONF}_\mathsf{wd}$.

Let $\mathcal{Q} \in \mathbb{S}^{ONF}_\mathsf{wd}$. By Definition 4.42, we have to distinguish only two cases (instead of three for $\mathbb{S}_\mathsf{wd}$), the case of basic graph patterns and the case of optional patterns. If $\mathcal{Q} = \mathcal{Q}_1$ OPT $\mathcal{Q}_2$, we will iterate through $\mathcal{Q}$ by the following equivalence: $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$ iff $[\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$. Thus, non-emptiness of $\mathcal{Q}$ shifts to non-emptiness of its mandatory clause $\mathcal{Q}_1$. Repeating this case distinction eventually yields a basic graph pattern (cf. Example 4.41). Therefore, non-emptiness of $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ eventually reduces to non-emptiness of the left-most basic graph pattern in $\mathcal{Q}$. As an auxiliary step, we define the function lmp (read as *left-most pattern*) for OPT normal form queries $\mathcal{Q}$ by

$$\mathsf{lmp}(\mathcal{Q}) := \begin{cases} \mathsf{lmp}(\mathcal{Q}_1) & \text{if } \mathcal{Q} = \mathcal{Q}_1 \text{ OPT } \mathcal{Q}_2 \\ \mathcal{Q} & \text{otherwise.} \end{cases}$$

The result of $\mathsf{lmp}(\mathcal{Q})$ is a basic graph pattern, i. e., $\mathsf{lmp} : \mathbb{S}^{ONF}_\mathsf{wd} \to \mathbb{S}_\mathsf{A}$.

Lemma 4.43 *Let $\mathcal{Q} \in \mathbb{S}^{ONF}_\mathsf{wd}$. For all graph databases DB,*

$$[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} \neq \emptyset \textit{ iff } [\![\mathsf{lmp}(\mathcal{Q})]\!]^{\mathsf{DS}}_{DB} \neq \emptyset.$$

PROOF: Let DB be a graph database. By induction on the structure of $\mathcal{Q}$.

Base: If $\mathcal{Q}$ is a BGP, $\mathcal{Q} = \mathsf{lmp}(\mathcal{Q})$ and, of course, $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$ iff $[\![\mathsf{lmp}(\mathcal{Q})]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$.

Hypothesis: Assume for $\mathcal{Q}_1 \in \mathbb{S}^{ONF}_\mathsf{wd}$ that $[\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$ iff $[\![\mathsf{lmp}(\mathcal{Q}_1)]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$.

Step: We need to show that for an optional pattern $\mathcal{Q} = \mathcal{Q}_1$ OPT $\mathcal{Q}_2$, $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$ iff $[\![\mathsf{lmp}(\mathcal{Q})]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$.

- $\Rightarrow$: There is at least one match $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$, i. e., there is a match $S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ and, possibly, a match $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$, such that $S = S_1(\cup S_2)$ (and $S_1 \rightleftarrows S_2$). Hence, $[\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$. By induction hypothesis, it holds that $[\![\mathsf{lmp}(\mathcal{Q}_1)]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$. As $\mathsf{lmp}(\mathcal{Q}) = \mathsf{lmp}(\mathcal{Q}_1)$, it follows that $[\![\mathsf{lmp}(\mathcal{Q})]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$.
- $\Leftarrow$: $\mathsf{lmp}(\mathcal{Q}) = \mathsf{lmp}(\mathcal{Q}_1)$ by definition of lmp. Furthermore, $[\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$ by induction hypothesis. Thus, there is at least one $S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$. Independent of an $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2$, it follows that there is a match $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \subseteq S$. Thus, $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$. Q. E. D.

Hence, tractability of non-emptiness of $(\mathbb{S}_\mathsf{wd}, [\![\cdot]\!]^{\mathsf{DS}}_{\_})$ is a consequence of the OPT normal form and Lemmas 4.25 and 4.43.

Theorem 4.44 NONEMPTY$(\mathbb{S}_\mathsf{wd}, [\![\cdot]\!]^{\mathsf{DS}}_{_})$ *is in* PTIME.

PROOF: It is sufficient to show that NONEMPTY($\mathbb{S}_{\mathsf{wd}}^{ONF}, [\![\cdot]\!]_{\_}^{\mathsf{DS}}$) is in PTIME. Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}^{ONF}$. By Lemma 4.43, $[\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}} \neq \emptyset$ iff $[\![\mathsf{Imp}(\mathcal{Q})]\!]_{DB}^{\mathsf{DS}} \neq \emptyset$. Since $\mathsf{Imp}(\mathcal{Q})$ is a basic graph pattern, the decision for non-emptiness of $[\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}}$ can be performed by deciding non-emptiness of the result set for the basic graph pattern $\mathsf{Imp}(\mathcal{Q})$, being performed in PTIME by Lemma 4.25. Q. E. D.

4.3.2 Tractability of Evaluation

An algorithm for the evaluation of $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]_{\_}^{\mathsf{DS}})$ also benefits from the OPT normal form. Therefore, we distinguish only two cases for $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}^{ONF}$:

1. $\mathcal{Q}$ is a basic graph pattern, i. e., $\mathcal{Q} \in \mathbb{S}_{\mathsf{A}}$ and
2. $\mathcal{Q}$ is an optional pattern, i. e., $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$ where $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{wd}}^{ONF}$.

Once again, handling basic graph patterns (case 1) is a task performed in PTIME by Lemma 4.25. If we get a PTIME result for optional patterns as well, our quest in establishing tractability of $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]_{\_}^{\mathsf{DS}})$ is complete.

Consider such an optional pattern $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$ with $\mathbb{S}_{\mathsf{wd}}^{ONF}$. Furthermore, let DB be a graph database and S be a dual simulation candidate, i. e., $S \subseteq \mathcal{V} \times \mathcal{U}$. It holds that $S \in [\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}}$ iff (a) $S \in [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}$ or (b) $S \in [\![\mathcal{Q}_1]\!]_{DB}^{\mathsf{DS}}$ and there is no $S' \in [\![\mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}$ with $S \rightleftarrows S'$. In order to show that (a) applies to S, there must be $S_1, S_2 \subseteq S$ with $S = S_1 \cup S_2$, such that $S_1 \in [\![\mathcal{Q}_1]\!]_{DB}^{\mathsf{DS}}$, $S_2 \in [\![\mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}$, and $S_1 \rightleftarrows S_2$. For sure, $dom(S_i)$ ($i \in \{1, 2\}$) may only include variables that also occur in $\mathcal{Q}_i$. Furthermore, for all variables $v \in dom(S_1) \cap dom(S_2)$, $vS_1 = vS_2$ because of compatibility. Therefore, S_1 and S_2 may have an overlap in variables belonging to both subpatterns. We choose S_1 (S_2, resp.) by reducing S to only contain those pairs $(v, o) \in S$ that are variables of $\mathcal{Q}_1$ ($\mathcal{Q}_2$, resp.). Extending the reduction of (partial) functions to dual simulations is a necessary step: For $S \subseteq \mathcal{V} \times \mathcal{U}$ and $V \subseteq \mathcal{V}$,

$$S{\restriction_V} := \{(\mathsf{v}, o) \in S \mid \mathsf{v} \in V\}.$$

Note that $S{\restriction_V} = S \cap (V \times \mathcal{U})$. Thus, we get $S_1 = S{\restriction_{vars(\mathcal{Q}_1)}}$ and $S_2 = S{\restriction_{vars(\mathcal{Q}_2)}}$. As we are still in case (a), we subsequently prove that S is a match for query $\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2$ iff S_1 and S_2 are matches for their respective subpatterns.

Lemma 4.45 *Let DB be a graph database, $(\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2) \in \mathbb{S}_{\mathsf{wd}}$, and $S \subseteq \mathcal{V} \times \mathcal{U}$. Furthermore, let $S_i = S{\restriction_{vars(\mathcal{Q}_i)}}$ ($i = 1, 2$).*

$$S \in [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}} \text{ iff } S_1 \in [\![\mathcal{Q}_1]\!]_{DB}^{\mathsf{DS}} \text{ and } S_2 \in [\![\mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}.$$

PROOF: Note that $S_1 \rightleftarrows S_2$ and $S = S_1 \cup S_2$ by construction of S_1 and S_2.

$\Leftarrow$: As $S_1 \rightleftarrows S_2$ and $S_i \in [\![\mathcal{Q}_i]\!]_{DB}^{\mathsf{DS}}$ ($i = 1, 2$), $S_1 \cup S_2 = S \in [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}$.

$\Rightarrow$: Suppose, $S_1 \notin [\![\mathcal{Q}_1]\!]_{DB}^{\mathsf{DS}}$ or $S_2 \notin [\![\mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}$. Thus, there are $S_i' \in [\![\mathcal{Q}_i]\!]_{DB}^{\mathsf{DS}}$ ($i = 1, 2$), distinct from S_1 and S_2, with $S_1' \rightleftarrows S_2'$ and $S = S_1' \cup S_2'$. For some $i \in \{1, 2\}$, $S_i' \subsetneq S_i$, i. e., for some $v \in vars(\mathcal{Q}_i)$, $vS_i' = \emptyset \neq vS_i$. By construction of S_1 and S_2, $v \in vars(\mathcal{Q}_1) \cap vars(\mathcal{Q}_2)$ and, therefore, there is an optional subpattern $P_1 \,\mathtt{OPT}\, P_2$ of $\mathcal{Q}_i$ with $v \in vars(P_2) \setminus vars(P_1)$. This subpattern contradicts the assumption that $\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2$ is a well-designed query. Thus, S_i' ($i = 1, 2$) are not any different from the S_i. Hence, $S_i \in [\![\mathcal{Q}_i]\!]_{DB}^{\mathsf{DS}}$ ($i = 1, 2$). Q. E. D.

Note that this proof relies on the assumption that $\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2$ is a well-designed query. For general SPARQL queries, we cannot assume S_1 and S_2 to be matches to $\mathcal{Q}_1/\mathcal{Q}_2$ and to

fully describe S. $\mathcal{Q}_1$ and $\mathcal{Q}_2$ may again be arbitrary well-designed queries, which is why we need to do the evaluation recursively.

In case (b), i. e., $S \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ but there is no $S' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ compatible with S, we need to show that verifying S_1 to be equal to S is sufficient. Analogously to Lemma 4.45, we prove that at least S_1 is actually a match for $\mathcal{Q}_1$.

Lemma 4.46 *Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}$ with $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$, DB be a graph database, and $S \subseteq \mathcal{V} \times \mathcal{U}$. If $S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$, then $S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ with $S_1 = S{\restriction}_{vars(\mathcal{Q}_1)}$.*

Proof: For $S \in [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$, $S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ follows from Lemma 4.45. Consider now $S \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ (and no $S' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ exists with $S \rightleftarrows S'$). Then $S_1 = S$. Thus, $S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$. Q. E. D.

We are almost done evaluating S w. r. t. optional pattern $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$ and graph database DB. Let us recapitulate how far we get in the evaluation with the so far achieved results:

- We first check whether $S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$. By Lemma 4.46, a negative answer means that $S \notin [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$.
- Otherwise, we distinguish whether $S = S_1$ holds.
  - If $S \neq S_1$ and $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$, Lemma 4.45 ensures us $S_1 \cup S_2 \in [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$. Thus, $S \in [\![\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$.
  - If $S \neq S_1$ and $S_2 \notin [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$, again by Lemma 4.45, $S \notin [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ implies $S \notin [\![\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$.
  - The open case is the one where $S = S_1$. Therefore, we are required to verify the absence of an $S' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ with $S \rightleftarrows S'$.

To obtain a positive answer to the open question, we could first check whether there is any match for $\mathcal{Q}_2$, i. e., whether $[\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ is empty (cf. Section 4.3.1). If $[\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$, looking through all the matches for $\mathcal{Q}_2$ is computationally too expensive. Fortunately, well-designedness and the $\mathtt{OPT}$ normal form of $\mathcal{Q}$ allow us to use an approximation. To disprove $S \in [\![\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ it is unnecessary to obtain a concrete $S' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ compatible with S.

If there is some $S' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ compatible with S, then there is a match for the left-most pattern of $\mathcal{Q}_2$ ($\mathsf{lmp}(\mathcal{Q}_2)$) which is also compatible with S. The match for the left-most pattern is a subset of any compatible match $S' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$. The reason why the match for the subpattern $\mathsf{lmp}(\mathcal{Q}_2)$ is also compatible with S is that $\mathcal{Q}$, and therefore $\mathcal{Q}_2$, is well-designed. The following lemma makes this relationship precise and also shows that if there is any match for $\mathsf{lmp}(\mathcal{Q}_2)$ in DB, that is compatible with S, then there is a match for $\mathcal{Q}_2$ that is compatible with S. While Lemmas 4.45 and 4.46 could be shown for general well-designed queries, the proof of the following result builds upon the $\mathtt{OPT}$ normal form of well-designed Sparql.

Lemma 4.47 *Let DB be a graph database, $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}^{ONF}_{\mathsf{wd}}$, and $S_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$. Furthermore, let $\widehat{S}$ be the maximal dual simulation between $\mathsf{lmp}(\mathcal{Q}_2)$ and DB. $S_1 \not\rightleftarrows S_2$ for all $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ iff $S_1{\restriction}_{vars(\mathsf{lmp}(\mathcal{Q}_2))} \not\subseteq \widehat{S}$.*

Proof: We handle the two directions separately.

$\Leftarrow$: Let $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$. We show that $S_1 \not\rightleftarrows S_2$ by induction on the structure of $\mathcal{Q}_2$.

Base: If $\mathcal{Q}_2 \in \mathbb{S}_\mathsf{A}$, $S_2 \subseteq \widehat{S}$ by Proposition 4.27. For every $v \in dom(S_1) \cap dom(S_2)$, it holds that $v \in vars(\mathcal{Q}_2)$. $S_1 \rightleftarrows S_2$ means $S_1{\restriction}_{vars(P_2)} \subseteq S_2$, which contradicts the assumption that $S_1{\restriction}_{vars(\mathsf{lmp}(P_2))} \not\subseteq \widehat{S}$ as $S_1{\restriction}_{vars(\mathsf{lmp}(P_2))} = S_1{\restriction}_{vars(P_2)}$. Thus, $S_1 \not\rightleftarrows S_2$.

Hypothesis: Suppose for $P_1, P_2 \in \mathbb{S}_\mathsf{wd}^{ONF}$, all $T_i \in [\![\mathcal{Q}_i]\!]_{DB}^\mathsf{DS}$ $(i = 1, 2)$ are incompatible with S_1.

Step: If $\mathcal{Q}_2 = P_1 \,\mathtt{OPT}\, P_2$, then $S_2 = T_1 \cup T_2$ with $T_1 \in [\![P_1]\!]_{DB}^\mathsf{DS}$ and $T_2 \in [\![P_2]\!]_{DB}^\mathsf{DS} \cup \{\emptyset\}$. If $T_2 = \emptyset$, $S_1 \not\rightleftarrows S_2$ follows from the induction hypothesis ($S_1 \not\rightleftarrows T_1$). If $T_2 \neq \emptyset$, $T_1 \rightleftarrows T_2$. Let $v \in dom(S_1) \cap dom(S_2)$. Then (i) $v \in dom(T_1) \setminus dom(T_2)$, (ii) $v \in dom(T_2) \setminus dom(T_1)$, or (iii) $v \in dom(T_1) \cap dom(T_2)$. In cases (i) and (ii), $vS_1 \neq vS_2$ follows from the induction hypothesis. In case (iii), $v(T_1 \cup T_2) = vT_1$ $(= vT_2$, resp.). By induction hypothesis $vS_1 \neq v(T_1 \cup T_2)$. Hence, $S_1 \not\rightleftarrows S_2$.

$\Rightarrow$: Let $S_2 \in [\![\mathcal{Q}_2]\!]_{DB}^\mathsf{DS}$. $S_1 \not\rightleftarrows S_2$ implies $S_1{\restriction}_{vars(\mathcal{Q}_2)} \not\subseteq S_2$. We show that $S_1{\restriction}_{vars(\mathsf{lmp}(\mathcal{Q}_2))} \not\subseteq \widehat{S}$ by induction on the structure of $\mathcal{Q}_2$:

Base: If $\mathcal{Q}_2 \in \mathbb{S}_\mathsf{A}$, $\widehat{S} \in [\![P_2]\!]_{DB}^\mathsf{DS}$ by Proposition 4.27 and Lemma 4.26. By assumption, all $S_2 \in [\![\mathcal{Q}_2]\!]_{DB}^\mathsf{DS}$ are incompatible with S_1. Thus, $S_1 \not\rightleftarrows \widehat{S}$ and $S_1{\restriction}_{vars(\mathsf{lmp}(\mathcal{Q}_2))} \not\subseteq \widehat{S}$ by $vars(\mathsf{lmp}(\mathcal{Q}_2)) = vars(\mathcal{Q}_2)$.

Hypothesis: For $P_1 \in \mathbb{S}_\mathsf{wd}^{ONF}$, $S_1{\restriction}_{vars(\mathsf{lmp}(P_1))} \not\subseteq \widehat{S}$.

Step: Here, $\mathcal{Q}_2 = P_1 \,\mathtt{OPT}\, P_2$. Since $\mathsf{lmp}(\mathcal{Q}_2) = \mathsf{lmp}(P_1)$, $S_1{\restriction}_{vars(\mathsf{lmp}(\mathcal{Q}_2))} \not\subseteq \widehat{S}$ follows directly from the induction hypothesis. Q. E. D.

Hence, if we have already established that $S_1 \in [\![\mathcal{Q}_1]\!]_{DB}^\mathsf{DS}$ and $S_1 = S$, checking for the absence of any $S' \in [\![\mathcal{Q}_2]\!]_{DB}^\mathsf{DS}$ with $S \rightleftarrows S'$ may be performed by computing the maximal dual simulation $\widehat{S}$ between $\mathsf{lmp}(\mathcal{Q}_2)$ and DB, and by establishing $S_1{\restriction}_{vars(\mathsf{lmp}(\mathcal{Q}_2))} \not\subseteq \widehat{S}$. We describe the full procedure in the following proof of tractability of the evaluation problem of $(\mathbb{S}_\mathsf{wd}, [\![\cdot]\!]_\_^\mathsf{DS})$.

Theorem 4.48 EVALUATION$(\mathbb{S}_\mathsf{wd}, [\![\cdot]\!]__^\mathsf{DS})$ *is in* PTIME.

PROOF: By Propositions 4.39 and 4.40, it is sufficient to show a PTIME procedure for EVALUATION$(\mathbb{S}_\mathsf{wd}^{ONF}, [\![\cdot]\!]__^\mathsf{DS})$. Let $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database, $\mathcal{Q} \in \mathbb{S}_\mathsf{wd}^{ONF}$, and $S \subseteq \mathcal{V} \times \mathcal{U}$.

Base: Evaluating $\mathcal{Q} \in \mathbb{S}_\mathsf{A}$ w. r. t. DB is in PTIME by Lemma 4.25.

Hypothesis: For $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_\mathsf{wd}^{ONF}$, checking whether any candidate $S' \subseteq \mathcal{V} \times \mathcal{U}$ is a match for $\mathcal{Q}_i$ $(i \in \{1, 2\})$ in DB can be performed in PTIME.

Step: If $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$, let $S_i = S{\restriction}_{vars(\mathcal{Q}_i)}$ $(i = 1, 2)$. Computing S_i $(i = 1, 2)$ can be performed in $\mathcal{O}(|vars(\mathcal{Q}_i)|^2 \cdot |O_{DB}|)$. If $S_1 \notin [\![\mathcal{Q}_1]\!]_{DB}^\mathsf{DS}$ (in PTIME by induction hypothesis), $S \notin [\![\mathcal{Q}]\!]_{DB}^\mathsf{DS}$ by Lemma 4.46. Thus, we reject S as a match for $\mathcal{Q}$ in PTIME.

If $S_2 \in [\![\mathcal{Q}_2]\!]_{DB}^\mathsf{DS}$ (in PTIME by induction hypothesis), $S \in [\![\mathcal{Q}]\!]_{DB}^\mathsf{DS}$ by Lemma 4.45. Thus, accepting S as a match for $\mathcal{Q}$, in this case, is again in PTIME.

If $S_2 \notin [\![\mathcal{Q}_2]\!]_{DB}^\mathsf{DS}$, we distinguish whether $S = S_1$ (in $\mathcal{O}((|vars(\mathcal{Q}_2)| \cdot |O_{DB}|)^2)$) or not:

1. If $S \neq S_1$, $S \notin [\![\mathcal{Q}]\!]_{DB}^\mathsf{DS}$ by Lemma 4.45 and we reject S as a match for $\mathcal{Q}$ in PTIME.

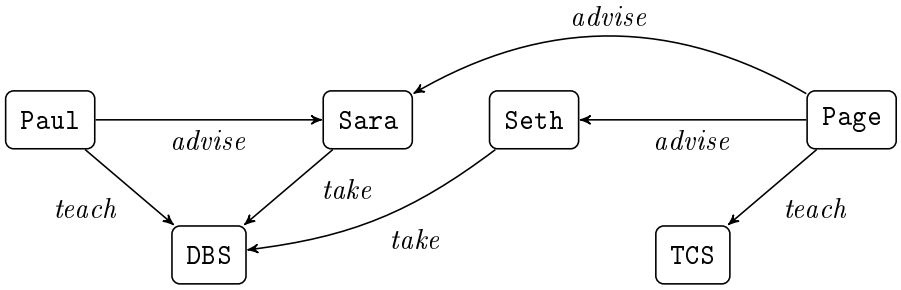


Figure 4.4: Another Graph Database Example

2. If $S = S_1$, we consider the basic graph pattern $P = \mathsf{Imp}(\mathcal{Q}_2)$ and compute the maximal dual simulation $\widehat{S}$ between P and DB (in PTIME by Lemma 4.28). If $S\restriction_{vars(P)} \subseteq \widehat{S}$, we reject S as a match for $\mathcal{Q}$ since by Lemma 4.47, there is a match $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ with $S \rightleftarrows S_2$. Otherwise, we accept S as a match for $\mathcal{Q}$ in DB. Q. E. D.

Thus, the dual simulation semantics is correct and tractable for well-designed SPARQL.

4.4 Maximal Dual Simulations for SPARQL

Throughout the last section, we have studied the semantic function $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$ w. r. t. different fragments of SPARQL. It primarily shifts SPARQL's matching semantics from graph homomorphisms (cf. Proposition 4.11) to dual simulations. For basic graph patterns (i. e., $\mathbb{S}_{\mathsf{A}}$) and well-designed SPARQL (i. e., $\mathbb{S}_{\mathsf{wd}}$), the new semantics turned out to be correct and tractable. As soon as arbitrary nesting of optional patterns was allowed, $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$ could not be shown correct anymore (cf. Proposition 4.34).

In this section, we catch up on the properties of dual simulations. More specifically, we are studying union-closedness of dual simulation matches. Recall from Section 2.3.2, that the maximal dual simulation between two graphs has been unique because of union-closedness of the set of all dual simulations. As far as basic graph patterns are concerned, we have already proven the uniqueness of the maximal dual simulation match by Proposition 4.27. The maximal dual simulation match necessarily contains every other dual simulation match. Thus, from a theoretical point of view, it represents a correct denotation of a basic graph pattern $\mathcal{Q} \in \mathbb{S}_{\mathsf{A}}$ w. r. t. a graph database DB.

Corollary 4.49 *Let DB be a graph database, $\mathcal{Q} \in \mathbb{S}_{\mathsf{A}}$, and $\widehat{S}$ the maximal dual simulation match for $\mathcal{Q}$ in DB. For all $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$, $\mu \subseteq \widehat{S}$.*

PROOF: First note that $\widehat{S} = \bigcup_{S \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}} S$ by Proposition 4.27. Let $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$. By Lemma 4.24, $\mu \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$. Thus, $\mu \subseteq \widehat{S}$. Q. E. D.

Let $[\![\cdot]\!]_{\_} : \mathbb{S}_{\mathsf{A}} \to \mathbf{2}^{\mathcal{V}\times\mathcal{U}}$ be any semantics for SPARQL's basic graph patterns. As soon as the maximal dual simulation match for $\mathcal{Q}$ in DB is contained in $[\![\mathcal{Q}]\!]_{DB}$, $[\![\cdot]\!]_{\_}$ is automatically correct. We denote by $[\![\cdot]\!]^{\mathsf{MDS}}_{\_}$ the semantic function that associates every basic graph pattern $\mathcal{Q} \in \mathbb{S}_{\mathsf{A}}$ and graph database DB with the singleton set containing the maximal dual simulation match for $\mathcal{Q}$ in DB. Of course, $(\mathbb{S}_{\mathsf{A}}, [\![\cdot]\!]^{\mathsf{MDS}}_{\_})$ is correct by Corollary 4.49. From a practical point of view, computing only the maximal dual simulation match is useful as a pruning step for SPARQL query processing.

Example 4.50 Consider $DB_{4.4}$, as depicted in Figure 4.4, and the basic graph pattern

$$\mathcal{Q} = (\mathsf{p}, \mathit{teach}, \mathsf{c})\ \texttt{AND}\ (\mathsf{p}, \mathit{advise}, \mathsf{s})\ \texttt{AND}\ (\mathsf{s}, \mathit{take}, \mathsf{c}),$$

where we ask for professors (p) teaching courses (c) and advising students (s), so that the students s have taken the courses c. There is a single match for $\mathcal{Q}$ in $DB_{4.4}$ under SPARQL's original interpretation, namely

$$\mu\ (\mathsf{p} \mapsto \mathtt{Paul}, \mathsf{s} \mapsto \mathtt{Sara}, \mathsf{c} \mapsto \mathtt{DBS}).$$

The maximal dual simulation match $\widehat{S}$ for $\mathcal{Q}$ in DB reflects on μ since

$$\widehat{S} = \{(\mathsf{p}, \mathtt{Paul}), (\mathsf{s}, \mathtt{Sara}), (\mathsf{c}, \mathtt{DBS})\}.$$

Observe that in this particular case μ is even equal to $\widehat{S}$. If we concentrate the search of a SPARQL query processor to the subgraph of $DB_{4.4}$ that is induced by the maximal dual simulation match, the processor considers a (much) smaller database instance where still all the matches can be found. In general, the maximal dual simulation match will be imprecise as soon as cycles are not boundedly matched (cf. Section 3.1.4).

In the best case, the maximal dual simulation match is empty. For instance, query

$$\mathcal{Q}' = (\mathsf{p}, \mathit{advise}, \mathsf{s})\ \mathtt{AND}\ (\mathsf{s}, \mathit{teach}, \mathsf{c})$$

has an empty maximal dual simulation match w. r. t. $DB_{4.4}$. Therefore, Corollary 4.49 ensures $[\![\mathcal{Q}']\!]^{\mathbb{S}}_{DB_{4.4}} = \emptyset$. There is no need for any further query processing. ■

This section is devoted to studying an extension of $[\![\cdot]\!]^{\mathsf{MDS}}_{-}$ to SPARQL queries with (at least) optional patterns, for which correctness is guaranteed. Although we identified a correct fragment for $[\![\cdot]\!]^{\mathsf{DS}}_{-}$, already a single optional operator renders dual simulation matches not union-closed. As a consequence, maximal dual simulation matches are not unique anymore. To overcome this issue, we introduce a weaker notion of compatibility for optional patterns in Section 4.4.1. Surprisingly, the insights of Section 4.4.1 may be transferred to SPARQL queries with arbitrary nesting of AND and OPT operators. The thus obtained semantics extends $[\![\cdot]\!]^{\mathsf{MDS}}_{-}$ to the full query language of $\mathbb{S}_{\mathsf{AO}}$ while keeping the overall semantics' correctness intact. We prove correctness and tractability for $[\![\cdot]\!]^{\mathsf{MDS}}_{-}$. Beyond the formal results, we evaluate the effectiveness of $[\![\cdot]\!]^{\mathsf{MDS}}_{-}$ as a pruning mechanism for SPARQL query processing, as sketched in Example 4.50.

4.4.1 Weak Compatibility and Well-Designed SPARQL

The following example shows that, in general, the maximal dual simulation match is not well-defined for well-designed SPARQL.

Example 4.51 Consider an optional pattern $\mathcal{Q} = \mathcal{Q}_1\ \mathtt{OPT}\ \mathcal{Q}_2$, where $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{A}}$ with $\mathit{vars}(\mathcal{Q}_1) \cap \mathit{vars}(\mathcal{Q}_2) \neq \emptyset$, i. e., there is at least one variable v shared by $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Note that $\mathcal{Q}$ is a well-designed query that is in OPT normal form. We consider some graph database DB, such that there are $S_1, S_1' \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ and $S_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1 \rightleftarrows S_2$ but there is no $S_2' \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$ with $S_1' \rightleftarrows S_2'$. The following observations hold for the candidates S_1, S_1', S_2:

(1) $S_1 \cup S_2 \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ and $S_1' \in [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$, both by Definition 4.29.

(2) $S_1 \cup S_1' \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ by Lemma 4.26.

(3) $S_1' \not\rightleftarrows S_2$ by assumption, which implies that $(S_1 \cup S_1') \not\rightleftarrows S_2$. This is because $\mathcal{Q}_1 \in \mathbb{S}_{\mathsf{A}}$ implies that $\mathit{dom}(S_1) = \mathit{dom}(S_1') = \mathit{vars}(\mathcal{Q}_1)$ and, therefore, for some $v \in \mathit{dom}(S_1 \cup S_1') \cap \mathit{dom}(S_2)$, $vS_2 \subsetneq v(S_1 \cup S_1')$. Thus, the unified match $(S_1 \cup S_2) \cup S_1'$ is not a match for $\mathcal{Q}$ in DB. ■

This means, even in case of well-designed queries $\mathcal{Q}$, $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ is not union-closed and generally has no unique maximal dual simulation match. The reason is that the notion of compatibility is too strong to allow partial, such as S'_1, and full matches, such as $S_1 \cup S_2$, for optional patterns to coexist in a dual simulation match. Although match S_2 was incompatible with $S_1 \cup S'_1$, we can still observe $vS_2 \subseteq v(S_1 \cup S'_1)$ for each $v \in dom(S_1 \cup S'_1) \cap dom(S_2)$. A compatibility notion based on this observation resolves the issue of union-closedness. We call it *weak compatibility*.

Definition 4.52 (Weak Compatibility)
Let $S_1, S_2 \subseteq \mathcal{V} \times \mathcal{U}$. S_1 *is weakly compatible with* S_2, denoted by $S_1 \Leftarrow S_2$, iff for all $v \in dom(S_1) \cap dom(S_2)$, $vS_2 \subseteq vS_1$. ▲

Weak compatibility is not symmetric and for every two dual simulation candidates S_i ($i = 1, 2$), $S_1 \rightleftarrows S_2$ implies $S_1 \Leftarrow S_2$. Furthermore, every dual simulation candidate S is weakly compatible with the empty dual simulation ($\emptyset$). We resolve the missing union-closedness of dual simulation matches for optional patterns by incorporating weak compatibility into the matching process.

Example 4.53 (Example 4.51 continued) Requiring weak compatibility in the evaluation of $\mathcal{Q}_1$ OPT $\mathcal{Q}_2$ does not affect the match $S_1 \cup S_2$ because compatibility implies weak compatibility. Furthermore, it does not change the state of S'_1 as a dual simulation match for $\mathcal{Q}_1$ (because of $\mathcal{Q}_1 \in \mathbb{S}_{\mathsf{A}}$).

Weak compatibility does allow for the unified match, i. e., $(S_1 \cup S_2) \cup S'_1$. The reason is that $S_1 \cup S'_1$ is a match for $\mathcal{Q}_1$ (by Lemma 4.26) and S_2 is a match for $\mathcal{Q}_2$. Although $(S_1 \cup S'_1)$ and S_2 are not compatible in the strong sense, $(S_1 \cup S'_1) \Leftarrow S_2$: Since $\mathcal{Q}_1$ is a basic graph pattern, $dom(S_1) = dom(S'_1) = vars(\mathcal{Q}_1)$. Let $v \in dom(S_1 \cup S'_1) \cap dom(S_2)$. It holds that $vS_1 \neq \emptyset$ and $vS'_1 \neq \emptyset$. Now, $(S_1 \cup S'_1) \Leftarrow S_2$ because $S_1 \rightleftarrows S_2$, i. e., $vS_1 = vS_2$. Thus, $vS_1 \subseteq v(S_1 \cup S'_1)$ implies $vS_2 \subseteq v(S_1 \cup S'_1)$. ■

The integration of weak compatibility into the dual simulation semantics of well-designed SPARQL yields a new intermediate semantics to achieving the maximal dual simulation semantics. We call this intermediate semantics *dual simulation approximation* for well-designed SPARQL, denoted by $[\![\cdot]\!]^{\forall\mathsf{DS}}_{\_}$. In light of Proposition 4.39, it is defined for queries $\mathcal{Q} \in \mathbb{S}^{ONF}_{\mathsf{wd}}$, only.

Definition 4.54 (Dual Simulation Approximation Semantics)
Let $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database. The *dual simulation approximation of* $\mathcal{Q} \in \mathbb{S}^{ONF}_{\mathsf{wd}}$ *w. r. t.* DB is defined inductively on the structure of $\mathcal{Q}$:

$$
\begin{aligned}
[\![\mathbb{G}]\!]^{\forall\mathsf{DS}}_{DB} &:= [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB} \\
[\![\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} &:= \left\{ S_1 \cup S_2 \;\middle|\; \begin{array}{l} S_1 \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB} \wedge \\ S_2 \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} \cup \{\emptyset\} \wedge \\ S_1 \Leftarrow S_2 \end{array} \right\}
\end{aligned}
$$

An $S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ is called an *approximate match for* $\mathcal{Q}$ *in* DB. ▲

First note that for optional patterns $\mathcal{Q} = \mathcal{Q}_1$ OPT $\mathcal{Q}_2$, $[\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB} \subseteq [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$, which does not hold for $[\![\cdot]\!]^{\mathbb{S}}_{\_}$ or $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$. The reason is that every match $S_1 \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$ is weakly compatible with the empty dual simulation ($\emptyset$). Thus, $S_1 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ even if there is an $S_2 \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB}$ with $S_1 \Leftarrow S_2$.

Example 4.55 (Example 4.51 continued) Recall that $S'_1 \in [\![\mathcal{Q}_1]\!]^{\mathsf{DS}}_{DB}$ did not have any compatible $S'_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$. Thus, S'_1 is a dual simulation match for $\mathcal{Q}$ in DB. Although it is not guaranteed that there is no weakly compatible match $S'_2 \in [\![\mathcal{Q}_2]\!]^{\mathsf{DS}}_{DB}$, our formulation of the approximation semantics ensures that $S_1 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.

Since $S_1 \rightleftarrows S_2$, $S_1 \Leftarrow S_2$ is implied and $S_1 \cup S_2 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. S_1 becomes an additional approximate match for $\mathcal{Q}$ because $S_1 \Leftarrow \emptyset$. ∎

If $[\![\cdot]\!]^{\forall\mathsf{DS}}_{\_}$ is union-closed, then additional matches due to the just observed matching behavior dissolve in the maximal dual simulation match.

Proposition 4.56 *Let DB be a graph database and $\mathcal{Q} \in \mathbb{S}^{ONF}_{\mathsf{wd}}$. If $S, S' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$, then $S \cup S' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.*

PROOF: By induction on the structure of $\mathcal{Q}$.

Base: The case of $\mathcal{Q}$ being a basic graph pattern follows from Lemma 4.26 because $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB} = [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$.

Hypothesis: For $S, S' \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$ $(i = 1, 2)$, $S \cup S' \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$.

Step: We need to show that matches for $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$ have the desired property. In this case, $S = S_1 \cup S_2$ and $S' = S'_1 \cup S'_2$, where $S_1, S'_1 \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$ and $S_2, S'_2 \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} \cup \{\emptyset\}$. By induction hypothesis, $S_1 \cup S'_1 \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$. We distinguish two cases for S_2 and S'_2:

(i) If $S_2 = S'_2 = \emptyset$, it holds that $S_1 \cup S'_1 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ because every dual simulation candidate is weakly compatible with the empty dual simulation and $S_1 \cup S'_1 \cup \emptyset = S_1 \cup S'_1$.

(ii) If $S_2 \neq \emptyset$ or $S'_2 \neq \emptyset$, it holds that $S_2 \cup S'_2 \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB}$ by induction hypothesis (if both are nonempty) or by assumption (if only one of them is empty). We need to show that $S_1 \cup S'_1 \Leftarrow S_2 \cup S'_2$ because only then $(S_1 \cup S'_1) \cup (S_2 \cup S'_2)$ (i. e., $S \cup S'$) is an approximate match for $\mathcal{Q}$. Let $v \in dom(S_1 \cup S'_1) \cap dom(S_2 \cup S'_2)$. Towards a contradiction, suppose $v(S_2 \cup S'_2) \not\subseteq v(S_1 \cup S'_1)$. Then there is an $o \in O_{DB}$ with $(v, o) \in S_2$ (or $(v, o) \in S'_2$) and $(v, o) \notin S_1 \cup S'_1$. As $S_1 \Leftarrow S_2$, $v \notin dom(S_1)$. Hence, $v \in dom(S'_1) \setminus dom(S_1)$. But then there is an optional subpattern $P_1 \,\mathtt{OPT}\, P_2$ in $\mathcal{Q}_1$ with $v \in vars(P_2) \setminus vars(P_1)$. As $v \in dom(S_2 \cup S'_2)$, $v \in vars(\mathcal{Q}_2)$ which contradicts the assumption that $\mathcal{Q}$ is a well-designed query. Thus, the assumption that $v(S_2 \cup S'_2) \not\subseteq v(S_1 \cup S'_1)$ is wrong and $(S_1 \cup S'_1) \Leftarrow (S_2 \cup S'_2)$ implies $(S_1 \cup S'_1) \cup (S_2 \cup S'_2) \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Q. E. D.

The union of all approximate dual simulation matches yields a unique greatest element per query, the *maximal dual simulation match*.

Theorem 4.57 *For every $\mathcal{Q} \in \mathbb{S}^{ONF}_{\mathsf{wd}}$ and DB, if $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB} \neq \emptyset$, then there is a unique greatest element $\widehat{S} \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$, i. e., $\widehat{S} \subseteq S$ implies $\widehat{S} = S$ for all $S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.*

PROOF: Suppose there are two distinct greatest elements $S_1, S_2 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Then by Proposition 4.56, $S_1 \cup S_2 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Since $S_1 \subseteq S_1 \cup S_2$ and $S_2 \subseteq S_1 \cup S_2$, $S_1 = S_1 \cup S_2 = S_2$. If S_1 and S_2 were distinct, they are not the greatest elements of $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Thus, there is no more than one greatest element. Q. E. D.

Theorem 4.57 justifies the *maximal dual simulation semantics* for well-designed SPARQL.

Definition 4.58 (Maximal Dual Simulation Semantics)
Let *DB* be a graph database and $\mathcal{Q} \in \mathbb{S}^{ONF}_{\mathsf{wd}}$. Then the *maximal dual simulation semantics for $\mathcal{Q}$ in DB* is defined by $[\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB} = \{\bigcup_{S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}} S\}$. ▲

Before we can make use of this semantics, we need to prove its correctness. We do this in two steps. First, we show that $[\![\cdot]\!]^{\forall\mathsf{DS}}_{\_}$ really is an approximation of $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$ for well-designed queries. Correctness of $[\![\cdot]\!]^{\mathsf{MDS}}_{\_}$ is then a mere consequence of the correctness of $[\![\cdot]\!]^{\mathsf{DS}}_{\_}$.

Lemma 4.59 *For all* $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}^{ONF}$ *and graph databases* DB, $[\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}} \subseteq [\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$.

PROOF: By induction on the structure of $\mathcal{Q}$.

Base: In case of basic graph patterns, $\mathcal{Q} = \mathbb{G}$ and $[\![\mathbb{G}]\!]_{DB}^{\forall\mathsf{DS}} = [\![\mathbb{G}]\!]_{DB}^{\mathsf{DS}}$ by Definition 4.54. Thus, $[\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}} \subseteq [\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$.

Hypothesis: For $\mathcal{Q}_i \in \mathbb{S}_{\mathsf{wd}}^{ONF}$ $(i = 1, 2)$, $[\![\mathcal{Q}_i]\!]_{DB}^{\mathsf{DS}} \subseteq [\![\mathcal{Q}_i]\!]_{DB}^{\forall\mathsf{DS}}$.

Step: We show for $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$, $[\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}} \subseteq [\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$. For $S \in [\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}}$ we distinguish (i) $S \in [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}$ and (ii) $S \in [\![\mathcal{Q}_1]\!]_{DB}^{\mathsf{DS}}$ for which no $S' \in [\![\mathcal{Q}_2]\!]_{DB}^{\mathsf{DS}}$ exists with $S \rightleftarrows S'$.

(i) Thus, there are $S_i \in [\![\mathcal{Q}_i]\!]_{DB}^{\mathsf{DS}}$ $(i = 1, 2)$ with $S_1 \rightleftarrows S_2$ and $S = S_1 \cup S_2$. By induction hypothesis, $S_i \in [\![\mathcal{Q}_i]\!]_{DB}^{\forall\mathsf{DS}}$ $(i = 1, 2)$. Since $S_1 \rightleftarrows S_2$, $S_1 \Leftarrow S_2$ is implied and $S_1 \cup S_2 \in [\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$.

(ii) By induction hypothesis, $S \in [\![\mathcal{Q}_1]\!]_{DB}^{\mathsf{DS}}$ implies $S \in [\![\mathcal{Q}_1]\!]_{DB}^{\forall\mathsf{DS}}$. As $S \Leftarrow \emptyset$ and $S = S \cup \emptyset$, $S \in [\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$. Q. E. D.

Theorem 4.60 $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]_{\_}^{\mathsf{MDS}})$ *is correct.*

PROOF: It is sufficient to show that $(\mathbb{S}_{\mathsf{wd}}^{ONF}, [\![\cdot]\!]_{\_}^{\mathsf{MDS}})$ is correct. Let DB be a graph database, $\mathcal{Q} \in \mathbb{S}_{\mathsf{wd}}^{ONF}$, and $\mu \in [\![\mathcal{Q}]\!]_{DB}^{\mathbb{S}}$. By Corollary 4.38, there is a dual simulation match $S \in [\![\mathcal{Q}]\!]_{DB}^{\mathsf{DS}}$ with $\mu \subseteq S$. Furthermore, $S \in [\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$ by Lemma 4.59. Since the maximal dual simulation match $\widehat{S}$ for $\mathcal{Q}$ in DB (i. e., $[\![\mathcal{Q}]\!]_{DB}^{\mathsf{MDS}} = \{\widehat{S}\}$) includes all matches $S' \in [\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$, $\mu \subseteq \widehat{S}$ follows from $S \subseteq \widehat{S}$. Q. E. D.

We have seen that $[\![\cdot]\!]_{\_}^{\forall\mathsf{DS}}$ approximates $[\![\cdot]\!]_{\_}^{\mathsf{DS}}$ in Lemma 4.59, and so does $[\![\cdot]\!]_{\_}^{\mathsf{MDS}}$ (cf. Theorem 4.60). As $[\![\cdot]\!]_{\_}^{\overline{\mathsf{DS}}}$ is correct, it approximates itself SPARQL's original semantics. Subsequently, we obtain a maximal dual simulation semantics for $\mathbb{S}_{\mathsf{AO}}$-queries beyond the well-designed case. Therefore, we transfer the approximation principle of this section to more general queries.

4.4.2 Compatibility and Mandatory Variables

The maximal dual simulation semantics has only been shown correct for well-designed SPARQL queries. A slight change in the compatibility notion successfully established that single matches from $[\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$ could be unified to larger matches within $[\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$.a For non-well-designed SPARQL queries $\mathcal{Q}$, $[\![\mathcal{Q}]\!]_{DB}^{\forall\mathsf{DS}}$ is not necessarily union-closed.

Example 4.61 Consider the query

$$\mathcal{Q} = \underbrace{(\mathsf{p}, \mathit{teach}, \mathsf{c})}_{M} \,\mathtt{OPT}\, \underbrace{(\mathsf{p}, \mathit{advise}, \mathsf{s})}_{O_1} \,\mathtt{OPT}\, \underbrace{(\mathsf{s}, \mathit{take}, \mathsf{c})}_{O_2},$$

primarily asking for professors (p) and the courses they teach (c). If there is any student (s), who is advised by the professor (cf. O_1), then these students also belong to the match. Additionally, the query checks whether a student has taken the course (cf. O_2). If there is no student advised by the professor, just those students who attended the course are returned.

Reconsider the graph database depicted in Figure 4.4, denoted by $DB_{4.4}$. Although $\mathcal{Q}$ is not well-designed, $[\![\mathcal{Q}]\!]_{DB_{4.4}}^{\forall\mathsf{DS}}$ may still be computed based on Definition 4.54, and contains the following approximate matches:

- $S = \{(\mathsf{p}, \texttt{Paul}), (\mathsf{s}, \texttt{Sara}), (\mathsf{c}, \texttt{DBS})\}$ and
- $S' = \{(\mathsf{p}, \texttt{Paul}), (\mathsf{s}, \texttt{Seth}), (\mathsf{c}, \texttt{DBS})\}$.

However, $S \cup S' \notin [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. In order to realize this statement, consider the approximate match S as a match for the subpattern M OPT O_1 of $\mathcal{Q}$. In order to get the pair $(\mathsf{s}, \texttt{Seth})$ added to S, we have to consider the approximate match $S_2 = \{(\mathsf{s}, \texttt{Seth}), (\mathsf{c}, \texttt{DBS})\} \in [\![O_2]\!]^{\forall\mathsf{DS}}_{DB}$. But $S \Leftarrow S_2$ does not hold since $\{\mathsf{s}, \mathsf{c}\} = dom(S) \cap dom(S_2)$ and $\mathsf{s}S_2 = \{\texttt{Seth}\} \not\subseteq \{\texttt{Sara}\} = \mathsf{s}S$. ■

Hence, what was working so well for well-designed SPARQL does not work at all for non-well-designed queries. Let us take a closer look at what weak compatibility changed for well-designed optional patterns w. r. t. Example 4.61. Variable s did not only establish the counterexample but is also the reason for $\mathcal{Q}$ being non-well-designed. To see this, note that $\mathsf{s} \in vars(O_1) \setminus vars(M)$ but $\mathsf{s} \in vars(O_2)$ as well. Hence, in a well-designed query in the shape of $\mathcal{Q}$ in Example 4.61, $vars(O_1) \cap vars(O_2) \neq \emptyset$ requires that the variables shared between O_1 and O_2 must also occur in M, i. e., the mandatory subpattern of the whole query. Generalizing this thought about the shared variables of mandatory and optional subpatterns yields the following result for well-designed queries.

Proposition 4.62 *Let DB be a graph database and $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}^{ONF}_{\mathsf{wd}}$ such that $\mathcal{Q}_1$ OPT $\mathcal{Q}_2 \in \mathbb{S}^{ONF}_{\mathsf{wd}}$. For every matches $S_1 \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$ and $S_2 \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB}$, $dom(S_1) \cap dom(S_2) \subseteq vars(\mathsf{lmp}(\mathcal{Q}_1))$.*

PROOF: Let $v \in dom(S_1) \cap dom(S_2)$. Hence, $v \in vars(\mathcal{Q}_1) \cap vars(\mathcal{Q}_2)$. We proceed by induction on the structure of $\mathcal{Q}_1$, showing that $v \in vars(\mathcal{Q}_1)$ implies $v \in vars(\mathsf{lmp}(\mathcal{Q}_1))$.

Base: In case of a basic graph pattern $\mathcal{Q}_1$, $v \in vars(\mathsf{lmp}(\mathcal{Q}_1))$ follows from $\mathsf{lmp}(\mathcal{Q}_1) = \mathcal{Q}_1$.

Hypothesis: Suppose for every $P_1 \in \mathbb{S}^{ONF}_{\mathsf{wd}}$, $v \in vars(P_1)$ implies that $v \in vars(\mathsf{lmp}(P_1))$.

Step: Let $P_2 \in \mathbb{S}^{ONF}_{\mathsf{wd}}$. For $\mathcal{Q}_1 = P_1$ OPT P_2, it remains to be shown that $v \in vars(P_1)$ because $\mathsf{lmp}(\mathcal{Q}_1) = \mathsf{lmp}(P_1)$ and $v \in vars(\mathsf{lmp}(P_1))$ by induction hypothesis. There are two cases to distinguish: First, $v \notin vars(P_2)$ implies $v \in vars(P_1)$ since, by assumption, $v \in vars(\mathcal{Q}_1)$. If $v \in vars(P_2)$, $v \in vars(P_1)$ because $v \in vars(\mathcal{Q}_2)$ and $\mathcal{Q}$ is a well-designed query. Q. E. D.

Hence, whenever we check for (weak) compatibility between two dual simulation candidates, at least in the case of well-designed SPARQL, we essentially compare the values of the variables occurring in the left-most pattern of the query. This result does not only hold for matches of the approximate semantics. By Corollary 4.38 and Lemma 4.59, every dual simulation match, as well as every SPARQL match, is included in the dual simulation approximation of a well-designed query. Because of their general impact on query evaluation, we call the variables that occur in the left-most pattern of a query *mandatory variables*.

Example 4.63 In query $\mathcal{Q}$ from Example 4.61, we have that $\mathsf{s} \in vars(O_2) \cap vars(O_1)$, but s is not a mandatory variable of $\mathcal{Q}$ because $\mathsf{s} \notin vars(M)$. p as well as c are mandatory variables of $\mathcal{Q}$. ■

Let $\mathcal{Q}_1$ OPT $\mathcal{Q}_2$ be a well-designed SPARQL query and let $S_i \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$ $(i = 1, 2)$ be dual simulation approximations. Proposition 4.62 tells us that instead of checking for weak compatibility between S_1 and S_2, as defined by Definition 4.52, asking for $vS_2 \subseteq vS_1$ for all mandatory variables v of $\mathcal{Q}$, that are those $v \in vars(\mathsf{lmp}(\mathcal{Q}_1))$, yields an equivalent decision. The reason is that for $v \in vars(\mathsf{lmp}(\mathcal{Q}_1))$, either $v \in dom(S_2)$ or $v \notin dom(S_2)$. In

the former case, Proposition 4.62 guarantees that $v \in dom(S_1) \cap dom(S_2)$. Since $S_1 \Leftarrow S_2$, $vS_2 \subseteq vS_1$. In the latter case, $vS_2 = \emptyset$. Thus, $vS_2 \subseteq vS_1$.

Leaving the realm of well-designed queries means leaving OPT normal form queries. Thus, Proposition 4.62 does not apply anymore. However, if we continue to use mandatory variables of subqueries while checking for weak compatibility, we obtain a variant of the approximate matching semantics that does entail the uniqueness of the maximal match.

Example 4.64 (Example 4.61 continued) We could not show that $S \cup S'$ was an approximate match in $DB_{4.4}$ because we were required to check for weak compatibility between $S = \{(\mathsf{p}, \texttt{Paul}), (\mathsf{s}, \texttt{Sara}), (\mathsf{c}, \texttt{DBS})\}$ and $S_2 = \{(\mathsf{s}, \texttt{Seth}), (\mathsf{c}, \texttt{DBS})\}$. Recall that $S \in [\![M \,\texttt{OPT}\, O_1]\!]_{DB}^{\forall\mathsf{DS}}$ and $S_2 \in [\![O_2]\!]_{DB}^{\forall\mathsf{DS}}$. However, S and S_2 are compatible in the following sense: For variable c, $\mathsf{c}S_2 = \{\texttt{DBS}\} = \mathsf{c}S$. c is a mandatory variable of $M \,\texttt{OPT}\, O_1$, but so is p. Let us observe the relationship between $\mathsf{p}S_2$ and $\mathsf{p}S$. It holds that $\mathsf{p}S_2 = \emptyset \subseteq \{\texttt{Paul}\} = \mathsf{p}S$. Hence, S and S_2 are compatible in the weak sense if we only consider the mandatory variables of the mandatory pattern. ■

What we reach by this focus on mandatory variables is a *what-if* interpretation. Let $\mathcal{Q} = M \,\texttt{OPT}\, O$ be any optional pattern. In particular, M can be any $\mathbb{S}_{\mathsf{AO}}$-query, e. g., $M = P_1 \,\texttt{OPT}\, P_2$. Comparing the matches of O with the matches of M only up to the mandatory variables of M assumes the matches of M to be only matches to P_1. The approximation is equipped for the case that matches of P_1 do not have compatible counterparts of P_2. If we cut off the weak compatibility criterion in the sense of Example 4.64, we regain union-closedness of the implied approximation semantics, and in consequence, well-definedness of the maximal element. Once more, additional matches will dissolve in the maximal dual simulation match.

Since $\mathbb{S}_{\mathsf{AO}}$-queries do not enjoy[6] the OPT normal form (cf. Definition 4.42), we have to reconsider conjunctions $\mathcal{Q} = \mathcal{Q}_1 \,\texttt{AND}\, \mathcal{Q}_2$. As neither of the subpatterns, $\mathcal{Q}_1$ or $\mathcal{Q}_2$, is superior over the other, mandatory variables of $\mathcal{Q}$ are the mandatory variables of $\mathcal{Q}_1$ and those of $\mathcal{Q}_2$. As they are that central, we give a formal definition of *mandatory variables*. Note that we are now back at $\mathbb{S}_{\mathsf{AO}}$-queries, i. e., triple patterns, conjunctions, and optional patterns.

Definition 4.65 (Mandatory Variables)
Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. The set of all *mandatory variables of* $\mathcal{Q}$, denoted by $\mathcal{M}(\mathcal{Q})$, is inductively defined by

$$\begin{aligned} \mathcal{M}(t) &:= vars(t) \\ \mathcal{M}(\mathcal{Q}_1 \,\texttt{OPT}\, \mathcal{Q}_2) &:= \mathcal{M}(\mathcal{Q}_1) \\ \mathcal{M}(\mathcal{Q}_1 \,\texttt{AND}\, \mathcal{Q}_2) &:= \mathcal{M}(\mathcal{Q}_1) \cup \mathcal{M}(\mathcal{Q}_2) \end{aligned}$$

▲

Example 4.64 provided us with a way to incorporate mandatory variables in optional pattern matching. We do not have a feasible counterpart for conjunctions $\mathcal{Q}_1 \,\texttt{AND}\, \mathcal{Q}_2$ so far. The original compatibility notion is, once again, too strong to cope with partial and complete matches to optional subpatterns of $\mathcal{Q}_1$ or $\mathcal{Q}_2$.

Example 4.66 As this example's graph database, take $DB_{4.5}$ depicted in Figure 4.5. In $DB_{4.5}$, Urs teaches the course CCS and advises the student John. Ute took the same course some years ago and now advises student Kristin. Steve has taken CCS and, afterwards, also taught it to others. Furthermore, consider the following query $\mathcal{Q}$:

$$\mathcal{Q} = (\underbrace{(\mathsf{p}_1, \textit{teach}, \mathsf{c})}_{M_1} \,\texttt{OPT}\, \underbrace{(\mathsf{p}_1, \textit{advise}, \mathsf{s})}_{O_1}) \,\texttt{AND}\, (\underbrace{(\mathsf{p}_2, \textit{take}, \mathsf{c})}_{M_2} \,\texttt{OPT}\, \underbrace{(\mathsf{p}_2, \textit{advise}, \mathsf{s})}_{O_2})$$

[6] ;-)

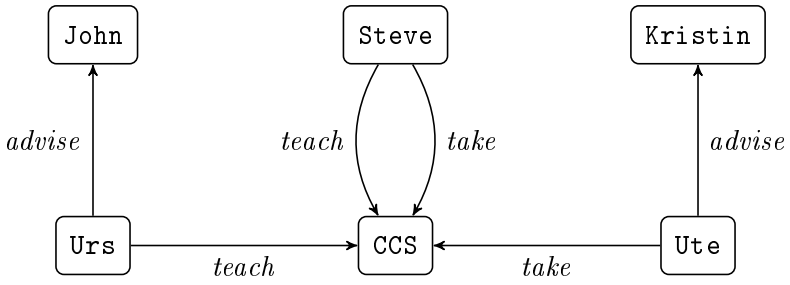


Figure 4.5: caption

This query is non-well-designed because, e.g., $\mathsf{s} \in \mathit{vars}(O_1) \setminus \mathit{vars}(M_1)$ but s is also a variable occurring in O_2. We know how to calculate $[\![M_i \,\mathtt{OPT}\, O_i]\!]^{\forall\mathsf{DS}}_{DB_{4.5}}$ $(i = 1, 2)$. If we provisionally apply the original compatibility notion, we obtain the following two results as matches of $\mathcal{Q}$:

- $S = \{(\mathsf{p}_1, \mathtt{Urs}), (\mathsf{c}, \mathtt{CCS}), (\mathsf{s}, \mathtt{John}), (\mathsf{p}_2, \mathtt{Steve})\}$ and
- $S' = \{(\mathsf{p}_1, \mathtt{Steve}), (\mathsf{c}, \mathtt{CCS}), (\mathsf{s}, \mathtt{Kristin}), (\mathsf{p}_2, \mathtt{Ute})\}$.

However, $S \cup S'$ cannot be a match of $\mathcal{Q}$ in $DB_{4.5}$. We get $S_1 \in [\![M_1 \,\mathtt{OPT}\, O_1]\!]^{\forall\mathsf{DS}}_{DB_{4.5}}$ with

$$S_1 = \{(\mathsf{p}_1, \mathtt{Urs}), (\mathsf{p}_2, \mathtt{Steve}), (\mathsf{c}, \mathtt{CCS}), (\mathsf{s}, \mathtt{John})\}$$

because $S_1^m \in [\![M_1]\!]^{\forall\mathsf{DS}}_{DB_{4.5}}$ and $S_1^o \in [\![O_1]\!]^{\forall\mathsf{DS}}_{DB_{4.5}}$, with

$$S_1^m = \{(\mathsf{p}_1, \mathtt{Urs}), (\mathsf{p}_1, \mathtt{Steve}), (\mathsf{c}, \mathtt{CCS})\} \text{ and } S_1^o = \{(\mathsf{s}, \mathtt{John}), (\mathsf{p}_1, \mathtt{Urs})\},$$

are weakly compatible. Analogously, $S_2 \in [\![M_2 \,\mathtt{OPT}\, O_2]\!]^{\forall\mathsf{DS}}_{DB}$ with

$$S_2 = \{(\mathsf{p}_2, \mathtt{Ute}), (\mathsf{p}_2, \mathtt{Steve}), (\mathsf{c}, \mathtt{CCS}), (\mathsf{s}, \mathtt{Kristin})\}.$$

S_1 and S_2 are the only two matches accounting fo $S \cup S'$, but they are incompatible because $\mathsf{s} \in \mathit{dom}(S_1) \cap \mathit{dom}(S_2)$ and $\mathsf{s}S_1 = \{\mathtt{John}\} \neq \{\mathtt{Kristin}\} = \mathsf{s}S_2$.

Some incorporation of weak compatibility would not help here, either, because $\mathsf{s}S_1$ and $\mathsf{s}S_2$ are incomparable. Recall that variable s is the reason for $\mathcal{Q}$ being non-well-designed. Thus, s is not mandatory in $\mathcal{Q}$ or its clauses: $\mathcal{M}(\mathcal{Q}) = \mathcal{M}(M_1 \,\mathtt{OPT}\, O_1) \cup \mathcal{M}(M_2 \,\mathtt{OPT}\, O_2) = \mathcal{M}(M_1) \cup \mathcal{M}(M_2) = \{\mathsf{p}_1, \mathsf{p}_2, \mathsf{c}\}$. Hence, if we focus on mandatory variables of the clauses, we should be successful in combining S and S', as desired. It holds that $vS_1 \subseteq vS_2$ for all $v \in \mathcal{M}(M_2 \,\mathtt{OPT}\, O_2)$ and $wS_2 \subseteq wS_1$ for all $w \in \mathcal{M}(M_1 \,\mathtt{OPT}\, O_1)$. If we allow this weak form of compatibility for conjunctions, $S_1 \cup S_2 = S \cup S'$ is regarded as a match of $\mathcal{Q}$ in $DB_{4.5}$.■

Thus, we need to ingest weak compatibility and the focus on mandatory variables in conjunctions. Due to the inherent symmetry of conjunctions, we have to ensure weak compatibility symmetrically. The following lemma shows that the notion of *mandatory variables* is justified by SPARQL's matching semantics.

Lemma 4.67 *Let DB be a graph database, $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$, and $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$. Then $\mathcal{M}(\mathcal{Q}) \subseteq \mathit{dom}(\mu)$.*

PROOF: By induction on the structure of $\mathcal{Q}$.

Base: Let $\mathcal{Q} = \mathbb{G} \in \mathbb{S}_{\mathsf{A}}$. Then $\mathcal{M}(\mathbb{G}) = \mathit{vars}(\mathbb{G})$ (by Definition 4.65) and $\mathit{dom}(\mu) = \mathit{vars}(\mathbb{G})$ (by (4.1) on Page 70). Thus, $\mathit{dom}(\mu) = \mathcal{M}(\mathbb{G})$.

Hypothesis: For $\mathcal{Q}_i \in \mathbb{S}_{\mathsf{AO}}$ $(i = 1, 2)$ and $\mu_i \in [\![\mathcal{Q}_i]\!]^{\mathbb{S}}_{DB}$, $\mathcal{M}(\mathcal{Q}_i) \subseteq \mathit{dom}(\mu_i)$.

Step: We need to show that $\mathcal{M}(\mathcal{Q}) \subseteq dom(\mu)$ if (i) $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2$ and (ii) $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$.

(i) For $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$, there are $\mu_1 \in [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB}$ and $\mu_2 \in [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$ with $\mu_1 \rightleftarrows \mu_2$ and $\mu = \mu_1 \cup \mu_2$. Thus, $dom(\mu) = dom(\mu_1) \cup dom(\mu_2)$. By induction hypothesis, $\mathcal{M}(\mathcal{Q}_i) \subseteq dom(\mu_i)$ $(i = 1, 2)$. Thus, $\mathcal{M}(\mathcal{Q}_1) \cup \mathcal{M}(\mathcal{Q}_2) = \mathcal{M}(\mathcal{Q}) \subseteq dom(\mu) = dom(\mu_1) \cup dom(\mu_2)$.

(ii) For $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$, either $\mu \in [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB}$ and there is no $\mu' \in [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$ with $\mu \rightleftarrows \mu'$, or $\mu \in [\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$. In the latter case, following the arguments of (i), we get $dom(\mu) = dom(\mu_1) \cup dom(\mu_2)$ and $\mathcal{M}(\mathcal{Q}_1) \subseteq dom(\mu_1)$. Thus, $\mathcal{M}(\mathcal{Q}) = \mathcal{M}(\mathcal{Q}_1) \subseteq dom(\mu_1) \cup dom(\mu_2) = dom(\mu)$. Furthermore, we have $\mathcal{M}(\mathcal{Q}) = \mathcal{M}(\mathcal{Q}_1) \subseteq dom(\mu_1) = dom(\mu)$ in the former case. Q. E. D.

In the following definitions, it first seems as if we were overloading the notions of *dual simulation approximation semantics*, *approximate matches*, and *maximal dual simulation semantics*. We will show that all the notions properly extend their counterparts defined in Definitions 4.54 and 4.58.

Definition 4.68 (Dual Simulation Approximation Semantics)
Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ and DB a graph database. The $\forall\mathsf{DS}$ *semantics of* $\mathcal{Q}$ *w. r. t.* DB is inductively defined by

$$
\begin{aligned}
[\![t]\!]^{\forall\mathsf{DS}}_{DB} &:= [\![t]\!]^{\mathsf{DS}}_{DB} \\
[\![\mathcal{Q}_1 \,\mathtt{AND}\, \mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} &:= \left\{ S_1 \cup S_2 \;\middle|\; \begin{array}{l} S_1 \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB} \wedge S_2 \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} \\ \forall v \in \mathcal{M}(\mathcal{Q}_1) : vS_2 \subseteq vS_1 \wedge \\ \forall v \in \mathcal{M}(\mathcal{Q}_2) : vS_1 \subseteq vS_2 \end{array} \right\} \\
[\![\mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} &:= \left\{ S_1 \cup S_2 \;\middle|\; \begin{array}{l} S_1 \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB} \wedge S_2 \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} \cup \{\emptyset\} \wedge \\ \forall v \in \mathcal{M}(\mathcal{Q}_1) : vS_2 \subseteq vS_1 \end{array} \right\}
\end{aligned}
$$

▲

Although this definition is quite different from Definition 4.54, it is just a conservative extension w. r. t. $\mathbb{S}^{ONF}_{\mathsf{wd}}$.

Proposition 4.69 *Let* $\mathcal{Q} \in \mathbb{S}^{ONF}_{\mathsf{wd}}$ *and* DB *be a graph database. Then the set of all approximate matches* $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ *w. r. t. Definition 4.54 is equal to the set of all approximate matches* $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ *according to Definition 4.68.*

Proof: We refer to the versions of the dual simulation approximation semantics in Definitions 4.54 and 4.68 by $[\![\cdot]\!]^{4.54}_{\_}$ and $[\![\cdot]\!]^{4.68}_{\_}$. Henceforth, we prove $[\![\mathcal{Q}]\!]^{4.54}_{DB} = [\![\mathcal{Q}]\!]^{4.68}_{DB}$ by induction on the structure of $\mathcal{Q}$.

Base: If $\mathcal{Q}$ is a basic graph pattern, i. e., $\mathcal{Q} = \mathbb{G} \in \mathbb{S}_{\mathsf{A}}$ or $\mathcal{Q} = t_1 \,\mathtt{AND}\, t_2 \,\mathtt{AND}\, \ldots \,\mathtt{AND}\, t_k$ $(\{t_1, t_2, \ldots, t_k\} = \mathbb{G})$. As $[\![\mathcal{Q}]\!]^{4.54}_{DB} = [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$, every match $S \in [\![\mathcal{Q}]\!]^{4.54}_{DB}$ has a sequence of dual simulation candidates $S_1, S_2, \ldots, S_k$, such that (i) $S_i \in [\![\mathcal{Q}_i]\!]^{\mathsf{DS}}_{DB}$, (ii) $S_i \rightleftarrows \bigcup_{0<j<i} S_j$ $(0 < i \leq k)$, and (iii) $S = S_1 \cup S_2 \cup \ldots \cup S_k$. Consider any $i \leq k$ $(i > 0)$. It holds that $\mathcal{M}(t_i) = vars(t_i)$ and $\mathcal{M}(t_1 \,\mathtt{AND}\, t_2 \,\mathtt{AND}\, \ldots \,\mathtt{AND}\, t_{i-1}) = \bigcup_{0<j<i} vars(t_j)$. For $v \in \mathcal{M}(t_i) \cap \mathcal{M}(t_1 \,\mathtt{AND}\, t_2 \,\mathtt{AND}\, \ldots \,\mathtt{AND}\, t_{i-1})$, $v(\bigcup_{0<j<i} S_j) = vS_i$ because of (ii) (i. e., $v(\bigcup_{0<j<i} S_i) \subseteq vS_i$ and $v(\bigcup_{0<j<i} S_i) \supseteq vS_i$). If $v \in \mathcal{M}(t_i) \setminus \mathcal{M}(t_1 \,\mathtt{AND}\, t_2 \,\mathtt{AND}\, \ldots \,\mathtt{AND}\, t_{i-1})$, $v(\bigcup_{0<j<i} S_j) = \emptyset \subseteq vS_i$. Conversely, $vS_i = \emptyset \subseteq v(\bigcup_{0<j<i} S_j)$ if $v \in \mathcal{M}(t_1 \,\mathtt{AND}\, t_2 \,\mathtt{AND} \ldots \mathtt{AND}\, t_{i-1}) \setminus \mathcal{M}(t_i)$. Thus, $S_i \cup \bigcup_{0<j<i} S_j \in [\![t_1 \,\mathtt{AND}\, t_2 \,\mathtt{AND} \ldots \mathtt{AND}\, t_{i-1} \,\mathtt{AND}\, t_i]\!]^{4.68}_{DB}$. By choosing $i = k$, we obtain the desired result.

Hypothesis: Suppose for $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}^{ONF}_{\mathsf{wd}}$, $[\![\mathcal{Q}_i]\!]^{4.54}_{DB} = [\![\mathcal{Q}_i]\!]^{4.68}_{DB}$ $(i = 1, 2)$.

Step: It remains to be shown that $[\![\mathcal{Q}]\!]^{4.54}_{DB} = [\![\mathcal{Q}]\!]^{4.68}_{DB}$ whenever $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$. We consider the two cases separately.

$\subseteq$: Let $S \in [\![\mathcal{Q}]\!]^{4.54}_{DB}$, i.e., there are $S_1 \in [\![\mathcal{Q}_1]\!]^{4.54}_{DB}$ and $S_2 \in [\![\mathcal{Q}_2]\!]^{4.54}_{DB} \cup \{\emptyset\}$, such that $S_1 \Leftarrow S_2$ and $S = S_1 \cup S_2$. By induction hypothesis (and by $\emptyset = \emptyset$), we get $S_1 \in [\![\mathcal{Q}_1]\!]^{4.68}_{DB}$ and $S_2 \in [\![\mathcal{Q}_2]\!]^{4.68}_{DB} \cup \{\emptyset\}$. We need to show that $vS_2 \subseteq vS_1$ for all $v \in \mathcal{M}(\mathcal{Q}_1)$. By $S_1 \Leftarrow S_2$, $v'S_2 \subseteq v'S_1$ for all $v' \in dom(S_1) \cap dom(S_2)$. As $dom(S_1) \cap dom(S_2) \subseteq \mathsf{Imp}(\mathcal{Q}_1)$ by Proposition 4.62 and $\mathsf{Imp}(\mathcal{Q}_1) = \mathcal{M}(\mathcal{Q}_1)$ by Definition 4.65 (and the fact that $\mathcal{Q}_1 \in \mathbb{S}^{ONF}_{\mathsf{wd}}$), $vS_2 \subseteq vS_1$ for all mandatory variables $v \in \mathcal{M}(\mathcal{Q}_1)$ that are jointly defined by S_1 and S_2. For a variable $v \in \mathcal{M}(\mathcal{Q}_1) \setminus (dom(S_1) \cap dom(S_2))$, we have that $vS_1 \neq \emptyset$ (a direct consequence of Lemma 4.46), so that $vS_2 = \emptyset$. Hence, $vS_2 \subseteq vS_1$ for all $v \in \mathcal{M}(\mathcal{Q}_1)$ and $S_1 \cup S_2 = S \in [\![\mathcal{Q}]\!]^{4.68}_{DB}$.

$\supseteq$: Let $S \in [\![\mathcal{Q}]\!]^{4.68}_{DB}$, i.e., there are $S_1 \in [\![\mathcal{Q}_1]\!]^{4.68}_{DB}$ and $S_2 \in [\![\mathcal{Q}_2]\!]^{4.68}_{DB} \cup \{\emptyset\}$, such that $vS_2 \subseteq vS_1$ for all $v \in \mathcal{M}(\mathcal{Q}_1)$ and $S = S_1 \cup S_2$. We need to show that $S_1 \Leftarrow S_2$, i.e., prove that $vS_2 \subseteq vS_1$ for all $v \in dom(S_1) \cap dom(S_2)$. By Proposition 4.62, $dom(S_1) \cap dom(S_2) \subseteq \mathsf{Imp}(\mathcal{Q}_1) = \mathcal{M}(\mathcal{Q}_1)$. Thus, the claim holds and $S_1 \cup S_2 = S \in [\![\mathcal{Q}]\!]^{4.54}_{DB}$. Q. E. D.

Thus, the purpose of the different style of semantics is found in handling arbitrarily nested `AND` and `OPT` operators in a correct and union-closed way. We first show that $[\![\cdot]\!]^{\forall\mathsf{DS}}_{\_}$ is correct.

Theorem 4.70 *Let DB be a graph database and $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$.*

$$[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \subseteq [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$$

Proof: Let $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$. By induction on $\mathcal{Q}$, we prove that $\mu \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.

Base: If $\mathcal{Q}$ is a triple pattern, then $\mu \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ because, by Lemma 4.24, $[\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB} \subseteq [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$ and, by definition, $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB} = [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$.

Hypothesis: For queries $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{AO}}$ and matches $\mu \in [\![\mathcal{Q}_i]\!]^{\mathbb{S}}_{DB}$ $(i = 1, 2)$, $\mu \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$.

Step: If $\mathcal{Q} = \mathcal{Q}_1$ `AND` $\mathcal{Q}_2$, there are (compatible) $\mu_i \in [\![\mathcal{Q}_i]\!]^{\mathbb{S}}_{DB}$ $(i = 1, 2)$ with $\mu = \mu_1 \cup \mu_2$. By induction hypothesis, $\mu_i \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$ $(i = 1, 2)$. As $\mu_1(v) = \mu_2(v)$ for all $v \in dom(\mu_1) \cap dom(\mu_2)$, $v\mu_1 \subseteq v\mu_2$ and $v\mu_2 \subseteq v\mu_1$. Let $v \in \mathcal{M}(\mathcal{Q}_1)$. By Lemma 4.67, $\mathcal{M}(\mathcal{Q}_1) \subseteq dom(\mu_1)$. Thus, if $v \in dom(\mu_2)$, $v\mu_2 \subseteq v\mu_1$ follows from compatibility of μ_1 and μ_2. If $v \notin dom(\mu_2)$, $v\mu_2 = \emptyset \subseteq v\mu_1$. Hence, $\mu_1 \cup \mu_2 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.

If $\mathcal{Q} = \mathcal{Q}_1$ `OPT` $\mathcal{Q}_2$, either (i) $\mu \in [\![\mathcal{Q}_1$ `AND` $\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB}$ or (ii) $\mu \in [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB}$. In case (i), the same argumentation as for $\mathcal{Q} = \mathcal{Q}_1$ `AND` $\mathcal{Q}_2$ applies. In case (ii), $\mu \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$ by induction hypothesis. $\mu = \mu \cup \emptyset$ and for all $v \in \mathcal{M}(\mathcal{Q}_1)$, $v\emptyset = \emptyset \subseteq v\mu$. Thus, $\mu \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Q. E. D.

In order to obtain a correct maximal dual simulation semantics, we have to further show that the approximation semantics is union-closed.

Lemma 4.71 *The set $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ is union-closed for all $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ and graph databases DB.*

Proof: Let $S, S' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ and let $DB = (O_{DB}, \Sigma, E_{DB})$. By induction on the structure of $\mathcal{Q}$, we show that $S \cup S' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.

Base: Since $[\![t]\!]^{\forall\mathsf{DS}}_{DB} = [\![t]\!]^{\mathsf{DS}}_{DB}$, $S \cup S' \in [\![t]\!]^{\forall\mathsf{DS}}_{DB}$ follows from $S \cup S' \in [\![t]\!]^{\mathsf{DS}}_{DB}$ by Lemma 4.26.

Hypothesis: For queries $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{AO}}$, if $S, S' \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$ $(i = 1, 2)$, then $S \cup S' \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$.

Step: In case $\mathcal{Q} = \mathcal{Q}_1 \,\texttt{AND}\, \mathcal{Q}_2$, there are $S_i, S_i' \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$ $(i = 1, 2)$, such that $S = S_1 \cup S_2$ and $S' = S_1' \cup S_2'$. By induction hypothesis, $S_i \cup S_i' \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$. We need to show that $v(S_j \cup S_j') \subseteq v(S_i \cup S_i')$ $(i, j = 1, 2,\, j \neq i)$ for all $v \in \mathcal{M}(\mathcal{Q}_i)$. Let $o \in v(S_j \cup S_j')$ for some $v \in \mathcal{M}(\mathcal{Q}_i)$. Then $o \in vS_j$ or $o \in vS_j'$. In the former case, $o \in vS_i$ since $vS_j \subseteq vS_i$. In the latter case, $o \in vS_i'$ since $vS_j' \subseteq vS_i'$. Thus, $o \in v(S_i \cup S_i')$. Thus, $S_1 \cup S_2 \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.

In case $\mathcal{Q} = \mathcal{Q}_1 \,\texttt{OPT}\, \mathcal{Q}_2$, there are $S_1, S_1' \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$ and $S_2, S_2' \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} \cup \{\emptyset\}$ with $S = S_1 \cup S_2$ and $S' = S_1' \cup S_2'$. As before, by induction hypothesis, $S_1 \cup S_1' \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$ and $S_2 \cup S_2' \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB} \cup \{\emptyset\}$. For $v \in \mathcal{M}(\mathcal{Q}_1)$, $v(S_2 \cup S_2') \subseteq v(S_1 \cup S_1')$ because $vS_2 \subseteq vS_1$ and $vS_2' \subseteq vS_1'$ (as in the case of $\mathcal{Q}_1 \,\texttt{AND}\, \mathcal{Q}_2$). Thus, $S \cup S' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$.

This argument carries over to any two $S, S' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$, making $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ a union-closed set. Q. E. D.

Thus, the greatest element of $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ is unique for all queries $\mathcal{Q}$ in $\mathbb{S}_{\mathsf{AO}}$.

Theorem 4.72 *Let $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ and DB a graph database. If $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB} \neq \emptyset$, then a greatest element (w. r. t. $\subseteq$) $\widehat{S} \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ exists. $\widehat{S}$ is unique and we call it the* maximal dual simulation between $\mathcal{Q}$ and DB.

Proof: We need to provide an $\widehat{S} \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ such that for all $S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ with $\widehat{S} \subseteq S$, $S = \widehat{S}$. Pick $\widehat{S} = \bigcup_{S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}} S$. By Lemma 4.71, $\widehat{S} \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Suppose there is a greatest element $\widehat{S}' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Then by Lemma 4.71, $\widehat{S} \cup \widehat{S}' \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ and $\widehat{S} \subseteq \widehat{S} \cup \widehat{S}'$. As $\widehat{S}$ is a greatest element of $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$, it holds that $\widehat{S} = \widehat{S} \cup \widehat{S}'$. Thus, a greatest element $\widehat{S}'$ distinct from $\widehat{S}$ cannot exist. Q. E. D.

This theorem justifies the following extension of the maximal dual simulation semantics for well-designed Sparql (cf. Definition 4.58) to arbitrary $\mathbb{S}_{\mathsf{AO}}$-queries.

Definition 4.73 (Maximal Dual Simulation Semantics)
Let DB be a graph database and $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. The *maximal dual simulation semantics of $\mathcal{Q}$ w. r. t. DB*, denoted $[\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB}$, is the singleton set containing the maximal dual simulation between $\mathcal{Q}$ and DB. ▲

By the results we obtained so far for the dual simulation approximation semantics, we deduce that we finally obtained a correct dual simulation denotation for arbitrary queries $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$.

Corollary 4.74 $(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathsf{MDS}}_{\_})$ *is correct.*

Proof: Let $\mu \in [\![\mathcal{Q}]\!]^{\mathbb{S}}_{DB}$. By Theorem 4.70, $\mu \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. The maximal dual simulation $\widehat{S}$ of $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ includes all elements of $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Thus, $\mu \subseteq \widehat{S}$. Q. E. D.

This result establishes half of our goals we formulated for query languages employing a semantic function whose foundation is established by dual simulations. The remaining goal is tractability of (a) the evaluation problem and (b) the non-emptiness problem. Regarding (b), the answer is unsatisfyingly positive: For every query $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ and every graph database DB, $[\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB} \neq \emptyset$. If $[\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB} = \emptyset$, $\bigcup_{S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}} S = \emptyset$ implies $[\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB} = \{\emptyset\}$. Hence, we may simply answer the non-emptiness problem by *yes* for every query $\mathcal{Q}$ and database DB (in $\mathcal{O}(1)$). The more interesting question is whether the maximal dual simulation match is empty. Therefore, this match must be computed and then checked for emptiness. The maximal dual simulation match can be computed in Ptime.

Theorem 4.75 *Computing the maximal dual simulation match for a query* $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ *w. r. t. graph database DB is in* PTIME.

The proof is postponed to Section 5.3.2.

Also for (a), the evaluation problem of $(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathsf{MDS}}_{\_})$, Theorem 4.75 delivers tractability. Let S be a candidate match. Checking, whether $S \in [\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB}$ is equivalent to the question whether $S = \widehat{S}$ where $\{\widehat{S}\} = [\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB}$. Therefore, we compute $\widehat{S}$ and compare it to S (for sure, in polynomial time). Thus, $(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathsf{MDS}}_{\_})$ has all the desired properties, from correctness up to tractability. In the following section, we identify and quantify the error due to the approximation of a query's result set. Therefore, we discuss and evaluate the effectiveness of $[\![\cdot]\!]^{\mathsf{MDS}}_{\_}$ as a pruning technique for $[\![\cdot]\!]^{\mathbb{S}}_{\_}$.

4.4.3 Effectiveness

Before diving into experiments about the quality of the maximal dual simulation semantics, let us first elaborate on the error we introduce when using it. The first error is due to cycles in the query (cf. Section 3.1.4). However, there are more because of the compatibility notion employed in dual simulation approximations.

Example 4.76 First, reconsider from Example 4.61 the query

$$\mathcal{Q}_1 = \underbrace{(\mathsf{p}, \mathit{teach}, \mathsf{c})}_{M} \,\mathtt{OPT}\, \underbrace{(\mathsf{p}, \mathit{advise}, \mathsf{s})}_{O_1} \,\mathtt{OPT}\, \underbrace{(\mathsf{s}, \mathit{take}, \mathsf{c})}_{O_2}.$$

If we switch O_1 with O_2 in $\mathcal{Q}_1$, we obtain a different query $\mathcal{Q}_2$, i. e.,

$$\mathcal{Q}_2 = \underbrace{(\mathsf{p}, \mathit{teach}, \mathsf{c})}_{M} \,\mathtt{OPT}\, \underbrace{(\mathsf{s}, \mathit{take}, \mathsf{c})}_{O_2} \,\mathtt{OPT}\, \underbrace{(\mathsf{p}, \mathit{advise}, \mathsf{s})}_{O_1}.$$

$\mathcal{Q}_1$ and $\mathcal{Q}_2$ are not equivalent up to SPARQL's semantics, e. g., $[\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB_{4.4}} \neq [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB_{4.4}}$ ($DB_{4.4}$ is depicted in Figure 4.4). $\mu \in [\![\mathcal{Q}_2]\!]^{\mathbb{S}}_{DB_{4.4}}$ with $\mu = \{(\mathsf{p}, \mathtt{Paul}), (\mathsf{c}, \mathtt{DBS}), (\mathsf{s}, \mathtt{Seth})\}$, but $\mu \notin [\![\mathcal{Q}_1]\!]^{\mathbb{S}}_{DB_{4.4}}$. Aiming for the most effective pruning, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ generally need different ones. However, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are equivalent up to the maximal dual simulation semantics, i. e., $[\![\mathcal{Q}_1]\!]^{\mathsf{MDS}}_{DB} = [\![\mathcal{Q}_2]\!]^{\mathsf{MDS}}_{DB}$ for all graph databases DB. In order to show equivalence, we prove the approximation denotations equivalent.

Proposition 4.77 *For all graph databases DB,* $[\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB} = [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB}$.

PROOF: We show the two directions, separately.

- $\subseteq$: Let $S \in [\![\mathcal{Q}_1]\!]^{\forall\mathsf{DS}}_{DB}$, i. e., there are $S_0, S_1, S_2 \subseteq \mathcal{V} \times \mathcal{U}$ with $S_0 \in [\![M]\!]^{\forall\mathsf{DS}}_{DB}$, $S_1 \in [\![O_1]\!]^{\forall\mathsf{DS}}_{DB} \cup \{\emptyset\}$, $S_2 \in [\![O_2]\!]^{\forall\mathsf{DS}}_{DB} \cup \{\emptyset\}$, and $S = S_0 \cup S_1 \cup S_2$. Furthermore, $\mathcal{M}(\mathcal{Q}_1) = \mathcal{M}(M \,\mathtt{OPT}\, O_1) = \mathcal{M}(M) = \{\mathsf{p}, \mathsf{c}\}$, $\mathsf{p}S_1 \subseteq \mathsf{p}S_0$ (i. e., $S_0 \cup S_1 \in [\![M \,\mathtt{OPT}\, O_1]\!]^{\forall\mathsf{DS}}_{DB}$), and $\mathsf{p}S_2 \subseteq \mathsf{p}(S_0 \cup S_1)$. Since $\mathcal{M}(\mathcal{Q}_2) = \mathcal{M}(M \,\mathtt{OPT}\, O_2) = \mathcal{M}(M)$, we need to show that (i) $\mathsf{p}S_2 \subseteq \mathsf{p}S_0$ (i. e., that $S_0 \cup S_2 \in [\![M \,\mathtt{OPT}\, O_2]\!]^{\forall\mathsf{DS}}_{DB}$) and (ii) $\mathsf{p}S_1 \subseteq \mathsf{p}(S_0 \cup S_2)$. From (i) and (ii) it follows that $S \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB}$. (i) itself follows from the fact that $\mathsf{p}S_2 \subseteq \mathsf{p}(S_0 \cup S_1)$. (ii) follows from $\mathsf{p}S_1 \subseteq \mathsf{p}S_0$. Thus, $S \in [\![\mathcal{Q}_2]\!]^{\forall\mathsf{DS}}_{DB}$.
- $\supseteq$: This direction follows the exact same steps as before, only backwards. Q. E. D.

Hence, although $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are different up to the SPARQL semantics, they have the same maximal dual simulation match. A quite similar argument can be found for the query pair $\mathcal{Q}_3, \mathcal{Q}_4$:

$$\begin{aligned}
\mathcal{Q}_3 &= (\underbrace{(\mathsf{p}, \mathit{teach}, \mathsf{c})}_{M_1} \,\mathtt{OPT}\, \underbrace{(\mathsf{p}, \mathit{advise}, \mathsf{s})}_{O}) \,\mathtt{AND}\, \underbrace{(\mathsf{s}, \mathit{take}, \mathsf{c})}_{M_2} \\
\mathcal{Q}_4 &= (\underbrace{(\mathsf{p}, \mathit{teach}, \mathsf{c})}_{M_1} \,\mathtt{AND}\, \underbrace{(\mathsf{s}, \mathit{take}, \mathsf{c})}_{M_2}) \,\mathtt{OPT}\, \underbrace{(\mathsf{p}, \mathit{advise}, \mathsf{s})}_{O}
\end{aligned}$$

Transforming $\mathcal{Q}_3$ into $\mathcal{Q}_4$ is exactly what we did for well-designed queries to obtain the OPT normal form. This transformation does only work for well-designed queries. Hence, the queries are not equivalent up to SPARQL's semantics. Once more, our maximal dual simulation semantics does not distinguish them.

Proposition 4.78 *For all graph databases DB,* $[\![\mathcal{Q}_3]\!]^{\forall \mathsf{DS}}_{DB} = [\![\mathcal{Q}_4]\!]^{\forall \mathsf{DS}}_{DB}$.

PROOF: We show the two directions, separately.

$\subseteq$: Let $S \in [\![\mathcal{Q}_3]\!]^{\forall \mathsf{DS}}_{DB}$, i. e., there are $S_0, S_1, S_2 \subseteq \mathcal{V} \times \mathcal{U}$ with $S_1 \in [\![M_1]\!]^{\forall \mathsf{DS}}_{DB}$, $S_0 \in [\![O]\!]^{\forall \mathsf{DS}}_{DB} \cup \{\emptyset\}$, $S_2 \in [\![M_2]\!]^{\forall \mathsf{DS}}_{DB}$, and $S = S_0 \cup S_1 \cup S_2$. Furthermore, $\mathcal{M}(\mathcal{Q}_1) = \mathcal{M}(M_1 \,\mathtt{OPT}\, O) \cup \mathcal{M}(M_2) = \mathcal{M}(M_1) \cup \mathcal{M}(M_2) = \mathcal{M}(M_1 \,\mathtt{AND}\, M_2) = \mathcal{M}(\mathcal{Q}_2) = \{\mathsf{p}, \mathsf{c}, \mathsf{s}\}$, so that

(i) $\mathsf{p}S_0 \subseteq \mathsf{p}S_1$ and $\mathsf{c}S_0 \subseteq \mathsf{c}S_1$, and

(ii) $\mathsf{p}(S_1 \cup S_0) = \mathsf{p}S_2$ and $\mathsf{c}(S_1 \cup S_0) = \mathsf{c}S_2$.

From (i) and (ii), we get that $\mathsf{p}S_1 = \mathsf{p}S_2$ and $\mathsf{c}S_1 = \mathsf{p}S_2$, i. e., $S_1 \cup S_2 \in [\![M_1 \,\mathtt{AND}\, M_2]\!]^{\forall \mathsf{DS}}_{DB}$. Moreover, $\mathsf{p}S_0 \subseteq \mathsf{p}(S_1 \cup S_2)$ and $\mathsf{c}S_0 \subseteq \mathsf{c}(S_1 \cup S_2)$ by (i). Hence, $S \in [\![\mathcal{Q}_4]\!]^{\forall \mathsf{DS}}_{DB}$.

$\supseteq$: This direction follows the same steps as before, only backward. Q. E. D.

Hence, some queries should be distinguished, but they are not by maximal dual simulations. The approximations introduced in Definition 4.68 are the reason for this. ∎

Thus, experiments on real-world and synthetic datasets must be performed, so that the actual error, introduced by maximal dual simulations, may be observed. Phrased differently, how close are we to a hypothetical, optimal pruning technique? How do the instance size and the number of optional patterns influence the prunings' quality?

Our tool SPARQLSIM[7] implements the maximal dual simulation semantics for SPARQL. Besides the computation of the maximal dual simulation match, the tool returns two numbers relevant for assessing the effectiveness of the procedure:

(1) The number of edges in the database that are covered by the maximal dual simulation match. An edge (o, a, o') of a give database DB is *covered by the maximal dual simulation match* $\widehat{S}$ of a query $\mathcal{Q}$ in DB iff $\mathcal{Q}$ contains a triple pattern $(\mathsf{x}, a, \mathsf{y})$, such that $o \in \mathsf{x}\widehat{S}$ and $o' \in \mathsf{y}\widehat{S}$. This is the number of edges (or triples) a (graph) query processor still has to consider after the maximal dual simulation match has been computed.

(2) The number of edges in DB with the labels occurring in $\mathcal{Q}$. A label $a \in \mathcal{P}$ occurs in $\mathcal{Q}$ iff $\mathcal{Q}$ has a triple pattern $(\mathsf{x}, a, \mathsf{y})$. This is the number of edges a graph query processor considers if no other (external) pruning mechanism applies.

We consider (2) as a naïve baseline. SPARQLSIM computed these numbers for all our queries (cf. Appendix A.3) on their respective datasets (cf. Appendix A.2). Since this is no performance evaluation, it is irrelevant on which machine these numbers were obtained. The results are identical on IS68 and IS69 (cf. Appendix A.1). We compiled the numbers in Appendix A.4. In the respective tables, (2) is represented by "Base"-columns while the values for (1) obtained the title "SPARQLSIM". For LUBM and DBpedia, we additionally collected the number of matches ("Result No."), as produced by Virtuoso or RDFox (cf. Appendix A.1), and

(3) the number of edges (or triples, resp.) required for producing the result set.

[7] available at GITHUB: `https://github.com/ifis-tu-bs/sparqlSim`

We titled the columns representing (3) by "Gold" since they refer to the theoretically best pruning approach. The four different LDBC datasets of different sizes allow us to reflect on the effectiveness of SPARQLSIM depending on different sizes of schematically similar instances. By the experiments on Wikidata, we observe the influence of different numbers of optional operators in the queries w. r. t. the baseline, additionally to the observations made on DBpedia and LUBM.

Because the gold standard is available for DBpedia and LUBM queries, we first discuss the results on these datasets (cf. Table A.4). The effectiveness of the maximal dual simulations (SPARQLSIM) on LUBM ranges from above 80% up to almost 100% (cf. $\mathcal{L}_4$, $\mathcal{L}_5$, $\mathcal{L}_6$). The baseline has an average effectiveness of 52%. On DBpedia our average effectiveness if is 99.999%, but the baseline also performs quite well with 95% effectiveness on average. The baseline's effectiveness values vary drastically between LUBM and DBpedia because only 18 predicates share more than 1 billion triples for LUBM, while less than 1 Billion triples are distributed over 65,000 predicates. In many cases, we still prune 10% more triples than the baseline approach on DBpedia, paired with a decent runtime to produce the pruning (cf. Section 5.3.4). Summarizing, the maximal dual simulation semantics has the potential to get a close-gold-standard pruning.

For the remaining datasets, we have only the baseline values and the SPARQLSIM values to compare. We declare our maximal dual simulation pruning having a bad quality if we are close to the baseline pruning. On Wikidata (cf. Table A.6), we first observe that the baseline/SPARQLSIM ratio is worse than we had on DBpedia: maximal dual simulations reached 12% of the baseline pruning on DBpedia and only 50% on Wikidata. One reason is that our DBpedia queries are prevalently basic graph patterns while only 12 out of 61 Wikidata queries have no optional operator. Therefore, the first type of error (cycles) paired with the second type of error (optional nesting) may apply, so that the overall pruning size of maximal dual simulations is generally closer to the baseline pruning. In contrast, only six out of twelve queries with more than two optional patterns show almost baseline qualities. Our extreme case is query $\mathcal{W}_9$ with 15 triple patterns and ten optional operators. Maximal dual simulations account for only six triple patterns in the pruning while the baseline approach differs by six orders of magnitude. Thus, the results regarding the influences of optional patterns do not allow for a final conclusion.

The pruning behavior on the differently sized LDBC datasets is rather interesting. The larger the dataset, the closer the maximal dual simulation pruning gets to the baseline. Note, the LDBC queries have no optional operators (cf. Appendix A.3). It seems that the number of cycles increases with the dataset size.

4.5 Summary

In this chapter, we have analyzed the impact of SPARQL's operator structure upon dual simulations. We reflected on two issues, namely correctness and tractability. While SPARQL's evaluation and non-emptiness problems are generally intractable, we aimed for tractable alternatives based on dual simulations. The results of this chapter are summarized for all non-intermediary query languages in Table 4.1.

Correctness was a notion we had to pin down to a variant of completeness employing subsumption of result sets. With regard to SPARQL queries, any new semantics producing a result set for them must contain matches reflecting on SPARQL's original semantics. Of course, the first three lines of Table 4.1 contain query languages that are correct by definition because the SPARQL semantics is correct w. r. t. itself. The first pure dual simulation-based semantics we obtained for $\mathbb{S}_{\mathsf{AO}}$ was, unfortunately, shown to be incorrect. As a remedy, the employed semantic function was shown correct for the SPARQL fragment of well-designed queries ($\mathbb{S}_{\mathsf{wd}}$) and tractability is also entailed. Both problems,

Table 4.1: Result Summary of Chapter 4

Language	Correctness	Evaluation	NonEmpty
$(\mathbb{S}, [\![\cdot]\!]^{\mathbb{S}}_{-})$	yes Definition 4.9	Pspace-complete Proposition 4.18	at least np-complete Proposition 4.20
$(\mathbb{S}_{\mathsf{A}}, [\![\cdot]\!]^{\mathbb{S}}_{-})$	yes Definition 4.9	Ptime Proposition 4.19	np-complete Proposition 4.20
$(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathbb{S}}_{-})$	yes Definition 4.9	conp-complete Proposition 4.18	at least np-complete Proposition 4.20
$(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathsf{DS}}_{-})$	no Proposition 4.34	—	—
$(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathsf{DS}}_{-})$	yes Corollary 4.38	Ptime Theorem 4.48	Ptime Theorem 4.44
$(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathsf{MDS}}_{-})$	yes Theorem 4.60	Ptime Theorem 4.75	$(\mathcal{O}(1))$ Definition 4.58
$(\mathbb{S}_{\mathsf{AO}}, [\![\cdot]\!]^{\mathsf{MDS}}_{-})$	yes Corollary 4.74	Ptime Theorem 4.75	$(\mathcal{O}(1))$ Definition 4.73

the evaluation problem and the non-emptiness problem of $(\mathbb{S}_{\mathsf{wd}}, [\![\cdot]\!]^{\mathsf{DS}}_{-})$, are solvable in Ptime. The structural feature we built upon was the so-called OPT *normal form*, which exists for all queries and is virtually effortless to obtain.

Guided by the principles of $\mathbb{S}_{\mathsf{wd}}$ that allow for correct dual simulation denotations of queries, we studied the property of union-closedness of dual simulations for Sparql queries. A correct Sparql semantics that is union-closed entails a pruning technique for Sparql query processing because the maximal match, guaranteed to exist by union-closedness, would contain all the matches of Sparql's original interpretation. Regrettably, the dual simulation semantics $([\![\cdot]\!]^{\mathsf{DS}}_{-})$ is not union-closed, even for well-designed Sparql. Therefore, we introduced an intermediary approximation semantics of $[\![\cdot]\!]^{\mathsf{DS}}_{-}$ that is union-closed. The key to union-closedness of the approximation is a new notion of compatibility for optional patterns. The weak compatibility principle could also be applied to non-well-designed queries, from which we obtain a correct and tractable semantics for $\mathbb{S}_{\mathsf{AO}}$, that uses dual simulations.

Concerning our goals, we succeeded in finding dual simulation-based semantic interpretations of Sparql queries fulfilling both our requirements. Although the naïve incorporation of dual simulations in Sparql introduces an inevitable error, its approximation preserves all Sparql matches and can be used as a Sparql query pre-processing mechanism. In our experiments, we saw that the maximal dual simulations semantics' effectiveness often removes more than 90% of the irrelevant triples and often improves upon the naïve baseline (sum of all predicate table entries) by 10%.

In recent years, the class of *weakly well-designed patterns*, which is a generalization of well-designed Sparql, has been studied [71, 72, 73]. Unfortunately, our negative results are not affected by this normal form because our counterexample showing incorrectness of the dual simulation semantics (cf. 4.34) is weakly well-designed.

CHAPTER 5

Graph Processing

The previous chapters were involved with applying different simulation notions as a method for various graph database management tasks, from object classification according to a graph schema, over graph pattern matching, up to graph query evaluation. Thereby, we always pointed to the tractable nature of simulations without actually explaining how to compute them. In this chapter, we introduce the basic algorithmic principles behind solving the *special dual simulation problem*. Therefore, two graphs, $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$, as well as a candidate relation $S_0 \subseteq V_Q \times V$, called the *dual simulation candidate*, are given as input. As in Chapter 3, Q is called the *pattern graph* while G is referred to as the *data graph*. An algorithm solving the problem returns the greatest dual simulation S included in S_0. Recall that dual simulations are union-closed (Theorem 2.29 (1)), making the output of dual simulation procedures unique.

Problem (DualSim)
Input: Labeled graphs $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$; dual simulation candidate $S_0 \subseteq V_Q \times V$.
Output: The greatest dual simulation $S \subseteq S_0$ between Q and G.

We use the notation $\textsc{DualSim}(Q, G, S_0)$ to denote the output of the DualSim problem for instance Q, G, S_0. In the (dual) simulation literature, e. g., [69, 22], we find solutions to a slightly different problem, which we call the *maximal (dual) simulation problem*, for which the candidate S_0 is not part of the input. Instead, a dedicated initialization step takes place, setting S_0 to $V_Q \times V$, which is the greatest possible (dual) simulation expecting every pattern node to be dual simulated by every data node. The major source of complexity remains with solving DualSim based on $S_0 = V_Q \times V$.

Problem (DualSimMax)
Input: Labeled graphs $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$.
Output: The maximal dual simulation $\widehat{S}$ between Q and G.

Continuing the notation above, we have $\textsc{DualSimMax}(Q, G) = \textsc{DualSim}(Q, G, V_Q \times V)$. Recall that maximal dual simulations are needed for obtaining object classifications, induced by a graph schema S onto a graph database DB (cf. Definition 2.27 on Page 23). The desired object classification is, thus, computed by $\vdash_S^{DB} = \textsc{DualSimMax}(DB, S)$. Object classification requires DB to be an instance of S. If DB is not an instance of S, the result of $\textsc{DualSimMax}(DB, S)$ is the empty dual simulation $R_\emptyset = \emptyset$. This way, a solution to DualSimMax provides us with a solution to the *non-emptiness problem of dual simulations*.

Problem (DualSimNonEmpty)
Input: Labeled graphs $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$.
Output: True if $Q \sqsubseteq_{\mathsf{Dsim}} G$. False, otherwise.

Corollary 2.32 justifies our decision procedure leveraging DualSimMax. Even the *evaluation problem of dual simulations* benefits from procedures for DualSim. Thus, any solution to the special dual simulation problem is a versatile tool.

Problem (DualSimEvaluation)
Input: Labeled graphs $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$; a simulation candidate $S \subseteq V_Q \times V$.
Output: True iff S is a dual simulation between Q and G.

As for the query evaluation problem, we ask whether a candidate (here a potential dual simulation $S \subseteq V_Q \times V$) actually is a dual simulation between the given graphs. Solving this problem may again utilize any DualSim procedure because DualSimEval(Q, G, S) evaluates to True iff DualSim$(Q, G, S) = S$, i. e., the greatest dual simulation between Q and G, included in S, is S itself.

Every one of the problems above is reducible to DualSim. Therefore, we concentrate on the solutions to that problem.

Contribution. Algorithms for solving DualSim are based on well-studied solutions to the *simulation problem* [86]. Therefore, we extend[1] and analyze existing solutions to the simulation problem in terms of runtime complexity. We include HHK [69], the most prominent similarity algorithm, based on its reception in the literature [69, 117, 29, 26, 50, 51, 85, 48, 53, 64, 56, 135]. Somewhat surprisingly, virtually all existing algorithms scale equally well with the size of the data graph G, including HHK (cf. Section 5.1.2). However, they are inflexible in how they compute (dual) simulations, making them neither extensible for, nor applicable to our Sparql semantics based on dual simulations (Sections 4.2 to 4.4). We propose and evaluate a new solution that we base on a *system of inequalities* representation of the dual simulation problem (cf. Section 5.2). The inequalities' view on DualSim allows us to naturally cope with Sparql's operator structure, implementing the maximal dual simulation semantics of Sparql (cf. Section 4.4). There, we also evaluate the efficiency of the maximal dual simulation semantics as a Sparql pruning approach. The system of inequalities has been published in the proceedings of the 35th IEEE International Conference on Data Engineering (ICDE 2019) [93]. Compared to our previous work [93, 92], we provide more in-depth material on existing simulation algorithms, allowing for a fine-grained comparison to our solution. Furthermore, we give a formal translation function mapping Sparql queries into systems of inequalities. We also sketch how unions and built-in filter conditions can be included.

Outline. Section 5.1 adapts and analyzes existing algorithms from the literature for solving the simulation problem. In Section 5.2, we present our algorithm, that regards dual simulations as solutions of a system of inequalities. We integrate Sparql's operator structure into the system of inequalities in Section 5.3. Thereby, we obtain an effective and efficient pruning procedure for Sparql queries through the maximal dual simulation semantics from Section 4.4. Finally, we summarize this chapter's findings in Section 5.4.

5.1 Simulation Algorithms

To the best of our knowledge, published algorithms for solving (dual) simulation problems [22, 69, 128, 60, 117, 132, 86] work all on the principle of *coinduction*, which is

[1] Simulation algorithms usually presume unlabeled directed graphs.

Algorithm 1: Naïve Coinductive Procedure

Input : $Q = (V_Q, \Sigma, E_Q)$, $G = (V, \Sigma, E)$, and $S_0 \subseteq V_Q \times V$.
Output: Greatest simulation $S \subseteq S_0$ between Q and G.

```
1 S ← S_0;
2 while there are nodes v, w ∈ V_Q and u ∈ vS, such that v E_Q^a w and uE^a ∩ wS = ∅
    do
3 |   S ← S \ {(v, u)};
4 end
```

implied by the coinductive nature of the definition itself (cf. Definition 2.16). Starting with the largest possible relation (often $S_0 = V_Q \times V$) between the two node sets, or some representation thereof, the algorithms incrementally disqualify pairs of nodes violating the requirements of (dual) simulations. The procedures terminate as soon as they can no longer disqualify any pair of nodes. The algorithms presented in Sections 5.1.1 and 5.1.2 follow this basic principle. Space-efficient solutions, which we sketch in Section 5.1.3, only indirectly use this approach in favor of reduced memory consumption. In this section, we discuss solutions to the *special simulation problem*, instead of its dual simulation counterpart.

Problem (Sim)
Input: Labeled graphs $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$; a simulation candidate $S_0 \subseteq V_Q \times V$.
Output: The greatest simulation $S \subseteq S_0$ between Q and G.

Focusing on the simulation problem makes the presentation of the algorithms more compact, especially in case of HHK covered in Section 5.1.2. The respective dual simulation procedures would copy the body of the algorithms to account for backward edges. The overall complexity is not affected.

5.1.1 Naïve Coinduction

The easiest way to achieve a solution is to implement the sketched procedure above directly. Henzinger et al. describe it as *schematic similarity* [69]. Algorithm 1 is a variant of the solution given by Henzinger et al. [69], extended to cope with edge labels and the distinction between pattern Q and data graph G. Furthermore, we do not intend to establish similarity (i. e., $\sqsubseteq_{\mathsf{sim}} \cap \sqsubseteq_{\mathsf{sim}}^{-1}$) between Q and G, but only find a simulation between Q and G that is included in S_0. Algorithm 1 iteratively checks whether the current relation S, initially $S = S_0$ (Line 1), fulfills the simulation properties of Definition 2.16. If not, Line 3 updates S, removing the false simulation candidate u for v. In the worst case, $S_0 = V_Q \times V$ (as in DualSimMax) so that Line 3 is passed $|V_Q| \cdot |V|$ times.

Example 5.1 Let us assume Q to be an a-loop, as given in Figure 5.1(a), and G a finite a-sequence, as in Figure 5.1(b), where k is some fixed positive integer. Initially,

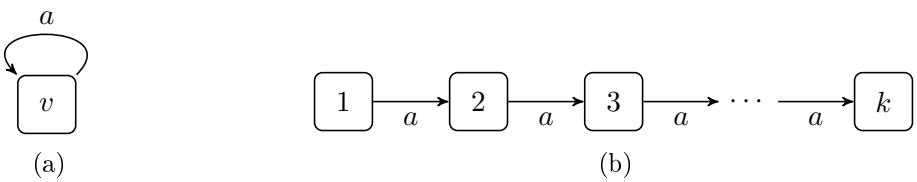


Figure 5.1: (a) An a-Loop Pattern and (b) A Finite a-Sequence

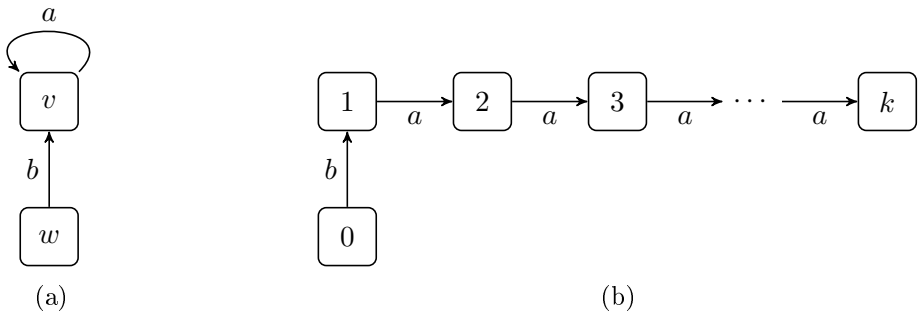


Figure 5.2: Figure 5.1 Prepended by b-labeled Edges

$S_0 = \{(v,1),(v,2),\ldots,(v,k)\}$ and $S = S_0$ is set in Line 1. G does not simulate Q. In every iteration, Algorithm 1 checks for edge $(v,a,v) \in E_Q$, whether there is some node $u \in V$ with $(v,u) \in S$ and $uE^a \cap vS = \emptyset$. In the first iteration, we remove pair (v,k) from S. Next, $(v,k-1)$ is removed, and so on until $S = \emptyset$. The final simulation is the empty simulation, verifying that G does not simulate Q. The number of iterations is k and, in fact, this is the minimal number of iterations every simulation algorithm must take for this particular example. ■

In every iteration, we check each edge $(v,a,w) \in E_Q$ and $u \in V$ with $(v,u) \in S$ (i. e., at most $|E_Q| \cdot |V|$ many) whether there is an edge $(u,a,u') \in E$ with $(w,u') \in S$. For one u, there are at most $|V|$ many u' with $(u,a,u') \in E$. Thus, there are at most $|E_Q| \cdot |V| \cdot |V|$ many checks of $(w,u') \in S$ to perform in a single iteration. In a naïve implementation, checking $uE^a \cap wS = \emptyset$ amounts to looking through all simulation candidates of w, which contributes time $\mathcal{O}(|V|)$ [69]. Algorithm 1 may thus be performed in $\mathcal{O}(|V_Q| \cdot |E_Q| \cdot |V|^4)$. In a less naïve implementation, we may get rid of a factor of $|V|$ by assuming sets uE^a and wS to be sorted. Thus, while checking for every $u' \in uE^a$, whether $u' \in wS$, the set wS is traversed at most once. This brings the algorithmic complexity of Algorithm 1 down to $\mathcal{O}(|V_Q| \cdot |E_Q| \cdot |V|^3)$.

As mentioned at the beginning of Chapter 4, Vardi calls the complexity we estimated above *combined complexity* [134] as both, pattern (or query) Q and data graph G, are part of the input. Regarding data complexity, graph pattern Q contributes only constant factors to the overall worst-case complexity. Algorithm 1 is, thus, in $\mathcal{O}(|V|^3)$ when regarding data complexity. Henzinger et al. add two algorithmic tweaks to the naïve approach, of which one of them directly tackles data complexity, promising an $\mathcal{O}(|V|^2)$ procedure. However, as we will argue, this optimization is infeasible in our setting. Thus, the data complexity of HHK remains in cubic-time.

5.1.2 The HHK Algorithm

The name of the algorithm, HHK, stems from the initials of the authors' last names, proposed as *EfficientSimilarity* by Henzinger, Henzinger, and Kopke [69]. The original algorithm contained a bug, pointed out and fixed by Ranzato [117]. Here, we refer to the fixed version of HHK.

As the name *EfficientSimilarity* suggests, HHK computes similarity classes. We adapt their algorithm to solve the simulation problem between labeled graphs Q and G with simulation candidate S_0. Any binary relation, $R \subseteq A \times B$, has a characteristic function $\chi_R : A \to \mathbf{2}^B$ with $\chi_R(a) := \{b \in B \mid (a,b) \in R\}$. For (dual) simulations $S \subseteq V_Q \times V$ between graph pattern Q and data graph G, χ_S associates with each node $v \in V_Q$ a set of (dual) simulating nodes $\chi_S(v) \subseteq V$. Updating $\chi_S(v)$ $(v \in V_Q)$ means updating S, e. g., $\chi_S(v) \setminus \{w\}$ translates to $S \setminus \{(v,w)\}$.

One problem of Algorithm 1, tackled by HHK, is that it always iterates over all edges of Q, no matter whether it is necessary or not.

Example 5.2 Consider the extension of Example 5.1 by a single b-labeled edge, as depicted in Figure 5.2. Q is the pattern depicted in Figure 5.2(a) and G the graph in Figure 5.2(b). Let us further assume $S_0 = \{(w,0),(v,1),(v,2),\ldots,(v,k)\}$, already an optimization upon $V_Q \times V$. As before, the first iteration removes (v,k), the second $(v,k-1)$, and so forth. After $k-1$ iterations, S contains $(w,0)$ and $(v,1)$. We already know that $(v,1)$ is removed, but only then also $(w,0)$ can be removed from S, leaving us with the empty simulation after at most $k+1$ iterations.

To complete a single iteration, Algorithm 1 checks for edges $(v,a,v) \in E_Q$ as well as $(w,b,v) \in E_Q$. The former edge leads to the necessary removals. However, consideration of the latter edge does not influence the computation at all, unless $(v,1)$ is removed. Hence, in up to k iterations, edge $(w,b,v) \in E_Q$ is traversed unnecessarily. ■

To overcome such unnecessary traversals, Henzinger et al. introduce so-called *remove sets*. In our setting, there is a remove set for every label $a \in \Sigma$. $\mathsf{Remove}_a(v)$ stores all nodes $u' \in V$ that cannot reach any simulating node of v, i. e., $\forall u' \in \mathsf{Remove}_a(v)$, there is no $v' \in \chi_S(v)$ with $u'\ E^a\ v'$. Thus, an edge $w\ E^a_Q\ v$ is only considered if $\mathsf{Remove}_a(v) \neq \emptyset$, i. e., if there is a potential to update the simulation candidates of w.

Example 5.3 Reconsider Q, G, and S_0 as in Example 5.2. The remove sets are initialized to $\mathsf{Remove}_a(v) = \{0,k\}$, $\mathsf{Remove}_b(v) = \{1,2,\ldots,k\}$, and $\mathsf{Remove}_a(w) = \mathsf{Remove}_b(w) = \emptyset$. The set $\mathsf{Remove}_a(v)$ collects all the nodes of G that cannot reach $\chi_{S_0}(v)$. In this particular case, it is the set of all nodes that do not have an outgoing a-labeled edge.

In the first iteration of HHK, $(w,b,v) \in E_Q$ is considered because $\mathsf{Remove}_b(v) \neq \emptyset$. By processing (w,b,v), we reduce $\chi_S(w)$ by all the nodes in $\mathsf{Remove}_b(v)$. In this case, $\chi_S(w)$ is not updated, but after processing $\mathsf{Remove}_b(v)$, there is no need for remembering the nodes we just removed. Thus, we set $\mathsf{Remove}_b(v)$ to $\emptyset$. Furthermore, we process $(v,a,v) \in E_Q$ which removes the pair (v,k) from S. HHK now recomputes $\mathsf{Remove}_a(v)$ to $\{k-1\}$. We explain the details of this update below.

It is important to notice that in the next iteration, $\mathsf{Remove}_a(v) = \{k-1\}$ and $\mathsf{Remove}_b(v) = \emptyset$. Thus, edge $(w,b,v) \in E_Q$ is not considered and will not be reconsidered until $(v,1)$ has been removed from S. ■

Correctly maintaining the remove sets is the key for improved combined complexity of HHK, depicted in Algorithm 2. The integration of remove sets reduces combined complexity of Algorithm 1 to $\mathcal{O}(|V_Q| \cdot |V|^3)$ [69]. Additionally to initializing the working relation S in Line 1, the remove sets are initialized in Lines 2 to 4 as explained in Example 5.3. The algorithm proceeds by picking nodes $v \in V_Q$ with non-empty remove sets, i. e., $\mathsf{Remove}_a(v) \neq \emptyset$ ($a \in \Sigma$), and considers every edge $u\ E^a_Q\ v$ for an update of $\chi_S(u)$. Recall that $\mathsf{Remove}_a(v)$ contains all nodes from the database that cannot reach a node by an a-labeled edge, simulating v. These nodes must be removed from $\chi_S(u)$ (Line 10). Upon removal of a node $w \in \mathsf{Remove}_a(v)$ from $\chi_S(u)$, w is not a candidate for simulating u anymore. This information must be propagated to all predecessors of u. Therefore, for every node w' that reaches w by some edge but no other node in $\chi_S(u)$, the respective remove set of u must include w' (cf. Line 13) as it represents a potential to update predecessor nodes of u.

Example 5.4 (Example 5.3 continued) Recall that, initially, $\mathsf{Remove}_a(v) = \{0,k\}$ and $\mathsf{Remove}_b(v) = \{1,2,\ldots,k\}$. Let us first process $\mathsf{Remove}_b(v)$. As there is only a single edge to consider, every element of $\mathsf{Remove}_b(v)$ is tested whether it is an element of $\chi_S(w)$ in Line 9. The tests will all be negative, $\mathsf{Remove}_b(v) = \emptyset$ afterwards, and we

Algorithm 2: The HHK Algorithm

Input : $Q = (V_Q, \Sigma, E_Q)$, $G = (V, \Sigma, E)$, and $S_0 \subseteq V_Q \times V$.
Output: Greatest simulation $S \subseteq S_0$ between Q and G.

```
1  S ← S_0;
2  forall v ∈ V_Q, a ∈ Σ do
3  |  Remove_a(v) ← V \ ⋃_{w∈χ_S(v)} E^a w;
4  end
5  while there are v ∈ V_Q and a ∈ Σ with Remove_a(v) ≠ ∅ do
6  |  Remove ← Remove_a(v);
7  |  Remove_a(v) ← ∅;
8  |  forall w ∈ Remove and u ∈ E_Q^a v do
9  |  |  if w ∈ χ_S(u) then
10 |  |  |  χ_S(u) ← χ_S(u) \ {w};
11 |  |  |  forall b ∈ Σ and w' ∈ E^b w do
12 |  |  |  |  if w'E^b ∩ χ_S(u) = ∅ then
13 |  |  |  |  |  Remove_b(u) ← Remove_b(u) ∪ {w'};
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
17 |  end
18 end
```

proceed with $\mathsf{Remove}_a(v)$ and edge $(v, a, v) \in E_Q$. Since $0 \notin \chi_S(v)$ but $k \in \chi_S(v)$, k is removed from $\chi_S(v)$ (in Line 10). Now every predecessor of k in G, here only $k-1$, is checked whether there is some a-labeled edge from $k-1$ to some node in $\chi_S(v)$ (which has just been updated). If not, $k-1$ is added to $\mathsf{Remove}_a(v)$. Hence, after this iteration, $\mathsf{Remove}_a(v) = \{k-1\}$ and all other remove sets are empty. Next, $k-1$ will be removed from $\chi_S(v)$ and $\mathsf{Remove}_a(v)$ is updated to $\{k-2\}$.

This procedure repeats until node 1 is removed from $\chi_S(v)$, i. e., eventually $\chi_S(v) = \emptyset$. Node 1 has a b-predecessor, namely node 0, and it cannot reach any node in $\chi_S(v)$ after 1 has been removed. Thus, $\mathsf{Remove}_b(v)$ must be updated to $\{0\}$. Next, this remove set together with the edge $(w, b, v) \in E_Q$ is considered and $\chi_S(w)$ is updated to $\emptyset$.

The resulting relation is $S = \emptyset$, verifying that G does not simulate Q. ■

Note that Lines 6 and 7 pick a remove set and mark it as processed. The reason for storing the current remove set in a local variable is that in a self-loop, $u = v$ and the nodes that were just removed are not to be mixed up with the nodes that must be removed after the iteration. Example 5.4 exemplifies this algorithmic behavior for node v and its remove set $\mathsf{Remove}_a(v)$.

The first key to the complexity of HHK is the observation, that if we pick $\mathsf{Remove}_a(v)$ in iteration i, then in any later iteration, $\mathsf{Remove}_a(v)$ is disjoint from its version in i [69]. Thus, for every node $v \in V_Q$ we have to consider at most $|V|$ many disjoint sets of $\mathsf{Remove}_a(v)$, that are $|\Sigma(Q)| \cdot |V|$ many in total[2]. Furthermore, each combination $w \in \mathsf{Remove}$ and $u\ E_Q^a\ v$ occurs at most once and the test $w \in \chi_S(u)$ evaluates to true at most once because w is removed from $\chi_S(u)$ (Line 10) after Line 9 has been passed. There are at most $|V|$ many a-predecessors of w (Line 11). Thus, the inner for-loop amounts to at most $|\Sigma(Q)| \cdot |V|$ iterations. In each of these iterations, the test in Line 12 is performed in $\mathcal{O}(|V|)$, once again assuming that $w'E^a$ and $\chi_S(u)$ are stored in sorted order.

[2] $\Sigma(Q) = \{a \mid (v, a, w) \in E_Q\}$

Thus, the overall combined complexity of HHK (Algorithm 2) is $\mathcal{O}(|V_Q| \cdot |\Sigma(Q)|^2 \cdot |V|^3)$. Compared to the result of Henzinger et al. [69], we get the factor of $|\Sigma(Q)|^2$ as our setting incorporates edge labels. Translated back to their setting, i. e., no edge labels and $Q = G$, we get a final complexity of $\mathcal{O}(|V|^4)$ or $\mathcal{O}(|E| \cdot |V|^2)$. Henzinger et al., however, conclude their algorithm to be in $\mathcal{O}(|E| \cdot |V|)$ because they make use of an additional data structure to perform Line 12 in $\mathcal{O}(1)$.

Consider a single label $a \in \Sigma$. Then $\mathsf{count}_a : V \times V_Q \to \mathbb{N}$ is a two-dimensional array of positive integers maintained to preserve (5.1).

$$\mathsf{count}_a(w', u) = |w'E^a \cap \chi_S(u)| \tag{5.1}$$

Upon removing w from $\chi_S(u)$, $\mathsf{count}_a(w', u)$ is decremented by 1 for every a-predecessor w' of w. The test in Line 12 is then reduced to checking whether $\mathsf{count}_a(w', u) = 0$.

On one hand, count_a is beneficial as it makes the factor of Line 12 constant. However, employing these matrices is quite memory-consuming as their entries are positive integer values. For every node $v \in V_Q$ and every label $a \in \Sigma(Q)$, $\mathsf{count}_a(v)$ stores $\mathcal{O}(|V|)$ many integer values to enable for the constant-time check in Line 12. Observing the current trend of the node set sizes in knowledge graphs, like Wikidata [87], lets the count structures appear infeasible. Furthermore, creating this auxiliary structure is quite time-consuming. They regain their benefits if query and data graph have similar extents, e. g., as for the original setting of similarity checking: comparing two (enormous) state spaces with each other. Hence, we conclude that HHK delivers a solution to the simulation problem in $\mathcal{O}(|V|^2)$ with count-arrays and $\mathcal{O}(|V|^3)$ without count data complexity.

5.1.3 On Space-Efficient Algorithms

HHK is the best known algorithm for computing simulations regarding time complexity. It has been designed for computing the maximal simulation relation within a single graph. Therefore, $S_0 = V \times V$ is constructed before Algorithm 2 is performed. Hence, HHK needs $\mathcal{O}(|V|^2)$ space, which is considered inefficient in environments where the main memory constitutes the principal bottleneck [132]. The algorithms, presented in [60, 117], fall in the class of space-efficient simulation algorithms. Their primary goal is to compute *simulation equivalence classes*, an important minimization technique for checking similarity between massive graphs. The simulation preorder is only a byproduct of their procedures. In our setting, since simulations are kept reasonably small by the size of the pattern graph Q, we will not cover these algorithms in-depth, but only note that their basic principle is founded on *partition refinement* [105].

Given a graph $G = (V, \Sigma, E)$ and let us assume the maximal simulation between G and G is $\widehat{R} \subseteq V \times V$. A *similarity block* is a subset of nodes, $B \subseteq V$, such that for every two nodes $v_i, v_j \in B$, $(v_i, v_j) \in \widehat{R}$ and $(v_j, v_i) \in \widehat{R}$, i. e., v_j simulates v_i and vice versa[3]. Thus, computing the coarsest partition of nodes, $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$, where every B_i $(i = 1, 2, \ldots, k)$ is a similarity block, is one of the algorithmic tasks followed by [60, 132, 117]. Computing simulations between blocks, instead of individual nodes, i. e., $R \subseteq \mathcal{B} \times \mathcal{B}$, reduces the overall memory-consumption of these algorithms. The computation time is not reduced but split up into the computation of the coarsest similarity partition and the simulation between the blocks. These algorithms require $\mathcal{O}(|\mathcal{B}|^2)$ space, which, in the worst case, is the same as using $|V|^2$ memory. The worst case here occurs if every data node describes a similarity block by its own, i. e., if $\mathcal{B} = \{\{v\} \mid v \in V\}$.

[3] $\widehat{R}$ is only used to define the notion of similarity block, but is not given as an input to the algorithms.

5.2 A System of Inequalities Approach

In this section, we develop a new solution to DualSim, based on a reformulation of the problem as a system of inequalities. Although the worst-case complexity of our algorithm remains unaltered, as compared to the others (cf. Section 5.2.4), we gain a degree of freedom allowing for a systematic reduction of iterations to eventually reach the final dual simulation. As we show in Section 5.2.6, the new procedure exhibits low computation times, a solid basis for query (pre-)processing. We engineer our solution in three steps. First, we define a set of inequalities, whose solutions are equivalent to the coinductive definition of dual simulation in Definition 2.23. We further show how to derive a fast implementation based on bit-vectors and bit-matrices in Section 5.2.3. Last, we provide a discussion on optimizations (Section 5.2.5) realized in our software prototype, called sparqlSim[4]. Before, we mention some assumptions regarding the data structures we use.

5.2.1 Preliminary Considerations

While the algorithms, presented in Section 5.1, are more or less independently formulated from the data structures in use, we base our solution on specific representations of the input parameters, especially that of the data graph G and that of the dual simulation (candidate) S. A concrete representation of the pattern graph is developed in Sections 5.2.2 and 5.2.3. This representation is, by no means, algorithm-guided but instead guided by the extents we get as input data.

First, recall from Section 5.1.2 that (dual) simulations $S \subseteq V_Q \times V$ are representable by functions $\chi_S : V_Q \to \mathbf{2}^V$. From an implementational point of view, $\chi_S(v)$ is a $|V|$-dimensional *row bit-vector*. To obtain a vector representation, formally, we assume a fixed total order $<_V$ on the elements of V to induce an index over $\mathbb{N}$. Node $v \in V$ has index $k \in \mathbb{N}$ if there are $k-1$ distinct nodes $v_1, v_2, \ldots, v_{k-1} \in V$, such that $v = \min_{<_V}(V \setminus \{v_1, v_2, \ldots, v_{k-1}\})$. To access the node of G with index k, we use the notation $V(k)$. If $|V| = n$, then its node set may, thus, be represented by $\{V(1), V(2), \ldots, V(n)\}$ or $\{v_1, v_2, \ldots, v_n\}$. If V_Q is indexed by the same method as described for V, χ_S may be represented as a $|V_Q| \times |V|$ bit-matrix.

In the course of our computations, we will check whether for some pair of nodes $v \in V_Q$ and $w \in V$, $w \in \chi_S(v)$. As V's elements are indexed, there is an index j for w, i. e., $V(j) = w$. Hence, checking $w \in \chi_S(v)$ boils down to testing whether the jth bit is set in the bit-vector representation of $\chi_S(v)$. Although explicit storage of the vectors promises constant-time access to its elements, the size of V determines the actual size of the bit-vectors. Employing *gap encodings*[5] drastically decreases memory consumption, while leaving the overall access time logarithmic in $|V|$, which is a feasible trade-off. Jumping from one entry to the next may be performed in constant time because, due to gap encodings, the next entry's index is accessible by a constant overhead from the current position in memory. Once the first entry in $\chi_S(v)$ has been found, iterating through all the elements of the set is in $\mathcal{O}(|\chi_S(v)|)$.

A labeled graph $G = (V, \Sigma, E)$ is completely characterized by its set of nodes V and a set of adjacency (forward) maps $\{\mathfrak{F}^a_G : V \to \mathbf{2}^V \mid a \in \Sigma\}$ with $\mathfrak{F}^a_G(v) = vE^a$. For a single node, forward maps associate a set of successors w. r. t. the edges with the associated label. Using backward maps $\mathfrak{B}^a_G$ ($a \in \Sigma$), with $\mathfrak{B}^a_G(v) = E^a v$, instead of forward maps yields an equivalent characterization. Backward maps consider sets of all predecessors. Let $\mathfrak{A}$ be an adjacency map of G, no matter whether it is a forward or a backward map. For $v \in V$, $\mathfrak{A}(v)$ denotes the row of v in $\mathfrak{A}$. Using this intuition jointly with an index on V, we represent $\mathfrak{A}$ as a $|V| \times |V|$ bit-matrix. Recall that the elements of V are indexed

[4] available at GitHub `https://github.com/ifis-tu-bs/sparqlSim`

[5] The library we use is called BitMagic `http://bitmagic.io/index.html`.

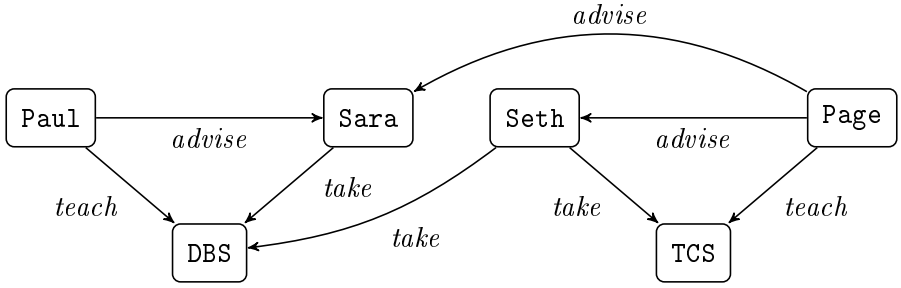


Figure 5.3: A Graph Structure

over $\{1, 2, \ldots, |V|\}$. An entry $\mathfrak{A}(i,j) = 1$ means that nodes $v_i = V(i)$ and $v_j = V(j)$ $(0 < i, j \leq |V|)$ are adjacent by the edge expressed in $\mathfrak{A}$. If $v_i E^a v_j$, then $\mathfrak{F}^a_G(i,j) = 1$ and $\mathfrak{B}^a_G(j,i) = 1$. Consequently, forward and backward matrices of the same label are related by transposition, i. e., $\mathfrak{F}^a_G = {\mathfrak{B}^a_G}^\mathsf{T}$ and $\mathfrak{B}^a_G = {\mathfrak{F}^a_G}^\mathsf{T}$, being the reason why either the forward maps or the backward maps characterize G's edge relation.

Example 5.5 Consider the example graph in Figure 5.3, which is a slightly altered version of the one we used throughout Section 4.4. For label *take*, this graph provides the two

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

(a) $\mathfrak{F}^{take}_{5.3}$ (b) $\mathfrak{B}^{take}_{5.3}$

Figure 5.4: Two Bit-Matrix Representations of Adjacency Maps

adjacency matrices in Figure 5.4, where we assume the set of nodes to be lexicographically ordered by their names, i. e., $v_1 = \texttt{DBS}$, $v_2 = \texttt{Page}$, $v_3 = \texttt{Paul}$, $v_4 = \texttt{Sara}$, $v_5 = \texttt{Seth}$, and $v_6 = \texttt{TCS}$. ■

In our implementation, we use an extended CRS (*Compressed Row Storage*) format [129, 27]. In the CRS format, we store matrices row-wise, and for each row, only non-zero values are considered. As graph matrices are usually sparse, it is beneficial to separate non-edges from edges, also called *implicit* and *explicit values* [39]. Our CRS version relies on the assumption that not only a graph's adjacency matrix is sparse, but also the majority of rows in a matrix $\mathfrak{A}$ contains no bits at all. We are dealing with labeled graphs $G = (V, \Sigma, E)$, entailing that the overall edge relation E is split up into $|\Sigma|$ adjacency maps.

Let $G = (V, \Sigma, E)$ with $|V| = n$ $(n > 0)$ and $\mathfrak{A}$ some adjacency matrix of G with m non-zero entries, i. e., $|\{(i,j) \mid \mathfrak{A}(i,j) = 1\}| = m$, and l rows with at least one non-zero entry, i. e., $|\{i \mid \exists j : \mathfrak{A}(i,j) = 1\}| = l$. We represent $\mathfrak{A}$ by three integer vectors, $\mathfrak{r}$, $\mathfrak{i}$, and $\mathfrak{c}$. The *row vector* $\mathfrak{r}$ is a list of row indices with at least one non-zero value, i. e., if row 42 contains some bit, then there is some k $(1 \leq k \leq l)$ with $\mathfrak{r}(k) = 42$. As there are n nodes and l non-zero rows in $\mathfrak{A}$, $\mathfrak{r} \in \{1, 2, \ldots, n\}^l$. The *interval vector* $\mathfrak{i}$ is an $(l+1)$-dimensional integer vector storing pointers to the positions in the *column index vector* $\mathfrak{c}$, which itself is an m-dimensional integer vector. If $\mathfrak{A}(42, 73) = 1$, then there is a k (as before) with $\mathfrak{r}(k) = 42$ and between positions $b_1 = \mathfrak{i}(k)$ and $b_2 = \mathfrak{i}(k+1)$ in vector $\mathfrak{c}$, value 73 will be found. Formally,

$$\mathfrak{A}(i,j) = 1 \Leftrightarrow \exists k \leq l : \mathfrak{r}(k) = i \wedge \exists v : \mathfrak{i}(k) \leq v < \mathfrak{i}(k+1) \wedge \mathfrak{c}(v) = j. \tag{5.2}$$

Note that variables k and v are positive integers in (5.2). Furthermore, the last position in $\mathfrak{i}$ points to the size of $\mathfrak{c}$, i. e., $\mathfrak{i}(l+1) = m$. Compared to the original CRS format, we added the row vector $\mathfrak{r}$ to account for compression of rows in $\mathfrak{A}$ with no bits set.

In our implementation, we guarantee the vector $\mathfrak{r}$ to be in sorted order, i. e., if $\mathfrak{r}(k) = i$, then for all $k' < k$, $\mathfrak{r}(k') < i$. Vector $\mathfrak{c}$ is *interval-sorted*, i. e., the values between $\mathfrak{i}(k)$ and $\mathfrak{i}(k+1)$ in $\mathfrak{c}$ are sorted. Thereby, we guarantee access to $\mathfrak{A}(i,j)$ in $\log_2(l)$, assuming that $l \gg \mathfrak{i}(k+1) - \mathfrak{i}(k)$ for $0 < k \leq l$.

Example 5.6 (Example 5.5 continued) Let us work out the $(\mathfrak{r}, \mathfrak{i}, \mathfrak{c})$ representation of $\mathfrak{F}^{take}_{5.3}$. There are two non-zero rows, namely 4 and 5. Thus,

$$\mathfrak{r} = (4, 5).$$

Row 4 contains one bit in column 1 and row 5 two bits, one in column 1 and one in 6, i. e.,

$$\mathfrak{c} = (1, 1, 6)$$

with interval vector

$$\mathfrak{i} = (1, 2, 4).$$

Note that the difference between consecutive components in $\mathfrak{i}$ describes the number of bits in the respective row. ■

Our algorithm for computing dual simulations will often perform a *bit-row vector-matrix multiplication*,

$$\mathfrak{v} \cdot \mathfrak{A} := \bigvee_{i:\mathfrak{v}(i)=1} \mathfrak{A}(i), \tag{5.3}$$

where $\mathfrak{v}$ is an n-dimensional bit-vector and $\mathfrak{A}$ an $n \times n$ bit-matrix. In other words, $\mathfrak{v} \cdot \mathfrak{A} = \mathfrak{w}$ where $\mathfrak{w}(j) = 1$ iff there is an i such that $\mathfrak{v}(i) = 1$ and $\mathfrak{A}(i,j) = 1$. In our setting, $\mathfrak{v}$ represents a set of nodes $V_\mathfrak{v} = \{v_i \in V \mid \mathfrak{v}(i) = 1\} \subseteq V$ of a graph $G = (V, \Sigma, E)$ and $\mathfrak{A}$ an adjacency matrix of G. Then $\mathfrak{v} \cdot \mathfrak{A}$ describes the subset of nodes reachable from $V_\mathfrak{v}$ by an edge described by $\mathfrak{A}$.

Example 5.7 Reconsider our example graph in Figure 5.3 and its adjacency matrices as exemplified in Figure 5.4. If we want to know which of the nodes are reachable by an edge traversal using *take*-forward edges and *teach*-backward edges, we begin with the bit-vector representation of V (cf. Example 5.5), i. e., $\mathfrak{v} = (1, 1, 1, 1, 1, 1)$. Then $\mathfrak{v} \cdot \mathfrak{F}^{take}_{5.3}$ describes the set of nodes reachable by a *take*-labeled edge, i. e., $\mathfrak{v} \cdot \mathfrak{F}^{take}_{5.3} = \mathfrak{w} = (1, 0, 0, 0, 0, 1)$. By using $\mathfrak{B}^{teach}_{5.3}$, we get to the desired result, $\mathfrak{w} \cdot \mathfrak{B}^{teach}_{5.3} = (0, 1, 1, 0, 0, 0)$. ■

Combined with the $(\mathfrak{r}, \mathfrak{i}, \mathfrak{c})$ representation of matrices, computing $\mathfrak{v} \cdot \mathfrak{A}$ is in $\mathcal{O}(n \cdot \log_2(l) \cdot \max\{\mathfrak{i}(k+1) - \mathfrak{i}(k) \mid 0 \leq k < l\})$. The last factor represents the maximal number of bits set in a row of $\mathfrak{A}$. The estimation is already aware of the sparse matrices setting and may even be reduced to $\mathcal{O}(n)$ because in graph databases, as we use them in our experiments, $l \ll n$ and $\mathfrak{i}(k+1) - \mathfrak{i}(k) \ll l$ (cf. Appendix A.2).

5.2.2 Characterizing Dual Simulations

Let $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$ be labeled graphs, and $S \subseteq V_Q \times V$ a dual simulation between Q and G. Consider an edge $(v, a, w) \in E_Q$ and a node $v' \in \chi_S(v)$. Since S is a dual simulation, we derive for χ_S from Definition 2.23 that

$$\exists w' \in V : (v', a, w') \in E \text{ and } w' \in \chi_S(w). \tag{5.4}$$

The problem with (5.4) is that there may be many w' qualifying for $(v', a, w') \in E$ but $w' \notin \chi_S(w)$. We pursue having a single operation which allows us to quickly verify the

existence of w' with the required property. Therefore, we exploit the forward adjacency map $\mathfrak{F}^a_G$ ($a \in \Sigma$) and prove the existence of a w' in (5.4) by intersecting the row of v' in $\mathfrak{F}^a_G$ and the nodes simulating w, i. e.,

$$\mathfrak{F}^a_{G_2}(v') \cap \chi_S(w) \neq \emptyset. \tag{5.5}$$

(5.5) is equivalent to (5.4). Unfortunately, (5.5) still only checks for one pair of nodes $(v, v') \in S$. Combining this equation for all $v' \in \chi_S(v)$ yields

$$\bigwedge_{v' \in \chi_S(v)} \quad \mathfrak{F}^a_{G_2}(v') \cap \chi_S(w) \neq \emptyset. \tag{5.6}$$

The same arguments apply to the second requirement of Definition 2.23, using the backward map in

$$\bigwedge_{w' \in \chi_S(w)} \quad \mathfrak{B}^a_{G_2}(w') \cap \chi_S(v) \neq \emptyset. \tag{5.7}$$

The combination of both, (5.6) and (5.7), yields two inequalities equivalent to Definition 2.23 and is the key aspect contributing to the efficient implementation we are going to obtain.

Lemma 5.8 *Let $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$ be labeled graphs with $(v, a, w) \in E_Q$. For a binary relation $S \subseteq V_Q \times V$, satisfying (5.6) and (5.7),*

$$\begin{array}{lrcll} (i) & \chi_S(w) & \subseteq & \bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_{G_2}(v') & \text{and} \\ (ii) & \chi_S(v) & \subseteq & \bigcup_{w' \in \chi_S(w)} \mathfrak{B}^a_{G_2}(w') & \end{array} \tag{5.8}$$

are satisfied.

Proof: Towards a contradiction of (i), assume $\chi_S(w) \not\subseteq \bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_G(v')$. Hence, there is a $w' \in \chi_S(w)$ such that for each $v' \in \chi_S(v)$, $w' \notin \mathfrak{F}^a_G(v')$, i. e., $(v', a, w') \notin E$. As a consequence, $\chi_S(v)$ and $\mathfrak{B}^a_G(w')$ are disjoint for each $v' \in \chi_S(v)$, contradicting our assumption that (5.7) holds. Therefore, such a w' cannot exist, allowing to conclude that $\chi_S(w) \subseteq \bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_G(v')$. Inequality (ii) is completely analogous using (5.6). Q. E. D.

Phrased differently, dual simulations S satisfy (5.8) for every edge (v, a, w) of the pattern graph Q. Conversely, every solution to (5.8) (for all pattern edges) is a dual simulation.

Theorem 5.9 *Let $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$ be labeled graphs. $S \subseteq V_Q \times V$ is a dual simulation between Q and G iff for every edge $(v, a, w) \in E_Q$, S satisfies (5.8).*

Proof: Showing the implication, i. e., a dual simulation S between Q and G satisfies (5.8), is analogous to the proof of Lemma 5.8. Assume that one of the inequalities, say (ii), is not satisfied for edge $(v, a, w) \in E_Q$. Then there is a $v' \in \chi_S(v)$ for which no $w' \in \chi_S(w)$ exists with $v' \in \mathfrak{B}^a_G(w')$. Hence, $(v, v') \in S$ with no $w' \in V$, such that $(v', a, w') \in E$ and $(w, w') \in S$. But this contradicts the assumption that S is a dual simulation between Q and G. Case (i), analogously.

Conversely, assume we have $S \subseteq V_Q \times V$ such that (5.8) holds for every $(v, a, w) \in E_Q$. We prove that S is a dual simulation. Let $(v, v') \in S$, i. e., $v' \in \chi_S(v)$, and $(v, a, w) \in E_Q$. We need to show that there is a $w' \in V$ such that $(v', a, w') \in E$ and $(w, w') \in S$. From (5.8)(ii) we get that for some $w' \in \chi_S(w)$ we have that $v' \in \mathfrak{B}^a_G(w')$. This w' completes the proof because $v' \in \mathfrak{B}^a_G(w')$ implies $(v', a, w') \in E$ and from $w' \in \chi_S(w)$ we get that $(w, w') \in S$. Case $(u, a, v) \in E_Q$ is completely analogous. Q. E. D.

Hence, (5.8) for all edges of the pattern graph characterizes dual simulations. We can use it as a basic fixpoint solution to DualSim as follows. Let S_i ($i \geq 0$) be the current

simulation candidate. For each edge of Q, check whether (5.8) is satisfied by S_i. If (5.8)(i) fails for edge $e = (v, a, w)$, then update S_i to S_{i+1} by computing

$$\chi_{S_{i+1}}(u) := \begin{cases} \chi_{S_i}(u) \cap \bigcup_{v' \in \chi_{S_i}(v)} \mathfrak{F}^a_G(v') & \text{if } u = w \\ \chi_{S_i}(u) & \text{otherwise.} \end{cases}$$

If (5.8)(ii) fails for e, compute S_{i+1} by

$$\chi_{S_{i+1}}(u) := \begin{cases} \chi_{S_i}(u) \cap \bigcup_{w' \in \chi_{S_i}(w)} \mathfrak{B}^a_G(w') & \text{if } u = v \\ \chi_{S_i}(u) & \text{otherwise.} \end{cases}$$

A single edge traversal removes all non-simulating nodes of w (v, resp.) relative to S_i. This procedure is repeated until we reach an S_k satisfying (5.8) for every edge of Q. As we will later show, S_k is the greatest dual simulation included in S_0.

Even though we maintain the PTIME nature of the other algorithms in Section 5.1, we still miss a way to quickly compute the union $\bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_G(v')$ and access $\chi_S(v)$ ($\bigcup_{w' \in \chi_S(w)} \mathfrak{B}^a_G(w')$ and $\chi_S(w)$, resp.). Therefore, the forthcoming implementation works with bit-representations of $\chi_S(v)$ and $\mathfrak{F}^a_G$ ($\mathfrak{B}^a_G$, resp.), paving the way for optimization in time and space consumption. In that setting, we derive a *system of inequalities* (SOI) from Theorem 5.9, for which dual simulations are the solutions.

5.2.3 Implementing Inequalities

We compute DUALSIM by a *system of inequalities* according to (5.8) from the last section. The challenge is to find a way to compute the unions quickly

$$\bigcup_{v' \in \chi_S(v)} \mathfrak{F}^a_{G_2}(v') \quad \text{and} \quad \bigcup_{w' \in \chi_S(w)} \mathfrak{B}^a_{G_2}(w'). \tag{5.9}$$

Combinations of vectors and matrices, especially when encoding information only bit-wise, promise fast computations (cf. Section 5.2.1). Hence, we use the adjacency maps of G as adjacency bit-matrices. Computing the bit-representation of the set of all successors (predecessors, resp.) of a set of nodes, which is also represented by a bit-vector, amounts to computing the vector-matrix product. Thus, we implement (5.9) by

$$\chi_S(v) \cdot \mathfrak{F}^a_{G_2} \quad \text{and} \quad \chi_S(w) \cdot \mathfrak{B}^a_{G_2}. \tag{5.10}$$

The results of the multiplications are used to check whether (5.8) (cf. Lemma 5.8) holds.

Example 5.10 Let us reconsider our example graph of Figure 5.3 and its adjacency matrices. Assume, we want to find all the nodes v and w connected by an *advise*-labeled edge, i. e., we check for dual simulations between the pattern $Q = (\{v, w\}, \Sigma, \{(v, \mathit{advise}, w)\})$ and the graph in Figure 5.3. According to Theorem 5.9, we need to check (5.8).

Consider first $S = V_Q \times V$, i. e., $\chi_S(v) = \chi_S(w) = (1, 1, 1, 1, 1, 1)$. Then

$$\chi_S(v) \cdot \mathfrak{F}^{\mathit{advise}}_{5.3} = (0, 0, 0, 1, 1, 0) = \mathfrak{w}. \tag{5.11}$$

Of course, $\chi_S(w) = (1, 1, 1, 1, 1, 1) \not\leq \mathfrak{w}$ which tells us S is not a dual simulation. If we begin with S' such that $\chi_{S'}(v) = (1, 1, 1, 1, 1, 1)$ and $\chi_{S'}(w) = \mathfrak{w}$, we obtain the desired property that $\chi_{S'}(v) \cdot \mathfrak{F}^{\mathit{advise}}_{5.3} = \mathfrak{w}$, i. e., $\chi_{S'}(w) \subseteq \bigcup_{v' \in \chi_{S'}(v)} \mathfrak{F}^{\mathit{advise}}_{5.3}(v')$. However, S' is not a dual simulation because $\chi_{S'}(w) \cdot \mathfrak{B}^{\mathit{advise}}_{5.3} = (0, 1, 1, 0, 0, 0)$ but $\chi_{S'}(v) = (1, 1, 1, 1, 1, 1)$ is a superset of the computed product. ∎

As $\chi_S(w)$ ($\chi_S(v)$, resp.) are also represented as bit-vectors, the inequalities in (5.8) may be reformulated as

$$\begin{array}{rcll} \chi_S(w) & \leq & \chi_S(v) \cdot \mathfrak{F}_G^a & \text{and} \\ \chi_S(v) & \leq & \chi_S(w) \cdot \mathfrak{B}_G^a & \end{array} \tag{5.12}$$

for each edge $(v, a, w) \in E_Q$. Comparing two n-dimensional row bit-vectors $\mathfrak{v}$ and $\mathfrak{w}$ by $\leq$ is interpreted as the component-wise comparison of $\mathfrak{v}$ and $\mathfrak{w}$ by $\leq$, i. e., $\mathfrak{v} \leq \mathfrak{w}$ iff $\mathfrak{v}(i) \leq \mathfrak{w}(i)$ for all $i \in \{1, 2, \ldots, n\}$. $\mathfrak{v}, \mathfrak{w}$ represent sets and $\mathfrak{v} \leq \mathfrak{w}$ indeed implements set inclusion. Hence, (5.12) are exact bit-vector/matrix representations of (5.8), which enables us to give a representation of DualSim as a *system of inequalities*, to be developed throughout the rest of this section.

A system of inequalities has a set of variables Var and a set of inequalities NEq, each of which is finite. We restrict the possible inequalities to the ones we will need for computing dual simulations between pattern graphs Q and data graphs G. Additionally, we include inequality types that are going to be used to encode SPARQL queries $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ in Section 5.3.

Definition 5.11 (System of Inequalities)
Let $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$ be labeled graphs. A pair $\mathcal{E} = (\mathsf{Var}, \mathsf{NEq})$ is a *system of inequalities (SOI) over Q and G* iff its *set of variables* $\mathsf{Var} \supseteq V_Q$ and its *set of inequalities* NEq contains inequalities of the shape

$$(1)\ \ w \leq v \cdot \mathfrak{A},\ (2)\ \ w \leq v, \text{ or } (3)\ \ w \leq \mathfrak{v},$$

where $w, v \in \mathsf{Var}$, $\mathfrak{A}$ is an adjacency matrix of G, and $\mathfrak{v}$ is a $|V|$-dimensional bit-vector. ▲

Inequalities range over by $\phi, \epsilon, \epsilon_1, \epsilon_2, \ldots$. The encoding of (5.12) will only use inequalities of shape (1). Shape (2) will be essential when we integrate optional patterns of SPARQL queries into the systems of inequalities in Section 5.3. (3) is useful, e. g., for optimizations to be discussed in Section 5.2.5.

Let $\mathcal{E} = (\mathsf{Var}, \mathsf{NEq})$ be a SOI over graph pattern $Q = (V_Q, \Sigma, E_Q)$ and data graph $G = (V, \Sigma, E)$. Assignments to the variables of $\mathcal{E}$ are relations $S \subseteq \mathsf{Var} \times V$. As $\mathsf{Var} \supseteq V_Q$ by Definition 5.11, assignments S may be dual simulations or candidates thereof. S is a *solution of $\mathcal{E}$* if it is *valid for* all inequalities NEq, i. e., when substituting every variable $v \in \mathsf{Var}$ by $\chi_S(v)$, the resulting inequalities shall hold.

Definition 5.12 (Valid Assignment, Solution)
Let $Q = (V_Q, \Sigma, E_Q)$ and $G = (V, \Sigma, E)$ be labeled graphs, and $\mathcal{E} = (\mathsf{Var}, \mathsf{NEq})$ an SOI over Q and G. A relation $S \subseteq \mathsf{Var} \times V$ is called an *assignment for $\mathcal{E}$*. S is a *valid assignment for inequality* $\epsilon \in \mathsf{NEq}$ iff

(1) $\epsilon = w \leq v \cdot \mathfrak{A}$ implies $\chi_S(w) \leq \chi_S(v) \cdot \mathfrak{A}$,

(2) $\epsilon = w \leq v$ implies that $\chi_S(w) \leq \chi_S(v)$, and

(3) $\epsilon = w \leq \mathfrak{v}$ implies that $\chi_S(w) \leq \mathfrak{v}$.

Otherwise, S is *invalid for* ϵ. Valid assignments for all inequalities $\epsilon \in \mathsf{NEq}$ are called *solutions of $\mathcal{E}$*. ▲

Example 5.13 (Example 5.10 continued) S is invalid for $w \leq v \cdot \mathfrak{F}_{5.3}^{advise}$ because

$$\chi_S(w) = (1,1,1,1,1,1) \not\leq (0,0,0,1,1,0) = \chi_S(v) \cdot \mathfrak{F}_{5.3}^{advise}.$$

S', on the other hand, is a valid assignment for $w \leq v \cdot \mathfrak{F}_{5.3}^{advise}$. Consider now S'' with $\chi_{S''}(v) = (0,1,0,0,0,0)$ and $\chi_{S''}(w) = \chi_{S'}(w)$. S'' is valid for $v \leq w \cdot \mathfrak{B}_{5.3}^{advise}$ because

$$\chi_{S''}(v) = (0,1,0,0,0,0) \leq (0,1,1,0,0,0) = \chi_{S''}(w) \cdot \mathfrak{B}_{5.3}^{advise},$$

which indicates that S'' is in fact a dual simulation between Q (Example 5.10) and the graph in Figure 5.3. ■

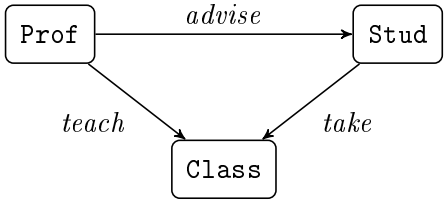


Figure 5.5: A Pattern Graph

Based on Equation (5.12), we define a SOI over Q and G for which every solution is a dual simulation between Q and G, and vice versa.

Definition 5.14 (Dual Simulation SOI)
Let $Q = (V_Q, \Sigma, E_Q)$ and G be labeled graphs. $\mathcal{E}(Q,G) = (\mathsf{Var}, \mathsf{NEq})$ is the *dual simulation SOI of Q and G* if $\mathsf{Var} = V_Q$ and NEq contains for each pattern edge $(v,a,w) \in E_Q$ the following equations:

$$w \leq v \cdot \mathfrak{F}_G^a \quad \text{and} \quad v \leq w \cdot \mathfrak{B}_G^a. \tag{5.13}$$

There are no other inequalities in NEq. ▲

Example 5.15 Subsequently, we give all six inequalities of the dual simulation SOI of pattern Q, depicted in Figure 5.5, and data graph G in Figure 5.3.

$$\begin{array}{llllllll} \epsilon_1: & \texttt{Stud} & \leq & \texttt{Prof} \cdot \mathfrak{F}_{5.3}^{advise} & \epsilon_2: & \texttt{Prof} & \leq & \texttt{Stud} \cdot \mathfrak{B}_{5.3}^{advise} \\ \epsilon_3: & \texttt{Class} & \leq & \texttt{Prof} \cdot \mathfrak{F}_{5.3}^{teach} & \epsilon_4: & \texttt{Prof} & \leq & \texttt{Class} \cdot \mathfrak{B}_{5.3}^{teach} \\ \epsilon_5: & \texttt{Class} & \leq & \texttt{Stud} \cdot \mathfrak{F}_{5.3}^{take} & \epsilon_6: & \texttt{Stud} & \leq & \texttt{Class} \cdot \mathfrak{B}_{5.3}^{take} \end{array} \tag{5.14}$$

■

The following result is a direct consequence of Definition 5.14 and Theorem 5.9.

Proposition 5.16 *Let Q and G be graphs and $\mathcal{E}(Q,G)$ the dual simulation SOI of Q and G. S is a solution of $\mathcal{E}(Q,G)$ iff it is a dual simulation between Q and G.*

Proof: Let S be a solution of $\mathcal{E}(Q,G) = (\mathsf{Var}, \mathsf{NEq})$. Then S is valid for $w \leq v \cdot \mathfrak{F}_G^a$ and $v \leq w \cdot \mathfrak{B}_G^a$ for all $(v,a,w) \in E_Q$. Thus, (5.8) holds for all $(v,a,w) \in E_Q$. By Theorem 5.9, S is a dual simulation between Q and G.

Conversely, let S be a dual simulation between Q and G. Then by Theorem 5.9, S satisfies (5.8) for all $(v,a,w) \in E_Q$. Hence, S is valid for all $w \leq v \cdot \mathfrak{F}_G^a$ and $v \leq w \cdot \mathfrak{B}_G^a$ ($(v,a,w) \in E_Q$), making S a solution of $\mathcal{E}(Q,G)$. Q. E. D.

We may now finalize the new algorithm computing DualSim as a fixpoint iteration over $\mathcal{E}(Q,G)$ with initial assignment S_0. The complete fixpoint iteration is given as Algorithm 3. Relation S is the current simulation candidate, initialized in Line 1. In set $\mathsf{Unstable}$, we store all inequalities from NEq, which we still need to process. Initially $\mathsf{Unstable} = \mathsf{NEq}$. In every iteration (from Lines 3 to 10), we first pick an unstable inequality $\epsilon = w \leq v \cdot \mathfrak{A}$ in Lines 3 and 4. We then compute $\chi_S(v) \cdot \mathfrak{A}$ and store it in row vector $\mathfrak{w}$. The condition in Line 6 evaluates positively if S is invalid for ϵ, i. e., if $\chi_S(w)$ contains nodes that cannot be reached by the edge described by $\mathfrak{A}$ from any node in $\chi_S(v)$. In this case, relation S is updated by Line 7 according to inequality ϵ. Afterwards, Line 8 adds to set $\mathsf{Unstable}$ all inequalities $\phi \in \mathsf{NEq}$ for which the state of validity of S could have changed due to the update in Line 7.

Example 5.17 (Example 5.10 continued) Let us compute the greatest dual simulation $\widehat{S}$ between the the graph pattern $Q = (V_Q, \Sigma, E_Q)$ depicted in Figure 5.5 and the graph $G = (V, \Sigma, E)$ depicted in Figure 5.3. Therefore, we follow the steps of Algorithm 3

Algorithm 3: SOI Iteration to Solve DUALSIM for Q, G, and S_0

input : $\mathcal{E}(Q,G) = (\mathsf{Var}, \mathsf{NEq})$ and assignment S_0.
output: Greatest solution $S \subseteq S_0$ of $\mathcal{E}(Q,G)$.

```
1  S ← S_0;
2  Unstable ← NEq;
3  while there is an inequality ϵ ∈ Unstable do
4  |  Unstable ← Unstable \ {ϵ};                          /* ϵ = w ≤ v · 𝔄 */
5  |  𝔴 ← χ_S(v) · 𝔄;
6  |  if χ_S(w) ⋦ 𝔴 then
7  |  |  χ_S(w) ← χ_S(w) ∧ 𝔴;                            /* update wS */
8  |  |  Unstable ← Unstable ∪ {ϕ ∈ NEq | ϕ = u ≤ w · 𝔈};
9  |  end
10 end
```

with the dual simulation SOI (5.14) from Example 5.15 as input. Furthermore, we start with the largest imaginable dual simulation candidate, namely $S_0 = V_Q \times V$, implying $\chi_{S_0}(\texttt{Prof}) = \chi_{S_0}(\texttt{Stud}) = \chi_{S_0}(\texttt{Class}) = (1,1,1,1,1,1)$.

After Line 2 has been passed, $S = S_0$ and $\mathsf{Unstable} = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_6\}$.

Iteration 1: Let us simply pick unstable inequalities by their indices, i. e., $\epsilon_1 = \texttt{Stud} \leq \texttt{Prof} \cdot \mathfrak{F}_{5.5}^{advise}$ first.

$$\begin{aligned} \chi_S(\texttt{Prof}) \cdot \mathfrak{F}_{5.5}^{advise} &= (1,1,1,1,1,1) \cdot \begin{pmatrix} 0&0&0&0&0&0 \\ 0&0&0&1&1&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \end{pmatrix} \\ &= (0,0,0,1,1,0) \not\geq (1,1,1,1,1,1) = \chi_S(\texttt{Stud}) \end{aligned} \tag{5.15}$$

Thus, the test in Line 6 is positive and we update $\chi_S(\texttt{Stud})$ according to the result, i. e., $\chi_S(\texttt{Stud}) = (1,1,1,1,1,1) \wedge (0,0,0,1,1,0) = (0,0,0,1,1,0)$. At last, we have to declare all inequalities unstable, having something to do with Stud, i. e., ϵ_2 and ϵ_5. However, they are already in Unstable.

Proceeding with $\epsilon_2 = \texttt{Prof} \leq \texttt{Stud} \cdot \mathfrak{B}_{5.5}^{advise}$, we get a similar update of $\chi_S(\texttt{Prof})$ to $(0,1,1,0,0,0)$. Once the update is performed, ϵ_1 returns to the set of unstable inequalities. At the end of this first iteration, we have $\chi_S(\texttt{Stud}) = (0,0,0,1,1,0)$, $\chi_S(\texttt{Prof}) = (0,1,1,0,0,0)$, and $\chi_S(\texttt{Class}) = (1,0,0,0,0,1)$. Furthermore, Unstable is set to $\{\epsilon_1\}$ because only ϵ_2 changes the state of $\chi_S(\texttt{Prof})$ for a potential update. ϵ_3 changes $\chi_S(\texttt{Class})$ directly and no other inequality changes it. Furthermore, $\chi_S(\texttt{Prof})$ and $\chi_S(\texttt{Stud})$ are not changed by any other inequality.

Iteration 2: Thus, we pick ϵ_1 one last time from set Unstable. This time we get

$$\begin{aligned} \chi_S(\texttt{Prof}) \cdot \mathfrak{F}_{5.5}^{advise} &= (0,1,1,0,0,0) \cdot \begin{pmatrix} 0&0&0&0&0&0 \\ 0&0&0&1&1&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \end{pmatrix} \\ &= (0,0,0,1,1,0) \geq (0,0,0,1,1,0) = \chi_S(\texttt{Stud}) \end{aligned} \tag{5.16}$$

Thus, S is valid for ϵ_1.

Since there is no more unstable inequality, the algorithm terminates after two iterations with the greatest solution, i. e., the greatest dual simulation between Q and G. ∎

Algorithm 3 computes solutions of a given dual simulation SOI of graphs Q and G.

Theorem 5.18 *Let $\mathcal{E}(Q,G)$ be a dual simulation SOI and S_0 an assignment for it. (I) Algorithm 3 terminates for inputs $\mathcal{E}(Q,G)$ and S_0, and (II) its final result S is the greatest solution of $\mathcal{E}(Q,G)$ with $S \subseteq S_0$.*

Proof: Every assignment for $\mathcal{E}(Q,G)$, including S_0, is finite. Due to Line 7, assignments after an update are guaranteed to be smaller than before the update. This is because Line 7 is only reached if $\chi_S(w) \not\leq \mathfrak{w}$ (in Line 6), i. e., there is a component i, such that $\chi_S(w)(i) = 1$ and $\mathfrak{w}(i) = 0$. Hence, after updating $\chi_S(w)$ by intersecting it with $\mathfrak{w}$, the resulting assignment has the property $\chi_S(w) = 0$. Update steps due to Line 7 occur finitely often because once $S = \emptyset$, the condition in Line 6 cannot be satisfied anymore. Hence, no inequality is added to set Unstable and finitely many executions of Line 4 will eventually empty the set because the set of inequalities of $\mathcal{E}(Q,G)$ is finite. Thus, the only possibility for non-termination occurs if some inequalities are added to Unstable infinitely often. But it is only Line 8 that adds inequalities to Unstable. In every such iteration, also Line 7 must be passed, so that S shrinks. Eventually, one of two conditions hold, (i) $S \neq \emptyset$ but the set Unstable is empty or (ii) $S = \emptyset$. As argued, the algorithm terminates in both cases, proving (I).

Next, we show that S is a solution of $\mathcal{E}(Q,G)$ after Algorithm 3 has terminated. If termination condition (ii) applies, S is a (trivial) solution of $\mathcal{E}(Q,G)$: Let $\epsilon = w \leq v \cdot \mathfrak{A}$ an inequality of the SOI. Then $\chi_S(w) = (0,0,\ldots,0) \leq \chi_S(v) \cdot \mathfrak{A}$ because the 0-vector is smaller or equal than any other vector. In fact, $\chi_S(v) = (0,0,\ldots,0)$ even implies that $\chi_S(v) \cdot \mathfrak{A} = (0,0,\ldots,0)$. In case (i), i. e., $S \neq \emptyset$ and Unstable $= \emptyset$, we need to show that S is a solution, i. e., S is a valid assignment for every inequality ϵ of $\mathcal{E}(Q,G)$. We denote the final S as S_f. Let $\epsilon = w \leq v \cdot \mathfrak{A}$ and consider the state of S after the last update of v (denoted S_v). If v has never been updated, $\chi_{S_v}(v) = \chi_{S_0}(v) = \chi_{S_f}(v)$ and ϵ is considered once (because of the initialization of Unstable in Line 2). Otherwise, $\chi_{S_v}(v) < \chi_{S_0}(v)$. The update of $\chi_S(v)$, performed in Line 7, is followed by an update of Unstable which then contains at least ϵ. Hence, ϵ is considered at least once/one last time. If $\mathfrak{w}$ is computed and the test in Line 6 fails, this means $\chi_{S_v}(w) \leq \chi_{S_v}(v) \cdot \mathfrak{A}$. As we considered the last update of v, ϵ does never occur again in Unstable. Furthermore, it holds that $\chi_{S_f}(v) = \chi_{S_v}(v)$ and $\chi_{S_f}(w) \leq \chi_{S_v}(w)$, which implies that S_f is valid for ϵ. If $\mathfrak{w}$ passes the test in Line 6, $\chi_{S_v}(w)$ is updated by $\mathfrak{w}$. The updated assignment is valid for ϵ, i. e., $\chi_{S_v}(w) \wedge \mathfrak{w} \leq \chi_{S_v}(v) \cdot \mathfrak{A}$. This is especially true if $v = w$ because we assumed S_v to be the assignment after the last update of v. Furthermore, $\chi_{S_f}(w) \leq \chi_{S_v}(w) \wedge \mathfrak{w}$ because every other update of w further reduces its value (w. r. t. $\leq$). Hence, $\chi_{S_f}(w) \leq \chi_{S_v}(w) \wedge \mathfrak{w} \leq \chi_{S_f}(v) \cdot \mathfrak{A}$, rendering S_f valid for ϵ. This argument may be repeated for all inequalities of $\mathcal{E}(Q,G)$.

It remains to be shown that S is the greatest solution with $S \subseteq S_0$. Since Algorithm 3 had S_0 as input, it holds that $S \subseteq S_0$. By Theorem 5.9 and Lemma 4.26, the greatest solution $\widehat{S} \subseteq S_0$ is unique. Thus, $\widehat{S} \subseteq S$ implies $\widehat{S} = S$. It remains to exclude that $S \subsetneq \widehat{S}$. In order to reach S, every iteration of the while-loop of Algorithm 3 removes pairs $(w,o) \in S_0 \setminus \widehat{S}$ or $(w,o) \in \widehat{S} \setminus S$. Towards a contradiction, assume we have reached S_ℓ with $\widehat{S} \subseteq S_\ell$ and the next update of S_ℓ yields S'_ℓ with $\widehat{S} \subsetneq S'_\ell$. Thus, there is $(w,o) \in \widehat{S}$, i. e., $o \in \chi_{\widehat{S}}(w)$, and an inequality $\epsilon \in \mathsf{NEq}$ with $\epsilon = w \leq v \cdot \mathfrak{A}$ for which S_ℓ is invalid. W. l. o. g., let $\mathfrak{A} = \mathfrak{F}^a_{DB}$ for some $a \in \Sigma$. The case of $\mathfrak{A} = \mathfrak{B}^a_{DB}$ is completely analogous. Since S_ℓ is invalid, $\chi_{S_\ell}(w) \not\leq \mathfrak{w}$ and, because o is removed from $\chi_{S_\ell}(w)$, there is no $o' \in O_{DB}$ with $o'\ E^a_{DB}\ o$ and $o' \in \chi_{S_\ell}(v)$. But this contradicts the assumption that $\widehat{S}$ is a solution of $\mathcal{E}(Q,G)$: As a solution, $\widehat{S}$ is a valid assignment for ϵ. Thus, $\chi_{\widehat{S}}(w) \leq \chi_{\widehat{S}}(v) \cdot \mathfrak{F}^a_{DB}$, which

implies the existence of an $o'' \in O_{DB}$ with $o'' \in \chi_{\widehat{S}}(v)$ and $o''\ E^a_{DB}\ o$ by Theorem 5.9 and Proposition 5.16. Because $\widehat{S} \subseteq S_\ell$, that o'' is an element of $\chi_{S_\ell}(v)$. Thus, $S'_\ell \subseteq \widehat{S}$, in contrast to the assumption.

Therefore, executing Algorithm 3 with input S_0 necessarily computes $\widehat{S}$, i. e., $S = \widehat{S}$ is the greatest solution of $\mathcal{E}(Q, G)$ included in S_0, which finalizes the proof for (II). Q. E. D.

Thus, by Proposition 5.16 and Theorem 5.18, Algorithm 3 computes the greatest dual simulation included in S_0.

5.2.4 Complexity Discussion

We assume the assignments in Lines 1, 2 and 4 to represent minor influences of the overall complexity. The significant influences to Algorithm 3 are estimated by if and how often the multiplication in Line 5 is performed. Each execution of Line 5 takes time $|\chi_S(v)| \cdot \mathcal{O}(|V|)$ as for each element of $\chi_S(v) \subseteq V$, row $\mathfrak{A}(v) \subseteq V$ must be added to $\mathfrak{w}$. Thus, Line 5 is bounded by $\mathcal{O}(|V|^2)$. When considering our matrix representation and the sparsity assumption, as explained in Section 5.2.1, Line 5 boils down to $\mathcal{O}(|V|)$. As before (cf. Sections 5.1.1 and 5.1.2), the value of a single variable may be subject to reduction at most $|V|$ times. Thus, a single inequality ϵ is subject to Lines 5 and 7 $\mathcal{O}(|V|)$ times.

Example 5.19 Let us assume Figure 5.1(a) to be Q and Figure 5.1(b) to be G, as in Example 5.1. The dual simulation SOI of Q and G is the following,

$$\mathcal{E}(Q, G) = \left(\{v\}, \left\{ \begin{array}{lcl} v & \leq & v \cdot \mathfrak{F}^a_G, \\ v & \leq & v \cdot \mathfrak{B}^a_G \end{array} \right\} \right).$$

Let us further assume $S_0 = \{(v, 1), (v, 2), \ldots, (v, k-1), (v, k)\}$. Picking the first inequality, i. e., $\epsilon_1 = v \leq v \cdot \mathfrak{F}^a_G$, yields a product of $\mathfrak{w} = (0, 1, \ldots, 1)$ in Line 5. Thus, $\chi_S(v) = (1, 1, \ldots, 1) \not\leq \mathfrak{w}$ and we update S to $\chi_S(v) = \chi_S(v) \wedge \mathfrak{w} = (0, 1, \ldots, 1)$. Next, we may pick ϵ_1 again or $\epsilon_2 = v \leq v \cdot \mathfrak{B}^a_G$. In both cases, $\chi_S(v)$ is reduced by at most one node of G per inequality. ∎

The worst case of updating iterations, a single inequality $\epsilon \in \mathsf{NEq}$ goes through Lines 5 to 8, is $\mathcal{O}(|V|)$. However, a single inequality $\epsilon \in \mathsf{NEq}$ may become unstable due to other inequalities, although the reached assignment S remains valid for ϵ (cf. Example 5.2).

Example 5.20 Take Figure 5.2(a) as Q and Figure 5.2(b) as G, as in Example 5.2. The respective dual simulation SOI is

$$\mathcal{E}(Q, G) = \left(\{v, w\}, \left\{ \begin{array}{lcllcl} v & \leq & v \cdot \mathfrak{F}^a_G, & v & \leq & w \cdot \mathfrak{F}^b_G, \\ v & \leq & v \cdot \mathfrak{B}^a_G, & w & \leq & v \cdot \mathfrak{B}^b_G \end{array} \right\} \right).$$

Let us assume, inequalities $\epsilon_1 = w \leq v \cdot \mathfrak{B}^b_G$ and $\epsilon_2 = v \leq v \cdot \mathfrak{B}^a_G$ are picked in Lines 3 and 4 alternatingly until they are both stable, i. e., $\epsilon_1, \epsilon_2 \notin \mathsf{Unstable}$. While ϵ_1 has no effect upon ϵ_2, an update of S due to ϵ_2 renders ϵ_1 unstable. In every such alternating iteration, the value of $\chi_S(w)$ is left unchanged and the value of $\chi_S(v)$ is reduced by one node per iteration, i. e., first k, then $k-1$, ..., up until 2, and 1. Only if 1 is removed from the set $\chi_S(v)$, ϵ_1 issues an update of $\chi_S(w)$. ∎

Combining Examples 5.19 and 5.20, we end up with an overall number of times, a single inequality may be seen in set $\mathsf{Unstable}$, is in $\mathcal{O}(|V| + |\mathsf{NEq}| \cdot |V|)$. These considerations leave us with an overall data complexity of Algorithm 3 of $\mathcal{O}(|V|^3)$ and $\mathcal{O}(|\mathsf{NEq}| \cdot (|V| + |\mathsf{NEq}| \cdot |V|) \cdot |V|^2)$ in combined complexity.

Theorem 5.21 *The special dual simulation problem (*DualSim*) is in* Ptime.

5.2.5 Optimizations

Optimal Initialization. An immediate optimization is given by altering the initial relation S_0, syntactically exploiting that for a variable/node v in V_Q, candidate nodes are only those supporting incident edges of v.

Example 5.22 In earlier examples, but especially in Example 5.17, we started with an unnecessarily *large* initial candidate, e. g., $S_0 = V_Q \times V$. Particularly, we had $\chi_{S_0}(\texttt{Prof}) = (1,1,1,1,1,1)$ although we could have observed that only two of the six graph nodes are possible values for node `Prof`, namely `Paul` and `Page`. These are the only two nodes with an outgoing *advise*-labeled, as well as a *teach*-labeled edge. The pattern node `Prof` requires both edges. Hence, we could have just started with S_0' where $\chi_{S_0'}(\texttt{Prof}) = (0,1,1,0,0,0)$ in which case the two iterations we needed in Example 5.17 boil down to a single one. ■

Hence, only those nodes exhibiting the given pattern's structure qualify as candidates. Therefore, let us denote by $\mathfrak{f}^a_G$ the bit-vector that summarizes the rows of $\mathfrak{F}^a_G$ in that $\mathfrak{f}^a_G(i) = 1$ iff there is a j with $\mathfrak{F}^a_G(i,j) = 1$, and $\mathfrak{f}^a_G(i) = 0$ otherwise. In the same lines, $\mathfrak{b}^a_G$ is defined as the summary of $\mathfrak{B}^a_G$. Then for each variable/node v in Var/V_Q, we compute the effect of inequality

$$v \;\leq\; \bigwedge_{(v,a,w)\in E_Q} \mathfrak{f}^a_G \wedge \bigwedge_{(u,a,v)\in E_Q} \mathfrak{b}^a_G \tag{5.17}$$

before considering any other inequality. Note that (5.17) follows shape (3) of Definition 5.11. Thus, when facing S_0 as input, the first step is to reduce $\chi_{S_0}(v)$ by those nodes that cannot (dual) simulate v because they are syntactically dissimilar to v.

Leaf Node Optimization. The second optimization is concerned with the *leaf nodes* of the pattern graph $Q = (V_Q, \Sigma, E_Q)$, which are all the nodes $v \in V_Q$ with a *degree* of at most 1, denoted as $deg_Q(v) \leq 1$. The *degree of a node* reflects on the number of incoming and outgoing edges in which the node participates. Formally for $G = (V, \Sigma, E)$ and $v \in V$,

$$deg_G(v) := \sum_{a\in\Sigma} (|vE^a| + |E^a v|). \tag{5.18}$$

Note that the degree also counts self-loops. A node exhibiting a self-loop will subsequently be excluded from the set of leaf nodes. The reason is that the optimization we pursue does not work correctly in case of self-loops.

The notion of degree may be naturally expanded to dual simulation SOIs $\mathcal{E}(Q,G) = (\mathsf{Var}, \mathsf{NEq})$. As for each edge of Q, NEq contains two inequalities, the degree of a variable $v \in \mathsf{Var}$ is defined by

$$deg_{\mathcal{E}(Q,G)}(v) := |\{\epsilon \in \mathsf{NEq} \mid \epsilon = v \leq w \cdot \mathfrak{A}\}|. \tag{5.19}$$

Thus, v is a *leaf variable* iff $deg_{\mathcal{E}(Q,G)}(v) \leq 1$. An inequality $\epsilon = v \leq w \cdot \mathfrak{A}$ is a *leaf inequality* iff v is a leaf variable. Note how the definition of degree excludes self-loops, e. g., for an edge (v,a,v) we would get $v \leq v \cdot \mathfrak{F}^a_G$ and $v \leq v \cdot \mathfrak{B}^a_G$. Thus, the degree of v is at least 2.

Leaf variables (or nodes, resp.) must be updated at most once due to their leaf inequalities.

Example 5.23 Consider the graph depicted in Figure 5.6, denoted as $G_{5.6}$. Furthermore, we append two edges to the pattern, depicted in Figure 5.5, asking for the e-mail addresses of `Prof` and `Stud`. Instead of another drawing of graph patterns, we simply extend the dual simulation SOI in Equation (5.14) (cf. Example 5.15) by the following inequalities:

$$\begin{array}{llcl@{\qquad}llcl} \epsilon_7: & \texttt{Mail1} & \leq & \texttt{Prof} \cdot \mathfrak{F}^{email}_{5.6} & \epsilon_8: & \texttt{Prof} & \leq & \texttt{Mail1} \cdot \mathfrak{B}^{email}_{5.6} \\ \epsilon_9: & \texttt{Mail2} & \leq & \texttt{Stud} \cdot \mathfrak{F}^{email}_{5.6} & \epsilon_{10}: & \texttt{Prof} & \leq & \texttt{Mail2} \cdot \mathfrak{B}^{email}_{5.6} \end{array} \tag{5.20}$$

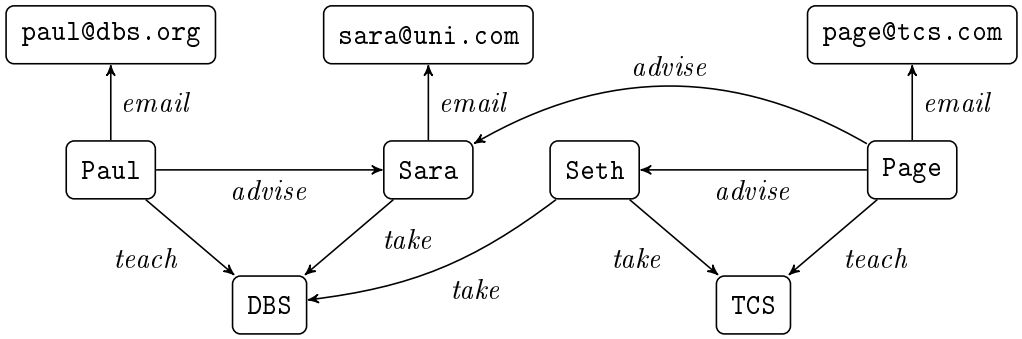


Figure 5.6: A Graph Structure

In the course of Algorithm 3, inequalities ϵ_7 to ϵ_{10} are going to reoccur in the set Unstable. Suppose we have reached some state of S, say S_1, and are going to update $\chi_{S_1}(\texttt{Mail1})$ due to a change of the variable assignment to `Prof`. If the final state of S, denoted by S_f, has the property that $\chi_{S_1}(\texttt{Prof}) = \chi_{S_f}(\texttt{Prof})$, the update of $\chi_{S_1}(\texttt{Mail1})$ is also final because ϵ_7 will not reappear as an unstable inequality. If $\chi_{S_1}(\texttt{Prof})$ is not final, ϵ_7 would issue potential changes to the assignment of `Mail1` over and over again. But this is unnecessary because after updating `Mail1`, the validity of ϵ_8 is still guaranteed. Hence, the update has no impact on the computation at this stage. ■

Picking leaf inequalities only after all the other non-leaf inequalities are stable may save a lot of unnecessary updates. Atre [15] reports on a similar technique borrowed from relational query optimization [130]. Note that after an update of a leaf variable, the status of its inequalities remains unchanged. If the variables are the same (i. e., self-loop), updates to one variable may have an impact on the other inequality (e. g., Examples 5.19 and 5.20).

Order Heuristics. Our SOI characterization of dual simulation and its implementation open up for dynamic evaluation strategies. The order in which the equations are evaluated has an impact on the overall runtime. For our experiments, we have chosen a fixed order that aims at shrinking the assignments as early as possible, e. g., by preferring inequalities with matrix components having more empty columns, an indicator for the sparsity of the resulting product. The strategy of one iteration may be adapted to a more efficient one in another iteration.

5.2.6 Comparison to State-of-the-Art

The (combined) complexity bound we gave for Algorithm 3 is polynomial in the size of (Q and) G. Every other algorithm, e. g., the ones we discussed throughout Section 5.1, would allow for the same conclusion, namely that the *special dual simulation problem* is in PTIME. Therefore, we want to know which of the presented algorithm is the fastest. We let them run on synthetic and real-world datasets. Thereby, we additionally see whether our assumptions and characterizations are, in fact, beneficial. We have implemented dual simulation versions of the naïve algorithm (cf. Section 5.1.1), HHK (cf. Section 5.1.2), and the SOI approach within our tool SPARQLSIM. All three implementations made use of our assumptions regarding the matrix data structures (cf. Section 5.2.1). We evaluate the total running times of the dual simulation procedures for computing the maximal dual simulations between all our queries and datasets (cf. Appendices A.2 and A.3). Since the competitor algorithms only consider graph pattern matching tasks, we used our queries in their BGP interpretation (cf. Appendix A.3). We solely measured the running times of the algorithms described in this chapter, leaving out the times for compiling the queries and initializing the candidate simulation (which, in our implementation, is part of the

Table 5.1: Five Runs of HHK with count-Structures

$\mathcal{Q}$	$T_{\texttt{count}}$	$T_{fixpoint}$	$T_{\sum}$
$\mathcal{L}_1^v$	2247.3300	1463.0800	3710.4100
$\mathcal{L}_2^v$	5491.2200	2714.8700	8206.0900
$\mathcal{L}_3^v$	6272.3500	401.6100	6673.9600
$\mathcal{L}_4^v$	2876.3700	123.2500	2999.6200
$\mathcal{L}_6^v$	4428.6500	743.9700	5172.6200

compilation time). We ran each query eleven times on IS68 (cf. Appendix A.1). We considered the median runtime (in general, the running times for a single query did not differ significantly for a particular algorithm). The results are presented in Table A.7 in Appendix A.4. Running times of the naïve algorithm are captioned by NAIVE, of HHK by HHK, and of the SOI algorithm by SPARQLSIM.

First of all, the performance of our SOI implementation almost always outperforms its competitors. Only in three cases, the naïve algorithm performs beats SPARQLSIM by approximately 0.0015 seconds. The HHK always performs worse than the naïve algorithm, which is puzzling at first. Intuitively speaking, it can be much more beneficial to start computing than pre-collecting a query-specific database statistics that reduces the overall fixpoint computation by a factor of $|O_{DB}|$ (or $|V|$, resp.):

- We have first tried to run HHK with its `count`-optimization. In these runs, after initializing the `remove` structures, `count` needs to be initialized. After having observed a first query set on the LUBM dataset, namely $\mathcal{L}_1^v$–$\mathcal{L}_6^v$, we deserted from `count`-optimizations. In Table 5.1, we splitted the count initialization time ($T_{\texttt{count}}$) from the computation time of the fixpoint ($T_{fixpoint}$). Due to the enormous sizes of graph database instances, exhibiting the `count`-structure exceeds the computation time of the naïve algorithm. Once the structures have been computed, the fixpoint iteration time if comparable if not even better than the naïve algorithm.

- After an intensive code review, we decided to work with the cubic-time version of the HHK that recomputes remove sets without constant-time lookups. This way, we obtained results for all our DBpedia queries. For the LUBM queries, we have set a timeout function: After at most 10,000 seconds, the computation of the simulation would stop. Not a single LUBM query finished the computation of the fixpoint before the timeout was reached.

- The reason is that the LUBM dataset is not as sparse as any of the other datasets, especially as DBpedia. Hence, when computing neighborhoods node-wise, as in the precomputation of `count` or the iterated computation of remove sets to be updated, HHK has to check many neighbors, possibly for many nodes.

Column $T_{fixpoint}$ of Table 5.1 reveals the actual fixpoint computation to be reasonably fast, i. e., at least comparable to what the naïve algorithm needed. However, the construction of the count arrays (cf. Section 5.1.2) is the bottleneck of the overall computation. Only in extreme cases, precomputing these arrays is beneficial to the overall computation. As our complexity estimation shows, HHK is a worst-case-optimal algorithm. This can be evaluated with the pattern graph in Figure 5.2 (a) and the graph database template depicted in Figure 5.7. For such graphs (for large $k > 10{,}000$), HHK produces the maximal dual simulation match (which is an empty dual simulation) much faster than the naïve algorithm and SPARQLSIM. In graph querying settings, HHK does not deliver a viable algorithm.

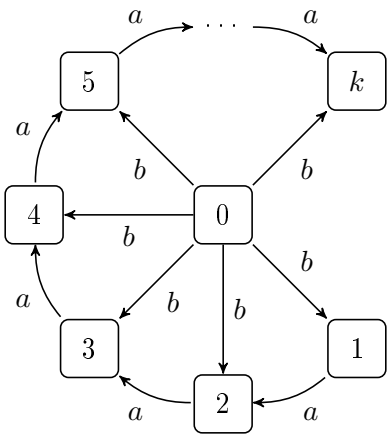


Figure 5.7: A Corner Case Data Graph for Dual Simulation Procedures

Concluding, our implementation shows indeed better runtime behavior than its competitors. Whenever sparqlSim is slower than the naïve algorithm, the computation times are extremely low for both implementations. Furthermore, our algorithm-independent characterization of dual simulations allows us to incorporate SPARQL's matching semantics, as we will show during the next section.

5.3 Pruning for SPARQL Queries

Having clarified the foundational and algorithmic aspects of dual simulations, we may now approach the realization of our maximal dual simulation semantics for SPARQL of Section 4.4. Our goal was to provide a correct and tractable pruning approach based on dual simulation principles. Section 4.4 already provides us with the fundamental result of correctness (cf. Theorem 4.72). Thus, by computing the maximal dual simulation match of $\mathcal{Q}$ in DB, we may obtain a subgraph of DB, which preserves all the SPARQL matches of $\mathcal{Q}$ in DB. We subsequently discuss systems of inequalities for queries $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. Beyond the material presented in Section 4.4, we also integrate the union operator and discuss built-in filter conditions. For each query language feature, we obtain a soundness result guaranteeing that the solutions of the constructed SOIs preserve SPARQL matches in the sense of correctness.

5.3.1 Triple and Basic Graph Patterns

Proposition 4.11 allows for the interpretation of basic graph patterns as graph structures with all their triple patterns constituting the edge relation. Thus, we derive an SOI representation of basic graph patterns (or single triple patterns, resp.) by the notion of dual simulation SOIs, now instantiated for a basic graph pattern $\mathbb{G}$ and a graph database DB.

Definition 5.24 (Dual Simulation SOI of BGPs)
Let $\mathbb{G} \in \mathbb{S}_{\mathsf{A}}$ and DB be a graph database. $\mathcal{E}(\mathbb{G}, DB) = (\mathsf{Var}, \mathsf{NEq})$ is the *dual simulation SOI of* $\mathbb{G}$ *and* DB if $\mathsf{Var} = vars(\mathbb{G})$ and

$$\mathsf{NEq} = \{\mathsf{y} \leq \mathsf{x} \cdot \mathfrak{F}^a_{DB}, \mathsf{x} \leq \mathsf{y} \cdot \mathfrak{B}^a_{DB} \mid (\mathsf{x}, a, \mathsf{y}) \in \mathbb{G}\}. \qquad \blacktriangle$$

We use the dual simulation SOI of basic graph pattern $\mathbb{G}$ and graph database DB to represent the constraints identifying dual simulation matches of $\mathbb{G}$ in DB. Therefore, solutions of $\mathcal{E}(\mathbb{G}, DB)$ are matches $S \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$ (cf. Definition 4.21). Analogously to the

case of graph patterns (cf. Proposition 5.16), we show that solutions to the dual simulation SOI of basic graph pattern $\mathbb{G}$ and graph database DB are dual simulation matches.

Lemma 5.25 *Let $\mathbb{G} \in \mathbb{S}_\mathsf{A}$ and DB be a graph database, such that $\mathcal{E}(\mathbb{G}, DB) = (\mathsf{Var}, \mathsf{NEq})$ is their dual simulation SOI. $S \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$ iff S is a solution of $\mathcal{E}(\mathcal{Q}, DB)$*

PROOF: By Proposition 4.22, $S \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$ is a dual simulation between $G(\mathbb{G})$ and DB. Furthermore, every dual simulation S between $G(\mathbb{G})$ and DB is a dual simulation match $S \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$. To prove the theorem, it suffices to show that S is a dual simulation between $G(\mathbb{G})$ and DB iff S is a solution of $\mathcal{E}(\mathbb{G}, DB)$. Note that the dual simulation SOIs $\mathcal{E}(\mathbb{G}, DB)$ and $\mathcal{E}(G(\mathbb{G}), DB)$ are exactly the same. The reason is that both sets of inequalities are built from the set $\mathbb{G}$. Hence, the claim directly follows from Proposition 5.16. Q. E. D.

This means, we compute the maximal dual simulation match (cf. Proposition 4.27) by Algorithm 3 with $\mathcal{E}(\mathcal{Q}, DB)$ and $vars(\mathcal{Q}) \times O_{DB}$ as candidate assignment.

Proposition 5.26 *Algorithm 3 returns the maximal dual simulation match for a basic graph pattern $\mathbb{G}$ in DB by the input of $\mathcal{E}(\mathbb{G}, DB)$ and $S_0 = vars(\mathbb{G}) \times O_{DB}$.*

PROOF: Let $\widehat{S} \in [\![\mathbb{G}]\!]^{\mathsf{DS}}_{DB}$ be the maximal dual simulation match of $\mathbb{G}$ in DB. $\widehat{S}$ is a solution of $\mathcal{E}(\mathbb{G}, DB)$ by Lemma 5.25. Note that every solution S' of $\mathcal{E}(\mathbb{G}, DB)$ is a subset of S_0, i. e., $S' \subseteq vars(\mathbb{G}) \times O_{DB}$. Algorithm 3 returns the greatest such solution S by Theorem 5.18. As $\widehat{S}$ is the maximal dual simulation, $S = \widehat{S}$ because

(i) $S \subsetneq \widehat{S}$ contradicts S being the greatest solution included in S_0 (cf. Theorem 5.18);

(ii) $\widehat{S} \subsetneq S$ contradicts the assumption that $\widehat{S}$ is the maximal dual simulation match since S would be a larger one;

(iii) incomparability of S and $\widehat{S}$ (up to $\subseteq$) contradicts the fact that the maximal dual simulation is unique (cf. Proposition 4.27). Q. E. D.

As the maximal dual simulation match between $\mathbb{G}$ and DB contains every SPARQL match of $\mathbb{G}$ in DB by Corollary 4.49, its computation by Algorithm 3 with the dual simulation SOI $\mathcal{E}(\mathbb{G}, DB)$ and candidate assignment $S_0 = vars(\mathbb{G}) \times O_{DB}$ is a *sound algorithmic solution* to computing the maximal dual simulation semantics of basic graph patterns. As the soundness of the result of Algorithm 3 highly depends on the construction of the system of inequalities, we transfer it to the system of inequalities $\mathcal{E}$ itself.

Definition 5.27 (Soundness of SOIs)
Let DB be a graph database, $\mathcal{Q}$ a SPARQL query, and $\mathcal{E} = (\mathsf{Var}, \mathsf{NEq})$ any SOI of $\mathcal{Q}$ and DB (cf. Definition 5.11). $\mathcal{E}$ is *sound w. r. t.* $\mathcal{Q}$ iff the greatest solution S of $\mathcal{E}$ is the maximal dual simulation match of $\mathcal{Q}$ in DB, i. e., $[\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB} = \{S\}$. ▲

As a direct consequence of Lemma 5.25 and Proposition 4.27, the dual simulation SOI of $\mathcal{Q} \in \mathbb{S}_\mathsf{A}$ and DB is sound.

Corollary 5.28 *Let DB be a graph database and $\mathcal{Q} \in \mathbb{S}_\mathsf{A}$. Then the dual simulation SOI $\mathcal{E}(\mathcal{Q}, DB)$ is sound.*

We have now reached a state of understanding of the complexity and implementation details to catch up on open proof obligations from Chapter 4. In particular, we subsequently prove that evaluation as well as non-emptiness of $(\mathbb{S}_\mathsf{A}, [\![\cdot]\!]^{\mathsf{DS}})$ are in PTIME (cf. Lemma 4.25). Furthermore, we show the PTIME computation of the maximal dual simulation match (cf. Lemma 4.28).

Proof of Lemma 4.25

Let $\mathcal{Q} \in \mathbb{S}_\mathsf{A}$ and $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database. It holds that $[\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB} \neq \emptyset$ iff the maximal dual simulation match of $\mathcal{Q}$ in DB is non-empty because, by Proposition 4.27, the maximal dual simulation match is the union of all dual simulation matches of $\mathcal{Q}$ in DB. Hence, if it is non-empty, there is a dual simulation match included. Otherwise, there is no dual simulation match of $\mathcal{Q}$ in DB. Consequently, we compute the maximal dual simulation match in PTIME (by Lemma 4.28) and check if it is non-empty.

Regarding evaluation, let $S_0 \subseteq \mathit{vars}(\mathcal{Q}) \times O_{DB}$. Algorithm 3 computes the greatest solution $S \subseteq S_0$ of $\mathcal{E}(\mathcal{Q}, DB)$ by Theorem 5.18. If $S = S_0$, then S_0 is a dual simulation match by Lemma 5.25. Otherwise, S_0 is not a solution, i. e., it is not a dual simulation match of $\mathcal{Q}$. By Theorem 5.21, the evaluation of S_0 is in PTIME. Q. E. D.

Proof of Lemma 4.28

By Proposition 5.26, Algorithm 3 returns the maximal dual simulation match of $\mathcal{Q} \in \mathbb{S}_\mathsf{A}$ and graph database DB by the input of $\mathcal{E}(\mathcal{Q}, DB)$ and $S_0 = \mathit{vars}(\mathcal{Q}) \times O_{DB}$. The computation takes PTIME by Theorem 5.21. Q. E. D.

5.3.2 Join Operators

When it comes to SPARQL's join operators, that are inner (conjunction) and left-outer joins (optional patterns), we have to be careful with the naming of variables in our SOI representation. Recall that we need to distinguish mandatory variable occurrences from optional occurrences (cf. Definition 4.73). The following example queries explain this requirement on hypothetical SOI representations.

Example 5.29 Let DB be any graph database, for which we subsequently create SOI representations of SPARQL queries. We begin with a conjunction that is a basic graph pattern representation of the pattern graph in Figure 5.5:

$$\mathcal{Q}_0 = (\mathsf{p}, \mathit{teach}, \mathsf{c})\ \texttt{AND}\ (\mathsf{p}, \mathit{advise}, \mathsf{s})\ \texttt{AND}\ (\mathsf{s}, \mathit{take}, \mathsf{c}).$$

Adopting the dual simulation SOI for basic graph patterns of the last section, we have to introduce two inequalities per triple pattern:

$$\begin{array}{llllllll} \epsilon_1: & \mathsf{c} & \leq & \mathsf{p} \cdot \mathfrak{F}^{\mathit{teach}}_{DB} & \epsilon_2: & \mathsf{p} & \leq & \mathsf{c} \cdot \mathfrak{B}^{\mathit{teach}}_{DB} \\ \epsilon_3: & \mathsf{s} & \leq & \mathsf{p} \cdot \mathfrak{F}^{\mathit{advise}}_{DB} & \epsilon_4: & \mathsf{p} & \leq & \mathsf{s} \cdot \mathfrak{B}^{\mathit{advise}}_{DB} \\ \epsilon_5: & \mathsf{c} & \leq & \mathsf{s} \cdot \mathfrak{F}^{\mathit{take}}_{DB} & \epsilon_6: & \mathsf{s} & \leq & \mathsf{c} \cdot \mathfrak{B}^{\mathit{take}}_{DB} \end{array} \tag{5.21}$$

We take the inequalities in (5.21) as our base inequalities for the subsequent examples. Next, recall query $\mathcal{Q}_1$ from Example 4.76,

$$\mathcal{Q}_1 = (\mathsf{p}, \mathit{teach}, \mathsf{c})\ \texttt{OPT}\ (\mathsf{p}, \mathit{advise}, \mathsf{s})\ \texttt{OPT}\ (\mathsf{s}, \mathit{take}, \mathsf{c}),$$

having the same triple patterns as $\mathcal{Q}_0$. Only the operator structure is different. This is why (5.21) is not the SOI representation of $\mathcal{Q}_1$ since, otherwise, our representation considered $\mathcal{Q}_0$ and $\mathcal{Q}_1$ to be equivalent. The maximal dual simulation semantics now requires that the assignments to p in ϵ_3/ϵ_4 form a subset of the assignments to p in ϵ_1/ϵ_2 (cf. Definition 4.73). Hence, p in ϵ_3/ϵ_4 is different from p in ϵ_1/ϵ_2. The same argumentation holds for c in ϵ_5/ϵ_6 and c in ϵ_1/ϵ_2.

$$\begin{array}{llllllll} \epsilon_1: & \mathsf{c} & \leq & \mathsf{p} \cdot \mathfrak{F}^{\mathit{teach}}_{DB} & \epsilon_2: & \mathsf{p} & \leq & \mathsf{c} \cdot \mathfrak{B}^{\mathit{teach}}_{DB} \\ \epsilon'_3: & \mathsf{s} & \leq & \mathsf{p}' \cdot \mathfrak{F}^{\mathit{advise}}_{DB} & \epsilon'_4: & \mathsf{p}' & \leq & \mathsf{s} \cdot \mathfrak{B}^{\mathit{advise}}_{DB} \\ \epsilon'_5: & \mathsf{c}' & \leq & \mathsf{s} \cdot \mathfrak{F}^{\mathit{take}}_{DB} & \epsilon'_6: & \mathsf{s} & \leq & \mathsf{c}' \cdot \mathfrak{B}^{\mathit{take}}_{DB} \end{array} \tag{5.22}$$

For now, we simply rename the respective SOI variables from (5.21) to obtain (5.22). Later, we will have introduced a formal translation process that guides these constructions. The requirements of optional patterns, here that p' must not exceed p and c' must not exceed c, is realized by two inequalities of shape (2) (cf. Definition 5.11):

$$\epsilon_7 : \quad \mathsf{p}' \quad \leq \quad \mathsf{p} \qquad \epsilon_8 : \quad \mathsf{c}' \quad \leq \quad \mathsf{c} \tag{5.23}$$

(5.22) and (5.23) together form a dual simulation SOI, but not for $\mathcal{Q}_1$ because variable s occurs in both optional clauses and is not distinguished in the SOI representation. The independent occurrences of s must also be independent in the SOI characterization. Therefore, another renaming of variable s takes place, e. g., yielding

$$\begin{array}{llllllll} \epsilon_1 : & \mathsf{c} & \leq & \mathsf{p} \cdot \mathfrak{F}_{DB}^{teach} & \epsilon_2 : & \mathsf{p} & \leq & \mathsf{c} \cdot \mathfrak{B}_{DB}^{teach} \\ \epsilon_3' : & \mathsf{s} & \leq & \mathsf{p}' \cdot \mathfrak{F}_{DB}^{advise} & \epsilon_4' : & \mathsf{p}' & \leq & \mathsf{s} \cdot \mathfrak{B}_{DB}^{advise} \\ \epsilon_5'' : & \mathsf{c}' & \leq & \mathsf{s}' \cdot \mathfrak{F}_{DB}^{take} & \epsilon_6'' : & \mathsf{s}' & \leq & \mathsf{c}' \cdot \mathfrak{B}_{DB}^{take} \\ \epsilon_7 : & \mathsf{p}' & \leq & \mathsf{p} & \epsilon_8 : & \mathsf{c}' & \leq & \mathsf{c} \end{array} \tag{5.24}$$

The additional variables $\mathsf{p}', \mathsf{c}', \mathsf{s}'$ have nothing to do with the original SPARQL variables. Therefore, the final solution must be interpreted by following the lines of the construction of (5.24). This means, a solution $S \subseteq \mathsf{Var} \times O_{DB}$ of (5.24) is transferred back to a dual simulation match $\widehat{S}$ of $\mathcal{Q}_1$ by $\mathsf{p}\widehat{S} := \mathsf{p}S \cup \mathsf{p}'S$, $\mathsf{s}\widehat{S} := \mathsf{s}S \cup \mathsf{s}'S$, and $\mathsf{c}\widehat{S} := \mathsf{c}S \cup \mathsf{c}'S$. Since S is valid for ϵ_7 and ϵ_8, $\mathsf{c}\widehat{S} = \mathsf{c}S$ and $\mathsf{p}\widehat{S} = \mathsf{p}S$. Note that (5.22) would be the SOI representation of

$$\mathcal{Q}_1' = (\mathsf{p}, \mathit{teach}, \mathsf{c})\ \texttt{OPT}\ ((\mathsf{p}, \mathit{advise}, \mathsf{s})\ \texttt{AND}\ (\mathsf{s}, \mathit{take}, \mathsf{c})),$$

where both occurrences of s are co-dependent.

The translation process is formalized, subsequently to this example. Before, we are considering another query from Example 4.76, namely

$$\mathcal{Q}_3 = ((\mathsf{p}, \mathit{teach}, \mathsf{c})\ \texttt{OPT}\ (\mathsf{p}, \mathit{advise}, \mathsf{s}))\ \texttt{AND}\ (\mathsf{s}, \mathit{take}, \mathsf{c}).$$

We begin, once more, with our base inequalities (5.21). Furthermore, we get an optional dependency between the occurrences of variable p, so that ϵ_3/ϵ_4 are replaced by ϵ_3'/ϵ_4' of (5.22) and we add ϵ_7 from (5.23). At last, we take care of variable s, which occurs as a mandatory variable of the right-hand side clause and as an optional variable of the left-hand side. Thus, Definition 4.73 requires that assignments to the optional occurrence must not exceed the ones to the mandatory occurrence. Therefore, we obtain the following final SOI representation of $\mathcal{Q}_3$:

$$\begin{array}{llllllll} \epsilon_1 : & \mathsf{c} & \leq & \mathsf{p} \cdot \mathfrak{F}_{DB}^{teach} & \epsilon_2 : & \mathsf{p} & \leq & \mathsf{c} \cdot \mathfrak{B}_{DB}^{teach} \\ \epsilon_3'' : & \mathsf{s}' & \leq & \mathsf{p}' \cdot \mathfrak{F}_{DB}^{advise} & \epsilon_4'' : & \mathsf{p}' & \leq & \mathsf{s}' \cdot \mathfrak{B}_{DB}^{advise} \\ \epsilon_5' : & \mathsf{c} & \leq & \mathsf{s} \cdot \mathfrak{F}_{DB}^{take} & \epsilon_6' : & \mathsf{s} & \leq & \mathsf{c} \cdot \mathfrak{B}_{DB}^{take} \\ \epsilon_7 : & \mathsf{p}' & \leq & \mathsf{p} & \epsilon_9 : & \mathsf{s}' & \leq & \mathsf{s} \end{array} \tag{5.25}$$

A literal translation of the maximal dual simulation semantics would require to distinguish between both occurrences of c as well. Hence, we would get c and c' together with inequalities

$$\mathsf{c} \quad \leq \quad \mathsf{c}' \qquad \mathsf{c}' \quad \leq \quad \mathsf{c}. \tag{5.26}$$

Every valid assignment S for (5.26) satisfies $\mathsf{c}S = \mathsf{c}'S$. Thus, the representation of $\mathcal{Q}_3$ we chose in (5.25) is complete. ■

We have learned that different variable occurrences in a query require distinct variables in the query's SOI representation. Therefore, we introduce an *intermediary SOI* that uses abstract variables in Var but binds them to actual query variables by a binding function β. Furthermore, every intermediary SOI lists a set of *mandatory variables*, which will be necessary when we encode conjunctions and optional patterns.

Definition 5.30 (Intermediary SOI)
Let $\mathcal{Q}$ be a SPARQL query and DB be a graph database. An *intermediary SOI of $\mathcal{Q}$ and DB* is a quadruple $(\mathsf{Var}, M, \beta, \mathsf{NEq})$, where Var is a set of variables, $M \subseteq \mathsf{Var}$ is a set of *mandatory variables*, $\beta : \mathsf{Var} \to \mathit{vars}(\mathcal{Q})$ is a binding function, and NEq is a set of *inequalities over* Var *and* DB[6]. ▲

Example 5.31 How should the intermediary SOI of $\mathcal{Q}_3$ from Example 5.29 and some graph database DB be constructed? We will consider variable names like $p, p', p_0, p_1, \ldots$ instead of their SPARQL variable counterparts. The binding function will associate these abstract variables with their SPARQL meaning. Therefore, we provide a different SOI variable for each occurrence of a query variable, i. e., $\mathsf{Var} = \{p_1, p_2, c_1, c_2, s_1, s_2\}$. NEq contains all the inequalities of (5.27).

$$\begin{array}{llllllll}
\phi_1: & c_1 & \leq & p_1 \cdot \mathfrak{F}_{DB}^{teach} & \phi_2: & p_1 & \leq & c_1 \cdot \mathfrak{B}_{DB}^{teach} \\
\phi_3: & s_2 & \leq & p_2 \cdot \mathfrak{F}_{DB}^{advise} & \phi_4: & p_2 & \leq & s_2 \cdot \mathfrak{B}_{DB}^{advise} \\
\phi_5: & c_2 & \leq & s_1 \cdot \mathfrak{F}_{DB}^{take} & \phi_6: & s_1 & \leq & c_2 \cdot \mathfrak{B}_{DB}^{take} \\
\phi_7: & p_2 & \leq & p_1 & \phi_8: & s_2 & \leq & s_1 \\
\phi_9: & c_1 & \leq & c_2 & \phi_{10}: & c_2 & \leq & c_1
\end{array} \tag{5.27}$$

Furthermore, β binds all the abstract variables in Var to actual query variables in $\mathit{vars}(\mathcal{Q}_3)$, i. e.,

$$\beta(x) := \begin{cases} \mathsf{p} & \text{if } x \in \{p_1, p_2\} \\ \mathsf{c} & \text{if } x \in \{c_1, c_2\} \\ \mathsf{s} & \text{if } x \in \{s_1, s_2\} \end{cases}$$

The set of mandatory variables M of the set of inequalities shall reflect on the mandatory variables of the query, i. e., the mandatory occurrence of p in $\mathcal{Q}_3$ is reflected by SOI variable p_1. Both occurrences of c are mandatory. Thus, we get $M = \{p_1, s_1, c_1, c_2\}$. ■

As already sketched, solutions of the intermediary SOI representations must be projected according to β. Otherwise, we will not be able to assess the soundness of the construction we are about to define. Assignments map the variables of the intermediary SOI to the universe of all objects $\mathcal{U}$.

Definition 5.32 (Assignment/Solution of Intermediary SOI)
Let $\mathcal{I} = (\mathsf{Var}, M, \beta, \mathsf{NEq})$ be an intermediary SOI. A relation $S \subseteq \mathsf{Var} \times \mathcal{U}$ is called an *intermediary assignment for $\mathcal{I}$*. An intermediary assignment S is valid for an inequality $\epsilon \in \mathsf{NEq}$ iff

(1) $\epsilon = w \leq v \cdot \mathfrak{A}$ implies $\chi_S(w) \leq \chi_S(v) \cdot \mathfrak{A}$,

(2) $\epsilon = w \leq v$ implies $\chi_S(w) \leq \chi_S(v)$, and

(3) $\epsilon = w \leq \mathfrak{v}$ implies $\chi_S(w) \leq \mathfrak{v}$.

S is a solution for $\mathcal{I}$ iff S is valid for all $\epsilon \in \mathsf{NEq}$. If S is a solution for $\mathcal{I}$, then $\beta(S)$ is the *projected solution* with

$$\beta(S) := \{(\beta(v), o) \mid (v, o) \in S\}.$$ ▲

Validity and solutions are formulated independently of a concrete database instance as the instances are expected to be encoded into the matrices $\mathfrak{A}$ used in shape (1) inequalities. As soon as we consider projected solutions, the actual names of the variables of an intermediary SOI are irrelevant. This observation is vital for the compositional construction

[6] The shapes follow those defined for general SOIs (cf. Definition 5.11). The matrices for shape (1) inequalities solely stem from DB.

of the intermediary SOI representation of SPARQL queries $\mathcal{Q}$, i.e., $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. When implementing the compatibility notion used in the maximal dual simulation semantics (cf. Definition 4.73), we will have to assume disjoint variable sets of the SOI components.

For basic graph patterns $\mathbb{G}$ (or triple patterns, resp.), we already know how to obtain a sound dual simulation SOI $\mathcal{E}(\mathbb{G}, DB) = (\mathsf{Var}, \mathsf{NEq})$ (cf. Definition 5.24). From this, we may derive a canonical intermediary SOI representation $\mathcal{I}(\mathbb{G}, DB)$ by $(\mathsf{Var}, \mathsf{Var}, \mathsf{id}_{\mathsf{Var}}, \mathsf{NEq})$. By an inductive argument, suppose we have intermediary SOIs $\mathcal{I}_1 = (\mathsf{Var}_1, M_1, \beta_1, \mathsf{NEq}_1)$ and $\mathcal{I}_2 = (\mathsf{Var}_2, M_2, \beta_2, \mathsf{NEq}_2)$ for queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$. As the projected solutions are independent of the actual identities of the variables (cf. Definition 5.32), we may assume Var_1 and Var_2 to be disjoint[7]. To obtain the intermediary SOI for $\mathcal{Q} = \mathcal{Q}_1 \ \mathtt{OPT} \ \mathcal{Q}_2$, we make use of the intermediary SOIs of the components by

$$\mathcal{I}_1 \ \mathtt{OPT} \ \mathcal{I}_2 := (\mathsf{Var}_1 \dot\cup \mathsf{Var}_2, \beta_1 \cup \beta_2, M_1, \mathsf{NEq}_1 \cup \mathsf{NEq}_2 \cup \mathsf{NEq}_2^1), \tag{5.28}$$

where $\mathsf{NEq}_2^1 = \{z_2 \leq z_1 \mid z_1 \in M_1 \wedge z_2 \in \mathsf{Var}_2 \wedge \beta_1(z_1) = \beta_2(z_2)\}$. Recall that $\mathcal{M}(\mathcal{Q}) = \mathcal{M}(\mathcal{Q}_1)$. By only carrying M_1 in $\mathcal{I}_1 \ \mathtt{OPT} \ \mathcal{I}_2$, we correctly implement $\mathcal{M}(\mathcal{Q})$. The new inequalities, due to NEq_2^1, implement the dependencies between the mandatory variables of $\mathcal{Q}_1$ and the variables of $\mathcal{Q}_2$.

Example 5.33 Let DB be some graph database. Once more, consider query $\mathcal{Q}_1$ from Example 4.76,

$$\mathcal{Q}_1 = (\mathsf{p}, \mathit{teach}, \mathsf{c}) \ \mathtt{OPT} \ (\mathsf{p}, \mathit{advise}, \mathsf{s}) \ \mathtt{OPT} \ (\mathsf{s}, \mathit{take}, \mathsf{c}).$$

To outline the construction process of the intermediary SOI representation of $\mathcal{Q}_1$, we perform the translation in two steps. First, we consider the translation of

$$\mathcal{Q}_z = \underbrace{(\mathsf{p}, \mathit{teach}, \mathsf{c})}_{t_1} \ \mathtt{OPT} \ \underbrace{(\mathsf{p}, \mathit{advise}, \mathsf{s})}_{t_2}.$$

Therefore, we get two intermediary SOIs, one for t_1 ($\mathcal{I}(t_1)$) and one for t_2 ($\mathcal{I}(t_2)$), as follows

- $\mathcal{I}(t_1) = (\{p_1, c_1\}, \underbrace{\{p_1, c_1\}}_{M_1}, \{(p_1, \mathsf{p}), (c_1, \mathsf{c})\}, \{\underbrace{p_1 \leq c_1 \cdot \mathfrak{B}_{DB}^{\mathit{teach}}}_{\phi_2}, \underbrace{c_1 \leq p_1 \cdot \mathfrak{F}_{DB}^{\mathit{teach}}}_{\phi_1}\})$ and
- $\mathcal{I}(t_2) = (\{p_2, s_2\}, \underbrace{\{p_2, s_2\}}_{M_2}, \{(p_2, \mathsf{p}), (s_2, \mathsf{s})\}, \{\underbrace{p_2 \leq s_2 \cdot \mathfrak{B}_{DB}^{\mathit{advise}}}_{\phi_4}, \underbrace{s_2 \leq p_2 \cdot \mathfrak{F}_{DB}^{\mathit{advise}}}_{\phi_3}\})$.

We already ensured the variable sets to be disjoint. Thus, we may now apply (5.28) to obtain the intermediary SOI representation of $\mathcal{Q}_z$ by

$$\mathcal{I}(\mathcal{Q}_z) = \left(\left\{ \begin{array}{c} p_1, c_1, \\ p_2, s_2 \end{array} \right\}, M_1, \left\{ \begin{array}{c} (c_1, \mathsf{c}), (s_2, \mathsf{s}), \\ (p_1, \mathsf{p}), (p_2, \mathsf{p}) \end{array} \right\}, \left\{ \begin{array}{c} \phi_1, \phi_2, \phi_3, \phi_4, \\ p_2 \leq p_1 \end{array} \right\} \right).$$

To obtain $\mathcal{I}(\mathcal{Q}_1)$, we use $\mathcal{I}(\mathcal{Q}_z)$ and have to combine it with the intermediary SOI of $t_3 = (\mathsf{s}, \mathit{take}, \mathsf{c})$:

- $\mathcal{I}(t_3) = (\{s_1, c_2\}, \underbrace{\{s_1, c_2\}}_{M_3}, \{(s_1, \mathsf{s}), (c_2, \mathsf{c})\}, \{\underbrace{s_1 \leq c_2 \cdot \mathfrak{B}_{DB}^{\mathit{take}}}_{\phi_6}, \underbrace{c_2 \leq s_1 \cdot \mathfrak{F}_{DB}^{\mathit{take}}}_{\phi_5}\})$.

Hence,

$$\mathcal{I}(\mathcal{Q}_1) = \left(\left\{ \begin{array}{c} p_1, c_1, \\ p_2, s_2, \\ s_1, c_2 \end{array} \right\}, M_1, \left\{ \begin{array}{c} (c_1, \mathsf{c}), (s_2, \mathsf{s}), \\ (p_1, \mathsf{p}), (p_2, \mathsf{p}), \\ (s_1, \mathsf{s}), (c_2, \mathsf{c}) \end{array} \right\}, \left\{ \begin{array}{c} \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \\ p_2 \leq p_1, c_2 \leq c_1 \end{array} \right\} \right),$$

being the final intermediary SOI representation of $\mathcal{Q}_1$ (cf. (5.24) in Example 5.29). ∎

[7] This will be reflected by the *disjoint union* operator ($\dot\cup$).

For conjunctions $\mathcal{Q} = \mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2$, recall that $\mathcal{M}(\mathcal{Q}) = \mathcal{M}(\mathcal{Q}_1) \cup \mathcal{M}(\mathcal{Q}_2)$ (cf. Definition 4.65). Thus, the mandatory variables of $\mathcal{I}_1 \texttt{ AND } \mathcal{I}_2$ will be $M_1 \cup M_2$. Furthermore, Definition 4.73 requires us to connect the mandatory variables of $\mathcal{Q}_1$ with corresponding variables of $\mathcal{Q}_2$, and the mandatory variables of $\mathcal{Q}_2$ with the corresponding ones of $\mathcal{Q}_1$. Thus, we are going to add inequality sets NEq_2^1 (as in (5.28)) and $\mathsf{NEq}_1^2 := \{z_1 \leq z_2 \mid z_2 \in M_2 \wedge z_1 \in \mathsf{Var}_1 \wedge \beta_2(z_2) = \beta_1(z_1)\}$. These thoughts culminate in

$$\mathcal{I}_1 \texttt{ AND } \mathcal{I}_2 := (\mathsf{Var}_1 \dot{\cup} \mathsf{Var}_2, \beta_1 \cup \beta_2, M_1 \cup M_2, \mathsf{NEq}_1 \cup \mathsf{NEq}_2 \cup \mathsf{NEq}_2^1 \cup \mathsf{NEq}_1^2). \tag{5.29}$$

Example 5.34 Reconsider

$$\mathcal{Q}_3 = ((\mathsf{p}, \mathit{teach}, \mathsf{c}) \texttt{ OPT } (\mathsf{p}, \mathit{advise}, \mathsf{s})) \texttt{ AND } (\mathsf{s}, \mathit{take}, \mathsf{c}).$$

The first part of the query, being $\mathcal{Q}_z$, has already been covered by Example 5.33. This time, we have to combine $\mathcal{I}(\mathcal{Q}_z)$ and $\mathcal{I}(t_3)$ conjunctively as defined by (5.29):

$$\mathcal{I}(\mathcal{Q}_3) = \left(\left\{ \begin{array}{c} p_1, c_1, \\ p_2, s_2, \\ s_1, c_2 \end{array} \right\}, M_1 \cup M_3, \left\{ \begin{array}{c} (c_1, \mathsf{c}), (s_2, \mathsf{s}), \\ (p_1, \mathsf{p}), (p_2, \mathsf{p}), \\ (s_1, \mathsf{s}), (c_2, \mathsf{c}) \end{array} \right\}, \left\{ \begin{array}{c} \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \\ p_2 \leq p_1, s_2 \leq s_1, \\ c_2 \leq c_1, c_1 \leq c_2 \end{array} \right\} \right).$$

Note, $s_2 \leq s_1$ is an inequality because $s_1 \in M_3$, $s_1 \in \mathsf{Var}_1 \cup \mathsf{Var}_2$, and $\beta_2(s_2) = \beta_3(s_1) = \mathsf{s}$. Similarly, $c_2 \leq c_1$ and $c_1 \leq c_2$. ■

Thus, while the variable bindings β (third component) are required for evaluating the solutions of intermediary SOIs, mandatory variables M (second component) are only needed during the construction of the SOI.

Definition 5.35 (Intermediary SOI for $\mathbb{S}_{\mathsf{AO}}$)
Let DB be a graph database and $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. The *intermediary SOI for $\mathcal{Q}$ and DB* is denoted by $\mathcal{I}(\mathcal{Q}, DB)$, and inductively defined by

$$\begin{array}{rcl} \mathcal{I}(t) & := & (\{x, y\}, \{x, y\}, \{(x, \mathsf{x}), (y, \mathsf{y})\}, \{y \leq x \cdot \mathfrak{F}^a_{DB}, x \leq y \cdot \mathfrak{B}^a_{DB}\}) \\ \mathcal{I}(\mathcal{Q}_1 \texttt{ OPT } \mathcal{Q}_2) & := & \mathcal{I}(\mathcal{Q}_1, DB) \texttt{ OPT } \mathcal{I}(\mathcal{Q}_2, DB) \\ \mathcal{I}(\mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2) & := & \mathcal{I}(\mathcal{Q}_1, DB) \texttt{ AND } \mathcal{I}(\mathcal{Q}_2, DB) \end{array}$$

where $t = (\mathsf{x}, a, \mathsf{y}) \in \mathbb{S}_{\mathsf{A}}$ and for $\mathcal{Q}_1, \mathcal{Q}_2 \in \mathbb{S}_{\mathsf{AO}}$. ▲

First note, with only slight adaptations, Algorithm 3 can still be used to compute the greatest solution within a candidate S_0. The new inequalities, due to the intermediary SOIs, only have the shape (2). If such an inequality $\epsilon = v \leq w$ is picked, we find the current state of S invalid (i. e., $\chi_S(v) \not\leq \chi_S(w)$) in polynomial time (because $|\chi_S(v)|, |\chi_S(w)| \leq |O_{DB}|$). If S is invalid for ϵ, update $\chi_S(v)$ w. r. t. $\chi_S(w)$ by computing S' through $\chi_{S'}(v) = \chi_S(v) \wedge \chi_S(w)$. Otherwise, S' remains equal to S.

We obtain the desired soundness result by looking at the projected solutions of the intermediary SOI construction for queries $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. Therefore, we show that every approximate dual simulation match $S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$ is a projected solution of the intermediary SOI $\mathcal{I}(\mathcal{Q}, DB)$. Conversely, if we have a solution of the intermediary SOI, then its projection is an approximate match.

Lemma 5.36 *Let $DB = (O_{DB}, \Sigma, E_{DB})$ be a graph database, $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$, and $\mathcal{I}(\mathcal{Q}, DB) = (\mathsf{Var}, M, \beta, \mathsf{NEq})$ the intermediary dual simulation SOI of $\mathcal{Q}$ and DB.*

(I) If $S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$, then there is a solution S' for $\mathcal{I}(\mathcal{Q}, DB)$ with $\beta(S') = S$.

(II) If S is a solution for $\mathcal{I}(\mathcal{Q}, DB)$, then $\beta(S) \in [\![\mathcal{Q}]\!]^{\forall \mathsf{DS}}_{DB}$.

PROOF: The proof strategy is an induction on the structure of $\mathcal{Q}$.

Base: If $\mathcal{Q}$ is a triple pattern, $\mathcal{Q} = t = (\mathsf{x}, a, \mathsf{y})$, then

$$\mathcal{I}(\mathcal{Q}, DB) = (\{x, y\}, \{x, y\}, \{(x, \mathsf{x}), (y, \mathsf{y})\}, \{y \leq x \cdot \mathfrak{F}^a_{DB}, x \leq y \cdot \mathfrak{B}^a_{DB}\}).$$

(I) Let $S \in [\![\mathcal{Q}]\!]^{\forall \mathsf{DS}}_{DB}$. By Definition 4.54, $[\![\mathcal{Q}]\!]^{\forall \mathsf{DS}}_{DB} = [\![\mathcal{Q}]\!]^{\mathsf{DS}}_{DB}$, and for each $(\mathsf{x}, o) \in S$ there is an $o' \in O_{DB}$ with $(o, a, o') \in E_{DB}$ and $(\mathsf{y}, o') \in S$. Conversely, for every $(\mathsf{y}, o) \in S$ there is an $o' \in O_{DB}$ with $(o', a, o) \in E_{DB}$ and $(\mathsf{x}, o') \in S$. Define

$$S' = \{(x, o) \mid (\mathsf{x}, o) \in S\} \cup \{(y, o) \mid (\mathsf{y}, o) \in S\},$$

for which $\beta(S') = S$ holds by construction. We need to show that S' is a solution for $\mathcal{I}(\mathcal{Q}, DB)$. Therefore, S' must be valid for $\epsilon_1 = y \leq x \cdot \mathfrak{F}^a_{DB}$ and $\epsilon_2 = x \leq y \cdot \mathfrak{F}^a_{DB}$. Recall from Section 5.2.1, $\chi_{S'}(x), \chi_{S'}(y) \subseteq O_{DB}$ and $\chi_{S'}(v) \cdot \mathfrak{F}^a_{DB}$ is the set of nodes reachable from $\chi_{S'}(v)$ by traversing an a-labeled edge in the forward direction. Suppose S' is invalid for ϵ_1, i. e., $\chi_{S'}(y) \not\subseteq \chi_{S'}(x) \cdot \mathfrak{F}^a_{DB}$, then there is a node $o' \in O_{DB}$, such that $(y, o') \in S'$ but $o' \notin \chi_{S'}(x) \cdot \mathfrak{F}^a_{DB}$. By construction, $(\mathsf{y}, o') \in S$ and as $o' \notin \chi_{S'}(x) \cdot \mathfrak{F}^a_{DB}$, there is no node $o \in O_{DB}$ with $(\mathsf{x}, o) \in S'$ and $(o, a, o') \in E_{DB}$, which contradicts the assumption that S is a dual simulation match. Thus, S' is valid for ϵ_1. By a similar argumentation, S' can be shown valid for ϵ_2. Hence, S' is a solution for $\mathcal{I}(\mathcal{Q}, DB)$.

(II) Let S be a solution for $\mathcal{I}(\mathcal{Q}, DB)$. Then S is valid for ϵ_1 and ϵ_2 (cf. case (I)). Let $(\mathsf{x}, o) \in \beta(S)$. Then we need to give some $o' \in O_{DB}$ with $(o, a, o') \in E_{DB}$ and $(\mathsf{y}, o') \in \beta(S)$. Since S is valid for ϵ_2, there is at least one $o' \in \chi_S(y)$, so that $(o, a, o') \in E_{DB}$. Hence, $(y, o') \in S'$ implies that $(\beta(y), o') = (\mathsf{y}, o') \in \beta(S)$. The case of $(\mathsf{y}, o) \in \beta(S)$ is completely analogous.

Hypothesis: For queries $\mathcal{Q}_i \in \mathbb{S}_{\mathsf{AO}}$ $(i = 1, 2)$, it holds that (I) $S_i \in [\![\mathcal{Q}_i]\!]^{\forall \mathsf{DS}}_{DB}$ implies a solution S'_i for $\mathcal{I}(\mathcal{Q}_i, DB)$ with $\beta(S'_i) = S_i$ and (II) every solution S_i for $\mathcal{I}(\mathcal{Q}_i, DB) = (\mathsf{Var}_i, M_i, \beta_i, \mathsf{NEq}_i)$ is an approximate match by $\beta_i(S_i) \in [\![\mathcal{Q}_i]\!]^{\forall \mathsf{DS}}_{DB}$.

Step: It remains to be shown that the claims also hold for conjunctions $\mathcal{Q} = \mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2$ and optional patterns $\mathcal{Q} = \mathcal{Q}_1 \texttt{ OPT } \mathcal{Q}_2$.

We first consider a conjunction $\mathcal{Q} = \mathcal{Q}_1 \texttt{ AND } \mathcal{Q}_2$ and its intermediary SOI $\mathcal{I}(\mathcal{Q}, DB) = (\mathsf{Var}_1 \dot\cup \mathsf{Var}_2, M_1 \cup M_2, \beta_1 \cup \beta_2, \mathsf{NEq})$ with $\mathsf{NEq}_1 \cup \mathsf{NEq}_2 \subseteq \mathsf{NEq}$.

(I) If $S \in [\![\mathcal{Q}]\!]^{\forall \mathsf{DS}}_{DB}$, there are $S_i \in [\![\mathcal{Q}_i]\!]^{\forall \mathsf{DS}}_{DB}$ $(i = 1, 2)$ with (a) $S = S_1 \cup S_2$, (b) $vS_2 \subseteq vS_1$ $(v \in \mathcal{M}(\mathcal{Q}_1))$, and (c) $vS_1 \subseteq vS_2$ $(v \in \mathcal{M}(\mathcal{Q}_2))$. By induction hypothesis, there are solutions S'_i for $\mathcal{I}(\mathcal{Q}_i, DB)$ $(i = 1, 2)$ with $\beta_i(S'_i) = S_i$. Define $S' = S'_1 \cup S'_2$ with $\beta(S') = S$ by construction. S' is valid for all inequalities in $\mathsf{NEq}_1 \cup \mathsf{NEq}_2$. Additionally, we have to check validity of S' w. r. t. inequalities $v \leq w \in \mathsf{NEq}$ where $w \in M_i$, $v \in \mathsf{Var}_j$, and $\beta_i(w) = \beta_j(v)$ $(\{i, j\} = \{1, 2\})$. Suppose S' is invalid for $v \leq w$ $(\beta(v) = \beta(w))$, i. e., $\chi_{S'}(v) \not\subseteq \chi_{S'}(w)$. Then there is some node $o' \in \chi_{S'}(\beta_j(v))$ with $o' \notin \chi_{S'}(\beta_i(w))$. A simple inductive argument shows that $w \in M_i$ implies $\beta_i(w) \in \mathcal{M}(\mathcal{Q}_i)$. Then o' constitutes a contradiction for (b) for $i = 1$ or (c) for $i = 2$. Hence, S' is valid for $v \leq w$.

(II) If S is a valid assignment for $\mathcal{I}(\mathcal{Q}, DB)$, S is valid for all inequalities in NEq. Hence, S is also valid for NEq_1 and NEq_2 (since $\mathsf{NEq}_1 \cup \mathsf{NEq}_2 \subseteq \mathsf{NEq}$). In fact, there is a largest assignment $S_1 \subseteq S$ valid for NEq_1 and a largest assignment $S_2 \subseteq S$ valid for NEq_2, so that $S = S_1 \cup S_2$ but $dom(S_1) \cap dom(S_2) = \emptyset$

(because $\mathsf{Var}_1 \cap \mathsf{Var}_2 = \emptyset$). For these assignments, $\beta_i(S_i) \in [\![\mathcal{Q}_i]\!]^{\forall\mathsf{DS}}_{DB}$ by induction hypothesis. We need to show that $\beta_1(S_1) \cup \beta_2(S_2) \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Therefore, let $v \in \mathcal{M}(\mathcal{Q}_1)$ (analogously, $v \in \mathcal{M}(\mathcal{Q}_2)$). Towards a contradiction, assume $v\beta_2(S_2) \not\subseteq v\beta_1(S_1)$. Again by an inductive argument, $v \in \mathcal{M}(\mathcal{Q}_1)$ implies some variable $x \in M_1$ with $\beta_1(x) = v$. Furthermore, $v \in vars(\mathcal{Q}_2)$ implies a variable $y \in \mathsf{Var}_2$ with $\beta_2(y) = v$. By construction of $\mathcal{I}(\mathcal{Q}, DB)$, $y \leq x \in \mathsf{NEq}$. Now $v\beta_2(S_2) \not\subseteq v\beta_1(S_1)$ implies $\beta_2(y)\beta_2(S_2) \not\subseteq \beta_1(x)\beta_1(S_1)$, which implies itself $\chi_S(y) \not\leq \chi_S(x)$ and, thereby, contradicts the assumption that S is a solution for $\mathcal{I}(\mathcal{Q}, DB)$, including the inequality $y \leq x$.

The case of $\mathcal{Q} = \mathcal{Q}_1 \,\mathtt{OPT}\, \mathcal{Q}_2$ is completely analogous, replacing the symmetric argument ($\{i, j\} = \{1, 2\}$) by the instance for $i = 1$ and $j = 2$. Q. E. D.

Finally, the maximal dual simulation match $\widehat{S} \in [\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB}$ is the greatest dual simulation $S \in [\![\mathcal{Q}]\!]^{\forall\mathsf{DS}}_{DB}$. Therefore, the greatest solution of the intermediary SOI $\mathcal{I}(\mathcal{Q}, DB)$ projects to the maximal dual simulation match $\widehat{S}$. We conclude the intermediary SOI of a query $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$ and graph database DB is sound (under projection).

Theorem 5.37 *Let DB be a graph database and $\mathcal{Q} \in \mathbb{S}_{\mathsf{AO}}$. The intermediary SOI of $\mathcal{Q}$ and DB is sound, i. e., the greatest solution $\widehat{S}$ of $\mathcal{I}(\mathcal{Q}, DB)$ possesses the property of $[\![\mathcal{Q}]\!]^{\mathsf{MDS}}_{DB} = \{\beta(\widehat{S})\}$.*

Proof: By Lemma 5.36, every approximate dual simulation match S of $\mathcal{Q}$ in DB has a solution S' for $\mathcal{I}(\mathcal{Q}, DB)$, so that $\beta(S') = S$, and vice versa. Thus, from Lemma 4.71, the greatest solution $\widehat{S}$ is the maximal dual simulation match by $\beta(\widehat{S})$. Q. E. D.

As a last open proof, we have to argue that computing the maximal dual simulation match is in polynomial time (Theorem 4.75).

Proof of Theorem 4.75

We give a Ptime procedure that computes the maximal solution of the intermediary SOI of $\mathcal{Q}$ and DB, $\mathcal{I}(\mathcal{Q}, DB)$. We apply Algorithm 3 to solve the $\mathcal{I}(\mathcal{Q}, DB) = (\mathsf{Var}, M, \beta, \mathsf{NEq})$ on initial candidate $S_0 \subseteq \mathsf{Var} \times \mathcal{U}$. For all inequalities $\epsilon \in \mathsf{NEq}$ of shape (1), i. e., $\epsilon = w \leq v \cdot \mathfrak{A}$ ($v, w \in \mathsf{Var}$), Algorithm 3 already produces the greatest solution in Ptime. Additionally, the algorithm must cope with inequalities ϕ of shape (2), i. e., $\phi = v \leq w$. Once all ϵ are stable, say at state S, we check S for validity of ϕ, i. e., whether $\chi_S(v) \leq \chi_S(w)$ in $\mathcal{O}(|O_{DB}|)$ if $\chi_S(v), \chi_S(w)$ are bit-vectors. If S is invalid, we update S to S' by

$$\chi_{S'}(x) := \begin{cases} \chi_S(v) \wedge \chi_S(w) & \text{if } x = v \\ \chi_S(x) & \text{otherwise.} \end{cases} \tag{5.30}$$

Updates due to (5.30) can be done in $\mathcal{O}(|O_{DB}|)$ by iterating over all positions of the vectors $\chi_S(v)$ and $\chi_S(w)$. If there is a j with $\chi_S(v)(j) = 1$ but $\chi_S(w)(j) = 0$, flip the jth bit in $\chi_S(v)$. Upon such an update, we have to declare all inequalities $u \leq v \cdot \mathfrak{A}$ unstable. After we updated all ϕ, either all inequalities are stable, or we begin with the *standard procedure* of Algorithm 3. After finitely many such iterations, all inequalities of shape (1) are stable, and S is valid for all inequalities of shape (2). Thus, the overall computation is in Ptime. The worst case adds a factor of $|\mathsf{Var}| \cdot |O_{DB}|$ because in each iteration, a single update as in (5.30) may reduce S by a single bit. After we have reached a solution S of $\mathcal{I}(\mathcal{Q}, DB)$, we have to compute the maximal dual simulation match by $\beta(S)$, which may be done in $\mathcal{O}(|\mathsf{Var}| \cdot |O_{DB}|)$ in a naïve implementation. Q. E. D.

5.3.3 The Union Operator and Some Filter Conditions

Even without a formal proof of soundness[8], we still want to sketch feasible constructions of SOIs for unions $\mathcal{Q}_u = \mathcal{Q}_1 \,\texttt{UNION}\, \mathcal{Q}_2$ and some built-in filter conditions, i. e., $\mathcal{Q}_f = \mathcal{Q}_1 \,\texttt{FILTER}\, R$. We assume for the query clauses, $\mathcal{Q}_i$ ($i \in \mathbb{N}$), to have already obtained intermediary SOI representations $\mathcal{I}(\mathcal{Q}_i, DB) = (\mathsf{Var}_i, M_i, \beta_i, \mathsf{NEq}_i)$. As before, we require that for every $i, j \in \mathbb{N}$, $\mathsf{Var}_i \cap \mathsf{Var}_j \neq \emptyset$ implies $i = j$.

Example 5.38 The most basic form of unions considers all queries $\mathcal{Q} \in \mathbb{S}_{\mathsf{AOU}}$ in the *union normal form* (cf. Example 4.16 and Proposition 4.17), i. e., $\mathcal{Q} = \mathcal{Q}_1 \,\texttt{UNION}\, \mathcal{Q}_2 \,\texttt{UNION}\, \ldots \,\texttt{UNION}\, \mathcal{Q}_k$ ($k > 1$) with $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k \in \mathbb{S}_{\mathsf{AO}}$. For these queries, our intermediary SOI representation can canonically be extended to capture $\mathcal{Q}$ by

$$\mathcal{I}(\mathcal{Q}, DB) := \left(\bigcup_{1 \leq i \leq k} \mathsf{Var}_i, \bigcup_{1 \leq i \leq k} M_i, \bigcup_{1 \leq i \leq k} \beta_i, \bigcup_{1 \leq i \leq k} \mathsf{NEq}_i \right). \tag{5.31}$$

The maximal dual simulation match for $\mathcal{Q}$ in DB surely is the union of the maximal dual simulation matches of $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k$. Suppose, we have a variable $\mathsf{v} \in \mathit{vars}(\mathcal{Q}_i) \cap \mathit{vars}(\mathcal{Q}_j)$ ($1 \leq i < j \leq k$). Then there are distinct variable $x \in \mathsf{Var}_i$ and $y \in \mathsf{Var}_j$ with $\beta_i(x) = \mathsf{v}$ and $\beta_j(y) = \mathsf{v}$. Hence, any projected maximal solution reflects on both variables independently because $\mathsf{Var}_i \cap \mathsf{Var}_j = \emptyset$. Summarizing, the constructs we allow for systems of inequalities suffice in case of union normal form queries.

If we do not pursue the exponential construction to obtain the union normal form, we need an additional inequality shape. Consider the query

$$\mathcal{Q}_u = \underbrace{(\mathsf{p}, \mathit{teach}, \mathsf{c})}_{t_1} \,\texttt{AND}\, \left(\underbrace{(\mathsf{p}, \mathit{advise}, \mathsf{s})}_{t_2} \,\texttt{UNION}\, \underbrace{(\mathsf{s}, \mathit{advisedBy}, \mathsf{p})}_{t_3} \right),$$

where *advisedBy* is thought of as the reverse property for *advise*. Hence, by this query we look for completeness as we assume that both kinds of triples might occur in the graph database. Intermediary SOIs for t_1, t_2, and t_3 are derived from Definition 5.35 and we may assume the occurrences of p to be reflected by variables p_1, p_2, and p_3 (in $\mathcal{I}(t_i, DB)$, $i = 1, 2, 3$). For sure, $p_2 \leq p_1$ and $p_3 \leq p_1$ (since $p_1 \in M_1$ and $p_2 \in \mathsf{Var}_2 / p_3 \in \mathsf{Var}_3$). However, p_1 is also bounded because of compatibility between matches of the union $t_2 \,\texttt{UNION}\, t_3$ and of t_1. p_1 must not exceed what p_2 and p_3 prescribe together, i. e.,

$$p_1 \leq p_2 + p_3 \tag{5.32}$$

would be an inequality reflecting on valid matches/assignments and solutions. ∎

The general construction is more involved as we need to keep track of union patterns in a compositional way. Think about a query $\mathcal{Q}_1 \,\texttt{AND}\, (\mathcal{Q}_2 \,\texttt{UNION}\, \mathcal{Q}_3)$ in which some variable, say x, is mandatory in $\mathcal{Q}_1$ and $\mathcal{Q}_2$ but optional in $\mathcal{Q}_3$.

Queries employing built-in filter conditions do not generally have SOI representations. Examples that work are single atoms of the shapes $\mathsf{x} = \mathsf{c}$ and $\mathsf{x} \neq \mathsf{c}$. In both cases, let $\mathcal{I} = (\mathsf{Var}, M, \beta, \mathsf{NEq})$ be a sound SOI representation of $\mathcal{Q}$. Then the former case is realized by

$$\mathcal{I} \,\texttt{FILTER}\, (\mathsf{x} = \mathsf{c}) := (\mathsf{Var}, M, \beta, \mathsf{NEq} \cup \{x \leq \mathsf{c} \mid x \in \mathsf{Var} \wedge \beta(x) = \mathsf{x}\}), \tag{5.33}$$

[8] Such a soundness proof would at least require a formal definition of the semantics of the additional operator structures.

where $\mathfrak{c}$ is an $|O_{DB}|$-dimensional vector with single component set to 1, i. e., $\mathfrak{c}(\mathsf{c}) = 1$ and $\mathfrak{c}(n) = 0$ $(n \neq \mathsf{c})$. In the latter case, we get a SOI realization by

$$\mathcal{I}\,\texttt{FILTER}\,(\mathsf{x} = \mathsf{y}) := \left(\mathsf{Var}, M, \beta, \mathsf{NEq} \cup \left\{ \begin{array}{l} \sum_{x \in \mathsf{Var}:\beta(x)=\mathsf{x}} x \leq \sum_{y \in \mathsf{Var}:\beta(y)=\mathsf{y}} y, \\ \sum_{y \in \mathsf{Var}:\beta(y)=\mathsf{y}} y \leq \sum_{x \in \mathsf{Var}:\beta(x)=\mathsf{x}} x \end{array} \right\}\right) \tag{5.34}$$

Derived from (5.33), we also handle $\mathsf{x} \neq \mathsf{c}$ by simply assuming the inverse vector $\mathfrak{c}$. Negation of $\mathsf{x} = \mathsf{y}$ and general negations of built-in filter conditions would require negated types of inequalities. However, conjunctions of the atomic built-in filter conditions above are manageable. Disjunctions only work as long as in $C_1 \vee C_2 \vee \ldots \vee C_k$, every of the clauses has the form $\mathsf{x} = c_i$ $(1 \leq i \leq k)$. As soon as the clauses are mixed, we can derive no restrictions at all in terms of additional inequalities. For instance, if a query is filtered by the condition $\mathsf{x} = \mathsf{c} \vee \mathsf{y} = \mathsf{c}$, we cannot assert additional constraints as in Equation (5.33) as x could be matched by anything, as long as c matches y. The same applies to arbitrary mixes of conjunctions and disjunctions.

As a last construct, we mention $\mathsf{bound}(\mathsf{x})$ and $\neg\mathsf{bound}(\mathsf{x})$. The latter is implemented by an additional constraint, namely

$$\sum_{x \in \mathsf{Var}:\beta(x)=\mathsf{x}} \leq \mathsf{o}. \tag{5.35}$$

The former case requires, once again, negated types of inequalities, which may be implemented, but leave the scope of this thesis.

For (5.34) and (5.35), we had to use inequalities with complex expressions on the left-hand side of the inequalities. But as long as summation is used, they are just shorthands. Let $\{x_1, x_2, \ldots, x_m\} = \{x \in \mathsf{Var} \mid \beta(x) = \mathsf{x}\}$. Then (5.35) unfolds to

$$\begin{array}{rcl} x_1 & \leq & \mathsf{o} \\ x_2 & \leq & \mathsf{o} \\ & \vdots & \\ x_k & \leq & \mathsf{o} \end{array} \tag{5.36}$$

5.3.4 Efficiency

In this set of experiments, we used our implementation SPARQLSIM and both database systems, Virtuoso and RDFox (Appendix A.1). To emulate a querying setting incorporating maximal dual simulation pruning for SPARQL, we first compute the pruning (DB_{prune}) by SPARQLSIM. Afterwards, we load the small database prune into the database and execute the query as described in Appendix A.1. All experiments were performed on IS69.

The results have recently been published in [93, 92]. We did conduct further experiments using Wikidata queries. However, the results are no more conclusive than the ones we already obtained on DBpedia and LUBM. The reasons for this potentially stem from the 10% sample of Wikidata and the implied selection procedure of Wikidata queries (cf. Appendix A.3).

To be self-contained, we include the result tables of our experiments in Table A.8 in Appendix A.4. Column $T_{\text{SPARQLSIM}}$ reports on the pruning time of our implementation. Columns $T(DB_0)$ and $T(DB_{prune})$ contains the running times of the database systems on the full dataset (DB_0) and the pruned dataset (DB_{pruned}). Columns for the combined times of SPARQLSIM's pruning and the system's querying on the prune is captioned by $\sum$.

Summarizing, the DBMS Virtuoso [47] that builds upon relational database system techniques to answer SPARQL queries is almost always faster than our the maximal dual simulation pruning combined with querying on the pruned database. Only for query $\mathcal{L}_2$, our pruning approach has a consistent impact on both database systems on LUBM. In 20

out of 31 queries on DBpedia, RDFox benefits from the reduction by SPARQLSIM's pruning. Although SPARQLSIM's effectiveness improves upon the baseline only by 5%, RDFox exhibits consistently lower computation times on the pruned database. With a maximal gain of more than 40 seconds and a minimal loss of less than 2 seconds, SPARQLSIM's method appears as a viable complementation of RDFox.

5.4 Summary

In this chapter, we started with a characterization of several *dual simulation*-related problems. As it turned out, each problem was reducible to DUALSIM, the *special dual simulation problem*, which, given two graphs, Q and G and a dual simulation candidate $S_0 \subseteq V_Q \times V$, asks for the greatest dual simulation $S \subseteq S_0$. Therefore, we concentrated on algorithmic principles and solutions for this particular problem.

To get an overview of existing solutions, we presented a naïve iteration and HHK, the most popular similarity algorithm, as measured by its reception in the literature[9]. Furthermore, we sketched more space-efficient solutions that are not required in our setting (small query/large data graph). Based on graph database-specific assumptions, we developed a *system of inequalities* (SOI) approach that first transforms the DUALSIM problem into an instance of an SOI. Finding dual simulations is the same task as finding solutions to the constructed SOI.

In a first experiment (cf. Section 5.2.6), we were able to show that our algorithm does indeed perform better than the naïve algorithm and HHK. Constructing the count-data structure, that is required for an efficient fixpoint retrieval, always exceeded the evaluation without using it. Therefore, we presented results only for the cubic version of HHK (cf. Section 5.1.2). In terms of a feasibility evaluation, our solution seems to be the best fit for tasks related to database querying.

Based on our SOI approach, we implemented and formally justified the maximal dual simulation semantics of Chapter 4 as a pruning semantics for SPARQL query processing. Its formalization is tightly coupled with our implementation SPARQLSIM. We gave sketches of how SPARQL's union operator and some built-in filter conditions can be reintegrated into the SOI representation.

Briefly, our efficiency evaluation shows that a full-fledged (commercial) database system like Virtuoso often works faster if no external mechanism interferes. Sometimes Virtuoso needed more time answering the query on the pruned database than on the full one. One reason for this may be that the heuristics used for finding the *right* query execution plan are out of reach when considering the (degenerated) statistics of the pruned database. Of course, the structure of our experimental setting presupposes that our tool runs as an integrated component in the database management system. Furthermore, the data structures we used are highly optimized towards the necessary fixpoint operations. If we cannot build upon similar indexing techniques, as we used them throughout this chapter, the running times will become much slower. In that case, our techniques are still feasible additions to database systems that already build on bit-matrix representations of the data, e. g., Redisgraph [31].

We observed that the order in which unstable inequalities are processed by Algorithm 3 plays an important rôle in the overall evaluation time. As discussed, we aimed for straightforward optimizations and ordering heuristics (cf. Section 5.2.5), but more involved insights, also based on the mathematical structures, might boost our implementation even further.

[9] according to the ACM Digital Library, 537 citations on Oct 21, 2019

CHAPTER 6

Conclusion

The individual chapters already provide summary sections. Therefore, we will only briefly reflect on the goals we set out in Chapter 1. Furthermore, we provide a perspective on simulations we obtained during the process of this thesis.

We believe the reason why tractable graph pattern matching up to (dual) simulations has not been conducted on extensive datasets because the existing algorithms, most of the works build upon [86], do not scale well with large data graphs. Tractability, i. e., the worst-case complexity is polynomial, only accounts for the worst imaginable case. In our experiments, we could show that the worst-case occurs only rarely in real, sometimes in synthetic data. The implementation of our new solution devised for solving the maximal dual simulation problem outperforms competitors by several orders of magnitude, although they sometimes exhibit a better worst-case complexity. Thereby, we could also show that graph pattern matching up to dual simulations handles the usual amount of data in large-scale real-world knowledge graphs. Although our software prototype cannot cope with industrial standard relational database technology in runtime, it provides enhancements for other software prototypes. Our tool can be used as a pruning mechanism for SPARQL using conjunctions, disjunctions, unions, and some built-in filter conditions. Thereby, we sketched the potential of dual simulation pattern matching.

Our implementation was grounded on a sound basis of formal results, justifying that our ultimate *maximal dual simulation semantics* is a correct approximation of the SPARQL semantics. At first, it seemed as if dual simulations must not be combined by SPARQL operators since, otherwise, tractability or correctness get lost. Each of the correct semantics we developed contributed an idea that could later be used to tackle the union-closed SPARQL semantics of maximal dual simulations.

6.1 Perspective

We devised several semantic notions based on simulations. Right from the beginning, we recaptured graph schemas and their semantics, as given by the modeled database instances. As early as in this step, the mathematical objects of simulations showed an enormous *fragility* towards change. While the graph schema model had a sound basis on classical simulations, the lack of root nodes in modern graph data immediately turned a well-known preorder into a reflexive relation with hardly any meaning. We had to adapt two assumptions in exchange for root nodes:

1. Connectedness of graph schemas, which is a property guaranteed by root nodes (cf. Section 2.1.1) and

2. incorporation of backward steps as simulation steps, culminating in dual simulations.

The second assumption, i. e., employment of dual simulations, also guarantees a property, formerly held by root nodes, namely reachability of all other nodes. Once we rectified our simulation preorder, and thereby, justified the semantics of graph schemas, we tried to add a small feature, namely mandatory edges, which immediately destroyed any useful properties for deriving a sensible semantics. It was only the fallback to deterministic graph schemas that helped here. According to the literature, the implications and the concrete limits of modality and alternating simulation (i. e., modal refinement) are open [79]. Going onward, the same issues as for simulations occurred with the notions of similarity and bisimilarity in Chapter 3. Also, essentially the same fixes applied. On our last quest in Chapter 4, we only added two operators and directly turned the resulting query semantics incomparable, where it was still comparable in case of basic graph patterns (cf. Theorem 3.24).

Based on this brief experience report, we are curious about the following:

1. Are there other mathematically well-founded notions that are *fragile* in the sense described above?

2. Are there other robust notions? We suspect graph isomorphisms to be a robust matching notion.

3. If SPARQL and general relational algebra is not the right query language for simulations, which one is?

6.2 Other Future Work

We aim at extending our graph schema and graph pattern matching models by incorporating data values. A first sketch has been part of our earlier work [90]. Although, as we discussed in [91], some form of heterogeneity (e. g., incomplete data by failures in Chapter 3) can be captured by means of pattern matching principles, what if the matches are drastically different from the pattern? We believe, in these cases, there is still hope for graph pattern matching notions, but they have to be devised according to an application context. A very general framework is known by the name of *isotactics* [111, 110], which were invented to find equivalent business process models (e. g., in BPMN notation), although they have quite different shape. One of the key components of the method are alignments as we sketched them in Section 2.1.1.

Another reason to go beyond pure pattern matching and also take the data, i. e., literals and annotations, in graph data seriously is the experienced rise of popularity of the property graph model[23, 7, 5], the data model of Neo4j. In this model, *key–value* pairs may be attached to nodes and concrete edges, raising the classical graph pattern matching to a hybrid mode of relational and graph technologies.

To potentially overcome the issue regarding the runtime of some queries, an immediate aid is Offline pre-processing of the data graphs. Instead of searching the whole data graph, we would run our dual simulation procedure on the smallest graph that is equivalent to the full data graph up to dual similarity. Techniques of partition refinement (cf. Section 5.1.3) and the algorithmic ideas of knowledge graph summarization (cf. Section 1.1) might help in producing such a small graph. However, as argued earlier, dual similarity is highly instable on updates. Single edge insertions or deletions may invalidate the equivalent small graph.

APPENDIX A

Evaluation Setup and Results

A.1 Environment

We have performed all our experiments on two different servers. Whenever necessary, we identify them by their names. IS68 is a server running Ubuntu 18.04.1 with an IntelCore i9 7940X, 3.1000 GHz, having 14 cores and 128 GB RAM. IS69 is a server running Ubuntu 18.04.2 with four XEON E7-8837, 2.6700 GHz, having eight cores each, 384 GB RAM, and a Kingston DCP1000 NVMe PCI-E SSD.

In our effectiveness and efficiency experiments, we used two RDF database systems, namely Virtuoso [47] and the high-performance in-memory database RDFox [102]. We deactivated caching for Virtuoso to achieve stable query evaluation times. RDFox is not using query caches.

For the evaluation of efficiency, we have run all queries ten times on each database and averaged the times. We did the same to get stable runtime results from our tool SPARQLSIM.

A.2 Datasets

We have summarized the dataset's characteristics in Table A.1. Thereby, every dataset forms a graph database $G = (V, \Sigma, E)$. The reported numbers, especially for $|\Sigma|$, reflect on the number of predicates used in the respective dataset, i. e., $\Sigma \subsetneq \mathcal{P}$ with $\Sigma = \{p \in \mathcal{P} \mid (s, p, o) \in E\}$.

DBpedia: We used DBpedia [17, 99, 82] in version 2015-10[1]. Compared to all the other datasets, we have more than 60,000 predicates in use in the dataset. Thus, we expect (1) the individual rows of the adjacency matrices to be extremely sparse and (2) also, the number of non-empty rows per adjacency rows to be lower than for the other dataset. Regarding (1), the node with the maximum degree of 51,328 incoming and outgoing edges references a Polish government Website, specifying how territorial information shall be represented. The average degree within the dataset is 1.8500. Observe, based on the numbers reported in Table A.1, the average $216132665 \times 216132665$ adjacency matrix would have approximately 11,500 entries. The forward matrices have an average number of non-empty rows of 6211 (with a median of 3) while the backward edges come with 4019 (with a median of 2) non-empty rows on average.

[1] https://wiki.dbpedia.org/Downloads2015-10

Table A.1: Used Datasets and Their Characteristics

Dataset	$\|V\|$	$\|\Sigma\|$	$\|E\|$
DBpedia	216,132,665	65,430	751,603,507
Wikidata	167,747,925	4681	318,444,534
LDBC 100	122,544	40	209,063
LDBC 500	424,673	40	957,026
LDBC 1000	904,466	40	2,152,183
LDBC 5000	7,066,988	40	17,254,242
LUBM	328,620,750	18	1,334,681,192

Wikidata: We picked a 10% random sample from Wikidata [87, 70], September edition of 2018 without metadata and references. The sample was constructed by picking 10% of the subjects and including all their outgoing edges. The sample covers two-thirds of the predicates of the original dataset. The average forward matrix has 29,844 non-empty rows (median of 66). The node with the most incoming/outgoing links (840 in total) is an article on *insertional mutation in embryonic stem cells of mice*. The average node has 2 neighbors.

LDBC: The LDBC datasets [46] represent our first synthetic datasets. The qualifier 100, 500, 1000, and 5000 represent the input parameter to the social network generator. Subsequently, we report on the numbers *maximum degree*/*average degree*/*average number of non-empty rows*:

100: 100 / 1.0900 / 4793.5000

500: 244 / 1.1500 / 20,758

1000: 602 / 1.1700 / 96,090.4000

5000: 2529 / 1.1900 / 362,246.9000

LUBM: The *Lehigh University Benchmark* [61] dataset is our extreme case, which we generated for 10,000 universities. More than one billion edges distribute over only 18 predicates. The average number of node neighbors is 1.2580, while the maximum number of neighbors is 12. An average forward adjacency matrix has 58,934,718.1700 (median of 19,795,772) non-empty rows.

A.3 Queries

We employed three interpretations of the queries in our evaluations. As an example, consider $\mathcal{Q} = (\mathtt{x}, \mathit{occupation}, \mathtt{Lawyer})$ `OPT` $(\mathtt{x}, \mathit{familyname}, \mathtt{Obama})$.

Original: The *original interpretation* keeps the query as it is.

Variable: The *variable interpretation* replaces constants in queries by variables, consistently. Hence, the variable version of $\mathcal{Q}$ is $(\mathtt{x}, \mathit{occupation}, \mathtt{y})$ `OPT` $(\mathtt{x}, \mathit{familyname}, \mathtt{z})$. If $\mathcal{Q}$ is a query, then we denote its variable version by $\mathcal{Q}^v$, $\mathcal{D}_1$ is the first DBpedia (original) query and $\mathcal{D}_1^v$ its variable interpretation.

BGP: The *BGP interpretation* of queries ignores all nesting of optional operators and just regards the triple patterns. For $\mathcal{Q}$, this means we consider the BGP

$$(\mathtt{x}, \mathit{occupation}, \mathtt{Lawyer})\ \mathtt{AND}\ (\mathtt{x}, \mathit{familyname}, \mathtt{Obama}),$$

or $\mathbb{G} = \{(\mathtt{x}, \mathit{occupation}, \mathtt{Lawyer}), (\mathtt{x}, \mathit{familyname}, \mathtt{Obama})\}$, equivalently.

Here, we report on the original query sets, from which variable as well BGP forms are automatically derived by flag options `-all-variable` and `-bgp` within our tool SPARQLSIM. If a SPARQL query $\mathcal{Q}$ does not contain a constant (i. e., node of the database), then its variable interpretation is identical to the original. Furthermore, if $\mathcal{Q}$ is a BGP, its BGP interpretation is the same as the original one.

All our queries are published in our GitHub repository and can be found in the folder `queries/`. We report on the querys' characteristics in Table A.2. The first query in the table refers to the first line in the respective repository file. Each query gets an identifier ($\mathcal{Q}$). Furthermore, we list for $\mathcal{Q}$ the number of variables ($|vars|$), the number of used labels ($|\Sigma|$), the number of triple patterns ($|t|$), the number of optional operators ($|$ `OPT` $|$), the query's optional depth (OD), and whether or not it is a well-designed query (WD). The optional depth is the maximal number of nested optional clauses, e. g., P_1 `OPT` P_2 `OPT` P_3 has an optional depth of 1, while query P_1 `OPT` (P_2 `OPT` P_3) has 2.

Our first set of queries comprises 6 queries, that have also been used by Medha Atre [15], and is referred to by `dbpedia.original.sparql`. The 19 DBpedia benchmark queries `dbpedia.benchmark.sparql` are drawn from the set Morsey et al. [99] used as showcases for their DBSP (DBpedia SPARQL Benchmark) methodology. Thereby, we excluded repetitions from our first set of queries. Our Wikidata queries (cf. `wikidata.sparql`) are a subset from the Wikidata SPARQL logs [87], Interval 1 (2017-06-12 till 2017-07-09)[2]. We randomly picked queries with 0 up to 10 optional operators. For each configuration of optional operators, we were looking for 20 different queries. Unfortunately, we could not find 20 feasible queries for all configurations. However, this procedure already returned 85 queries. After the queries have been cleaned to match our fragment $\mathbb{S}_{\mathsf{AO}}$, we checked whether we would find any match in our 10% sample of Wikidata (cf. Appendix A.2). Only 11 of the 85 queries matched because the constants asked for belonged to the sample. Therefore, we changed the interpretation of all the other queries to *variable*, i. e., we treated every constant as if it was a variable. After that, 22 queries for Wikidata were left. 13 of those queries consisted of a single triple pattern, which we also had to remove. Finally, we had 9 of formerly 85 queries, presented as $\mathcal{W}_1$–$\mathcal{W}_9$ in Table A.2. Therefore, we repeated the procedure on the same interval for queries with at least two triple patterns. After applying the same procedure, as we did for the first Wikidata query set, to 217 queries, we obtained 52 additional queries ($\mathcal{W}_{10}$–$\mathcal{W}_{61}$). The LDBC queries (`ldbc.sparql`) stem from an LDBC GitHub repository[3], from which we included the 7 queries from the `interactive-short` query set. Our LUBM queries (`lubm.sparql`) are again those, used by Medha Atre [15]. Additionally, we used the 14 benchmark queries, the LUBM Website[4] lists as benchmark queries (`lubm.benchmark.sparql`).

A.4 Evaluation Results

In this section, we present the result tables that we interpret in Sections 4.4.3, 5.2.6 and 5.3.4. Thereby, Tables A.4 to A.6 belong to our evaluation of effectiveness in Section 4.4.3. Table A.7 list the runtime results of different dual simulation algorithms computing the maximal dual simulation (cf. Section 5.2.6). Finally, Table A.8 shows our results for the efficiency evaluation of Section 5.3.4.

[2] `https://iccl.inf.tu-dresden.de/web/Wikidata_SPARQL_Logs/en`

[3] `https://github.com/ldbc/ldbc_snb_implementations/tree/master/sparql/queries`

[4] `http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt`

Table A.2: The Queries

#	$\mathcal{Q}$	$\lvert vars\rvert$	$\lvert\Sigma\rvert$	$\lvert t\rvert$	$\lvert$ OPT $\rvert$	OD	#	$\mathcal{Q}$	$\lvert vars\rvert$	$\lvert\Sigma\rvert$	$\lvert t\rvert$	$\lvert$ OPT $\rvert$	OD	#	$\mathcal{Q}$	$\lvert vars\rvert$	$\lvert\Sigma\rvert$	$\lvert t\rvert$	$\lvert$ OPT $\rvert$	OD
dbpedia.original.sparql																				
1	$\mathcal{D}_1$	14	9	9	4	1	2	$\mathcal{D}_2$	9	7	7	1	1	3	$\mathcal{D}_3$	7	5	5	1	1
4	$\mathcal{D}_4$	10	6	7	2	1	5	$\mathcal{D}_5$	5	3	3	1	1	6	$\mathcal{D}_6$	21	12	12	8	1
dbpedia.benchmark.sparql																				
1	$\mathcal{D}_1^B$	7	5	5	1	1	2	$\mathcal{D}_2^B$	3	2	2	0	0	3	$\mathcal{D}_3^B$	3	2	2	0	0
4	$\mathcal{D}_4^B$	5	4	4	0	0	5	$\mathcal{D}_5^B$	6	5	5	0	0	6	$\mathcal{D}_6^B$	4	3	3	0	0
7	$\mathcal{D}_7^B$	3	2	2	0	0	8	$\mathcal{D}_8^B$	3	2	2	0	0	9	$\mathcal{D}_9^B$	5	3	4	0	0
10	$\mathcal{D}_{10}^B$	3	2	2	0	0	11	$\mathcal{D}_{11}^B$	6	3	3	2	1	12	$\mathcal{D}_{12}^B$	4	2	2	1	1
13	$\mathcal{D}_{13}^B$	4	2	2	1	1	14	$\mathcal{D}_{14}^B$	5	3	4	0	0	15	$\mathcal{D}_{15}^B$	3	2	2	0	0
16	$\mathcal{D}_{16}^B$	11	6	6	4	1	17	$\mathcal{D}_{17}^B$	3	2	2	0	0	18	$\mathcal{D}_{18}^B$	5	3	4	0	0
19	$\mathcal{D}_{19}^B$	6	5	5	0	0														
wikidata.sparql																				
1	$\mathcal{W}_1$	3	2	2	0	0	2	$\mathcal{W}_2$	4	2	2	1	1	3	$\mathcal{W}_3$	9	5	5	3	1
4	$\mathcal{W}_4$	14	7	9	4	1	5	$\mathcal{W}_5$	10	5	5	4	1	6	$\mathcal{W}_6$	12	6	6	5	1
7	$\mathcal{W}_7$	15	8	8	6	1	8	$\mathcal{W}_8$	16	8	8	7	1	9	$\mathcal{W}_9$	26	15	15	10	1
10	$\mathcal{W}_{10}$	4	2	2	1	1	11	$\mathcal{W}_{11}$	4	2	2	1	1	12	$\mathcal{W}_{12}$	4	2	2	1	1
13	$\mathcal{W}_{13}$	4	2	2	1	1	14	$\mathcal{W}_{14}$	4	2	2	1	1	15	$\mathcal{W}_{15}$	4	2	2	1	1
16	$\mathcal{W}_{16}$	4	2	2	1	1	17	$\mathcal{W}_{17}$	4	2	2	1	1	18	$\mathcal{W}_{18}$	4	2	2	1	1
19	$\mathcal{W}_{19}$	4	2	2	1	1	20	$\mathcal{W}_{20}$	4	2	2	1	1	21	$\mathcal{W}_{21}$	4	2	2	1	1
22	$\mathcal{W}_{22}$	4	2	2	1	1	23	$\mathcal{W}_{23}$	4	2	2	1	1	24	$\mathcal{W}_{24}$	4	2	2	1	1
25	$\mathcal{W}_{25}$	4	2	2	1	1	26	$\mathcal{W}_{26}$	4	2	2	1	1	27	$\mathcal{W}_{27}$	4	2	2	1	1
28	$\mathcal{W}_{28}$	4	2	2	1	1	29	$\mathcal{W}_{29}$	4	2	2	1	1	30	$\mathcal{W}_{30}$	4	2	2	1	1
31	$\mathcal{W}_{31}$	4	2	2	1	1	32	$\mathcal{W}_{32}$	4	2	2	1	1	33	$\mathcal{W}_{33}$	4	2	2	1	1
34	$\mathcal{W}_{34}$	4	2	2	1	1	35	$\mathcal{W}_{35}$	4	2	2	1	1	36	$\mathcal{W}_{36}$	3	2	2	0	0

#	$\mathcal{Q}$	$\lvert vars\rvert$	$\lvert\Sigma\rvert$	$\lvert t\rvert$	$\lvert$ OPT $\rvert$	OD	#	$\mathcal{Q}$	$\lvert vars\rvert$	$\lvert\Sigma\rvert$	$\lvert t\rvert$	$\lvert$ OPT $\rvert$	OD	#	$\mathcal{Q}$	$\lvert vars\rvert$	$\lvert\Sigma\rvert$	$\lvert t\rvert$	$\lvert$ OPT $\rvert$	OD
37	$\mathcal{W}_{37}$	4	2	2	1	1	38	$\mathcal{W}_{38}$	4	2	2	1	1	39	$\mathcal{W}_{39}$	4	2	2	1	1
40	$\mathcal{W}_{40}$	4	2	2	1	1	41	$\mathcal{W}_{41}$	4	2	2	1	1	42	$\mathcal{W}_{42}$	3	2	2	0	0
43	$\mathcal{W}_{43}$	5	2	3	1	1	44	$\mathcal{W}_{44}$	3	2	2	0	0	45	$\mathcal{W}_{45}$	3	2	2	0	0
46	$\mathcal{W}_{46}$	4	3	3	0	0	47	$\mathcal{W}_{47}$	4	2	2	1	1	48	$\mathcal{W}_{48}$	3	2	2	0	0
49	$\mathcal{W}_{49}$	6	3	3	2	1	50	$\mathcal{W}_{50}$	3	2	2	0	0	51	$\mathcal{W}_{51}$	6	3	3	2	1
52	$\mathcal{W}_{52}$	5	3	3	1	1	53	$\mathcal{W}_{53}$	6	3	3	2	1	54	$\mathcal{W}_{54}$	4	3	3	0	0
55	$\mathcal{W}_{55}$	3	2	2	0	1	56	$\mathcal{W}_{56}$	5	3	4	0	0	57	$\mathcal{W}_{57}$	6	3	3	2	1
58	$\mathcal{W}_{58}$	5	3	3	1	1	59	$\mathcal{W}_{59}$	5	3	3	1	1	60	$\mathcal{W}_{60}$	4	3	3	0	0
61	$\mathcal{W}_{61}$	8	5	5	2	1														
ldbc.sparql																				
1	$\mathcal{C}1$	10	9	9	0	0	2	$\mathcal{C}2$	11	9	10	0	0	3	$\mathcal{C}3$	7	6	6	0	0
4	$\mathcal{C}4$	5	4	4	0	0	5	$\mathcal{C}5$	5	4	4	0	0	6	$\mathcal{C}6$	9	8	8	0	0
7	$\mathcal{C}7$	9	7	8	0	0														
lubm.sparql																				
1	$\mathcal{L}_1$	9	6	7	2	1	2	$\mathcal{L}_2$	15	11	13	3	1	3	$\mathcal{L}_3$	15	10	13	3	1
4	$\mathcal{L}_4$	6	5	5	1	1	5	$\mathcal{L}_5$	6	5	5	1	1	6	$\mathcal{L}_6$	7	5	5	1	1
lubm.benchmark.sparql																				
1	$\mathcal{L}_1^B$	3	2	2	0	0	2	$\mathcal{L}_2^B$	6	4	6	0	0	3	$\mathcal{L}_3^B$	3	2	2	0	0
4	$\mathcal{L}_4^B$	6	5	5	0	0	5	$\mathcal{L}_5^B$	3	2	2	0	0	6	$\mathcal{L}_6^B$	2	1	1	0	0
7	$\mathcal{L}_7^B$	5	3	4	0	0	8	$\mathcal{L}_8^B$	6	4	5	0	0	9	$\mathcal{L}_9^B$	6	4	6	0	0
10	$\mathcal{L}_{10}^B$	3	2	2	0	0	11	$\mathcal{L}_{11}^B$	3	2	2	0	0	12	$\mathcal{L}_{12}^B$	5	3	4	0	0
13	$\mathcal{L}_{13}^B$	2	1	1	0	0														

Table A.4: Effectiveness Evaluation on LUBM and DBpedia

Query	Result No.	Gold	Base	SPARQLSIM	Query	Result No.	Gold	Base	SPARQLSIM
LUBM Dataset					DBpedia Dataset				
$\mathcal{L}_1$	10,448,905	3,276,841	505,801,654	10,181,730	$\mathcal{D}_1$	523,066	3,139,273	91,456,630	3,141,102
$\mathcal{L}_2$	226,641	114,989	757,247,049	25,429,750	$\mathcal{D}_2$	0	0	68,555,187	0
$\mathcal{L}_3$	32,828,280	15,416,012	683,467,844	50,237,812	$\mathcal{D}_3$	12	60	82,491,252	60
$\mathcal{L}_4$	11	35	587,712,255	126	$\mathcal{D}_4$	5794	28,704	81,004,705	28,704
$\mathcal{L}_5$	10	33	587,712,255	101	$\mathcal{D}_5$	25,102,459	22,630,477	37,296,555	22,691,521
$\mathcal{L}_6$	7	35	674,284,193	35	$\mathcal{D}_6$	365,693	79,943	101,844,668	79,944
$\mathcal{L}_1^v$	10,448,905	3,276,841	505,801,654	10,181,730	$\mathcal{D}_1^v$	10,159,863	14,565,628	91,456,630	14,591,682
$\mathcal{L}_2^v$	226,641	114,991	757,247,049	26,112,456	$\mathcal{D}_2^v$	0	0	68,555,187	0
$\mathcal{L}_3^v$	110,521,609	26,915,573	683,467,844	242,767,878	$\mathcal{D}_3^v$	37,453	50,003	82,491,252	50,003
$\mathcal{L}_4^v$	7,788,533	22,276,006	587,712,255	152,664,613	$\mathcal{D}_4^v$	1,165,353	150,227	81,004,705	150,284
					$\mathcal{D}_5^v$	25,102,459	22,630,477	37,296,555	22,691,521
$\mathcal{L}_6^v$	7,199,781	35,850,845	674,284,193	35,998,905	$\mathcal{D}_6^v$	365,693	79,943	101,844,668	79,944
DBpedia Dataset (Benchmark Queries)									
$\mathcal{D}_1^B$	12	60	82,491,252	60	$\mathcal{D}_2^B$	859,751	726,749	5,362,530	726,812
$\mathcal{D}_3^B$	913,786	1,587,731	5,362,530	1,588,127	$\mathcal{D}_4^B$	438,542	386,000	81,176,300	386,020
$\mathcal{D}_5^B$	0	0	81,176,302	0	$\mathcal{D}_6^B$	0	0	67,165,169	0
$\mathcal{D}_7^B$	815,522	886,826	66,505,605	886,939	$\mathcal{D}_8^B$	34,991	37,965	66,422,294	37,965
$\mathcal{D}_9^B$	8416	30,258	80,259,440	30,258	$\mathcal{D}_{10}^B$	8247	13,116	406,598	13,116
$\mathcal{D}_{11}^B$	8061	12,642	742,007	12,642	$\mathcal{D}_{12}^B$	9849	8955	16,165	8955
$\mathcal{D}_{13}^B$	9554	8660	16,165	8660	$\mathcal{D}_{14}^B$	123,467	365,131	80,274,588	365,154
$\mathcal{D}_{15}^B$	22,673,220	27,652,055	37,296,555	27,747,192	$\mathcal{D}_{16}^B$	2	4	18,005,367	4
$\mathcal{D}_{17}^B$	7,898,331	8,285,964	67,144,769	8,294,385	$\mathcal{D}_{18}^B$	66,903	41,808	66,461,056	41,808
$\mathcal{D}_{19}^B$	879,460	292,531	74,149,022	292,541					

Table A.5: Effectiveness Results on the LDBC Datasets

	100		500		1000		5000	
Query	Base	SPARQLSIM	Base	SPARQLSIM	Base	SPARQLSIM	Base	SPARQLSIM
$\mathcal{C}_1$	85,776	792	400,498	4014	889,398	8127	6,799,411	40,635
$\mathcal{C}_2$	107,398	0	451,120	0	989,787	0	7,587,208	0
$\mathcal{C}_3$	50,489	756	214,871	8318	479,215	21,941	3,906,783	198,501
$\mathcal{C}_4$	57,829	0	254,867	0	563,510	0	4,410,731	0
$\mathcal{C}_5$	46,033	19,530	165,450	135,412	347,982	313,344	2,428,553	2,357,826
$\mathcal{C}_6$	85,333	2732	272,192	35,242	556,019	93,136	3,755,637	892,254
$\mathcal{C}_7$	62,387	9706	317,376	136,420	726,750	356,716	5,987,946	3,458,852

Table A.6: Effectiveness on the Wikidata Sample

Query	Base	sparqlSim	Query	Base	sparqlSim	Query	Base	sparqlSim	Query	Base	sparqlSim
$\mathcal{W}_1$	30,701,618	3,810,056	$\mathcal{W}_2$	271,993	13,432	$\mathcal{W}_3$	7,058,477	57,040	$\mathcal{W}_4$	7,326,621	3457
$\mathcal{W}_5$	5,002,684	4,785,432	$\mathcal{W}_6$	4,787,804	4,764,043	$\mathcal{W}_7$	5,841,508	189,705	$\mathcal{W}_8$	6,702,321	4,956,191
$\mathcal{W}_9$	6,127,069	6	$\mathcal{W}_{10}$	2	1	$\mathcal{W}_{11}$	5	4	$\mathcal{W}_{12}$	7	6
$\mathcal{W}_{13}$	9	1	$\mathcal{W}_{14}$	10	9	$\mathcal{W}_{15}$	10	7	$\mathcal{W}_{16}$	11	8
$\mathcal{W}_{17}$	12	9	$\mathcal{W}_{18}$	12	12	$\mathcal{W}_{19}$	12	11	$\mathcal{W}_{20}$	21	8
$\mathcal{W}_{21}$	21	19	$\mathcal{W}_{22}$	24	23	$\mathcal{W}_{23}$	25	24	$\mathcal{W}_{24}$	26	17
$\mathcal{W}_{25}$	29	3	$\mathcal{W}_{26}$	38	37	$\mathcal{W}_{27}$	41	41	$\mathcal{W}_{28}$	46	46
$\mathcal{W}_{29}$	47	46	$\mathcal{W}_{30}$	54	37	$\mathcal{W}_{31}$	58	57	$\mathcal{W}_{32}$	68	68
$\mathcal{W}_{33}$	70	67	$\mathcal{W}_{34}$	73	71	$\mathcal{W}_{35}$	135	126	$\mathcal{W}_{36}$	84,620	8913
$\mathcal{W}_{37}$	84,620	48,368	$\mathcal{W}_{38}$	259,503	308	$\mathcal{W}_{39}$	271,993	13,432	$\mathcal{W}_{40}$	280,557	280,554
$\mathcal{W}_{41}$	280,557	280,554	$\mathcal{W}_{42}$	504,172	50,361	$\mathcal{W}_{43}$	1,313,454	674,683	$\mathcal{W}_{44}$	4,803,540	8645
$\mathcal{W}_{45}$	4,805,362	9056	$\mathcal{W}_{46}$	4,813,606	78	$\mathcal{W}_{47}$	4,818,549	4,767,157	$\mathcal{W}_{48}$	4,851,711	17,711
$\mathcal{W}_{49}$	4,863,584	4,771,917	$\mathcal{W}_{50}$	4,900,823	27,479	$\mathcal{W}_{51}$	4,962,167	812	$\mathcal{W}_{52}$	5,062,736	8773
$\mathcal{W}_{53}$	5,252,988	4,810,560	$\mathcal{W}_{54}$	5,264,685	7500	$\mathcal{W}_{55}$	5,406,767	131,460	$\mathcal{W}_{56}$	5,407,223	0
$\mathcal{W}_{57}$	5,754,187	4,856,051	$\mathcal{W}_{58}$	6,008,218	190,146	$\mathcal{W}_{59}$	6,417,127	194,913	$\mathcal{W}_{60}$	30,731,883	4660
$\mathcal{W}_{61}$	31,065,525	5169									

Table A.7: Dual Simulations: Comparison to State-of-the-Art on LUBM and DBpedia

$\mathcal{Q}$	Naive	HHK	sparqlSim	$\mathcal{Q}$	Naive	HHK	sparqlSim	$\mathcal{Q}$	Naive	HHK	sparqlSim
LUBM Dataset											
$\mathcal{L}_1$	1326.4800	—	9.6985	$\mathcal{L}_2$	580.2890	—	5.3251	$\mathcal{L}_3$	1264.6000	—	35.3777
$\mathcal{L}_4$	442.1960	—	1.5902	$\mathcal{L}_5$	402.0810	—	1.5767	$\mathcal{L}_6$	786.8520	—	0.8215
$\mathcal{L}_1^v$	1370.6600	—	9.6837	$\mathcal{L}_2^v$	2539.8700	—	7.8567	$\mathcal{L}_3^v$	1488.7700	—	34.9070
$\mathcal{L}_4^v$	680.5930	—	20.1893	$\mathcal{L}_6^v$	888.6590	—	9.6929	$\mathcal{L}_1^B$	676.5460	—	0.1604
$\mathcal{L}_2^B$	302.5910	—	1.3451	$\mathcal{L}_3^B$	830.9850	—	0.4802	$\mathcal{L}_4^B$	71.5194	—	0.6089
$\mathcal{L}_5^B$	648.2550	—	0.0123	$\mathcal{L}_6^B$	1373.2200	—	0.0063	$\mathcal{L}_7^B$	609.5130	—	1.0110
$\mathcal{L}_8^B$	655.9270	—	0.1016	$\mathcal{L}_9^B$	248.6370	—	5.7442	$\mathcal{L}_{10}^B$	639.1510	—	0.0124
$\mathcal{L}_{11}^B$	28.8619	—	0.0308	$\mathcal{L}_{12}^B$	40.5184	—	0.0735	$\mathcal{L}_{13}^B$	1124.5400	—	0.4453
DBpedia Dataset											
$\mathcal{D}_1$	131.8180	1210.1300	0.1645	$\mathcal{D}_2$	0.0491	4.9914	0.0023	$\mathcal{D}_3$	87.8867	510.3250	0.0368
$\mathcal{D}_4$	82.7732	2468.6000	0.0374	$\mathcal{D}_5$	99.8869	133.0480	1.1803	$\mathcal{D}_6$	0.0006	11.3141	0.0021
$\mathcal{D}_1^v$	133.8560	1688.6300	0.6205	$\mathcal{D}_2^v$	0.0495	5.6256	0.0023	$\mathcal{D}_3^v$	88.4062	788.5090	0.4993
$\mathcal{D}_4^v$	81.2308	3563.2400	0.0311	$\mathcal{D}_5^v$	97.5897	135.1350	1.1722	$\mathcal{D}_6^v$	0.0007	11.2269	0.0020
$\mathcal{D}_1^B$	88.9172	501.3390	0.0369	$\mathcal{D}_2^B$	25.8251	135.9880	0.0176	$\mathcal{D}_3^B$	27.2936	94.0591	0.1354
$\mathcal{D}_4^B$	66.0301	1135.2500	0.5018	$\mathcal{D}_5^B$	0.0000	2.9788	0.0000	$\mathcal{D}_6^B$	0.2047	2211.1400	0.0106
$\mathcal{D}_7^B$	0.0006	25.8374	0.0022	$\mathcal{D}_8^B$	2.5369	267.7570	0.4502	$\mathcal{D}_9^B$	1.9976	580.7460	0.4733
$\mathcal{D}_{10}^B$	77.9261	8258.7900	0.0521	$\mathcal{D}_{11}^B$	0.7098	9.6570	0.0141	$\mathcal{D}_{12}^B$	3.8993	25.6195	0.0159
$\mathcal{D}_{13}^B$	0.0709	6.6031	0.0105	$\mathcal{D}_{14}^B$	151.6360	392.4540	0.1020	$\mathcal{D}_{15}^B$	142.4860	208.7410	1.5962
$\mathcal{D}_{16}^B$	0.0000	1.1351	0.0000	$\mathcal{D}_{17}^B$	10.2253	131.5840	0.4946	$\mathcal{D}_{18}^B$	2.1530	1587.6800	0.4849
$\mathcal{D}_{19}^B$	34.7248	486.9790	0.5007								

Table A.8: Efficiency on LUBM and DBpedia

$\mathcal{Q}$	$T_{\text{SPARQLSIM}}$	$T(DB)$	$T(DB_{prune})$	$\sum$	$T(DB)$	$T(DB_{prune})$	$\sum$
		Virtuoso			RDFox		
$\mathcal{L}_1$	15.6839	5.1261	2.2609	17.9448	19.1000	1.4010	17.0849
$\mathcal{L}_2$	5.2648	50.8528	0.9709	6.2357	25,900.0000	888.0000	893.2648
$\mathcal{L}_3$	49.8689	56.6760	26.7672	76.6361	161.0000	15.6900	65.5589
$\mathcal{L}_4$	2.4795	0.0007	0.0001	2.4796	0.0000	0.0000	2.4795
$\mathcal{L}_5$	1.8955	0.0003	0.0001	1.8956	0.0000	0.0000	1.8955
$\mathcal{L}_6$	1.4653	0.0003	0.0001	1.4654	0.0000	0.0000	1.4653
$\mathcal{L}_1^v$	15.6582	5.1261	2.2609	17.9191	19.1000	1.4010	17.0592
$\mathcal{L}_2^v$	10.8789	50.8528	0.9709	11.8498	25,900.0000	888.0000	898.8789
$\mathcal{L}_3^v$	50.4400	105.7132	76.4734	126.9134	240.4000	157.6000	208.0400
$\mathcal{L}_4^v$	30.7849	55.0371	49.6343	80.4192	59.4000	38.6700	69.4549
$\mathcal{L}_6^v$	14.1073	5.3266	3.9156	18.0229	12.7000	7.5740	21.6813
$\mathcal{D}_1$	0.6910	0.3945	0.3590	1.0500	1.4000	1.1150	1.8060
$\mathcal{D}_2$	0.0036	0.0005	0.0000	0.0036	0.0000	0.0000	0.0036
$\mathcal{D}_3$	0.0523	0.0019	0.0003	0.0526	1.1000	0.0030	0.0553
$\mathcal{D}_4$	0.0688	0.0104	0.0033	0.0721	0.6200	0.0020	0.0708
$\mathcal{D}_5$	2.2756	2.1480	4.0081	6.2837	5.9600	3.4930	5.7686
$\mathcal{D}_6$	0.5392	0.0393	0.0210	0.5602	3.2300	0.0160	0.5552
$\mathcal{D}_1^v$	1.5046	3.4059	3.8493	5.3539	49.5000	5.6930	7.1976
$\mathcal{D}_2^v$	0.0034	0.0017	0.0000	0.0034	0.0000	0.0000	0.0034
$\mathcal{D}_3^v$	0.1725	0.0172	0.0049	0.1774	4.4000	0.0060	0.1785
$\mathcal{D}_4^v$	1.3464	0.0608	0.0341	1.3805	30.3000	0.1780	1.5244
$\mathcal{D}_5^v$	1.5604	2.1399	3.6889	5.2493	5.9000	4.3480	5.9084
$\mathcal{D}_6^v$	0.5376	0.0364	0.0211	0.5587	3.3000	0.0160	0.5536
$\mathcal{D}_1^B$	0.0625	0.0024	0.0000	0.0625	1.4680	0.0000	0.0625
$\mathcal{D}_2^B$	0.0267	0.0426	0.0086	0.0353	0.0990	0.0300	0.0567
$\mathcal{D}_3^B$	0.1617	0.0687	0.0637	0.2254	0.3480	0.1100	0.2717
$\mathcal{D}_4^B$	0.9985	0.0450	0.0127	1.0112	0.1040	0.0120	1.0105
$\mathcal{D}_5^B$	0.0000	0.0001	0.0000	0.0000	0.0330	0.0000	0.0000
$\mathcal{D}_6^B$	0.0182	0.0001	0.0001	0.0183	0.0000	0.0000	0.0182
$\mathcal{D}_7^B$	0.6252	0.0222	0.0134	0.6386	12.8300	0.0420	0.6672
$\mathcal{D}_8^B$	0.7034	0.0034	0.0011	0.7045	14.4100	0.0020	0.7054
$\mathcal{D}_9^B$	0.1437	0.0208	0.0053	0.1490	0.7930	0.0010	0.1447
$\mathcal{D}_{10}^B$	0.0730	0.0025	0.0010	0.0740	0.1170	0.0010	0.0740
$\mathcal{D}_{11}^B$	0.0210	0.0033	0.0028	0.0238	0.0040	0.0010	0.0220
$\mathcal{D}_{12}^B$	0.0257	0.0011	0.0017	0.0274	0.0010	0.0000	0.0257
$\mathcal{D}_{13}^B$	0.0200	0.0011	0.0018	0.0218	0.0010	0.0010	0.0210
$\mathcal{D}_{14}^B$	0.0196	0.0541	0.0308	0.0504	0.6430	0.0220	0.0416
$\mathcal{D}_{15}^B$	0.1334	1.0824	0.4405	0.5739	3.2820	1.9980	2.1314
$\mathcal{D}_{16}^B$	1.6231	0.0000	0.0002	1.6233	0.0000	0.0000	1.6231
$\mathcal{D}_{17}^B$	0.0138	0.1212	0.0989	0.1127	0.7580	0.3100	0.3238
$\mathcal{D}_{18}^B$	0.8411	0.0116	0.0033	0.8444	0.1190	0.0010	0.8421
$\mathcal{D}_{19}^B$	0.6780	0.1019	0.0556	0.7336	18.7500	0.0480	0.7260

Bibliography

[1] S. Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, Reading, Mass, 1995.

[2] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the 6th International Conference on Database Theory*, ICDT '97, pages 1–18, London, UK, UK, 1997. Springer-Verlag.

[3] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[4] Sergio Abriola, Pablo Barceió, Diego Figueira, and Santiago Figueira. Bisimulations on data graphs. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'16, pages 309–318. AAAI Press, 2016.

[5] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017.

[6] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of modern graph query languages. *CoRR*, abs/1610.06264, 2016.

[7] Renzo Angles, Marcelo Arenas, Pablo Barcelo, Peter Boncz, George Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan Sequeda, Oskar van Rest, and Hannes Voigt. G-core: A core for future graph query languages. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1421–1432, New York, NY, USA, 2018. ACM.

[8] Renzo Angles and Claudio Gutierrez. Querying rdf data from a graph database perspective. In *ESWC 2005*, LNCS, pages 346–360, Berlin, Heidelberg, 2005. Springer.

[9] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, 2008.

[10] Renzo Angles and Claudio Gutierrez. An introduction to graph data management. *arXiv:1801.00036 [cs]*, 2017.

[11] Renzo Angles and Claudio Gutierrez. An introduction to graph data management. In George Fletcher, Jan Hidders, and Josep Lluís Larriba-Pey, editors, *Graph Data Management: Fundamental Issues and Recent Developments*, Data-Centric Systems and Applications, pages 1–32, Cham, 2018. Springer International Publishing.

[12] Marcelo Arenas, Claudio Gutierrez, Daniel P. Miranker, Jorge Pérez, and Juan F. Sequeda. Querying semantic data on the web? *SIGMOD Rec.*, 41(4):6–17, 2013.

[13] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. Foundations of rdf databases. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems: 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, Lecture Notes in Computer Science, pages 158–204, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[14] 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*, PODS '11, pages 305–316, New York, NY, USA, 2011. ACM.

[15] Medha Atre. Left bit right: For sparql join queries with optional patterns (left-outer-joins). In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1793–1808, New York, NY, USA, 2015. ACM.

[16] Medha Atre, Vineet Chaoji, Mohammed J. Zaki, and James A. Hendler. Matrix "bit" loaded: A scalable lightweight join query processor for rdf data. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 41–50, New York, NY, USA, 2010. ACM.

[17] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[18] José Balcázar, Joaquim Gabarró, and Miklós Sántha. Deciding bisimilarity is p-complete. *Formal Aspects of Computing*, 4(1):638–648, 1992.

[19] Pablo Barceló Baeza. Querying graph databases. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '13, pages 175–188, New York, NY, USA, 2013. ACM.

[20] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 171–177. Springer Berlin Heidelberg, 2011.

[21] Catriel Beeri and Tova Milo. Schemas for integration and translation of structured and semi-structured data. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Catriel Beeri, and Peter Buneman, editors, *Database Theory - ICDT'99*, volume 1540, pages 296–313, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[22] Bard Bloom and Robert Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24(3):189–220, 1995.

[23] Angela Bonifati, George Fletcher, Hannes Voigt, Nikolay Yakovets, and H. V. Jagadish. *Querying Graphs*. Morgan & Claypool Publishers, 2018.

[24] Dan Brickley and R.V. Guha. Rdf schema 1.1. W3c recommendation, 2014.

[25] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[26] Joel Brynielsson, Johanna Högberg, Lisa Kaati, Christian Mårtenson, and Pontus Svenson. Detecting social positions using simulation. In *2010 International Conference on Advances in Social Networks Analysis and Mining*, pages 48–55, 2010.

[27] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '09, pages 233–244, New York, NY, USA, 2009. ACM.

[28] Peter Buneman. Semistructured data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '97, pages 117–121, New York, NY, USA, 1997. ACM.

[29] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Database Theory - ICDT '97*, Lecture Notes in Computer Science, pages 336–350. Springer Berlin Heidelberg, 1997.

[30] Peter Buneman and Sławek Staworko. Rdf graph alignment with bisimulation. *Proc. VLDB Endow.*, 9(12):1149–1160, 2016.

[31] Pieter Cailliau, Tim Davis, Vijay Gadepally, Jeremy Kepner, Roi Lipman, Jeffrey Lovitz, and Keren Ouaknine. Redisgraph graphblas enabled graph database. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 285–286, 2019.

[32] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Extending semi-structured data. In *SEBD*, 1998.

[33] Marco A. Casanova. Keyword search over rdf datasets. In *Conceptual Modeling*, Lecture Notes in Computer Science, pages 7–10, Cham, 2019. Springer International Publishing.

[34] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM.

[35] Qun Chen, Andrew Lim, and Kian Win Ong. D(k)-index: An adaptive structural summary for graph-structured data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 134–144, New York, NY, USA, 2003. ACM.

[36] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[37] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[38] Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax. W3c recommendation, 2014.

[39] Timothy A Davis. Algorithm 9xx: Suitesparse:graphblas: graph algorithms in the language of sparse linear algebra. 1(1):24.

[40] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 337–340. Springer Berlin Heidelberg, 2008.

[41] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, pages 601–610, New York, New York, USA, 2014. ACM Press.

[42] Xin Luna Dong. Building a broad knowledge graph for products. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 25–25, 2019.

[43] A. Dovier and C. Piazza. The subgraph bisimulation problem. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):1055–1056, 2003.

[44] M. Duerst and M. Suignard. Internationalized resource identifiers (iris). RFC 3987, 2005.

[45] Ramez Elmasri and Sham Navathe. *Fundamentals of database systems.* Pearson, Hoboken, NJ, seventh edition edition, 2016.

[46] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. The ldbc social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 619–630, New York, NY, USA, 2015. ACM.

[47] Orri Erling and Ivan Mikhailov. Rdf support in the virtuoso dbms. In *Networked Knowledge - Networked Media: Integrating Knowledge Management, New Media Technologies and Semantic Systems*, pages 7–24, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[48] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 8–21, New York, NY, USA, 2012. ACM.

[49] Wenfei Fan, Zhe Fan, Chao Tian, and Xin Luna Dong. Keys for graphs. *Proc. VLDB Endow.*, 8(12):1590–1601, 2015.

[50] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB Endow.*, 3(1-2):264–275, 2010.

[51] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. Graph pattern matching: from intractable to polynomial time. *Proceedings of the VLDB Endowment*, 3(1-2):264–275, 2010.

[52] Wenfei Fan, Jianzhong Li, Shuai Ma, Hongzhi Wang, and Yinghui Wu. Graph homomorphism revisited for graph matching. *Proc. VLDB Endow.*, 3(1-2):1161–1172, 2010.

[53] Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. Query preserving graph compression. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 157–168, New York, NY, USA, 2012. ACM.

[54] Wenfei Fan and Ping Lu. Dependencies for graphs. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems - PODS '17*, pages 403–416, Chicago, Illinois, USA, 2017. ACM Press.

[55] Wenfei Fan, Xin Wang, Yinghui Wu, and Dong Deng. Distributed graph simulation: Impossibility and possibility. *Proc. VLDB Endow.*, 7(12):1083–1094, 2014.

[56] Wenfei Fan, Wenyuan Yu, Jingbo Xu, Jingren Zhou, Xiaojian Luo, Qiang Yin, Ping Lu, Yang Cao, and Ruiqi Xu. Parallelizing sequential graph computations. *ACM Trans. Database Syst.*, 43(4):18:1–18:39, 2018.

[57] Mary F. Fernández and Dan Suciu. Optimizing regular path expressions using graph schemas. In *ICDE*, 1997.

[58] Nadime Francis, Alastair Green, Paolo Guargliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Martin Schuster, Petra Selmer, and Andrés Taylor. Formal semantics of the language cypher. *CoRR*, abs/1802.09984, 2018.

[59] Brian Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. In *Papers from the AAAI FS '06*, pages 45–53, 2006.

[60] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. Simulation as coarsest partition problem. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '02, pages 415–430, London, UK, UK, 2002. Springer-Verlag.

[61] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158 – 182, 2005.

[62] Claudio Gutierrez, Carlos A. Hurtado, Alberto O. Mendelzon, and Jorge Pérez. Foundations of semantic web databases. *Journal of Computer and System Sciences*, 77(3):520–541, 2011.

[63] Claudio Gutiérrez, Jan Hidders, and Peter T. Wood. Graph data models. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*, pages 830–835, Cham, 2019. Springer International Publishing.

[64] Abdelmalek Habi, Brice Effantin, and Hamamache Kheddouci. Fast top-k search with relaxed graph simulation. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 495–502, 2018.

[65] Steve Harris and Andy Seaborne. Sparql 1.1 query language. W3c recommendation, 2013.

[66] Jonathan Hayes and Claudio Gutierrez. Bipartite graphs as intermediate model for rdf. In *The Semantic Web - ISWC 2004*, Lecture Notes in Computer Science, pages 47–61. Springer Berlin Heidelberg, 2004.

[67] Patrick J. Hayes and Peter F. Patel-Schneider. Rdf 1.1 semantics. W3c recommendation, 2014.

[68] Pavol Hell and Jaroslav Nešetřil. On the complexity of h-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92 – 110, 1990.

[69] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 453–462, 1995.

[70] Ali Ismayilov, Dimitris Kontokostas, Sören Auer, Jens Lehmann, and Sebastian Hellmann. Wikidata through the eyes of dbpedia. *Semantic Web*, 9(4):493–503, 2018.

[71] Mark Kaminski and Egor V. Kostylev. Beyond well-designed sparql. In *19th International Conference on Database Theory (ICDT 2016)*, volume 48 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:18, Dagstuhl, Germany, 2016. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[72] Mark Kaminski and Egor V. Kostylev. Complexity and expressive power of weakly well-designed sparql. *Theory of Computing Systems*, 62(4):772–809, 2018.

[73] Mark Kaminski and Egor V. Kostylev. Subsumption of weakly well-designed sparql patterns is undecidable. *arXiv:1901.09353 [cs]*, 2019.

[74] Nils Karlund and Fred B. Scheider. Verifying safety properties using non-deterministic infinite-state automata. Technical report, Cornell University, NY, USA, 1989.

[75] Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, and Henry F Korth. Covering indexes for branching path queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 133–144, New York, NY, USA, 2002. ACM.

[76] Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, and Ehud Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 129–140, Washington, DC, USA, 2002. IEEE Computer Society.

[77] Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin Heidelberg, 2008.

[78] K. G. Larsen and B. Thomsen. A modal process logic. In *[1988] Proceedings. Third Annual Symposium on Logic in Computer Science*, pages 203–210, 1988.

[79] Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. On modal refinement and consistency. In *Proceedings of the 18th International Conference on Concurrency Theory*, CONCUR'07, pages 105–119, Berlin, Heidelberg, 2007. Springer-Verlag.

[80] Kim G. Larsen, Ulrik Nyman, and Andrzej Wąsowski. On modal refinement and consistency. In Luís Caires and Vasco T. Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory*, volume 4703, pages 105–119, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[81] Kim Guldstrand Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science, pages 232–246. Springer Berlin Heidelberg, 1990.

[82] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Soren Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. page 29.

[83] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *J. ACM*, 63(2):14:1–14:53, 2016.

[84] N. Lynch and F. Vaandrager. Forward and backward simulations. *Information and Computation*, 121(2):214–233, 1995.

[85] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. Capturing topology in graph pattern matching. *Proc. VLDB Endow.*, 5(4):310–321, 2011.

[86] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. Strong simulation: Capturing topology in graph pattern matching. *ACM Trans. Database Syst.*, 39(1):4:1–4:46, 2014.

[87] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In *The Semantic Web - ISWC 2018*, Lecture Notes in Computer Science, pages 376–394, Cham, 2018. Springer International Publishing.

[88] József Marton, Gábor Szárnyas, and Dániel Varró. Formalising opencypher graph queries in relational algebra. In *Advances in Databases and Information Systems*, pages 182–196, Cham, 2017. Springer.

[89] Stephan Mennicke. Modal schema graphs for graph databases. In *Conceptual Modeling*, Lecture Notes in Computer Science, pages 498–512, Cham, 2019. Springer International Publishing.

[90] Stephan Mennicke, Jan-Christoph Kalo, and Wolf-0Tilo Balke. Querying graph databases: What do graph patterns mean? In *Conceptual Modeling: 36th International Conference, ER 2017, Valencia, Spain, November 6-9, 2017, Proceedings*, pages 134–148, Cham, 2017. Springer International Publishing.

[91] Stephan Mennicke, Jan-Christoph Kalo, and Wolf-Tilo Balke. Using queries as schema-templates for graph databases. *Datenbank-Spektrum*, 18(2):89–98, 2018.

[92] Stephan Mennicke, Jan-Christoph Kalo, Denis Nagel, Hermann Kroll, and Wolf-Tilo Balke. Fast dual simulation processing of graph database queries (supplement). *arXiv:1810.09355 [cs]*, 2018.

[93] Stephan Mennicke, Jan-Christoph Kalo, Denis Nagel, Hermann Kroll, and Wolf-Tilo Balke. Fast dual simulation processing of graph database queries. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 244–255, 2019.

[94] Stephan Mennicke, Denis Nagel, Jan-Christoph Kalo, Niklas Aumann, and Wolf-Tilo Balke. Reconstructing graph pattern matches using sparql. In *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Rostock, Germany, September 11-13, 2017.*, page 152, 2017.

[95] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[96] Robin Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 481–489, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[97] Robin Milner. *Communicating and Mobile Systems: The &pgr;-calculus*. Cambridge University Press, New York, NY, USA, 1999.

[98] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 277–295, London, UK, UK, 1999. Springer-Verlag.

[99] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. Dbpedia sparql benchmark-performance assessment with real queries on real data. In *ISWC 2011*, 2011.

[100] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. Exemplar queries: A new way of searching. *The VLDB Journal*, 25(6):741–765, 2016.

[101] L. De Nardo, F. Ranzato, and F. Tapparo. The subgraph similarity problem. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):748–749, 2009.

[102] Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. Rdfox: A highly-scalable rdf store. In *The Semantic Web - ISWC 2015*, Lecture Notes in Computer Science, pages 3–20, Cham, 2015. Springer International Publishing.

[103] Svetlozar Nestorov, Serge Abiteboul, and Rajeev Motwani. Extracting schema from semistructured data. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 295–306, New York, NY, USA, 1998. ACM.

[104] Thomas Neumann and Gerhard Weikum. Rdf-3x: A risc-style engine for rdf. *Proc. VLDB Endow.*, 1(1):647–659, 2008.

[105] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[106] Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.

[107] David Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, Lecture Notes in Computer Science, pages 167–183. Springer Berlin Heidelberg, 1981.

[108] François Picalausa, George H. L. Fletcher, Jan Hidders, and Stijn Vansummeren. Principles of guarded structural indexing. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 245–256, 2014.

[109] François Picalausa, Yongming Luo, George H. L. Fletcher, Jan Hidders, and Stijn Vansummeren. A structural approach to indexing triples. In *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications*, ESWC'12, pages 406–421, Berlin, Heidelberg, 2012. Springer-Verlag.

[110] Artem Polyvyanyy, Jan Sürmeli, and Matthias Weidlich. Interleaving isotactics - an equivalence notion on behaviour abstractions. *Theoretical Computer Science*, 737:1–18, 2018.

[111] Artem Polyvyanyy, Matthias Weidlich, and Mathias Weske. Isotactics as a foundation for alignment and abstraction of behavioral models. In *Business Process Management*, pages 335–351, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[112] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *The Semantic Web - ISWC 2006*, pages 30–43, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[113] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.

[114] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. Technical report, W3C, 2008.

[115] Prakash Ramanan. Efficient algorithms for minimizing tree pattern queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 299–309, New York, NY, USA, 2002. ACM.

[116] Prakash Ramanan. Covering indexes for xml queries: Bisimulation - simulation = negation. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 165–176. VLDB Endowment, 2003.

[117] F. Ranzato and F. Tapparo. A new efficient simulation equivalence algorithm. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 171–180, 2007.

[118] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing. *Proc. VLDB Endow.*, 11(4):420–431, 2017.

[119] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4):15:1–15:41, 2009.

[120] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[121] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of sparql query optimization. In *Proceedings of the 13th International Conference on Database Theory*, ICDT '10, pages 4–33, New York, NY, USA, 2010. ACM.

[122] Guus Schreiber and Yves Raimond. Rdf 1.1 primer. Technical report, W3C, 2014.

[123] Alexander Schätzle, Antony Neu, Georg Lausen, and Martin Przyjaciel-Zablocki. Large-scale bisimulation of rdf graphs. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, SWIM '13, pages 1:1–1:8, New York, NY, USA, 2013. ACM.

[124] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[125] Q. Song, Y. Wu, P. Lin, L. X. Dong, and H. Sun. Mining summaries for knowledge graph search. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1887–1900, 2018.

[126] Colin Stirling. The joys of bisimulation. In *Mathematical Foundations of Computer Science 1998*, Lecture Notes in Computer Science, pages 142–151. Springer Berlin Heidelberg, 1998.

[127] Keishi Tajima. Schemaless semistructured data revisited. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Val Tannen, Limsoon Wong, Leonid Libkin, Wenfei Fan, Wang-Chiew Tan, and Michael Fourman, editors, *In Search of Elegance in the Theory and Practice of Computation*, volume 8000, pages 466–482, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[128] Li Tan and Rance Cleaveland. Simulation revisited. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 480–495. Springer Berlin Heidelberg, 2001.

[129] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55(11):1801–1809, 1967.

[130] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies*. W. H. Freeman & Co., New York, NY, USA, 1990.

[131] Johan Van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic: Volume II: Extensions of Classical Logic*, Synthese Library, pages 167–247, Dordrecht, 1984. Springer Netherlands.

[132] Rob van Glabbeek and Bas Ploeger. Correcting a space-efficient simulation algorithm. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 517–529. Springer Berlin Heidelberg, 2008.

[133] Rob J. van Glabbeek. The linear time - branching time spectrum. In *CONCUR 1990*, pages 278–297, Berlin, Heidelberg, 1990. Springer.

[134] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 137–146, New York, NY, USA, 1982. ACM.

[135] Miao Xie, Sourav S. Bhowmick, Gao Cong, and Qing Wang. Panda: toward partial topology-based search on large networks in a single machine. *The VLDB Journal*, 26(2):203–228, 2017.