Bisimulations on Data Graphs

Sergio Abriola, Pablo Barceló, Diego Figueira, Santiago Figueira

Bisimulations on Data Graphs

Sergio Abriola SABRIOLA@DC.UBA.AR
Universidad de Buenos Aires & ICC-CONICET

Pablo Barceló PBARCELO@DCC.UCHILE.CL
Center for Semantic Web Research &
DCC, University of Chile

Diego Figueira DIEGO.FIGUEIRA@LABRI.FR
CNRS, LaBRI, France

Santiago Figueira SANTIAGO@DC.UBA.AR
Universidad de Buenos Aires & ICC-CONICET

Abstract

Bisimulation provides structural conditions to characterize indistinguishability from an external observer between nodes on labeled graphs. It is a fundamental notion used in many areas, such as verification, graph-structured databases, and constraint satisfaction. However, several current applications use graphs where nodes also contain data (the so called "data graphs"), and where observers can test for equality or inequality of data values (e.g., asking the attribute 'name' of a node to be different from that of all its neighbors). The present work constitutes a first investigation of "data aware" bisimulations on data graphs. We study the problem of computing such bisimulations, based on the observational indistinguishability for XPath —a language that extends modal logics like PDL with tests for data equality— with and without transitive closure operators. We show that in general the problem is PSPACE-complete, but identify several restrictions that yield better complexity bounds (CO-NP, PTIME) by controlling suitable parameters of the problem, namely the amount of *non-locality* allowed, and the class of models considered (graphs, DAGs, trees). In particular this analysis yields a hierarchy of tractable fragments.

1. Introduction

1.1 The notion of bisimulation

Bisimulation is a fundamental notion that establishes when two nodes (or states) in a labeled graph (e.g., a transition system) cannot be distinguished by an external observer. It was independently discovered in the areas of computer science and philosophical logic during the 1970s —see the work of Sangiorgi, 2009 for a thorough historical revision of the notion of bisimulation. In both contexts, bisimulation (and its "half" version, *simulation*) appeared as a refinement of the notion of morphism, i.e, "structure-preserving" mappings. In the case of computer science, bisimulation was developed by Milner (and refined by Park) in the context of concurrency theory as a way to study the behavior of programs (Milner, 1971; Park, 1981). In philosophical logic, it was introduced by van Benthem in order to characterize the expressive power of the basic modal logic in terms of a fragment of first-order logic (van Benthem, 1976).

Nowadays, different fields of computer science apply domain-specific notions of (bi)simulation. For instance, (bi)simulation is used in concurrency to study behavioral equality for processes (Milner, 1999); in model checking to tackle the state-explosion problem (Clarke, Grumberg, & Peled, 2001); in databases as a method for indexing and compressing semi-structured data (Milo & Su-

ciu, 1999; Fan, Li, Wang, & Wu, 2012); in stochastic planning to solve Markov decision processes efficiently (Givan, Dean, & Greig, 2003); in description logics to understand the expressiveness of some languages (Kurtonina & de Rijke, 1999); in natural language generation to define semantic counterparts to the notion of referring expression (Areces, Figueira, & Gorín, 2011); and, with a data-aware definition of bisimulation, in the verification of data-aware processes as a way to obtain decidability under suitable semantic restrictions (Hariri, Calvanese, De Giacomo, Deutsch, & Montali, 2013a; Hariri, Calvanese, Montali, De Giacomo, De Masellis, & Felli, 2013b). Also, in constraint satisfaction the closely related notion of arc consistency is used as an approximation of satisfiability (Dechter, 1992, 2003) and as a method for finding tractable instances of SAT (Kolaitis & Vardi, 2000; Dalmau, Kolaitis, & Vardi, 2002).

Let us quickly recall the notion of bisimulation for the basic multi-modal logic, here called ML. An ML-structure is a tuple $\langle G, E, V\rangle$, where G is a set of nodes, $E \subseteq G \times \mathbb{A} \times G$ are directed edges labeled with a letter from a given fixed and finite alphabet $\mathbb{A}$, and V is a function mapping every node in G with a set of propositional symbols.

Let $\mathcal{G} = \langle G, E, V\rangle$ and $\mathcal{G}' = \langle G', E', V'\rangle$ be two ML-structures. A bisimulation between $\mathcal{G}$ and $\mathcal{G}'$ is a relation $Z \subseteq G \times G'$ such that for each pair $(u, u') \in Z$ we have:

- **(Atomic harmony)** $V(u) = V(u')$
- **(Zig)** if $\mathcal{G}$ contains an edge $u \xrightarrow{a} v$ (for $a \in \mathbb{A}$) then $\mathcal{G}'$ contains an edge $u' \xrightarrow{a} v'$ such that $(v, v') \in Z$.
- **(Zag)** if $\mathcal{G}'$ contains an edge $u' \xrightarrow{a} v'$ (for $a \in \mathbb{A}$) then $\mathcal{G}$ contains an edge $u \xrightarrow{a} v$ such that $(v, v') \in Z$.

The nodes u in $\mathcal{G}$ and u' in $\mathcal{G}'$ are *bisimilar* if there is a bisimulation between $\mathcal{G}$ and $\mathcal{G}'$ that contains the pair (u, u').

The following are two important properties of this notion:

- First, bisimulation can be restated in terms of greatest fixed points, which in turn yields a simple polynomial time algorithm for checking if u and u' are bisimilar (more specifically, for computing the *maximal* bisimulation between $\mathcal{G}$ and $\mathcal{G}'$).
- Second, the notion of bisimulation captures, in a precise sense, the expressiveness of ML on finite models. Formally, Hennesy-Milner's Theorem establishes that nodes u and u' are bisimilar if and only if they cannot be distinguished by ML formulas (see, e.g., Blackburn, de Rijke, & Venema, 2001). This result is robust, as it continues to hold if we replace ML by more expressive navigational logics used in the analysis of programs (e.g., PDL, Fischer & Ladner, 1979) and model checking (e.g., CTL*, Clarke et al., 2001).

1.2 Data-aware bisimulations

A distinguishing feature of bisimulations is that they are defined in terms of the *topology* of the graph structure only, i.e., the way in which nodes are linked by labeled edges. This is good enough for applications on which this topology is the only relevant feature in their model. However, it is not sufficient for other applications that impose higher demands on such models and query languages. We are thinking here, in particular, of "data-aware" models such as *data* or *property graphs*, which have become *de-facto* standard in the area of graph databases (see, e.g. Angles & Gutiérrez, 2008;

Libkin & Vrgoč, 2012; Robinson, Webber, & Eifrem, 2013), or XML documents such as *data trees* (Bojańczyk, Muscholl, Schwentick, & Segoufin, 2009). In addition to the topology defined by labeled edges, such graph-based models allow nodes to be *attributed*, i.e., to be associated with a set of property/value pairs. Moreover, languages over these models are endowed with the capability of testing for equality of such data values.

Example 1. Consider a data graph representation of a movie database, in which (a) nodes represent actors, directors, and movies, (b) edges establish relationships between such nodes, e.g. movie m is directed by director d and casts actor a, and (c) nodes contain attributes, such as the name of the actor and its age, or the title of the movie, its duration, and the company who produced it. □

An important feature of the query languages for data graphs is that they combine topology and data to express relevant properties. An example is the query which asks whether a director has two movies produced by different companies. This query cannot be expressed in a purely navigational language such as ML, PDL or CTL* (simply because they cannot compare the attribute values of two nodes), but can in turn be expressed in the "data-aware" language XPath$_=$ (Libkin, Martens, & Vrgoc, 2016). This language extends the navigational core of XPath with data comparison formulas of the form $\langle \alpha_1 = \alpha_2 \rangle$ and $\langle \alpha_1 \neq \alpha_2 \rangle$. Intuitively, when evaluated on a node u these formulas ask whether there are paths π_1 and π_2 starting in u such that π_i satisfies the condition given by path expression α_i (for $i = 1, 2$) and the final nodes of π_1 and π_2 have the same (resp., different) data value (we assume for the sake of simplicity that each node is attributed with a single data value, given by function $data$).

As it has been recently shown in the context of XML/data trees, the language XPath$_=$ allows for a Hennessy-Milner's style characterization in terms of a natural class of "data-aware" bisimulations (Figueira, Figueira, & Areces, 2015). We notice in this article that such characterization extends in a straightforward way to the class of data graphs. Let us explain intuitively how such "data-aware" bisimulation Z (called XPath$_=$-bisimulation in the paper) between data graphs $\mathcal{G} = \langle G, E, data \rangle$ and $\mathcal{G}' = \langle G', E', data' \rangle$ is defined. The **(Zig$_=$)** property establishes that for each pair (u, u') in Z and paths π_1 and π_2 in $\mathcal{G}$ starting from u, there must be paths π'_1 (having the same length as π_1) and π'_2 (having the same length as π_2) in $\mathcal{G}'$ starting from u' such that:

- **Topology-preserving property:** The sequence of labels of π_i and π'_i (for $i = 1, 2$) are the same and the jth node of π_i is in the Z-relation with the jth node of π'_i, for every j.
- **Data-awareness property:** If the data values of the final nodes of π_1 and π_2 are equal (resp., different), so is the case for the data values of the final nodes of π'_1 and π'_2.

This is graphically depicted in Figure 1.2. The **(Zag$_=$)** property is, of course, symmetric.

It is worth remarking here that, in general, languages for data graphs, such as XPath$_=$ and others —e.g., Bojańczyk et al., 2009; Figueira, 2010; Libkin & Vrgoč, 2012; David, Gheerbrant, Libkin, & Martens, 2013— allow to test for (in)equality of data values only, abstracting away the concrete data. This means, in particular, that such query languages cannot check if a node is attributed with a particular data value d. The reason is that meaningful properties of the graph topology are naturally closed under renaming of data values through bijections. While the use of constants may be essential for data retrieval, from an observational perspective the infinite domain of data values is merely a source of unique names to relate nodes. This is why we work with languages and bisimulation notions that are closed under bijections of data values and, therefore, domain-independent.

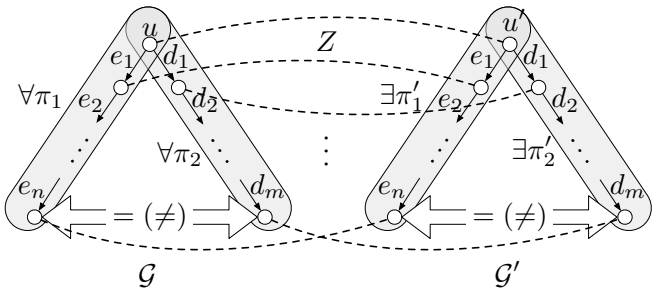

Figure 1: The **(Zig$_=$)** clause for XPath$_=$ on data graphs. In the picture, $(e_i)_{i\leq n}$ and $(d_j)_{j\leq m}$ are labels.

1.3 Potential applications

Data-aware bisimulations have been used to study the expressive power of XPath$_=$ on data trees (Figueira, Figueira, & Areces, 2014). We foresee several other potential applications of them when interpreted over data graphs:

- **Indexing:** Finding bisimilar nodes over graph-structured data is the first step in many approaches to building indexing data structures for efficient evaluation of navigational languages (Milo & Suciu, 1999; Fan et al., 2012). These approaches are based on the following idea: If x and y are bisimilar and x is in the output of a query, also y is in the output. Extending this to "data-aware" bisimulations might then serve as a building-block over which index structures for XPath$_=$ expressions can be constructed.

- **Clustering:** Another motivation stems from the task of *clustering* in graph data mining (Getoor & Diehl, 2005), i.e.,, the division of data into groups of similar objects. One way to define similarity on data graphs is based on *observational indistinguishability*, that is, grouping together elements x, y that cannot be distinguished with a data aware logic L: $x \equiv_L y$. For the logic XPath$_=$, this notion corresponds to "data-aware" bisimilarity. Further, in cases when the previous notion is too strict, it might prove useful to compute a *degree* of similarity, where more similar elements are elements that are distinguished through more complex formulas. This degree of similiarity can be defined, in turn, by restricting suitable parameters in the definition of "data-aware" bisimulations (e.g., the amount of *non-locality* allowed, as studied in this paper).

- **Referring expressions generation:** A basic and active task in natural language generation is *referring expressions generation* (REG); for a complete picture of the area, see the survey of Krahmer & Van Deemter, 2012. The problem of REG can be stated as follows: given a scene and a target element in that context, generate a grammatically correct expression, called *referring expression* (RE), in a given language that uniquely represents the element. It has been proposed by Krahmer, van Erk, & Verleg, 2003 to use labeled directed graphs for representing the scene, while the work of Areces, Koller, & Striegnitz, 2008 resorts to

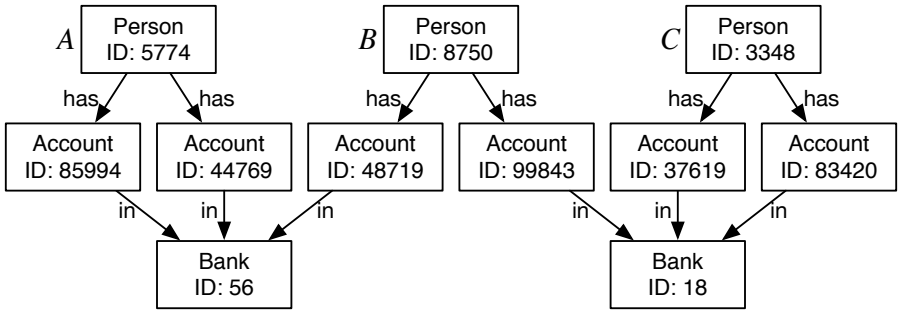

Figure 2: A scene with people, accounts and banks.

description logics (DLs) as a formalism for representing a RE. Finally, the work of Areces et al., 2011 shows that the latter approach can be efficiently implemented using bisimulations.

In some cases, though, a scene for the REG problem is better modeled as a data graph. Imagine, e.g., a scene modeling clients, accounts, and banks (Figure 2). Each object has an ID. Suppose we look for a RE for target B. It is impossible to distinguish nodes A and B using ML or the DLs used in previous works, since they are bisimilar (assuming, of course, that IDs are not part of the language). However, the RE for B "the person who has accounts in different banks" can be formalized in XPath$_=$ as follows:

$$\mathsf{Person} \wedge \langle \downarrow_{\mathsf{has}}[\mathsf{Account}]\downarrow_{\mathsf{in}}[\mathsf{Bank}] \neq \downarrow_{\mathsf{has}}[\mathsf{Account}]\downarrow_{\mathsf{in}}[\mathsf{Bank}] \rangle.$$

Thus, extending the work of Areces et al., 2011, "data-aware" bisimulations might then be an efficient tool for REG in cases when REs are expressed in the language of XPath$_=$.

1.4 Computing "data-aware" bisimulations

In any of the previous cases one is faced with the fundamental problem of determining whether two nodes are "data-aware" bisimilar (more in general, checking if there is a "data-aware" bisimulation relating two data graphs). Recall that this problem can be solved in PTIME for usual (i.e., purely topological) bisimulations. One of the reasons that explains this is that such bisimulations are *local*, in the sense that the **(Zig)** and **(Zag)** conditions for a pair (u, u') are defined in terms of nodes which are adjacent to u and u', resp. But this no longer holds for "data-aware" bisimulations, as the **(Zig$_=$)** and **(Zag$_=$)** conditions are defined in terms of arbitrarily long paths (i.e., in a *non-local* way). As it turns out, this makes the problem of computing "data-aware" bisimulations intractable.

1.5 Contributions

Our main contribution is an in-depth study of the complexity of computing "data-aware" bisimulations by fine-tuning on the level of *non-locality* allowed. This non-locality is measured in terms of (a) the lengths of the paths considered in the definition of bisimulation, and (b) the classes of models over which bisimulations are computed. In particular, we show the following:

- In full generality, checking whether two data graphs are "data-aware" bisimilar is PSPACE-complete. This is obtained by showing that the problem is polynomially equivalent to *equiva-*

lence of nondeterministic finite automata, which is PSPACE-complete (Meyer & Stockmeyer, 1972). In particular, there are cases in which the smallest witness (π_1, π_2) to the fact that two data graphs are not bisimilar is a pair of paths of exponential size.

- The previous observation naturally calls for a restriction on the length of paths to be inspected in the definition of "data-aware" bisimulation (restriction (a) above). We start by considering paths of polynomial length only. While this decreases the complexity of the problem to the class CO-NP, we show that it still does not yield tractability.

  We thus restrict to paths of constant length only and show that this condition does guarantee tractability. Interestingly, this restricted notion of bisimulation characterizes an important fragment of the XPath language; namely, the one of *bounded length data comparisons*. This fragment restricts the length of expressions α_1 and α_2 in formulas of the form $\langle \alpha_1 = \alpha_2 \rangle$ and $\langle \alpha_1 \neq \alpha_2 \rangle$ only —but does not restrict the navigational expressions of the form $\langle \alpha \rangle$, where $\langle \alpha \rangle$ is true if there is a path starting at the current node of evaluation satisfying α.

- We then study how the underlying graphs affect the complexity of the problem (restriction (b) above), and look at the two most important classes of acyclic graphs: trees and DAGs. We show that checking "data-aware" bisimilarity is tractable for the former and CO-NP-complete for the latter.

- Finally, we look at *two-way* XPath$_=$, which allows to traverse edges in both directions. The problem of checking "data aware" bisimilarity in this context remains PSPACE-complete. The upper bound follows easily, but the lower bound needs a new proof. As before, the restriction to paths of polynomial length yields a CO-NP bound, and for paths of constant length we obtain tractability. Interestingly, the simulation problem remains PSPACE-hard even over DAGs, establishing a contrast with the case when inverses are not allowed.

A preliminary version of the present paper appeared in the work of Abriola, Barceló, Figueira, & Figueira, 2016. This article extends that version as follows:

- We provide full proofs of all results. In particular, we explain in detail in the proof of Theorem 9 how the (bi)simulation problem for all the logics we consider can be reduced in polynomial time to automata containment (resp., equivalence). This yields a general PSPACE upper bound for the problem studied in the paper. It is worth noticing that such a reduction was only sketched in the proceedings of Abriola et al., 2016. While not technically difficult, we believe that it provides interesting insights into the nature of the problem.

- We provide a PSPACE lower bound for the complexity of (bi)simulation for two-way XPath$_=$ over DAGs. This problem had been left open in the work of Abriola et al., 2016. The proof is of independent interest, as it involves proving a PSPACE lower bound for the complexity of the containment problem for a restricted class of automata.

1.6 Related work

There are many different approaches to define logics that compare data values. Some of these formalisms include: first-order logic with two variables (FO^2) with access to an equivalence relation that represents equality of data values (Bojańczyk et al., 2009); extensions of modal temporal logics,

such as LTL, CTL or μ-calculus, with a data binding mechanism for storing and comparing values (Demri, Lazić, & Nowak, 2005; Jurdziński & Lazić, 2007; Kara, Schwentick, & Zeume, 2010); formalisms based on regular expressions or automata equipped with registers (Libkin et al., 2016; Demri & Lazić, 2009; Jurdziński & Lazić, 2011); and logics based on XPath (Libkin et al., 2016; Figueira, 2013, 2012), similar to the one we study in this work. All these different approaches are in general incomparable in terms of expressive power, and the corresponding bisimulation notions for such languages would therefore differ from ours. Some of these languages, e.g., FO^2, have a strong modal flavour, and thus they may relate to more traditional (i.e., local) notions of bisimulation. Others, on the other hand, such as register automata or modal logics with binding mechanisms, may need some extra structure in the definition of the bisimulation relation (Murawski, Ramsay, & Tzevelekos, 2015). One exception to this rule is the so-called *logic of repeating values* studied in the works of Demri, D'Souza, & Gascon, 2007; Demri, Figueira, & Praveen, 2016; Abriola, Figueira, & Figueira, 2017b, which can be seen as a common factor between the temporal approach and $\mathrm{XPath}_{=}$. This logic, however, does not feature the non-local flavour of $\mathrm{XPath}_{=}$ when testing for data values, and thus its corresponding bisimulation notion (which has not been studied as of now) is of different nature than ours.

Noticeably, our intractability results are in line with the intractability of other *non-local* notions of bisimulations, such as the *fair bisimulations* studied in verification (Kupferman & Vardi, 1998). An important point of departure, though, is that such notions are defined with respect to infinite paths in transition systems, while our notion considers finite paths only. Since our notion of bisimulation requires comparing the data values of the last nodes in two paths, it simply does not make sense to consider infinite paths.

Interestingly, different notions of of data-aware bisimulations have been recently introduced in the literature on verification of *data-centric dynamic systems* (Hariri et al., 2013a, 2013b; Belardinelli, Lomuscio, & Patrizi, 2014). While such notions are defined "locally" – in the same way than the usual notion of bisimulation – a "non-local" condition is externally imposed by requiring pairs in the bisimulation relation to be isomorphic with respect to a bijection between the data domains of the systems. It is difficult to study how such notions of bisimulation compare to ours, as they are defined over a different data model that is suitable for the problems at hand. However, it is clear that they allow to impose conditions that our notion cannot; e.g., that there is a path along which n different data values occur. On the other hand, they include no explicit way of comparing the data values of the last nodes of two paths, which is in turn allowed by our notion. Thus, it seems that these notions are incomparable to ours. To the best of our knowledge, there has been no analysis of the complexity of the notions of bisimulation used in the study data-centric dynamic systems.

Finally, the logics we study are related to description logics with concrete domains (Lutz, 2003; Lutz, Areces, Horrocks, & Sattler, 2005) —a family of modal logics designed for the representation of conceptual knowledge, equipped with means that allow to describe "concrete qualities" of real-world objects such as their weight, temperature, and spatial extension. One can draw a connection between, on the one hand, $\mathcal{ALC}$ when restricted to having binary predicates $=, \neq$ on a concrete domain, and, on the other hand, data-aware XPath without transitive axes on data graphs. The study of such logics has concentrated, however, on static analysis tasks such as satisfiability and subsumption, which have no direct relationship with bisimulations.

1.7 Organization of the paper

We present basic notions in Section 2. The XPath$_=$-bisimulations are introduced in Section 3. The complexity of XPath$_=$-bisimulations is studied in Section 4. Restrictions on paths are presented in Section 5, and those on data models in Section 6. The two-way version of XPath$_=$ is studied in Section 7. Finally, Section 8 is devoted to conclusions.

2. Data Graphs and XPath

As is customary in graph-structured data, we work with edge-labeled data graphs, i.e., finite graphs whose edges are labeled with an element of a finite alphabet $\mathbb{A}$ and whose nodes contain a single value of an infinite domain $\mathbb{D}$ (Libkin & Vrgoč, 2012; Barceló, 2013). Formally, a *data graph* $\mathcal{G}$ over $\mathbb{A}$ is a tuple $\langle G, E, data\rangle$, where G is a finite set of nodes, $E \subseteq G \times \mathbb{A} \times G$, and $data : G \to \mathbb{D}$ assigns values to nodes. Intuitively, an edge $(x, a, y) \in E$ (for x, y nodes in G and a a symbol in $\mathbb{A}$) represents that there is an a-labeled edge from x to y. Also, $data(x) = d$ iff the data value of node x is d. By convention, the set of nodes of a data graph $\mathcal{G}$ will be denoted by G, the set of nodes of a data graph $\mathcal{G}'$ by G', and so on. When E is clear from the context, we write $x \xrightarrow{a} y$ instead of $(x, a, y) \in E$.

We work with the language XPath$_=$, a simplification of XPath, stripped of its syntactic sugar, and adapted to reasoning with data values. While XPath$_=$ was initially designed for querying XML documents with data values, suitably represented as data trees (Figueira et al., 2015), it is natural to evaluate this language also over data graphs (see, e.g., Libkin, Martens, & Vrgoč, 2013). While some works have used a different terminology to refer to the versions of XPath that are evaluated over graphs (e.g., Libkin et al., 2013 calls such family of languages GXPath), we keep calling the language XPath$_=$ for the sake of simplicity (and as, after all, the syntax remains unchanged).

XPath$_=$ is a two-sorted language, with *path expressions* (denoted α, β, γ) representing binary relations on nodes, and *node expressions* (denoted φ, ψ, η) representing unary relations, or properties. Its syntax is defined by mutual recursion as follows:

$$\begin{aligned} \alpha, \beta &::= \varepsilon \mid \downarrow_a \mid \alpha\beta \mid \alpha \cup \beta \mid [\varphi] && (a \in \mathbb{A}) \\ \varphi, \psi &::= \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle\alpha\rangle \mid \langle\alpha = \beta\rangle \mid \langle\alpha \neq \beta\rangle. \end{aligned}$$

A node expression is *positive* if it contains no negation.

We formally define the semantics of XPath$_=$ in Figure 3. Intuitively, $\downarrow_a$ corresponds to going to a neighbor of the current node via an edge labeled with a, the formula $[\varphi]$ is used to check that φ holds in some node along a path, and $\langle\alpha = \beta\rangle$ checks that we have witnesses for the path expressions α and β such that their ending node have same data values (or different ones for $\langle\alpha \neq \beta\rangle$).

Example 2. In order to ease the understanding of the semantics of XPath$_=$, we present some examples of the interpretation of some path and node expressions of XPath$_=$. Recall that the semantics of a node expression correspond to a set of nodes, while the semantics for path expressions correspond to a set of pairs of nodes. Consider then the formulas

$$\alpha = \downarrow_a\downarrow_b \tag{1}$$
$$\beta = [\varphi] \tag{2}$$
$$\varphi = \langle\epsilon = \alpha\rangle \tag{3}$$

$$
\begin{aligned}
[\![\varepsilon]\!]^{\mathcal{G}} &= \{(x,x) \mid x \in G\} \\
[\![\downarrow_a]\!]^{\mathcal{G}} &= \{(x,y) \mid (x,a,y) \in E\} \\
[\![[\varphi]]\!]^{\mathcal{G}} &= \{(x,x) \mid x \in [\![\varphi]\!]^{\mathcal{G}}\} \\
[\![\alpha\beta]\!]^{\mathcal{G}} &= \{(x,z) \mid \exists y:\ (x,y) \in [\![\alpha]\!]^{\mathcal{G}}, (y,z) \in [\![\beta]\!]^{\mathcal{G}}\} \\
[\![\alpha \cup \beta]\!]^{\mathcal{G}} &= [\![\alpha]\!]^{\mathcal{G}} \cup [\![\beta]\!]^{\mathcal{G}} \\
[\![\langle\alpha\rangle]\!]^{\mathcal{G}} &= \{x \in G \mid \exists y:\ (x,y) \in [\![\alpha]\!]^{\mathcal{G}}\} \\
[\![\langle\alpha = \beta\rangle]\!]^{\mathcal{G}} &= \{x \in G \mid \exists y,z:\ (x,y) \in [\![\alpha]\!]^{\mathcal{G}}, (x,z) \in [\![\beta]\!]^{\mathcal{G}}, data(y) = data(z)\} \\
[\![\langle\alpha \neq \beta\rangle]\!]^{\mathcal{G}} &= \{x \in G \mid \exists y,z:\ (x,y) \in [\![\alpha]\!]^{\mathcal{G}}, (x,z) \in [\![\beta]\!]^{\mathcal{G}}, data(y) \neq data(z)\} \\
[\![\varphi \wedge \psi]\!]^{\mathcal{G}} &= [\![\varphi]\!]^{\mathcal{G}} \cap [\![\psi]\!]^{\mathcal{G}} \\
[\![\neg\varphi]\!]^{\mathcal{G}} &= G \setminus [\![\varphi]\!]^{\mathcal{G}}
\end{aligned}
$$

Figure 3: Semantics of XPath$_=$ for a data graph $\mathcal{G} = \langle G, E, data\rangle$.

$$\psi \;=\; \langle \alpha \neq \alpha \rangle \tag{4}$$

The intuitive interpretation of such formulas is as follows.

1. The pairs (u,v) of nodes such that v can be reached from u by first following an edge labeled a and then another one labeled b.
2. The pairs (u,u) such that u satisfies φ.
3. The nodes u such that from u there is a path to some node v for which it holds that (a) (u,v) satisfies α, and (b) u and v have the same data value.
4. The nodes u such that from u there are paths to nodes v and v' for which it holds that (a) both (u,v) and (u,v') satisfy α, and (b) v and v' have different data values. □

For a data graph $\mathcal{G}$ and $u \in G$, we write $\mathcal{G}, u \models \varphi$ to denote $u \in [\![\varphi]\!]^{\mathcal{G}}$, and we say that $\mathcal{G}, u$ *satisfies* φ.

Definition 3 (Indistinguishability)**.** We write $\mathcal{G}, u \equiv \mathcal{G}', u'$ if

$$\mathcal{G}, u \models \varphi \iff \mathcal{G}', u' \models \varphi, \quad \text{for every node expression } \varphi \text{ of XPath}_=.$$

Similarly, we write $\mathcal{G}, u \Rrightarrow \mathcal{G}', u'$ if

$$\mathcal{G}, u \models \varphi \implies \mathcal{G}', u' \models \varphi, \quad \text{for every positive node expression } \varphi \text{ of XPath}_=. \quad \square$$

As it is always the case, the notion of one-way indistinguishability ($\Rrightarrow$), which is not symmetric, applies only to formulas which are positive. If we considered formulas with negation in its definition, one would have that $\mathcal{G}', u' \models \varphi$ iff $\mathcal{G}', u' \not\models \neg\varphi$, and this would imply $\mathcal{G}, u \not\models \neg\varphi$, i.e., $\mathcal{G}, u \models \varphi$; henceforth, the notion of $\Rightarrow$ in such a case would coincide with that of $\equiv$.

In the next section we introduce a notion of (bi)simulation that characterizes, in a precise sense, logical *indistinguishability* (i.e., the relations $\equiv$ and $\Rrightarrow$) for XPath$_=$.

3. Bisimulations on Data Graphs

The notion of (bi)simulation for XPath$_=$ over *data trees* was developed in the work of Figueira et al., 2014 and later extended by Abriola, Descotte, & Figueira, 2014. As we observe next, this notion is robust and extends in a straightforward way to data graphs.[1]

Definition 4 (XPath$_=$-bisimulations)**.** Let $\mathcal{G}$ and $\mathcal{G}'$ be data graphs over $\mathbb{A}$. An XPath$_=$-*bisimulation* between $u \in G$ and $u' \in G'$ (written $\mathcal{G}, u \leftrightarrows \mathcal{G}', u'$) is a relation $Z \subseteq G \times G'$ such that uZu' and for all $(x, x') \in G \times G'$ such that xZx' we have:

- **(Zig$_=$)** For every pair of paths in $\mathcal{G}$ of the form

$$\pi_1 = x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \text{ and } \pi_2 = x \xrightarrow{d_1} y_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y_m \quad (\text{for } e_i, d_j \in \mathbb{A}),$$

  there are paths in $\mathcal{G}'$ of the form

$$\pi_1' = x' \xrightarrow{e_1} x_1' \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n' \text{ and } \pi_2' = x' \xrightarrow{d_1} y_1' \xrightarrow{d_2} \ldots \xrightarrow{d_m} y_m',$$

  such that the following holds:

  1. $x_i Z x_i'$ for all $1 \leq i \leq n$, and $y_j Z y_j'$ for all $1 \leq j \leq m$.
  2. $data(x_n) = data(y_m) \Leftrightarrow data(x_n') = data(y_m')$.

- **(Zag$_=$)** For every paths in $\mathcal{G}'$ of the form

$$\pi_1' = x' \xrightarrow{e_1} x_1' \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n' \text{ and } \pi_2' = x' \xrightarrow{d_1} y_1' \xrightarrow{d_2} \ldots \xrightarrow{d_m} y_m', \quad (\text{for } e_i, d_j \in \mathbb{A}),$$

  there are paths in $\mathcal{G}$ of the form

$$\pi_1 = x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \text{ and } \pi_2 = x \xrightarrow{d_1} y_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y_m$$

  such that conditions 1 and 2 above are verified.

Furthermore, an XPath$_=$-*simulation* from $u \in G$ to $u' \in G'$ (denoted $\mathcal{G}, u \rightrightarrows \mathcal{G}', u'$) is a relation $Z \subseteq G \times G'$ such that uZu' and for all $(x, x') \in G \times G'$ such that xZx' it is the case that condition **(Zig$_=$)** above is verified.

The nodes u in $\mathcal{G}$ and u' in $\mathcal{G}'$ are XPath$_=$*-(bi)similar* if there is an XPath$_=$-(bi)simulation between $\mathcal{G}, u$ and $\mathcal{G}', u'$. □

It is worth comparing our XPath$_=$-bisimulation with the classical bisimulation notion for ML. There are two simple ways of transforming a data graph into an ML-structure.

1. We erase the data values, obtaining a ML-structure with empty valuation for the propositional variables. Under this interpretation, it is clear that if two nodes are XPath$_=$-bisimilar then they are ML-bisimilar, while the converse implication is not true in general. This is explained in the following example.

1. Strictly speaking, the notions presented below for data graphs are slightly different from the ones presented in the work of Figueira et al., 2014 for data trees. This is because the data trees studied in such article are *node-labeled* while our data graphs are *edge-labeled*. The difference is inessential to the results, as it is easy to define (bi)simulation invariant translations from node-labeled data graphs to edge-labeled graphs, and viceversa.

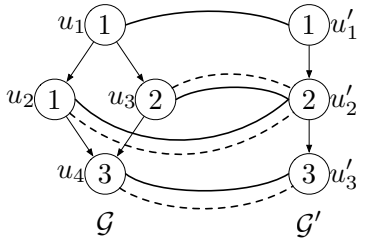

Figure 4: Two data graphs with same label in all edges (not shown). The dotted lines represent the maximal bisimulation for XPath$_=$. The solid lines represent an ML-bisimulation when data is absent.

Example 5. Consider the data graphs $\mathcal{G}$ and $\mathcal{G}'$ in Figure 4. It can be checked that $Z = \{(u_1, u'_1), (u_2, u'_2), (u_3, u'_2), (u_4, u'_3)\}$ (represented by the solid lines) is an ML-bisimulation on the ML-structures resulting from erasing the nodes' data. However u_1 and u'_1 are not bisimilar for XPath$_=$. Indeed, the paths $u_1 \to u_2$ and $u_1 \to u_3$, arriving at nodes with different data values, cannot be copied in $\mathcal{G}'$, as the rule **(Zig$_=$)** imposes. However one can check that $Z \setminus \{(u_1, u'_1)\}$ (represented by the dotted lines) is an XPath$_=$-bisimulation. □

2. We assign a propositional letter p_d to every node of the data graph with data value d in the data graph. In contrast with the previous case, now the existence of an ML-bisimulation over the ML-structure implies the existence of a data-aware bisimulation in the original data graph, but the converse does not hold. See Figure 4: u_2 and u'_2 are XPath$_=$-bisimilar but clearly not ML-bisimilar as they do not verify **(Atomic harmony)**. This is because XPath$_=$ (and, therefore, the data-aware bisimulations studied in the paper) cannot speak about a particular data value (it cannot express "data value d holds in the current node"), but can only check whether two paths finish in nodes with the same (or different) data value.

XPath$_=$-(bi)simulations are closed under union: if $Z_1 \subseteq G \times G'$ and $Z_2 \subseteq G \times G'$ are XPath$_=$-(bi)simulations between $u \in G$ and $u' \in G$, then so is $Z_1 \cup Z_2$. This immediately implies the following:

Proposition 6. *If there is an XPath$_=$-(bi)simulation between $u \in G$ and $u' \in G$, then there is a maximal such XPath$_=$-(bi)simulation.*

3.1 The characterization

One can verify that the following theorem, originally stated in terms of data trees, holds also in the general case of data graphs. It establishes the desired, Henessy-Milner-style characterization of the notion of logical indistinguishability for XPath$_=$ in terms of XPath$_=$-(bi)simulations.

Theorem 7. *Let $\mathcal{G}$ and $\mathcal{G}'$ be data graphs over the same alphabet $\mathbb{A}$, and u and u' nodes in G and G', respectively. Then $\mathcal{G}, u \equiv \mathcal{G}', u'$ iff $\mathcal{G}, u \leftrightarrows \mathcal{G}', u'$, and $\mathcal{G}, u \Rrightarrow \mathcal{G}', u'$ iff $\mathcal{G}, u \rightrightarrows \mathcal{G}', u'$.*

Proof. We only prove the theorem for the case of bisimulation, as for simulation the proof is analogous. The idea is to reduce the case of data graphs to data trees by unravelling, and then use the result of Figueira et al., 2015.

Let $\mathcal{G}$ be a data graph and u a node in G. We define $\widetilde{\mathcal{G}}_u$ as the data graph that corresponds to the *unraveling* of $\mathcal{G}$ from node u. The idea is that nodes of $\widetilde{\mathcal{G}}_u$ are finite sequences of nodes from G representing paths of $\mathcal{G}$ starting at u. Formally, the set of nodes of $\widetilde{\mathcal{G}}_u$ corresponds to the least subset $\widetilde{G}_u$ of G^* such that (i) $u \in \widetilde{G}_u$, and (ii) for any $\sigma \in G^*$ and $x, y \in G$ it is the case that

$$\sigma xy \in \widetilde{G}_u \iff \sigma x \in \widetilde{G}_u \text{ and } x \xrightarrow{a} y \text{ is an edge of } \mathcal{G}, \text{ for some label } a \in \mathbb{A}.$$

The set of edges of $\widetilde{\mathcal{G}}_u$ is defined as follows. For $x, y \in G$, $\sigma \in G^*$, and a label a we have

$$\sigma x \xrightarrow{a} \sigma xy \in \widetilde{\mathcal{G}}_u \iff \sigma x \in \widetilde{G}_u \text{ and } x \xrightarrow{a} y \in \mathcal{G}.$$

Finally, the data value of a node σx in $\widetilde{\mathcal{G}}_u$ is the data value of x in $\mathcal{G}$. Observe that the unraveling of a data graph is thus a possibly infinite (but finitely branching) data tree. The fact that the structure is infinite does not change at all the semantics of XPath$_=$ given in Figure 3.

It can be seen that $\mathcal{G}, u \leftrightarrows \widetilde{\mathcal{G}}_u, u$; in particular, that the relation that contains all pairs of the form $(x, \sigma x) \in G \times \widetilde{G}_u$ is an XPath$_=$ bisimulation between u in $\mathcal{G}$ and u in $\widetilde{\mathcal{G}}_u$. On the other hand, it can also be proved that $\mathcal{G}, u \equiv \widetilde{\mathcal{G}}_u, u$. In fact, one can show that $\mathcal{G}, u \models \varphi$ iff $\widetilde{\mathcal{G}}_u, \sigma u \models \varphi$ for every formula φ in XPath$_=$. The proof goes by induction on the structural complexity of φ. The key points are the following:

- Let $x_0 \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n$ be a path in $\mathcal{G}$ such that x_0 is reachable form u. Then from every $\sigma_0 \in \widetilde{G}_u$ whose last symbol is x_0 there is a path $\sigma_0 \xrightarrow{e_1} \sigma_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} \sigma_n$ in $\widetilde{\mathcal{G}}_u$ such that the data value of x_i coincides with that of σ_i, for each $0 \leq i \leq n$.
- Analogously, for any path $\sigma_0 \xrightarrow{e_1} \sigma_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} \sigma_n$ in $\tilde{\mathcal{G}}_u$ there is a path $x_0 \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n$ in $\mathcal{G}$, where x_0 (reachable from u) is the last symbol of σ_0, and such that the data value of x_i coincides with that of σ_i for each $0 \leq i \leq n$.

To obtain the desired result, we lift our data trees $\mathcal{G}, u$ and $\mathcal{G}', u'$ to their corresponding unravelings and then use Theorem 8 from the work of Figueira et al., 2015, which states the coincidence of logical equivalence and bisimulation over possibly infinite but finitely branching data trees. □

Example 8. (Example 5 continued). Consider the data graphs of Figure 4. These data graphs contain the same letter a in all its edges (not shown in the figure). In what follows we write $\downarrow$ instead of $\downarrow_a$. We have that $\mathcal{G}, u_1 \not\equiv \mathcal{G}', u'_1$ since $\mathcal{G}, u_1 \models \langle \downarrow \neq \downarrow \rangle$ but $\mathcal{G}', u'_1 \not\models \langle \downarrow \neq \downarrow \rangle$. Another distinguishing node expression could be $\langle \varepsilon = \downarrow \rangle$. Notice that u_2 and u'_2 cannot be distinguished in XPath$_=$ though they have different data values (as this cannot be expressed in the logic). □

It is worth remarking that adding the transitive reflexive closure $\downarrow_a^*$ of $\downarrow_a$ to XPath$_=$ does not change the notion of (bi)simulation, and, in particular, Theorem 7 continues to hold (when $\equiv$ and $\Rrightarrow$ are replaced with the corresponding indistinguishability notion in the extended language).

4. Computing XPath$_=$-(Bi)Simulations

As mentioned in the introduction, a fundamental problem when dealing with (bi)simulations is checking whether a pair of nodes is (bi)similar. In this section we study the complexity of such problem for XPath$_=$-(bi)simulations and show it to be PSPACE-complete. This establishes an important difference with the problem of computing bisimulations in the absence of data, which can be

solved in polynomial time, as it is essentially equal to computing bisimulations for the basic modal logic.

Formally, we study the following problems:

XPATH$_=$-(BI)SIMILARITY
INPUT : Data graphs $\mathcal{G}$ and $\mathcal{G}'$, nodes $u \in G$ and $u' \in G'$. QUESTION : $\mathcal{G}, u \rightrightarrows \mathcal{G}', u'$? ($\mathcal{G}, u \leftrightarrows \mathcal{G}', u'$?, resp.)

Our main result establishes the following:

Theorem 9. *The problems of* XPATH$_=$-BISIMILARITY *and* XPATH$_=$-SIMILARITY *are* PSPACE*-complete.*

From Theorem 7 we obtain:

Corollary 10. *The problem of checking $\mathcal{G}, u \equiv \mathcal{G}', u'$ or $\mathcal{G}, u \Rrightarrow \mathcal{G}', u'$, given data graphs $\mathcal{G}$ and $\mathcal{G}'$ and nodes $u \in G$ and $u' \in G'$, is* PSPACE*-complete.*

Furthermore, the proof of Theorem 9 will imply that the PSPACE lower bound is quite resilient, as it holds even when checking indistinguishability for the restricted class of formulas of the form $\langle \varepsilon = \downarrow_{e_1} \ldots \downarrow_{e_n} \rangle$, for $e_1 \ldots e_n \in \mathbb{A}$. These formulas simply check if, starting from a node u, it is possible to follow a path labeled $e_1 \ldots e_n$ and reach a node with the same data value as u. Formally, let us write $\mathcal{G}, u \equiv_{\text{paths}} \mathcal{G}', u'$ if $\mathcal{G}, u$ and $\mathcal{G}', u'$ are indistinguishable with respect to this class of formulas; that is, if for every $e_1, \ldots, e_n \in \mathbb{A}$ we have that

$$\mathcal{G}, u \models \langle \varepsilon = \downarrow_{e_1} \ldots \downarrow_{e_n} \rangle \;\Leftrightarrow\; \mathcal{G}', u' \models \langle \varepsilon = \downarrow_{e_1} \ldots \downarrow_{e_n} \rangle.$$

Analogously, we define $\mathcal{G}, u \Rrightarrow_{\text{paths}} \mathcal{G}', u'$. We then obtain the following corollary.

Corollary 11. *Checking $\mathcal{G}, u \equiv_{\text{paths}} \mathcal{G}', u'$ or $\mathcal{G}, u \Rrightarrow_{\text{paths}} \mathcal{G}', u'$ is* PSPACE*-complete.*

The proof of Theorem 9 shows that (essentially) XPATH$_=$-(BI)SIMILARITY is polynomially equivalent to the *containment problem* (resp., *equivalence problem*) for nondeterministic finite automata (NFAs). Although the proof is not very involved, both directions are non-trivial, and, in particular, the reduction from containment to bisimilarity requires a clever encoding. Below, we briefly recall basic notions on NFA and its containment and equivalence problems.

Basics on automata. Recall that an NFA over a finite alphabet $\mathbb{A}$ is given as a tuple $\mathcal{A} = (Q, q_0, F, \delta)$, where Q is a finite set of *states*, $q_0 \in Q$ is the *initial* state, $F \subseteq Q$ is the set of *final* states, and $\delta \subseteq Q \times \mathbb{A} \times Q$ is the *transition relation*. A word $w \in \mathbb{A}^*$ is *accepted* by $\mathcal{A}$ if there is an *accepting run* of $\mathcal{A}$ over w that respects the transition relation δ. Formally, if $w = a_1 \ldots a_n$, where each a_i is a symbol in $\mathbb{A}$, an *accepting run* of $\mathcal{A}$ over w is a sequence $q_0 \ldots q_n$ of states in Q such that (i) $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for each $0 \leq i < n$, and (ii) $q_n \in F$. (Notice that the first state of this accepting run corresponds to the initial state of $\mathcal{A}$). The set of words in $\mathbb{A}^*$ that are accepted by $\mathcal{A}$ is the *language* of $\mathcal{A}$, denoted with $L(\mathcal{A})$.

The containment problem for NFAs is defined as follows. Given NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ over $\mathbb{A}$, is $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$? Respectively, we define the equivalence problem for NFAs, but this time we ask whether $L(A_1) = L(A_2)$. Both containment and equivalence of NFAs are PSPACE-complete (Meyer & Stockmeyer, 1972). We prove Theorem 9 below by reductions to and from these problems.

4.1 Upper bound

We start by explaining how to obtain a PSPACE upper bound for XPATH$_=$-SIMILARITY. The algorithm receives a pair of graphs $\mathcal{G}, \mathcal{G}'$ and a pair of nodes u, u', it guesses a candidate relation $Z \subseteq G \times G'$ containing (u, u'), and then checks that Z satisfies the **(Zig$_=$)** property. Since Z is of polynomial size and PSPACE = NPSPACE from Savitch's Theorem (Savitch, 1970), i.e., PSPACE is closed under non-determinism, we only need to show that the latter can be checked in PSPACE. This is done by reducing the problem in polynomial time to containment of NFAs. Since NFA containment is in PSPACE the result follows (as PSPACE computable functions are closed under composition).

Let us explain now the reduction to containment of NFAs. Given a node $x \in G$, we construct (details below) in polynomial time an NFA $\mathcal{A}_{\mathcal{G},x}$ over the alphabet $\mathbb{A} \times G \cup \{=, \neq\}$ that accepts precisely the language L of words of the form:

$$(e_1, x_1) \dots (e_n, x_n) \star (f_1, y_1) \dots (f_m, y_m), \tag{5}$$

for $(e_i, x_i), (f_j, y_j) \in \mathbb{A} \times G$ and $\star \in \{=, \neq\}$, such that $\mathcal{G}$ contains paths:

$$x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} x_n \quad \text{and} \quad x \xrightarrow{f_1} y_1 \xrightarrow{f_2} \dots \xrightarrow{f_m} y_m$$

for which $data(x_n) \star data(y_m)$.

We also construct (details below) an NFA $\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}$ over the alphabet $\mathbb{A} \times G \cup \{=, \neq\}$ that accepts precisely the language L' of words of the form (5) such that $\mathcal{G}'$ contains paths:

$$x' \xrightarrow{e_1} x'_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} x'_n \quad \text{and} \quad x' \xrightarrow{d_1} y'_1 \xrightarrow{d_2} \dots \xrightarrow{d_m} y'_m,$$

and the following holds:

- $x_i Z x'_i$ for each $1 \leq i \leq n$, and $y_j Z y'_j$ for each $1 \leq j \leq m$.
- $data(x'_n) \star data(y'_m)$.

Finally, we verify (details below) that

$$Z \text{ satisfies } \textbf{(Zig}_=\textbf{)} \iff L(\mathcal{A}_{\mathcal{G},x}) \subseteq L(\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x}), \text{ for each } (x, x') \in Z. \tag{6}$$

Since Z is of polynomial size, Equation (6) tells us that in order to check whether **(Zig$_=$)** holds we only need to check containment for a polynomial number of pairs of NFAs. This establishes the upper bound since each such containment can be checked in PSPACE.

The proof for bisimilarity is analogous. In fact, Z satisfies **(Zig$_=$)** and **(Zag$_=$)** if and only if for each $(x, x') \in Z$ it is the case that

$$L(\mathcal{A}_{\mathcal{G},x}) \subseteq L(\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}) \text{ and } L(\mathcal{A}_{\mathcal{G}',x'}) \subseteq L(\mathcal{A}^{\mathcal{G}',Z^{-1}}_{\mathcal{G},x}).$$

This can clearly be checked in PSPACE.

In what follows, we give the definitions of $\mathcal{A}_{\mathcal{G},x}$ and $\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}$, and the verification that (6) holds.

Definition of $\mathcal{A}_{\mathcal{G},x}$. The set of states of $\mathcal{A}_{\mathcal{G},x}$ is defined as

$$\{y,\ y_d^{=},\ y_d^{\neq}\ \mid\ y \in G \text{ and } d \text{ is the data value of some node in } \mathcal{G}\}.$$

The initial state is x and the set of final states corresponds to:

$$\{y_d^{=}\ \mid\ data(y) = d\}\ \cup\ \{y_d^{\neq}\ \mid\ data(y) \neq d\}.$$

Finally, the transition relation of $\mathcal{A}_{\mathcal{G},x}$ corresponds to the union of the following sets:

1. $\{(y,\ (e,z)\ ,z)\ \mid\ (y \xrightarrow{e} z) \text{ is an edge in } \mathcal{G}\}$.
2. $\{(y_d^{=},\ (e,z)\ ,z_d^{=})\ \mid\ (y \xrightarrow{e} z) \text{ is an edge in } \mathcal{G} \text{ and } d \text{ is a data value in } \mathcal{G}\}$.
3. $\{(y_d^{\neq},\ (e,z)\ ,z_d^{\neq})\ \mid\ (y \xrightarrow{e} z) \text{ is an edge in } \mathcal{G} \text{ and } d \text{ is a data value in } \mathcal{G}\}$.
4. $\{(y,\ (=)\ ,x_d^{=})\ \mid\ data(y) = d\}$.
5. $\{(y,\ (\neq)\ ,x_d^{\neq})\ \mid\ data(y) = d\}$.

Clearly, $\mathcal{A}_{\mathcal{G},x}$ can be constructed in polynomial time from $\mathcal{G}$. We prove next that $L(\mathcal{A}_{\mathcal{G},x}) = L$, where L is the language defined above.

Notice first that any accepting run of $\mathcal{A}_{\mathcal{G},x}$ must be a sequence of states of the form

$$x\ x_1\ \ldots\ x_n\ x_d^{\star}\ (y_1)_d^{\star}\ \ldots\ (y_m)_d^{\star}.$$

By definition, the word w accepted by this run is of the form:

$$(e_1, x_1) \ldots (e_n, x_n) \star (f_1, y_1) \ldots (f_m, y_m),$$

for $e_i, f_j \in \mathbb{A}$ and $\star \in \{=, \neq\}$, and it is the case that $\mathcal{G}$ contains paths

$$x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \quad \text{and} \quad x \xrightarrow{f_1} y_1 \xrightarrow{f_2} \ldots \xrightarrow{f_m} y_m$$

such that $data(x_n) = d$ if and only if $\star$ corresponds to the symbol $=$. Since $(y_m)_d^{\star}$ is an accepting state, it must be the case then that

$$data(x_n) = data(y_m) \iff \star \text{ corresponds to the symbol } =.$$

We conclude that $data(x_n) \star data(y_m)$, and therefore that $w \in L$. This tells us that $L(\mathcal{A}_{\mathcal{G},x}) \subseteq L$ since w was chosen arbitrarily.

On the other hand, consider a word w in L of the form

$$(e_1, x_1) \ldots (e_n, x_n) \star (f_1, y_1) \ldots (f_m, y_m).$$

By definition, $\mathcal{G}$ contains paths:

$$x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \quad \text{and} \quad x \xrightarrow{f_1} y_1 \xrightarrow{f_2} \ldots \xrightarrow{f_m} y_m$$

for which $data(x_n) \star data(y_m)$. It is easy to see then that

$$x\ x_1\ \ldots\ x_n\ x_d^{\star}\ (y_1)_d^{\star}\ \ldots\ (y_m)_d^{\star},$$

for $d = data(x_n)$, is an accepting run of $\mathcal{A}_{\mathcal{G},x}$ over w. Thus, $w \in L(\mathcal{A}_{\mathcal{G},x})$, and, therefore, $L(\mathcal{A}_{\mathcal{G},x}) \subseteq L$ since w was chosen arbitrarily.

Definition of $\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}$. The set of states of $\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}$ is defined by:

$$\{(y,y'),\ (y,y')^{=}_{d},\ (y,y')^{\neq}_{d}\ \mid\ (y,y') \in G \times G' \text{ and } d \text{ is the data value of some node in } \mathcal{G}'\}.$$

The initial state is (x,x') and the set of final states corresponds to:

$$\{(y,y')^{=}_{d}\ \mid\ data(y') = d\}\ \cup\ \{(y,y')^{\neq}_{d}\ \mid\ data(y') \neq d\}.$$

Finally, the transition relation of $\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}$ corresponds to the union of the following sets:

1. $\{\big((y,y'),\ (e,z)\ ,(z,z')\big)\ \mid\ (y \xrightarrow{e} z)$ is an edge in $\mathcal{G}$, $(y' \xrightarrow{e} z')$ is an edge in $\mathcal{G}'$, and $zZz'\}$.
2. $\{\big((y,y')^{=}_{d},\ (e,z)\ ,(z,z')^{=}_{d}\big)\ \mid\ (y \xrightarrow{e} z)$ is an edge in $\mathcal{G}$, $(y' \xrightarrow{e} z')$ is an edge in $\mathcal{G}'$, zZz', and d is a data value in $\mathcal{G}'\}$.
3. $\{\big((y,y')^{\neq}_{d},\ (e,z)\ ,(z,z')^{\neq}_{d}\big)\ \mid\ (y \xrightarrow{e} z)$ is an edge in $\mathcal{G}$, $(y' \xrightarrow{e} z')$ is an edge in $\mathcal{G}'$, zZz', and d is a data value in $\mathcal{G}'\}$.
4. $\{\big((y,y'),\ (=)\ ,(x,x')^{=}_{d}\big)\ \mid\ data(y') = d\}$.
5. $\{\big((y,y'),\ (\neq)\ ,(x,x')^{\neq}_{d}\big)\ \mid\ data(y') = d\}$.

Clearly, $\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}$ can be constructed in polynomial time from $\mathcal{G}$. It is also quite easy to prove that $L(\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'}) = L'$

Verification. Note that Z satisfies **(Zig$_=$)** if and only if for every $(x,x') \in Z$ and paths in $\mathcal{G}$ of the form

$$\pi_1 = x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \ \text{ and } \ \pi_2 = x \xrightarrow{d_1} y_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y_m, \ \text{ where } e_i, d_j \in \mathbb{A},$$

there are paths in $\mathcal{G}'$ of the form

$$\pi'_1 = x' \xrightarrow{e_1} x'_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x'_n \ \text{ and } \ \pi'_2 = x' \xrightarrow{d_1} y'_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y'_m,$$

such that the following holds:

1. $x_i Z x'_i$ for all $1 \leq i \leq n$, and $y_j Z y'_j$ for all $1 \leq j \leq m$.
2. $data(x_n) = data(y_m) \Leftrightarrow data(x'_n) = data(y'_m)$.

In other words, Z satisfies **(Zig$_=$)** if and only if for every $(x,x') \in Z$ and word over $\mathbb{A} \times G \cup \{=, \neq\}$ of the form (5) such that $\mathcal{G}$ contains paths:

$$x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \quad \text{and} \quad x \xrightarrow{f_1} y_1 \xrightarrow{f_2} \ldots \xrightarrow{f_m} y_m$$

for which $data(x_n) \star data(y_m)$, it is the case that $\mathcal{G}'$ contains paths:

$$x' \xrightarrow{e_1} x'_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x'_n \quad \text{and} \quad x' \xrightarrow{d_1} y'_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y'_m,$$

for which the following holds:

- $x_i Z x'_i$ for each $1 \leq i \leq n$, and $y_j Z y'_j$ for each $1 \leq j \leq m$.
- $data(x'_n) \star data(y'_m)$.

That is, Z satisfies **(Zig$_=$)** if and only if for every $(x,x') \in Z$ it is the case that every word in $L(\mathcal{A}_{\mathcal{G},x})$ is also in $L(\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'})$, i.e., $L(\mathcal{A}_{\mathcal{G},x}) \subseteq L(\mathcal{A}^{\mathcal{G},Z}_{\mathcal{G}',x'})$.

4.2 Lower bound

We start by showing the lower bound for XPATH$_=$-SIMILARITY. We proceed by constructing a polynomial time reduction from containment of NFAs to XPATH$_=$-SIMILARITY. Let $\mathcal{A}_i = (Q_i, q_i, F_i, \delta_i)$ be NFAs over alphabet $\mathbb{A}$, for $i = 1, 2$. Here, Q_i is the finite set of states of $\mathcal{A}_i$, q_i is the initial state, $F_i \subseteq Q_i$ is the set of final states, and $\delta_i \subseteq Q_i \times \mathbb{A} \times Q_i$ is its transition relation. We assume, without loss of generality, that q_i has no incoming transitions and F_i consists of a single state $f_i \neq q_i$ without outgoing transitions. Furthermore, we assume that f_i (for $i = 1, 2$) is reachable from every state in Q_i and that $Q_1 \cap Q_2 = \emptyset$. It is easy to see that containment of NFAs continues being PSPACE-hard even under such restrictions.

Let $u_1, u_2, v_1, v_2, w_1, w_2$ be fresh elements that do not belong to $Q_1 \cup Q_2$. For $i = 1, 2$, we define a data graph $\mathcal{G}_i = (G_i, E_i, data_i)$ as follows (see Figure 5).

1. $G_i = Q_i \cup \{u_i, v_i, w_i\}$.
2. $(x, a, y) \in E_i$ if and only if one of the following holds:
   - $(x, a, y) \in \delta_i$
   - $x \in Q_i \setminus \{q_i, f_i\}$ and $y \in \{u_i, v_i, w_i\}$
   - $x = q_i$ and $y = u_i$
   - $x = u_i$ and $y \in \{u_i, v_i, w_i\}$
3. $data_i(q) = \begin{cases} 1 & \text{if } q \in \{q_i\} \cup F_i, \\ 2 & \text{if } q \in \{u_i\} \cup Q_i \setminus \{q_i, f_i\}, \\ 3 & \text{if } q = v_i, \\ 4 & \text{if } q = w_i. \end{cases}$

Clearly, $\mathcal{G}_i$ can be constructed in polynomial time from $\mathcal{A}_i$, for $i = 1, 2$. We show next that

$$L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2) \iff \mathcal{G}_1, q_1 \rightrightarrows \mathcal{G}_2, q_2.$$

We start with the right-to-left direction. Take an arbitrary word $e_1 \dots e_n \in L(\mathcal{A}_1)$. Therefore, $n > 0$ since $\mathcal{A}_1$ does not accept the empty word. Let $\varphi := \langle \varepsilon = \downarrow_{e_1} \dots \downarrow_{e_n} \rangle$. Then by construction $\mathcal{G}_1, q_1 \models \varphi$ (as there is a path from q_1 to f_1 in $\mathcal{G}_1$ labeled $e_1 \dots e_n$ and $data_1(q_1) = data_1(f_1)$). Hence, since $\mathcal{G}_1, q_1 \rightrightarrows \mathcal{G}_2, q_2$ and φ is a positive node XPath$_=$ expression, Theorem 7 tells us that $\mathcal{G}_2, q_2 \models \varphi$. That is, there exists a path

$$\pi = q_2 \xrightarrow{e_1} u_1 \dots \xrightarrow{e_n} u_n \text{ in } \mathcal{G}_2 \text{ such that } data_2(q_2) = data_2(u_n).$$

Since $n > 0$, the node u_n can only be f_2. By construction, then, all internal nodes of π must belong to $Q_2 \setminus \{q_2, f_2\}$ (as there is no path linking nodes u_2, v_2, w_2 with f_2). This implies that $e_1 \dots e_n \in L(\mathcal{A}_2)$.

For the left-to-right direction, let us define $Z \subseteq G_1 \times G_2$ as follows (see Figure 5): xZy iff one of the following holds:

1. $x = q_1$ and $y = q_2$.
2. $x \in \{u_1\} \cup Q_1 \setminus \{q_1, f_1\}$ and $y \in \{u_2\} \cup Q_2 \setminus \{q_2, f_2\}$.

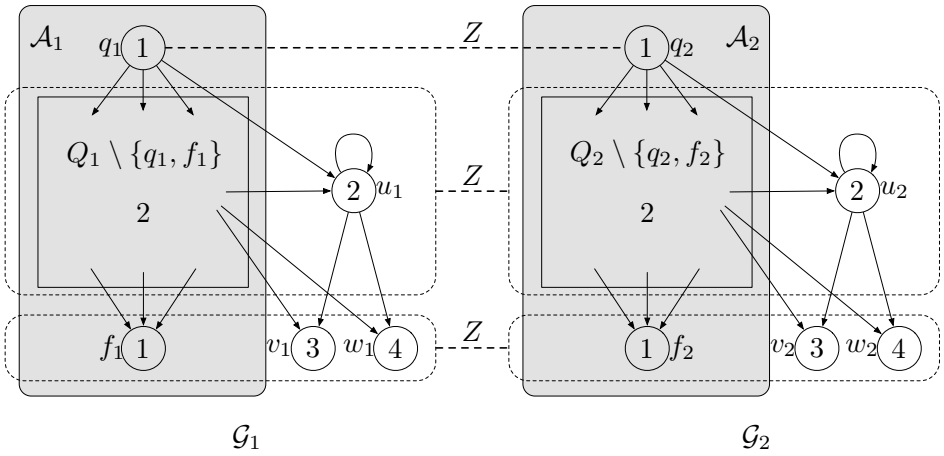

Figure 5: The data graphs $\mathcal{G}_i = (G_i, E_i, data_i)$ $(i = 1, 2)$, constructed from NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, and the bisimulation $Z \subseteq G_1 \times G_2$ used in the left-to-right direction of the reduction. All nodes inside a dotted area on $\mathcal{G}_1$ are related to all nodes inside a dotted on $\mathcal{G}_2$ area via Z.

3. $x \in \{v_1, w_1, f_1\}$ and $y \in \{v_2, w_2, f_2\}$.

We show next that Z satisfies the **(Zig$_=$)** clause for any pair $(x_1, x_2) \in G_1 \times G_2$ such that $x_1 Z x_2$. We do this by cases:

a. If $x_1 \in \{f_1, v_1, w_1\}$ and $x_2 \in \{f_2, v_2, w_2\}$, then **(Zig$_=$)** holds trivially as there are no outgoing paths from f_1, v_1 or w_1.

b. If $x_1 \in Q_1 \setminus \{q_1, f_1\} \cup \{u_1\}$ and $x_2 \in Q_2 \setminus \{q_2, f_2\} \cup \{u_2\}$, let α, β be two paths starting in x_1. Suppose first that $\alpha = x_1$ (*i.e.* the empty path starting at x_1) and

$$\beta = x_1 \xrightarrow{e_1} y_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} y_m \xrightarrow{e_m} z \quad \text{(resp. } x_1 \xrightarrow{e_1} y_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} y_m\text{)},$$

where the y_i's are in $(Q_1 \setminus \{q_1, f_1\}) \cup \{u_1\}$ and $z \in \{f_1, v_1, w_1\}$. Then the corresponding β' in $\mathcal{G}_2$ is

$$x_2 \xrightarrow{e_1} u_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} u_2 \xrightarrow{e_m} v_2 \quad \text{(resp., } x_2 \xrightarrow{e_1} u_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{m-1}} u_2\text{)},$$

where there are $m - 1$ occurrences of u_2. If both α and β have length greater than 0, then the procedure is similar, but one path may end in w_2 if the data values of the endpoints of α, β are different elements in $\{1, 3, 4\}$.

c. Finally, if $x_1 = q_1$ and $x_2 = q_2$, there are two main cases for the type of paths α, β in $\mathcal{G}_1$ to be replicated in $\mathcal{G}_2$ while respecting the **(Zig$_=$)** condition with respect to Z. If both α and β are of length greater than 0, then copying them is straightforward using the nodes u_2, v_2, w_2. If both are of length 0 the result is trivial. Suppose one of them is length 0 and the other one is of length greater than 0, say $\alpha = q_1$ and

$$\beta = q_1 \xrightarrow{e_1} y_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{n-1}} y_n \xrightarrow{e_n} z.$$

We need to find paths α' and β' in $\mathcal{G}_2$, both starting in q_2, which "copy" α and β respectively. Clearly α' is just q_2, as it has to have length 0. The definition of β' depends on z. If $z \in Q_1 \setminus \{q_1, f_1\} \cup \{u_1, v_1, w_1\}$, then β' is of the form

$$\beta' = q_2 \xrightarrow{e_1} y'_2 \xrightarrow{e_2} \dots \xrightarrow{e_{n-1}} y'_n \xrightarrow{e_n} z', \tag{7}$$

where $y'_i = u_2$ $(i = 2 \dots n)$ and z' is either u_2, v_2, or w_2 if z is u_1, v_1, or w_1, respectively. If $z = f_1$, then the endpoints of α and β have both data value 1. Since the word $e_1 \dots e_n \in L(\mathcal{A}_1)$ and by hypothesis $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$, we conclude that $e_1 \dots e_n$ is accepted by $\mathcal{A}_2$. This means that there is a path of the form (7) where $y'_i \in Q \setminus \{q_2, f_2, u_2, v_2, w_2\}$ and $z' = f_2$. This path satisfies the required condition of **(Zig$_=$)**, as the condition on Z is verified and the data value of q_2 and f_2 (the endpoints of α' and β') are equal, namely have data value 1.

For XPATH$_=$-BISIMILARITY the proof is analogous, but using this time a reduction from NFA equivalence. In fact, it can be easily checked that with exactly the same construction shown above we obtain that

$$L(\mathcal{A}_1) \equiv L(\mathcal{A}_2) \iff \mathcal{G}_1, q_1 \leftrightarrows \mathcal{G}_2, q_2.$$

Notice that this construction also immediately implies Corollary 11. In fact, for the right-to-left direction of the reduction to hold we only require invariance of $(\mathcal{G}_2, q_2)$ with respect to $(\mathcal{G}_1, q_1)$ referred to those formulas of the form $\langle \varepsilon = \downarrow_{e_1} \dots \downarrow_{e_n} \rangle$, for $e_1 \dots e_n \in \mathbb{A}$. In other words, we only require $(\mathcal{G}_1, q_1) \Rrightarrow_{\text{paths}} (\mathcal{G}_2, q_2)$ or $(\mathcal{G}_1, q_1) \equiv_{\text{paths}} (\mathcal{G}_2, q_2)$ depending on whether we are reducing from containment or equivalence of NFAs, respectively.

5. Restricting Paths in Bisimulations

The smallest witness to the fact that two NFAs are not equivalent (resp., one NFA is not contained in another one) might be a path of exponential length (Meyer & Stockmeyer, 1972). As a corollary to the proof of Theorem 9, we obtain then that the smallest witness to the fact that a given relation $Z \subseteq G \times G'$ does not satisfy the **(Zig$_=$)** condition might also be a pair (π_1, π_2) of paths of exponential length. This naturally calls for a restriction on the length of paths considered in the definition of XPath$_=$-(bi)simulation as a way to obtain better complexity bounds. We consider this restriction natural for the following reasons:

- Long witnesses correspond to large distinguishing formulas in XPath$_=$. But rarely will users be interested in checking if nodes are distinguishable by formulas that they cannot even write. Thus, the restriction to shorter paths can be seen as an approximation to the notion of XPath$_=$-(bi)similarity based on user-understandable formulas.

- In practice, algorithms for computing (bi)simulations in the absence of data stop after a few iterations (Luo, Fletcher, Hidders, Wu, & Bra, 2013b; Luo, Fletcher, Hidders, Bra, & Wu, 2013a). This tells us that when nodes in real-world graphs are distinguishable by ML formulas, they are distinguishable by some small formula. One might expect a similar behavior for XPath$_=$, implying that the restriction to shorter paths provides a fair approximation of the problem in practice.

In this section we study the complexity of XPath$_=$-(bi)similarity for paths of restricted length. We show that the problem becomes CO-NP-complete for paths of polynomial length, and tractable

for paths of constant length. This notion of bisimilarity further captures the expressive power of a natural fragment of XPath$_=$; namely, the one formed by expressions of *bounded length*. This fragment only restricts formulas of the form $\langle \alpha \star \beta \rangle$, for $\star \in \{=, \neq\}$.

5.1 Bounded bisimulation and equivalence

Let $f : \mathbb{N} \to \mathbb{N}$ be a positive, non-decreasing function. We define the notion of f-XPath$_=$-*(bi)simulation* as in Definition 4, but now in the **(Zig$_=$)** (resp., **(Zag$_=$)**) condition we only consider paths π_1 and π_2 (resp., π'_1 and π'_2) of length at most $f(\max(|G|, |G'|))$, where $|G|$ denotes the number of edges in $\mathcal{G}$. We call the new conditions **(Zig$^f_=$)** and **(Zag$^f_=$)**, respectively.

More formally, an f-XPath$_=$-*bisimulation* between $u \in G$ and $u' \in G'$ (written $\mathcal{G}, u \leftrightarrows_f \mathcal{G}', u'$) is a relation $Z \subseteq G \times G'$ such that uZu' and for all $(x, x') \in G \times G'$ such that xZx' we have:

- **(Zig$^f_=$)** For every paths in $\mathcal{G}$ of the form

  $$\pi_1 = x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \text{ and } \pi_2 = x \xrightarrow{d_1} y_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y_m$$

  such that $m, n \leq f(\max(|G|, |G'|))$, there are paths in $\mathcal{G}'$ of the form

  $$\pi'_1 = x' \xrightarrow{e_1} x'_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x'_n \text{ and } \pi'_2 = x' \xrightarrow{d_1} y'_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y'_m,$$

  such that the following holds:

  1. $x_i Z x'_i$ for all $1 \leq i \leq n$, and $y_j Z y'_j$ for all $1 \leq j \leq m$.
  2. $data(x_n) = data(y_m) \Leftrightarrow data(x'_n) = data(y'_m)$.

- **(Zag$^f_=$)** For every paths in $\mathcal{G}'$ of the form

  $$\pi'_1 = x' \xrightarrow{e_1} x'_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x'_n \text{ and } \pi'_2 = x' \xrightarrow{d_1} y'_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y'_m$$

  such that $m, n \leq f(\max(|G|, |G'|))$, there are paths in $\mathcal{G}$ of the form

  $$\pi_1 = x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} x_n \text{ and } \pi_2 = x \xrightarrow{d_1} y_1 \xrightarrow{d_2} \ldots \xrightarrow{d_m} y_m,$$

  such that conditions 1 and 2 above are verified.

- **(Zig)** For every $y \in G$ and $e \in \mathbb{A}$ such that $x \xrightarrow{e} y$ there is $y' \in G'$ such that $x' \xrightarrow{e} y'$ and yZy'.

- **(Zag)** For every $y' \in G'$ and $e \in \mathbb{A}$ such that $x' \xrightarrow{e} y'$ there is $y \in G$ such that $x \xrightarrow{e} y$ and yZy'.

The reason why we add the one-step, comparison-free rules of **(Zig)** and **(Zag)** will become clearer below, in the characterization theorem for a fragment of XPath$_=$ to be studied next (Theorem 13). The idea is that we want to restrict with f the length of pairs of paths which compare data values at their terminating nodes, but we do not want to restrict the length of single paths which do not compare data values.

Similarly, an f-XPath$_=$-*simulation* from $u \in G$ to $u' \in G'$ (denoted $\mathcal{G}, u \rightrightarrows_f \mathcal{G}', u'$) is a relation $Z \subseteq G \times G'$ such that uZu' and for all $(x, x') \in G \times G'$ such that xZx' it is the case that condition **(Zig$^f_=$)** and **(Zig)** above are verified.

The characterization. We define the logical counterpart of this refined notion of bisimulation. We aim at an analog of Theorem 7 relative to the adequate restriction of indistinguishability (*cf.* Definition 3). As we show below, this restriction is defined by the fragment of XPath$_=$ whose path expressions α occurring in an expression of the form $\langle \alpha \star \beta \rangle$ (for $\star \in \{=, \neq\}$) have length bounded by f. In the following we formalize this idea.

The *length* of a path expression α, denoted $\mathsf{len}(\alpha)$, was defined in the work of Abriola, Descotte, & Figueira, 2017a. It corresponds to the number of $\downarrow_a$'s occurring in α *at the uppermost level*, i.e., outside any test of the form $[\varphi]$. E.g., $\mathsf{len}(\downarrow_a[\langle \downarrow_b = \downarrow_a \downarrow_b \downarrow_c \rangle]\downarrow_b) = \mathsf{len}(\downarrow_a \downarrow_b) = 2$. We use this notion to define the *maximum length* of a node or path expression. This represents the maximum length of paths that are involved in expressions of the form $\langle \alpha \star \beta \rangle$, for $\star \in \{=, \neq\}$.

Definition 12 (Maximum length)**.** Given a node or path expression θ, we write $\mathsf{ml}(\theta)$ to denote the *maximum length* of θ. Formally, ml is recursively defined as follows:

$$\begin{array}{rcl}
\mathsf{ml}(\lambda) & = & 0 \\
\mathsf{ml}(\varepsilon\alpha) & = & \mathsf{ml}(\alpha) \\
\mathsf{ml}([\varphi]\alpha) & = & \max\{\mathsf{ml}(\varphi), \mathsf{ml}(\alpha)\} \\
\mathsf{ml}(\downarrow_a \alpha) & = & \mathsf{ml}(\alpha) \\
\mathsf{ml}(\varphi \wedge \psi) & = & \max\{\mathsf{ml}(\varphi), \mathsf{ml}(\psi)\} \\
\mathsf{ml}(\neg\varphi) & = & \mathsf{ml}(\varphi) \\
\mathsf{ml}(\langle\alpha\rangle) & = & \mathsf{ml}(\alpha) \\
\mathsf{ml}(\langle \alpha \star \beta \rangle) & = & \max\{\mathsf{len}(\alpha), \mathsf{len}(\beta), \mathsf{ml}(\alpha), \mathsf{ml}(\beta)\},
\end{array}$$

where α is any path expression or the empty string λ. □

As an example,

$$\mathsf{ml}(\langle \downarrow_a [\langle \downarrow_b \downarrow_a \downarrow_c \rangle] \downarrow_b = \downarrow_b [\langle \downarrow_a \downarrow_b \downarrow_a \downarrow_b \rangle] \rangle) = 2.$$

We now introduce the notion of f-XPath$_=$-indistinguishability which coincides with f-XPath$_=$-bisimulation (*cf.* Definition 3). We write $\mathcal{G}, u \equiv_f \mathcal{G}', u'$ (resp. $\mathcal{G}, u \Rrightarrow_f \mathcal{G}', u'$) if $\mathcal{G}, u \models \varphi \Leftrightarrow \mathcal{G}', u' \models \varphi$ (resp. $\mathcal{G}, u \models \varphi \Rightarrow \mathcal{G}', u' \models \varphi$) for every (positive) node expression φ of XPath$_=$ such that $\mathsf{ml}(\varphi) \leq f(\max(|G|, |G'|))$.

As in Theorem 7 we obtain the following characterization:

Proposition 13. *$\mathcal{G}, u \equiv_f \mathcal{G}', u'$ iff $\mathcal{G}, u \leftrightarrows_f \mathcal{G}', u'$, and $\mathcal{G}, u \Rrightarrow_f \mathcal{G}', u'$ iff $\mathcal{G}, u \rightrightarrows_f \mathcal{G}', u'$.*

Proof. It follows using the idea in the proof of Theorem 8 from the work of Figueira et al., 2015. Formulas of the form $\langle \alpha \star \beta \rangle$ of ml bounded by $f(\max(|G|, |G'|))$ (for $\star \in \{=, \neq\}$) are handled by rules **(Zig$^f_=$)** and **(Zag$^f_=$)**. Formulas of the form $\langle \downarrow \alpha \rangle$ are equivalent to $\langle \downarrow [\langle \alpha \rangle] \rangle$, and these are handled by rules **(Zig)** and **(Zag)**. □

5.2 Computing f-XPath$_=$-bisimulations

Here we study the complexity of computing f-XPath$_=$-(bi)simulations:

f-XPATH$_=$-(BI)SIMILARITY
INPUT : Data graphs $\mathcal{G}$ and $\mathcal{G}'$, nodes $u \in G$ and $u' \in G'$. QUESTION : $\mathcal{G}, u \rightrightarrows_f \mathcal{G}', u'$? ($\mathcal{G}, u \leftrightarrows_f \mathcal{G}', u'$?, resp.)

Since this problem is PSPACE-complete when f is an exponential function, it is natural to start by restricting f to be a polynomial. We show next that while this restriction lowers the complexity of our problem, it still does not yield tractability:

Proposition 14. *The following holds:*

1. *The problem* p-XPATH$_=$-(BI)SIMILARITY *is in* CO-NP *for every non-decreasing polynomial* $p : \mathbb{N} \to \mathbb{N}$.
2. *The problem* p-XPATH$_=$-(BI)SIMILARITY *can be* CO-NP*-hard even if* $p : \mathbb{N} \to \mathbb{N}$ *is the identity function.*

Proof. We only prove the claim for p-XPATH$_=$-SIMILARITY. The proof for bisimilarity is analogous. We start with item 1. Let $\mathcal{G}$ and $\mathcal{G}'$ be data graphs and u, u' nodes in G, G', respectively. In order to check whether there is an XPath$_=$-simulation from u to u' we can use the standard greatest fixed point algorithm for computing the maximal simulation in the absence of data. We adapt it here to the **(Zig$^p_=$)** condition of XPath$_=$-simulations.

The algorithm computes the maximal XPath$_=$-simulation from $\mathcal{G}$ to $\mathcal{G}'$. We start by defining $Z = G \times G'$. At each step we choose an arbitrary pair $(x, x') \in Z$. If **(Zig$^p_=$)** fails when evaluated on this pair we simply remove it from Z. We proceed iteratively until we reach a fixed point. Finally, we check whether $(u, u') \in Z$. Only if this is the case we declare that there is an XPath$_=$-simulation from u to u'.

Thus, in order to check that there is *no* XPath$_=$-simulation from u to u', we can simply guess a computation of the algorithm that removes the pair (u, u') from Z. Such computation consists of (a) pairs $(x_1, x'_1), \ldots, (x_m, x'_m)$; (b) relations $Z_0, \ldots, Z_m$ such that: $Z_0 = G \times G'$, $Z_i = Z_{i-1} \setminus \{(x_i, x'_i)\}$ for each $1 \leq i \leq m$, and Z_m does not contain (u, u'); and (c) suitable witnesses for the fact that (x_i, x'_i) does not satisfy **(Zig$^p_=$)** with respect to Z_{i-1}, for each $1 \leq i \leq m$. Such witness consists of a pair (π_1, π_2) of paths of length at most $p(\max(|\mathcal{G}|, |\mathcal{G}'|))$ in $\mathcal{G}$ starting from x_i, and yet another witness for the fact that no pair (π'_1, π'_2) of paths in $\mathcal{G}'$ starting from x'_i satisfies **(Zig$^p_=$)** with respect to (π_1, π_2). The latter can be represented by an accepting run of the complement of the NFA $\mathcal{A}_{\mathcal{G}', x'_i, Z_{i-1}}$ (as defined in the proof of the upper bound of Theorem 9) over the word that represents the pair (π_1, π_2) in $\mathcal{A}_{\mathcal{G}, x_i}$. Clearly, each one of the components of this guess can be represented using polynomial space. Further, it can be checked in polynomial time that the guess satisfies the required conditions It follows that checking whether there is *no* XPath$_=$-simulation from u to u' is in NP (and, thus, that our problem is in CO-NP).

For item 2 we use the following claim:

Claim 15. *The problem of checking containment of NFA* $\mathcal{A}_1$ *in* $\mathcal{A}_2$*, restricted to words of length at most* $\max(|\mathcal{A}_1|, |\mathcal{A}_2|)$ *is* CO-NP*-hard. (Here,* $|\mathcal{A}_i|$ *defines the number of tuples in the transition relation of* $\mathcal{A}_i$*, for* $i = 1, 2$*).*

Proof. We use a reduction from the complement of 3CNF satisfiability. Given a 3CNF formula φ of the form

$$(\ell^1_1 \vee \ell^2_1 \vee \ell^3_1) \wedge \ \ldots \ \wedge (\ell^1_n \vee \ell^2_n \vee \ell^3_n),$$

over the set $\{h_1, \ldots, h_m\}$ of propositional symbols, we construct in polynomial time NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ over the alphabet $\mathbb{A} := \{h_1, \ldots, h_m, \neg h_1, \ldots, \neg h_m\}$ such that

$$\varphi \text{ is satisfiable} \iff L(\mathcal{A}_1) \not\subseteq L(\mathcal{A}_2). \tag{8}$$

The NFA $\mathcal{A}_1$ consists of states $q_0, \ldots, q_n$, with q_0 and q_n being the initial and final state. The transitions of $\mathcal{A}_1$ are the ones in the set:

$$\{(q_i, \ell^j_{i+1}, q_{i+1}) \mid 0 \leq i < n,\ j = 1, 2, 3\}.$$

That is, the words accepted by $\mathcal{A}_1$ codify the different ways in which we can choose one literal from each clause in φ. One of these words encodes a satisfying assignment of φ if and only if it does not contain an "error", i.e., it does not mention a propositional variable and its negation. We then take $\mathcal{A}_2$ as the NFA that accepts those words over the alphabet $\mathbb{A}$ that do mention a propositional variable in $\{h_1, \ldots, h_m\}$ and its negation, i.e.,

$$L(\mathcal{A}_2) \ = \ \bigcup_{1 \leq i \leq m} \mathbb{A}^* h_i \mathbb{A}^* \neg h_i \mathbb{A}^* \ \cup \ \mathbb{A}^* \neg h_i \mathbb{A}^* h_i \mathbb{A}^*.$$

Such $\mathcal{A}_2$ can be easily defined as follows. The states of $\mathcal{A}_2$ are r_0 and s_i, t_i, s'_i, t'_i, for each $1 \leq i \leq m$. The initial state is r_0 and the set of final states is $\{t_i, t'_i \mid 1 \leq i \leq m\}$. For each symbol $a \in \mathbb{A}$ and state q in $\mathcal{A}_2$, there is a transition (q, a, q). Moroever, for each $1 \leq i \leq m$ the NFA $\mathcal{A}_2$ contains transitions

$$(r_0, h_i, s_i), (s_i, \neg h_i, t_i), (r_0, \neg h_i, s'_i), (s'_i, h_i, t'_i).$$

It is easy to see that (8) holds. In fact, φ is satisfiable if and only if there is a word in $L(\mathcal{A}_1)$ that does not contain an "error", that is, it does not belong to $L(\mathcal{A}_2)$.

Notice, in addition, that $|\mathcal{A}_1| = 3n$ and $|\mathcal{A}_2| = 8m^2 + 6m \leq 14m^2$. Moreover, the words in $L(\mathcal{A}_1)$ are of length at most n. This tells us that not only (8) holds, but it holds even if the containment of $\mathcal{A}_1$ into $\mathcal{A}_2$ is restricted to words of length at most $\max(|\mathcal{A}_1|, |\mathcal{A}_2|)) \geq 3n > n$. □

We then reduce the restricted containment problem from Claim 15 to p-XPATH$_=$-SIMILARITY, where $p : \mathbb{N} \to \mathbb{N}$ is the identity function, by using the same construction than in the proof of the lower bound of Theorem 9. In fact, it can be readily checked that, starting from NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, such reduction constructs data graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with distinguished nodes q_1 and q_2, respectively, such that the following are equivalent:

1. $\mathcal{A}_1$ is contained in $\mathcal{A}_2$ up to words of length at most $\max(|\mathcal{A}_1|, |\mathcal{A}_2|))$.
2. $(\mathcal{G}, q_1) \rightrightarrows_{\max(|\mathcal{G}_1|, |\mathcal{G}_2|))} (\mathcal{G}_2, q_2)$.

This finishes the proof of the proposition. □

The reason why the previous restriction is not sufficient to obtain tractability is that there are too many paths of polynomial length in a data graph. We solve this issue by restricting to paths of constant length only. In the following we identify the function that takes constant value $c \in \mathbb{N}$ with the letter c.

For $c \geq 0$, we call XPath$_=(c)$ the syntactical fragment of XPath$_=$ of ml bounded by c. Further, fragments of the form XPath$_=(c)$ (for $c \geq 1$) extend multi-modal logic, which in turn coincides with XPath$_=(0)$:

Proposition 16. *XPath$_=(0)$ is semantically equivalent to multi-modal logic with no propositions and only atoms $\top$ and $\bot$.*

Proof. In the jargon of ML, we have a language with modalities $\langle a \rangle$ for each $a \in \mathbb{A}$. On the one hand, any node expression of the form $\langle \alpha \star \beta \rangle$ in XPath$_=(0)$ is also of the form

$$\langle [\varphi_1] \dots [\varphi_n] \star [\psi_1] \dots [\psi_m] \rangle,$$

which is equivalent to $\bigwedge_i \varphi_i \wedge \bigwedge_j \psi_j$, if $\star$ is $=$, and to a contradiction (e.g., $\neg\langle\varepsilon\rangle$) otherwise. On the other hand, a node expression $\langle \downarrow_a \alpha \rangle$ is equivalent to $\langle \downarrow_a [\langle \alpha \rangle] \rangle$. Any node expression of the form $\langle \downarrow_a [\varphi] \rangle$ of XPath$_=(0)$ can be straightforwardly translated (in a truth preserving way) to ML via T as $\langle a \rangle T(\varphi)$. The translation from modal logic to XPath$_=(0)$ is obvious. □

Proposition 17. *The problem of* c-XPATH$_=$-(BI)SIMULATION *is* PTIME*-complete for each constant* $c > 0$*.*

Proof. We use the same algorithm as in the proof of the previous upper bound. The difference now is that checking whether a pair $(x, x') \in Z$ satisfies **(Zig$^c_=$)** can be solved efficiently for each $c > 0$. This is because there is at most a polynomial number of paths of length $\leq c$ in $\mathcal{G}$ starting from x. We conclude that checking whether $\mathcal{G}, u \rightrightarrows_c \mathcal{G}', u'$ is in PTIME. The same holds for $\mathcal{G}, u \leftrightarrows_c \mathcal{G}', u'$. The lower bound follows straightforwardly from Proposition 16 and PTIME-hardness for usual ML bisimulations (Balcázar, Gabarro, & Santha, 1992). □

Proposition 13 establishes that c-XPath$_=$-simulations characterize the expressive power of the fragment of XPath$_=$ defined by formulas of ml bounded by the constant c. The following corollary to the proof of Proposition 17 states that when two nodes are not c-XPath$_=$-bisimilar, it is possible to compute in polynomial time a node expression in this fragment that distinguishes them.

Corollary 18. *There is a polynomial time algorithm which given* $\mathcal{G}, u \not\rightrightarrows_c \mathcal{G}', u'$ *(resp.,* $\mathcal{G}, u \not\leftrightarrows_c \mathcal{G}', u'$*), constructs*[2] *a (positive) node expression* φ *of XPath*$_=(c)$ *such that* $\mathcal{G}, u \models \varphi$ *and* $\mathcal{G}', u' \not\models \varphi$*.*

Proof. We adapt the algorithm given in the work of Areces et al., 2011 for the basic ML to our notion of bisimulation. We only explain the case of c-XPath$_=$-simulation, as for bisimulation the proof is analogous.

We construct a polynomial time algorithm which, on input $\mathcal{G}$ and $\mathcal{G}'$, computes, for each $x \in G$, a set $S(x) = \{x' \in G' \mid x \rightrightarrows_c x'\}$ and a positive node expression $N(x)$ such that $\mathcal{G}, x \models N(x)$ and $[\![N(x)]\!]^{\mathcal{G}'} = S(x)$. The existence of this algorithm is enough to prove the desired result: if $\mathcal{G}, u \not\rightrightarrows_c \mathcal{G}', u'$, we have that $u' \notin S(u)$ and therefore $\mathcal{G}, x \models N(x)$ and $\mathcal{G}', x' \not\models N(x)$, hence satisfying the statement of the Corollary.

The algorithm is as follows. We start by setting, for all $x \in G$, $S(x) := G'$ and $N(x) := \top$ (representing any positive tautology, *e.g.*$\langle\varepsilon\rangle$).We repeat the following process until none of the items below are true:

- There are $(x, x') \in G \times G'$ such that **(Zig$^c_=$)** does not hold at (x, x'), in the sense that there are paths in $\mathcal{G}$ of the form

  $$\pi_1 = x \xrightarrow{e_1} x_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} x_n \text{ and } \pi_2 = x \xrightarrow{d_1} y_1 \xrightarrow{d_2} \dots \xrightarrow{d_m} y_m$$

  with $m, n \leq c$, and such that there are no paths in $\mathcal{G}'$ of the form

  $$\pi'_1 = x' \xrightarrow{e_1} x'_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} x'_n \text{ and } \pi'_2 = x' \xrightarrow{d_1} y'_1 \xrightarrow{d_2} \dots \xrightarrow{d_m} y'_m,$$

2. Provided an adequate representation for such formula is chosen (Figueira & Gorín, 2010).

satisfying $x'_i \in S(x_i)$ for all $1 \leq i \leq n$, $y'_i \in S(y_i)$ for all $1 \leq j \leq m$, and $data(x_n) = data(y_m)$ iff $data(x'_n) = data(y'_m)$. In this case we set $S(x) := S(x) \setminus \{x'\}$ and

$$N(x) := N(x) \wedge \langle \downarrow_{e_1} [N(x_1)] \downarrow_{e_2} \ldots \downarrow_{e_n} [N(x_n)] \star \downarrow_{d_1} [N(y_1)] \downarrow_{d_2} \ldots \downarrow_{d_m} [N(y_m)] \rangle,$$

where $\star$ is $=$ in case $data(x_n) = data(y_m)$ and $\neq$ otherwise.

- There are $(x, x') \in G \times G'$ such that **(Zig)** does not hold at (x, x'), in the sense that there is $y \in G$ such that $x \xrightarrow{e} y$ and there is no $y' \in G'$ such that $y \xrightarrow{e} y'$ and $y' \in S(y)$. In this case, we set $S(x) := S(x) \setminus \{x'\}$ and $N(x) := N(x) \wedge \langle \downarrow_e [N(y)] \rangle$.

The idea is that at each step, if either **(Zig$^f_=$)** or **(Zig)** are false, we shrink $S(x)$ for some x, and the "reason" behind the falsity of **(Zig$^f_=$)** or **(Zig)** is encoded in the node expression $N(x)$. The invariant of the cycle states that for all $x \in G$ we have $[\![N(x)]\!]^{\mathcal{G}'} \subseteq S(x)$ and $\mathcal{G}, x \models N(x)$. Since at execution of the body of the cycle, one element is removed from $S(x)$, for some $(x, x') \in G \times G'$, the total number of iterations is polynomial in the size of the data graphs. Furthermore, at each iteration, we only search for paths of length at most c, and so the total number of steps taken by this algorithm is polynomial. □

6. Restricting the Models

Here we follow a different approach from the one in Section 5: We constrain the topology of graphs instead of the (bi)simulations. Since our goal is restricting the number and/or length of the paths considered in the analysis of (bi)simulations, it is natural to look into acyclic graphs; namely, trees and DAGs.

Let us start with data trees, i.e., data graphs whose underlying graph is a directed tree. This case is relevant as XML documents are (essentially) data trees, and the study of XPath$_=$-bisimulations was started in such context. Notice that for data trees both the number and the length of paths one needs to consider when checking the **(Zig$_=$)** and **(Zag$_=$)** conditions are polynomial. This implies that the problem of computing XPath$_=$-(bi)simulations over data trees is tractable:

Theorem 19. *The problem of* XPATH$_=$-(BI)SIMULATION *for data trees is in* PTIME.

As a second case, let us consider *data DAGs*, which allow for undirected cycles only. In this case the length, but not the number, of the paths one needs to consider at the moment of checking the **(Zig$_=$)** and **(Zag$_=$)** conditions is polynomial. The first observation helps lowering the complexity of computing XPath$_=$-(bi)simulations in this context from PSPACE to CO-NP, while the second one prevents us from obtaining tractability.

Proposition 20. *The problem* XPATH$_=$-(BI)SIMULATION *for data DAGs is* CO-NP*-complete.*

Proof. The following result is instrumental for our proof.

Lemma 21. *The problems of containment or equivalence of NFAs whose underlying graph is a DAG are both* CO-NP-COMPLETE.

Proof. The CO-NP-hardness of the NFA DAG containment problem follows from Claim 15, since the number of tuples in the transition relations of a DAG form an upper bound to the length of words accepted by it. The completeness of the problem is then immediate, as a witness of $L(\mathcal{A}_1) \not\subseteq$

$L(\mathcal{A}_2)$ can be checked in polynomial time, since there is a linear bound on the length of words accepted by DAGs. For the hardness of the equivalence problem, we can reduce the problem of DAG containment to equivalence by adding a disjoint copy of $\mathcal{A}_2$ to $\mathcal{A}_1$ and then joining by a same parent node: $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1 \sqcup \mathcal{A}_2) \equiv L(\mathcal{A}_2)$. The completeness follows as in the case of containment. □

The problem is in CO-NP as a consequence of the first item of Proposition 14, since paths in DAGs are of linear size. To prove CO-NP-hardness, we reduce the problem of DAG containment (equivalence) to the problem of data DAG (bi)simulation, and then use Lemma 21. We will see the proof only for simulation, with a construction that can also be used for the analogous case of bisimulation.

Given two DAG NFAs $\mathcal{A}_i = (Q_i, q_i, F_i, \delta_i)$, for $i = 1, 2$, (*i.e.*, NFA whose transition graphs have no cycles) we construct data DAGs $\mathcal{G}_i = (G_i, E_i, data_i)$, with $i = 1, 2$, as in the proof of the lower bound of Theorem 9. The main difference is that the node u_i of Figure 5 will be replaced by some DAGs that we will now construct. Let $n = \max\{|Q_1|, |Q_2|\}$, and let Q_i^j be the set of nodes $q \in Q_i$ at "maximum distance j" from f_i, that is, so that there is a directed path from q to f_i of length j but there is no such path of length $> j$ (note that $Q_i = \bigcup_{j \leq n} Q_i^j$). Now, each u_i of Figure 5 is replaced with n fresh nodes $u_i^1, \ldots, u_i^n$ of data value 2, with every possible edge from Q_i^j to $u_i^{j'}$ for every $1 \leq j' < j$, from q_i to u_i^n, from u_i^j to v_i and w_i, and from u_i^j to $u_i^{j'}$ for every $j' < j$. It is straightforward to check that the resulting data graphs are DAGs, and that as in the lower bound proof of Theorem 9 we have that $\mathcal{G}_1, q_1 \rightrightarrows \mathcal{G}_2, q_2$ if and only if $L(A_1) \subseteq L(A_2)$; here the simulation differs in that all nodes from a Q_1^j will be related with u_2^j (and those of Q_2^j with u_1^j), and each u_1^j is related with the corresponding u_2^j.

□

7. Two-way XPath$_=$

A common expressive extension for languages on graphs is to consider a *two-way* version that allows to traverse edges in both directions (see, e.g., Calvanese, De Giacomo, Lenzerini, & Vardi, 2000; Libkin et al., 2013). We call XPath$_=^{\updownarrow}$ the extension of XPath$_=$ with path expressions of the form $\uparrow_a$, for $a \in \mathbb{A}$. The semantics of these expressions over $\mathcal{G} = \langle G, E, data \rangle$ is as follows:

$$[\![\uparrow_a]\!]^{\mathcal{G}} = \{(x, y) \mid (y, a, x) \in E\}.$$

We write $\mathcal{G}, u \equiv^{\updownarrow} \mathcal{G}', u'$ (resp. $\mathcal{G}, u \Rightarrow^{\updownarrow} \mathcal{G}', u'$) if $\mathcal{G}, u \models \varphi \Leftrightarrow \mathcal{G}', u' \models \varphi$ (resp. $\mathcal{G}, u \models \varphi \Rightarrow \mathcal{G}', u' \models \varphi$) for every (positive) node expression φ of XPath$_=^{\updownarrow}$.

A notion of (bi)simulation for XPath$_=^{\updownarrow}$ *over data trees* was introduced in the work of Figueira et al., 2015, and turns out to be tractable. It heavily relies on the determinism of $\uparrow_a$ over trees, and hence does not fit in the context of data graphs. However, there is a simple way to adapt XPath$_=$-bisimulations to the case of two-way XPath$_=$.

Given a data graph $\mathcal{G} = \langle G, E, data \rangle$ over $\mathbb{A}$, we define its *completion* over $\mathbb{A} \cup \mathbb{A}^-$, where $\mathbb{A}^- := \{a^- \mid a \in \mathbb{A}\}$, as the data graph $\mathcal{G}_c = \langle G, E_c, data \rangle$, where E_c extends E by adding the edge (v, a^-, u), for each edge $(u, a, v) \in E$. That is, $\mathcal{G}_c$ extends $\mathcal{G}$ with the "inverse" of every edge in E. We also define a bijection $\varphi \mapsto \varphi_c$ mapping node expressions of XPath$_=$ over $\mathbb{A}$ to node expressions of XPath$_=^{\updownarrow}$ over $\mathbb{A} \cup \mathbb{A}^-$ as follows: φ_c is the result of replacing each occurrence of $\uparrow_a$ in φ (for $a \in \mathbb{A}$) with $\downarrow_{a^-}$. The following proposition is straightforward:

Proposition 22. *$\mathcal{G}, u \models \varphi$ iff $\mathcal{G}_c, u \models \varphi_c$.*

We say that there is an *XPath$^{\updownarrow}_{=}$-bisimulation* between $u \in G$ and $u' \in G'$ (denoted $\mathcal{G}, u \leftrightarrows^{\updownarrow} \mathcal{G}', u'$) if $\mathcal{G}_c, u \leftrightarrows \mathcal{G}'_c, u'$ (over the extended alphabet $\mathbb{A} \cup \mathbb{A}^-$). Similarly, we define XPath$^{\updownarrow}_{=}$-simulations $\rightrightarrows^{\updownarrow}$. Analogously to Theorem 7, one can show the following result.

Theorem 23. *$\mathcal{G}, u \equiv^{\updownarrow} \mathcal{G}', u'$ iff $\mathcal{G}, u \leftrightarrows^{\updownarrow} \mathcal{G}', u'$, and $\mathcal{G}, u \Rrightarrow^{\updownarrow} \mathcal{G}', u'$ iff $\mathcal{G}, u \rightrightarrows^{\updownarrow} \mathcal{G}', u'$.*

We study the complexity of the following problem:

XPATH$^{\updownarrow}_{=}$-(BI)SIMILARITY
INPUT : Data graphs $\mathcal{G}$ and $\mathcal{G}'$, nodes $u \in G$ and $u' \in G'$. QUESTION : $\mathcal{G}, u \rightrightarrows^{\updownarrow} \mathcal{G}', u'$? ($\mathcal{G}, u \leftrightarrows^{\updownarrow} \mathcal{G}', u'$?, resp.)

The bounded notions of bisimulation introduced in §5 are defined over XPath$^{\updownarrow}_{=}$ and an alphabet $\mathbb{A}$ in the expected way: reducing to XPath$_{=}$ over the signature $\mathbb{A} \cup \mathbb{A}^-$ and the corresponding completion of the data graphs. We use symbols $\rightrightarrows^{\updownarrow}_{f}$ (resp. $\leftrightarrows^{\updownarrow}_{f}$) for referring to the f-XPath$^{\updownarrow}_{=}$ (bi)simulations. Observe that in this 'two-way' case the f bounds the total number of edges traversed, regardless of whether their labels are in $\mathbb{A}$ or in $\mathbb{A}^-$ We then study:

f-XPATH$^{\updownarrow}_{=}$-(BI)SIMILARITY
INPUT : Data graphs $\mathcal{G}$ and $\mathcal{G}'$, nodes $u \in G$ and $u' \in G'$. QUESTION : $\mathcal{G}, u \rightrightarrows^{\updownarrow}_{f} \mathcal{G}', u'$? ($\mathcal{G}, u \leftrightarrows^{\updownarrow}_{f} \mathcal{G}', u'$?, resp.)

The identification of XPath$^{\updownarrow}_{=}$ over $\mathbb{A}$ with XPath$_{=}$ over $\mathbb{A} \cup \mathbb{A}^-$ and the corresponding completions of graphs allows us to straightforwardly transfer some upper bounds:

- XPATH$^{\updownarrow}_{=}$-(BI)SIMILARITY is in PSPACE (§4.1).
- p-XPATH$^{\updownarrow}_{=}$-(BI)SIMILARITY, for p a non-decreasing polynomial, is in CO-NP (item 1 of Proposition 14).
- c-XPATH$^{\updownarrow}_{=}$-(BI)SIMILARITY, for c a constant function, is in PTIME (Proposition 17)

Regarding the lower bounds, we focus here on the general case of XPath$^{\updownarrow}_{=}$. One can check that the proof given in §4.2 does not work because in the two way context, more nodes in the graphs can be reached through the accessibility relations. The main result of this section is the following:

Theorem 24. XPATH$^{\updownarrow}_{=}$-(BI)SIMILARITY *is* PSPACE*-complete.*

Proof. We only need to prove the lower bound. We will actually prove something stronger later in Theorems 25 and 28, where we show that similarity and bisimilarity for XPath$^{\updownarrow}_{=}$, respectively, remain PSPACE-hard even over DAG data graphs. However, such proofs are quite more complicated, and we believe it is useful to show that over arbitrary data graphs one can obtain a simpler proof. We only give the proof for bisimilarity. We use a reduction from the PSPACE-complete problem of *universality* for NFA (i.e., does an NFA accept all words?). Let $\mathcal{A}$ be a NFA over $\mathbb{A}$, with initial state q_0 and final state q_f. We build data graphs $\mathcal{G}$ and $\mathcal{G}'$ as in Figure 6. We claim that $\mathcal{G}, u_1 \leftrightarrows^{\updownarrow} \mathcal{G}', u'_1$ if

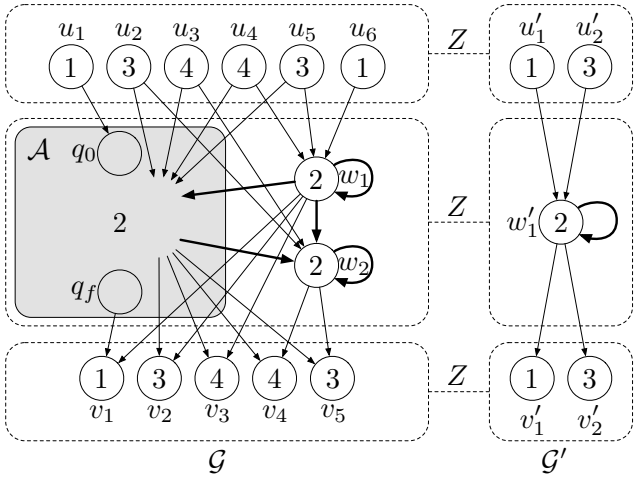

Figure 6: Definition of the data graphs $\mathcal{G}$ and $\mathcal{G}'$ based on an NFA $\mathcal{A}$ over alphabet $\mathbb{A}$. Boldface arrows have, as label, all symbols from $\mathbb{A}$. Lightface arrows have all the same label $e \notin \mathbb{A}$ for some e. All nodes of $\mathcal{A}$ (the grey area) have data value 2. All nodes inside a dotted area on $\mathcal{G}$ are related to all nodes inside a dotted area on $\mathcal{G}'$ via Z.

and only if $L(\mathcal{A}) = \Sigma^*$.

For the left-to-right direction we proceed as follows. Given a word $\omega = a_1 \dots a_n$ in Σ^*, we consider the formula $\varphi = \langle \epsilon = \downarrow_e \downarrow_{a_1} \dots \downarrow_{a_n} \downarrow_e \rangle$. By construction, $\mathcal{G}', u'_1 \models \varphi$, and from the hypothesis $\mathcal{G}, u_1 \leftrightarrows^{\updownarrow} \mathcal{G}', u'_1$ and Theorem 23, we have that $\mathcal{G}, u_1 \models \varphi$. On the one hand, the only way to transit from u_1 is through e, getting to q_0. On the other hand, in order to satisfy φ in u_1, there is a path:

$$u_1 \xrightarrow{e} q_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} z_1 \xrightarrow{e} z_2$$

where $data(z_2) = data(u_1) = 1$. One can see that the only possibility is that $z_2 = v_1$. One could arrive to v_1 from q_f or w_1, but w_1 is downwardly inaccessible from $\mathcal{A}$, and therefore $z_1 = q_f$. Even more, all the nodes in the path $q_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} z_1$ are in $\mathcal{A}$, and so $\omega \in L(\mathcal{A})$.

For the right-to-left direction, one can check that the relation Z depicted in Figure 6 is an XPath$^{\updownarrow}_{=}$-bisimulation. In all cases, the **(Zig$_=$)** condition is easily satisfied thanks to the construction of $\mathcal{G}'$, so we will only check **(Zag$_=$)** for every pair $(x, x') \in G \times G'$. We do this by cases.

- To prove **(Zag$_=$)** for the pair (u_1, u'), with $u' \in \{u'_1, u'_2\}$, let $\alpha = u' \xrightarrow{e_1} x_2 \xrightarrow{e_2} \dots \xrightarrow{e_{m-1}} x_m$ and $\beta = u' \xrightarrow{e'_1} y_2 \xrightarrow{e'_2} \dots \xrightarrow{e'_{n-1}} y_n$, where $e_i, e'_i \in \mathbb{A} \cup \mathbb{A}^-$. The interesting case is that where one of the paths has length 0, say $\alpha = u'$; y_n has the same data value as u'; and all e'_i belong to $\mathbb{A}$. In this case, we only need to use the hypothesis that $L(\mathcal{A}) = \Sigma^*$, and here it is important that w_2 cannot reach down into v_1.

- For all other pairs (u, u') with $u \in \{u_2, u_3, u_4, u_5, u_6\}$, $u' \in \{u'_1, u'_2\}$ in the topmost level of the figure, the general strategy for the construction of our needed paths is to mimic the paths in $\mathcal{G}'$ by going into w_1 or w_2 whenever the original paths go to w'_1. Whenever the paths on $\mathcal{G}'$ go to u'_1, u'_2, v'_1, or v'_2, and the path does not end there, the path on $\mathcal{G}$ mirrors the move

by going into any node in the top or bottom part of $\mathcal{G}$, as appropriate. Finally, depending on whether the original paths end in different or same data value, the path in $\mathcal{G}$ can go to two nodes with data value 3 and 4 or to the same node, respectively.

- To prove that **(Zag$_=$)** holds for all pairs (w, w') with $w \in \mathcal{A} \cup \{w_1, w_2\}, w' = w'_1$ of the middle level, the hard case is when w is a node of the $\mathcal{A}$ portion of the graph. Here, the idea of the procedure is the same as in the previous item.

- Finally, for pairs (v, v') with $v \in \{v_1, v_2, v_3, v_4, v_5\}$, $v' \in \{v'_1, v'_2\}$ of the bottom level, we also proceed as before, going upwards into w_1 as soon as the path in $\mathcal{G}'$ goes upwards into w'_1, and afterward mimicking the behaviours of the paths as needed.

This concludes the proof. □

7.1 Restricting the models

Interestingly, for two-way (bi)similarity the PSPACE-hardness holds even over data DAGs. This establishes a stark difference with the one-way (bi)similarity problem for XPath$_=$ over data DAGs, which lies in CO-NP (as stated in Proposition 20). Our lower bound proofs are more involved than the ones we have seen so far, as they require to establish PSPACE-hardness for the containment/equivalence problems over restricted classes of NFAs. We prove the lower bound for similarity and bisimilarity separately. We believe that this improves the clarity of the presentation, as it allows us to incrementally construct the more complicated gadgets used for the bisimilarity lower bound in terms of the simpler ones used for similarity.

We first show the following result.

Theorem 25. XPATH$^{\updownarrow}_{=}$-SIMILARITY *for data DAGs is* PSPACE*-hard.*

Proof. The proof consists of two parts. In Lemma 26 we use the idea of the proof of PSPACE-hardness of NFA containment (implicit in the work of Kozen, 1977) to show that hardness holds even when the transition graphs of the automata are 'almost' DAGs. Then we prove in Lemma 27 that the containment problem for such a class of NFAs can be reduced to a data DAG simulation problem for XPath$^{\updownarrow}_{=}$.

We say that $\mathcal{G}$ is a *#-quasi-DAG* (or simply a *quasi-DAG* when the symbol # is implicit) if it is an edge-labeled graph over an alphabet with a distinguished symbol #, such that reversing all those edges labeled # in $\mathcal{G}$ results in a DAG.

Lemma 26. *The containment problem for NFAs is* PSPACE*-complete, even if the input consists of two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ whose underlying graphs are quasi-DAGs.*

Proof of Lemma 26. We show that there is a polynomial-time computable translation such that, given inputs $n \in \mathbb{N}$ (in unary) and a Turing machine $M = \langle Q, \{0,1\}, \delta, q_0, F\rangle$, constructs two automata $\mathcal{A}^1_{n,M}, \mathcal{A}^2_{n,M}$ over the alphabet $\{0, 1, \#\} \cup Q$ such that:

1. $L(\mathcal{A}^1_{n,M}) \subseteq L(\mathcal{A}^2_{n,M})$ iff there does not exists an accepting run of M in SPACE(n).

2. If we reverse the arrows of the transitions reading the symbol # in $\mathcal{A}^1_{n,M}$ and $\mathcal{A}^2_{n,M}$ we obtain the NFAs $\widehat{\mathcal{A}}^1_{n,M}$ and $\widehat{\mathcal{A}}^2_{n,M}$. These automata are DAGs.

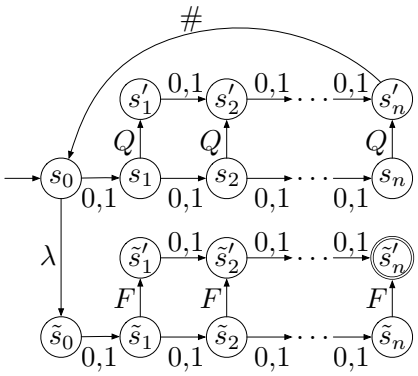

Figure 7: The automaton $\mathcal{A}^1_{n,M}$, with initial state s_0 and final state $\tilde{s}'_n$, accepts all non-empty sequences of configurations $c_1 \dots c_i q c_{i+1} \dots c_n$ separated with the symbol $\#$, and such that in the last configuration the symbol chosen from Q is actually from the set F of final states of M.

The lemma then follows since PSPACE-complete is closed under complement.

The main idea is that $\mathcal{A}^1_{n,M}$ accepts encodings of all possible sequences of tape configurations (in a syntactic sense, not necessarily obtainable in M) such that the ending configuration contains a final state (some $q_f \in F$), while $\mathcal{A}^2_{n,M}$ accepts all words that are *not* valid sequences of configurations for M (for example, because two consecutive configurations are not related via a valid transition of δ). Thus, $L(\mathcal{A}^1_{n,M})$ being included in $L(\mathcal{A}^2_{n,M})$ means that no possible run of M restricted to tape space n can reach a final state.

As we restrict the space of the Turing machine M to n cells, we can encode each configuration C in the form of $C = c_1 \dots c_i q c_{i+1} \dots c_n$, where $c_j \in \{0, 1\}$ for each $1 \leq i \leq n$, and $q \in Q$; here we interpret that the head of the machine is over cell i and the machine is in state q. We can then encode a run of M as a sequence $C_1 \# C_2 \dots \# C_m$, where C_1 must satisfy validity conditions corresponding to the initialization of M, and where each C_{i+1} must be reachable from C_i using the transition δ of M. The main difficulty of this Lemma lies in building the NFA $\mathcal{A}^2_{n,M}$ satisfying condition 2 and accepting all sequences of configurations that are not valid. While we show the construction of $\mathcal{A}^i_{n,M}$ using empty transitions (i.e., λ-transitions) for ease of understanding, it is straightforward to modify these NFA to avoid such transitions.

The construction of $\mathcal{A}^1_{n,M}$ is simple, and is graphically given in Figure 7. Observe that reversing its $\#$ arrow turns the structure of the automata into a DAG, satisfying condition 2.

For the construction of $\mathcal{A}^2_{n,M}$, we will divide the problem into various parts, building different automata that accept sequences of configurations that are not valid runs. We will construct different automata accepting various sequences:

- An automaton $\mathcal{E}_{init}$ that accepts sequences such that the first $n+1$ symbols do not correspond to an initial tape configuration of M.

- An automaton $\mathcal{E}_{local}$ that detects when two successive configurations are not valid because some cell has changed its value without having the head of the machine over it.

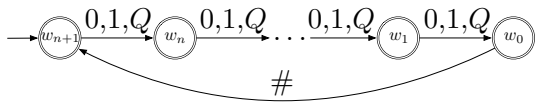

Figure 8: The graphical representation of $\mathcal{W}_{n+1}$. In general, $\mathcal{W}_i$ is defined in the same way, but having as the sole initial state the state w_i. All states are final states.

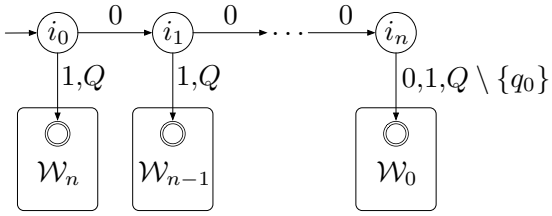

Figure 9: The automaton $\mathcal{E}_{init}$ has no final states, except for all those in the $\mathcal{W}_i$. It only accepts words if the initial $n+1$ symbols are not the word $0 \dots 0q_0$, which corresponds to the initial configuration of the tape of M.

- n automata $\mathcal{E}_{head(i)}$ that detect an error in successive configurations, when first the head is over cell i, and the following position or state of the head and the symbol on the cell i are not consistent with transition δ.

The automaton $\mathcal{A}^2_{n,M}$ simply consists of an initial state that can make a λ-transition to the initial states of $\mathcal{E}_{init}$, $\mathcal{E}_{local}$ or some $\mathcal{E}_{head(i)}$; if these automata satisfy the property 2, then so does $\mathcal{A}^2_{n,M}$.

We now proceed to the construction of these automata. First, we construct auxiliary automata $\mathcal{W}_i$, for each $0 \leq i \leq n+1$. Intuitively, they will serve the function of, once an error has been found, potentially writing the remaining i symbols from the alphabet $\{0,1\} \cup Q$ and then proceeding to write new groups of $n+1$ symbols separated with $\#$. See Figure 8 for details.

For the construction of $\mathcal{E}_{init}$, see Figure 9.

For the construction of $\mathcal{E}_{local}$, we first build the automata $\mathcal{C}_i$, $0 \leq i \leq n+1$, which reads a word of length $n+1$ such that the i-th letter is a 0 or 1, and the following symbol, if any, is not in Q. The idea for the construction of $\mathcal{E}_{local}$ is to start reading groups of words of length $n+1$, separated with a $\#$, and then nondeterministically choose one position with a 0 or 1, such that the next letter is not a symbol in Q, and such that the next 'configuration' has a 1 or 0 in that position, respectively; this should not be a valid run of M, as in one step there has been a change to a cell where the head was not positioned. See Figure 10 for more details.

For the construction of $\mathcal{E}_{head(i)}$, the idea is as in $\mathcal{E}_{local}$, only that now we need to take into account all possible local inconsistencies (around the cell position i in two successive 'configurations') with respect to the transition function δ. At some point $\mathcal{E}_{head(i)}$ reads an $n+1$ symbols word, and the automaton nondeterministically takes many different paths according to the symbol 0 or 1 corresponding to the cell position i and according to the symbol q representing the head of M over the cell i. Each of these disjoint paths, after reading a $\#$, check that the next 'configuration' is not consistent with the previously seen data and the transition function δ: there is some error in the

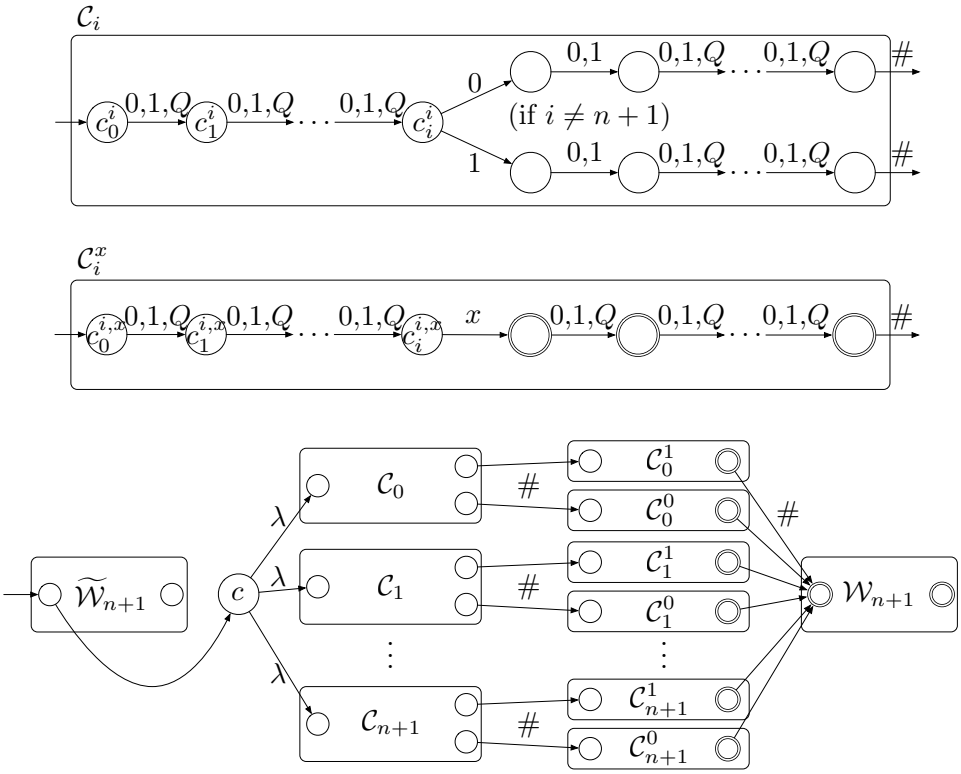

Figure 10: $\mathcal{E}_{local}$ accepts words that include two consecutive configurations where the tape changes in a position that did not have the head of M over it. $\widetilde{\mathcal{W}}_{n+1}$ is like $\mathcal{W}_{n+1}$, but with an empty set of final states. $\mathcal{C}_i$ and $\mathcal{C}_i^x$ have lengths of $n+2$, reading configurations of $n+1$ characters.

combination of the current symbol on the i-th position of the cell, the current position of the head, or the current state in Q of the machine.

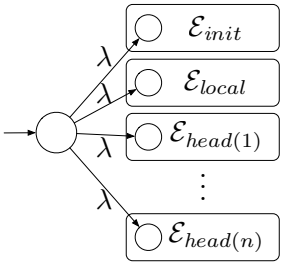

Figure 11: The automaton $\mathcal{A}_{n,M}^2$.

Finally, as mentioned before, the automaton $\mathcal{A}_{n,M}^2$ is assembled with these automata, as shown in Figure 11. We remark that $\mathcal{E}_{init}$, $\mathcal{E}_{local}$ and all the $\mathcal{E}_{head(i)}$ are DAGs when the arrows reading $\#$

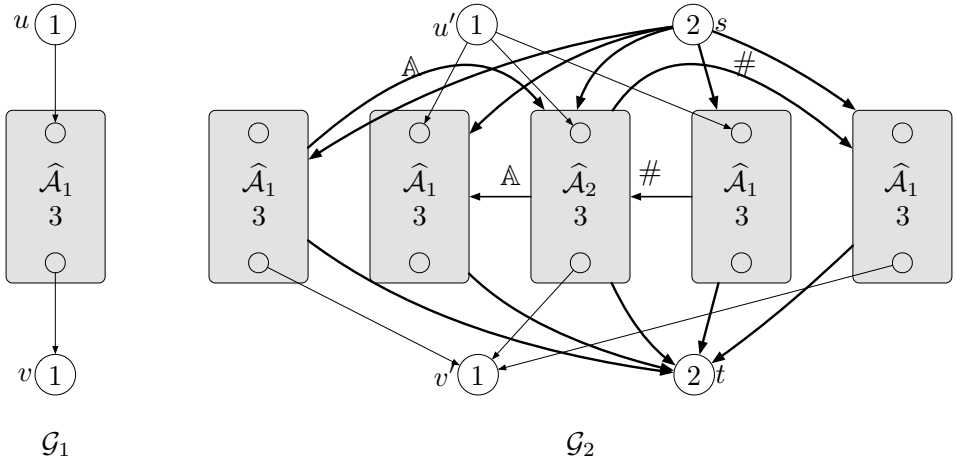

Figure 12: A graphical representation of the data graphs $(\mathcal{G}_1, u)$ and $(\mathcal{G}_2, u')$. The automata $\widehat{\mathcal{A}}_1$ and $\widehat{\mathcal{A}}_2$ are defined as in Item 2 of Lemma 26. Lightface arrows have all the same empty label disjoint from $\mathbb{A} \cup \{\#\}$. Bold arrows between boxes represent that there is a transition arrow from every state of the first box into every state of the second one. Bold arrows between a state and a box represent that there is a transition arrow from such state into every state of the box. All nodes represented in the grey region have data value 3.

are reversed in direction. Therefore, applying that reversion of $\#$-arrows to $\mathcal{A}^2_{n,M}$ also results in a DAG, as we wanted. □

We now finish the proof of Theorem 25. To do this, it suffices to prove the following lemma.

Lemma 27. *Given two NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ (without λ-transitions), whose underlying graphs are quasi-DAGs over an alphabet $\mathbb{A} \sqcup \{\#\}$, and such that all their states can reach a final state, we have that*

$$L(\mathcal{A}_1) \subseteq L(\mathcal{A}_1) \quad \iff \quad \mathcal{G}_1, u \rightrightarrows^{\updownarrow} \mathcal{G}_2, u',$$

where $(\mathcal{G}_1, u)$ and $(\mathcal{G}_2, u')$ are defined as in Figure 12.

Note that the conditions of being free of λ-transitions and that all the states can reach a final state do not restrict the generality of this result; for any NFA $\mathcal{A}$, we can construct a NFA $\mathcal{A}'$ such that it satisfies those two conditions, with $L(\mathcal{A}) = L(\mathcal{A}')$, and such that the underlying graph of $\mathcal{A}'$ is a quasi-DAG if the one of $\mathcal{A}$ is a quasi-DAG.

Proof of Lemma 27. First, assume that $\mathcal{G}_1, u \rightrightarrows^{\updownarrow} \mathcal{G}_2, u'$. Let us consider an arbitrary word $w \in L(\mathcal{A}_1)$. In particular, w is of the form

$$w_1 \# \cdots \# w_m,$$

where each non-empty w_i is of the form

$$c_{i,1} \dots c_{i,n_i}$$

with $c_{i,j} \in \mathbb{A}$. We take a corresponding path expression

$$\alpha = \downarrow\downarrow_{c_{1,1}} \cdots \downarrow_{c_{1,n_1}} \uparrow_{\#} \cdots \uparrow_{\#} \downarrow_{c_{m,1}} \cdots \downarrow_{c_{m,n_i}} \downarrow.$$

Since $\mathcal{G}_1, u \rightleftarrows^{\updownarrow} \mathcal{G}_2, u'$, and $\varphi := \langle \epsilon = \alpha \rangle$ is satisfied in $\mathcal{G}_1, u$, then φ is also satisfied in $\mathcal{G}_2, u'$ by Theorem 7. However, the only way for that formula to be satisfied at u' is for the path α to go solely through $\widehat{\mathcal{A}}_2$, as the labels of the arrows do not allow the opportunity to go to the leftmost or rightmost copies of $\widehat{\mathcal{A}}_1$, and the path cannot enter the innermost copies of $\widehat{\mathcal{A}}_1$ or it would similarly be unable to leave that region and end in a node with data value 1. Therefore, we conclude that $w \in L(\mathcal{A}_2)$. Hence, $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$.

For the converse implication, let us assume that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$. We define a simulation Z from $\mathcal{G}_1, u$ to $\mathcal{G}_2, u'$ as follows. The relation Z over $\mathcal{G}_1 \times \mathcal{G}_2$ consists of the pairs $(u, u'), (u, s), (v, v'), (v, t)$, as well as every pair (z, z') so that z is a node of $\widehat{\mathcal{A}}_1$ in $\mathcal{G}_1$ and z' is either (a) the corresponding node in any of the four copies of $\widehat{\mathcal{A}}_1$ in $\mathcal{G}_2$, or (b) *any* node from the sole copy of $\widehat{\mathcal{A}}_2$ in $\mathcal{G}_2$.

By definition, $(u, u') \in Z$. Let $x_1 Z x_2$. We want to prove that the conditions of **(Zig$_=$)** are satisfied for the pair (x_1, x_2). We can divide this proof into three main cases, depending on the region where x_1 lies:

- The case where $x_1 = u$: If $x_2 = s$, then we can simply mirror everything from $\mathcal{G}_1$ in some copy of $\widehat{\mathcal{A}}_1$ plus the nodes s and t, so we only consider the case $x_2 = u'$. Assume we have two concrete paths starting from u, and ending in nodes y and z, respectively. We analyse the different possibilities for y and z (ignoring symmetric cases); our separation into subcases automatically indicates if the data values of these ending nodes are equal or not:
  - If $y = z = u$, then on $\mathcal{G}_2$ we can simulate the paths of $\mathcal{G}_1$ as follows. If at any moment the original paths pass through $\widehat{\mathcal{A}}_1$, then on $\mathcal{G}_2$ we go to the corresponding node in any inner copy of $\widehat{\mathcal{A}}_1$ and from then on we simply mirror the paths (if at some point a path in $\mathcal{G}_1$ goes to v, we mimic it in $\mathcal{G}_2$ by going to t).
  - Similarly if $y = u$ and $z \in \widehat{\mathcal{A}}_1$, or if both y and z belong to $\widehat{\mathcal{A}}_1$.
  - If $y \in \widehat{\mathcal{A}}_1$ and $z = v$, we can proceed in the same way as before, as only data inequality (and not the particular data values) is important, so we can match v with t. Analogously for $y = z = v$.
  - The remaining subcase is that where $y = u$ and $z = v$. Note that here the pair of paths cannot be mirrored using the previous idea. However, if the paths do not use $\uparrow_a$ with $a \in \mathbb{A}$ nor $\downarrow_{\#}$, then we can mirror them in $\widehat{\mathcal{A}}_2$ using the hypothesis that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$. Analogously, if any of the paths ever uses an arrow $\uparrow_a$ with $a \in \mathbb{A}$, or a $\downarrow_{\#}$, then we can mirror the initial part of that path in $\widehat{\mathcal{A}}_2$ (using the hypothesis that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ and that the set of final states of $\mathcal{A}_1$ is reachable from any of its states), and then continue the mirroring in one of the external copies of $\widehat{\mathcal{A}}_1$ (which are directly connected to v', unlike the inner copies).
- The case where $x_1 = v$: This case is analogous to the previous one. We only have to remark, for the case where $x_2 = v', y = v, z = u$, that if any path has any arrow $\downarrow_a$ with $a \in \mathbb{A}$ or an arrow $\uparrow_{\#}$, then we can mirror the path from that point forward by going into an inner copy of $\widehat{\mathcal{A}}_1$. Otherwise, the paths correspond to words in $\mathcal{A}_1$, and we can use the hypothesis.

- The case where x_1 is a node in the copy of $\widehat{\mathcal{A}}_1$: If x_2 lies in some copy of $\widehat{\mathcal{A}}_1$, then **(Zig$_=$)** is trivially satisfied. Otherwise, if x_2 lies in the copy of $\widehat{\mathcal{A}}_2$, the strategy to simulate the paths of $\mathcal{G}_1$ is simply to go towards s or t or towards the appropriate copy $\widehat{\mathcal{A}}_1$, depending on how the paths begin; as any of those copies can access nodes s and t (which have the same data value but different data value from the nodes in the central regions), we are done.

This concludes the proof of Lemma 27. □

This concludes the proof of Theorem 25. □

An analogous of Theorem 25 also holds for bisimilarity, but proving this is not such a straightforward extension as was the case for Theorem 9 or Proposition 20.

Theorem 28. XPATH$^{\updownarrow}_{=}$-BISIMILARITY *for data DAGs is* PSPACE*-hard.*

Proof. First of all, we can easily prove that quasi-DAG equivalence also is PSPACE-complete:

Lemma 29. *The equivalence problem for NFAs is* PSPACE*-complete, even if the input consists of two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ whose underlying graphs are quasi-DAGs.*

Proof. That this problem is in PSPACE follows immediately from Lemma 26. On the other hand, the completeness follows from that same lemma by reducing from the problem of containment to the one of equivalence using that, given two quasi-DAGs $\mathcal{A}_1$ and $\mathcal{A}_2$, we have that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1 \sqcup \mathcal{A}_2) \equiv L(\mathcal{A}_2)$. Note that if $\mathcal{A}_1$ and $\mathcal{A}_2$ are quasi-DAGs, then so is $\mathcal{A}_1 \sqcup \mathcal{A}_2$ via the standard construction. □

To prove the PSPACE-hardness of the problem of XPath$^{\updownarrow}_{=}$-bisimilarity for data DAGs, we first define some useful terms. A *root* of a DAG is a node that has no incoming edges, the *height* of a DAG is the maximum downward distance between a root and a node, and the *level* i of a DAG is the set of nodes whose maximum distance to some root is i.

Now, to obtain the hardness for Theorem 28, we observe that it is sufficient to prove the following. Given two NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ over alphabet $\mathbb{A} \sqcup \{\#\}$ whose underlying graphs are $\#$-quasi-DAGs, and such that the heights of $\widehat{\mathcal{A}}_1$ and of $\widehat{\mathcal{A}}_2$ are equal (where $\widehat{\mathcal{A}}_i$ is the automaton that reverses the $\#$-transitions of $\mathcal{A}_i$), and both $\widehat{\mathcal{A}}_1$ and of $\widehat{\mathcal{A}}_2$ have no unreachable states or states that cannot reach a final state, we can construct data DAGs $\mathcal{D}_1$ and $\mathcal{D}_2$ such that

$$L(\mathcal{A}_1) \equiv L(\mathcal{A}_2) \qquad \Longleftrightarrow \qquad \mathcal{D}_1, u_1 \leftrightarrow^{\updownarrow} \mathcal{D}_2, u_2,$$

for specific nodes $u_1 \in \mathcal{D}_1$ and $u_2 \in \mathcal{D}_2$ to be defined below. That proving this is sufficient for our purposes easily follows Lemma 29 and the fact that the quasi-DAGs $\mathcal{A}^1_{n,M} \sqcup \mathcal{A}^2_{n,M}$ and $\mathcal{A}^2_{n,M}$ have the same height (as $\widehat{\mathcal{A}}^1_{n,M} \sqcup \widehat{\mathcal{A}}^2_{n,M}$ have both height n, and so does $\widehat{\mathcal{A}}^2_{n,M}$).

Now, we proceed to the construction of $\mathcal{D}_1, u_1$ and $\mathcal{D}_2, u_2$ by applying two transformations to the NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$:

- **The transformation $\mathcal{T}_d(\mathcal{B})$.** It standardizes some properties relating their initial and final states. Given $d > 0$ and an automaton $\mathcal{B}$ with an underlying DAG, we define the DAG-automaton $\mathcal{T}_d(\mathcal{B})$ as in Figure 13, adding new states and changing the initial state and set of final states. Observe that for any $\mathcal{B}$ and a large enough d, $\mathcal{T}_d(\mathcal{B})$ has only one state in level 0 (the new initial state) and the new final state is the only state at its level. Furthermore, note that $L(\mathcal{T}_d(\mathcal{B})) = e^d L(\mathcal{B}) e^d$.

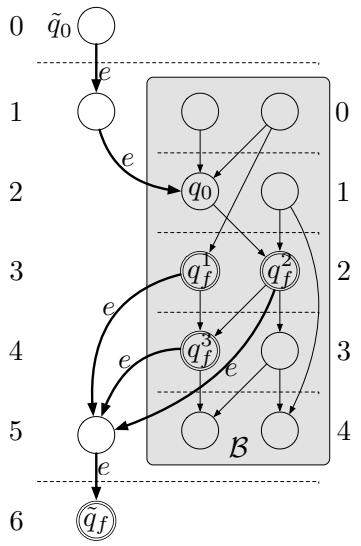

Figure 13: In gray, a DAG-automaton $\mathcal{B}$ over a given alphabet Σ of height 4 (transition letters not shown) with initial state q_0 in level 1, and final states q_f^1, q_f^2 in level 2, and q_f^3 in level 3. We add 4 new nodes and 6 new transitions with label $e \notin \Sigma$ to obtain a new DAG-automaton $\mathcal{T}_2(\mathcal{B})$ of height 6, whose initial state is $\tilde{q}_0$ in level 0 and the only final state is $\tilde{q}_f$ is in level 6. Observe that $L(\mathcal{T}_2(\mathcal{B})) = eeL(\mathcal{B})ee$.

- **The transformation $\Theta(\mathcal{B})$.** Given an automaton $\mathcal{B}$ with an underlying DAG, with initial node in level 0 and a unique final state in its last level, Θ converts $\mathcal{B}$ into a data DAG $\Theta(\mathcal{B})$ with a distinguished node as shown in Figure 14.

We then define $\mathcal{D}_1, u_1 := \Theta(\mathcal{T}_d(\widehat{\mathcal{A}}_1))$ and $\mathcal{D}_2, u_2 := \Theta(\mathcal{T}_d(\widehat{\mathcal{A}}_2))$, where d is the height of $\widehat{\mathcal{A}}_1$ and $\widehat{\mathcal{A}}_2$.

To prove that $\mathcal{D}_1, u_1 \leftrightarrows^{\updownarrow} \mathcal{D}_2, u_2$ implies $L(\mathcal{A}_1) \equiv L(\mathcal{A}_2)$, we proceed as in the proof of Theorem 25. We only prove that $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$, as the other inclusion is symmetrically obtained; to see this, we take u_1 and u_2, which must be related via some two-way bisimulation by hypothesis. Let $\omega \in L(\mathcal{A}_1)$, and $\tilde{\omega} = e^d \omega e^d$ (where e is the fresh symbol that appears in the construction of $\mathcal{T}_d(\widehat{\mathcal{A}}_1)$) and let α be the corresponding path expression in the vein of what was done in the proof of Theorem 25. By **(Zig$_=$)**, there must be a a path satisfying α in $\mathcal{D}_2$ starting in u_2 and ending in a node with value 1. By construction, this implies that the path goes solely through $\mathcal{T}_d(\widehat{\mathcal{A}}_2)$; from this we derive that $\omega \in L(\mathcal{A}_2)$, as we wanted.

On the other hand, to prove that $L(\mathcal{A}_1) \equiv L(\mathcal{A}_2)$ implies $\mathcal{D}_1, u_1 \leftrightarrows^{\updownarrow} \mathcal{D}_2, u_2$, we construct a a two-way bisimulation Z between $\mathcal{D}_1$ and $\mathcal{D}_2$. The definition of Z is given in Figure 15. In particular, Z relates a node x_1 in $\mathcal{D}_1$ with a node x_2 in $\mathcal{D}_2$ if and only if x_1 is at the same level as x_2 (recall that $\mathcal{D}_1$ and $\mathcal{D}_2$ have the same height).

We must prove that the Z so defined is indeed a two-way bisimulation. To do this, we generally proceed in the same fashion as in the proof of Theorem 25. We do not give an exhaustive case-by-case proof, as the strategies to find adequate witnesses for **(Zig$_=$)** and **(Zag$_=$)** are analogous to those

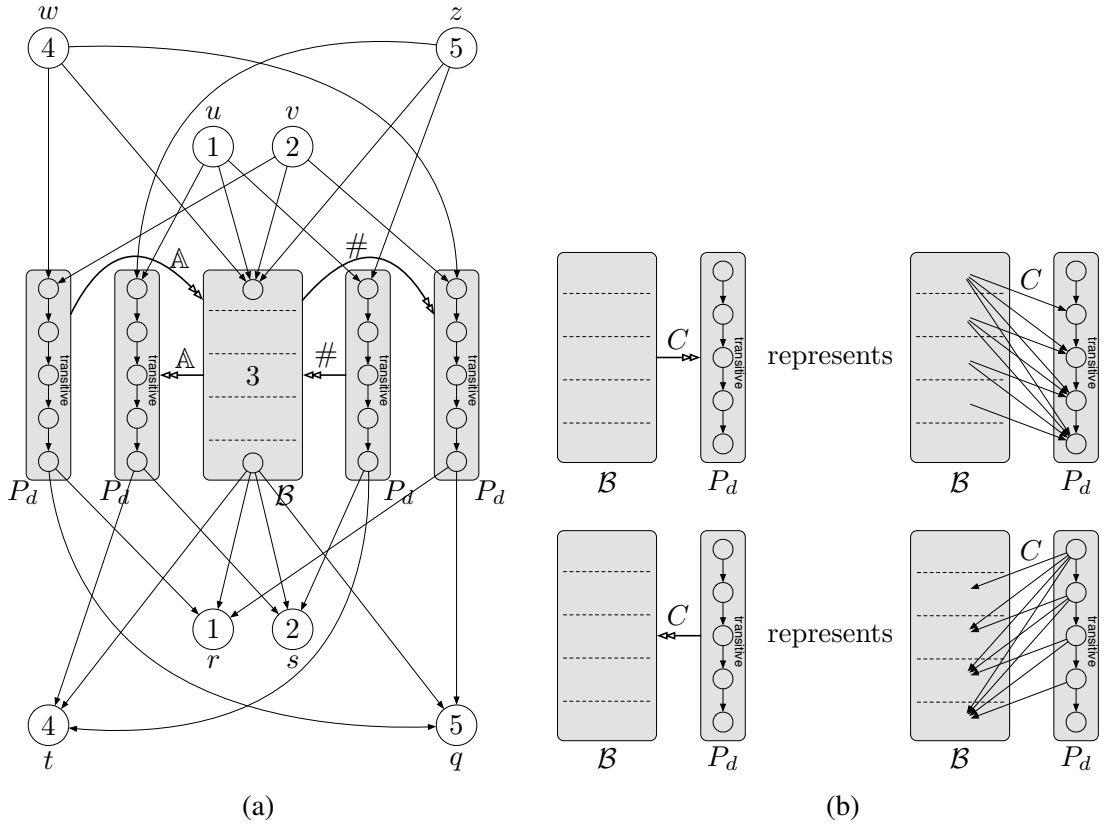

Figure 14: (a) The transformation Θ from a "normalized" NFA $\mathcal{B}$ with an underlying DAG into a data DAG $\Theta(\mathcal{B})$. The NFA $\mathcal{B}$ has its initial node in level 0 and a unique final state in its last level. The lines between the uppermost nodes and the initial state of $\mathcal{B}$, as well as the lines between the sole final state of $\mathcal{B}$ and the lowermost nodes, are edges with a single new label, which we will refer to as an 'idle' label to avoid notation clutter. The dotted lines represent the levels of $\mathcal{B}$, and d represents the height of $\mathcal{B}$ (in the diagram $d = 5$). P_d consists of d nodes connected linearly and transitively via edges with all labels in $\mathbb{A} \sqcup \{\#\}$. The edges between the nodes in the copies of P_d and $\mathcal{B}$ go to all nodes of a certain level, as indicated in (b). All nodes in gray areas have data value 3. The distinguished node in the data DAG is u.

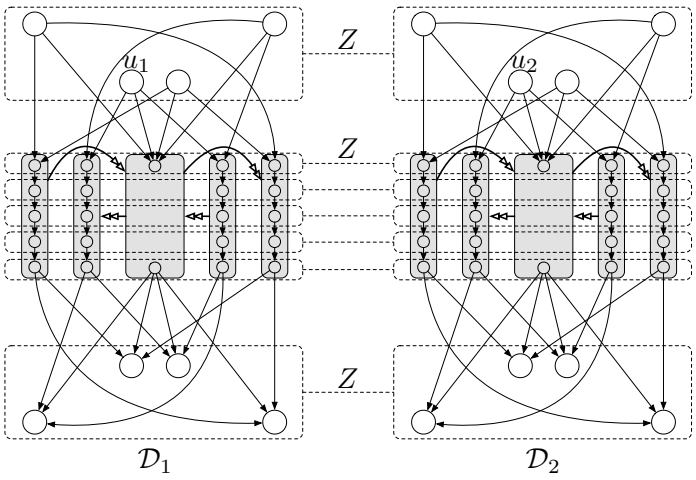

Figure 15: The data DAGs $\mathcal{D}_1$ and $\mathcal{D}_2$ have the same height. The relation Z connects each node at level i in $\mathcal{D}_1$ with each node at level i in $\mathcal{D}_2$. Z is a bisimulation between $\mathcal{D}_1, u_1$ and $\mathcal{D}_2, u_2$.

previously used. Instead, we highlight the salient points of the construction of $\mathcal{D}_1$ and $\mathcal{D}_2$, giving an idea of the main properties that are needed for the proof:

- From every one of the upper nodes $\{u_1, v_1, w_1, z_1\}$ or $\{u_2, v_2, w_2, z_2\}$, it is possible to descend via an idle edge to the uppermost node of the central portion of the corresponding data DAG (namely, the portion with $\mathcal{T}_d(\widehat{\mathcal{A}}_1)$ or $\mathcal{T}_d(\widehat{\mathcal{A}}_2)$, respectively). In particular, we can always start from one of the upper nodes, navigate 'inside $\mathcal{T}_d(\widehat{\mathcal{A}}_i)$' (using the hypothesis $L(\mathcal{A}_1) \equiv L(\mathcal{A}_2)$ to mimic 'words' from one automaton with the other), and end up in a node with the same data value as the starting one. This point is important to verify that all upper nodes are bisimilar.

- Analogously, there is an edge from the lowest node of the central portion towards all lower nodes. This is used to verify that all the lower nodes are bisimilar with r_i.

- From any of the upper nodes, we can always descend via two paths ending in different data values and having the following forms: they begin and end with idle-labeled edges, and in the middle they have edges with labels in $\mathbb{A} \sqcup \{\#\}$, in number at most that of the height of $\mathcal{T}_d(\widehat{\mathcal{A}}_i)$. This is another key point for proving the bisimilarity of all upper nodes.

- Analogously for pair of paths beginning in a lower node, 'ascending' to end in two nodes of different data values.

This concludes the proof of Theorem 28. □

8. Conclusions

As we have seen, while in general computing (bi)simulations on data graphs is PSPACE-complete, better bounds can be obtained by either restricting the topology of the graph, or by relaxing the

conditions for bisimulation. Further, several upper bounds continue to hold when inverses are added, save for the ones when the underlying graph is a data DAG. The following table summarizes our results:

Model	Problem	Logic	
		$\text{XPath}_{=}$	$\text{XPath}^{\updownarrow}_{=}$
Graph	(bi)simulation	PSPACE-c	PSPACE-c
	p-(bi)simulation	CO-NP	CO-NP
	c-(bi)simulation	PTIME-c	PTIME
DAG	(bi)simulation	CO-NP-c	PSPACE-c
Tree	(bi)simulation	PTIME	PTIME

In the future we would like to consider XPath with reflexive-transitive axes. Instead of having $\downarrow_a$, we then have $\downarrow^*_A$ denoting pairs of nodes that can be reached through paths with labels from a set A. Although having both $\downarrow_a$ and $\downarrow^*_A$ does not change the indistinguishability (nor bisimulation) relation, having only $\downarrow^*_A$ in the absence of $\downarrow_a$ gives rise to a different bisimulation relation, somewhat akin to ML-bisimulation over transitive frames (Dawar & Otto, 2009).

Acknowledgements

This work was partially supported by the Laboratoire International Associé "INFINIS", by grants ANPCyT-PICT-2013-2011 and UBACyT 20020150100002BA, and STIC AmSud 2015 *Foundations of Graph Structured Data*. Barceló is funded by Millennium Nucleus Center for Semantic Web Research under Grant NC120004 and Fondecyt grant 1170109.

References

Abriola, S., Barceló, P., Figueira, D., & Figueira, S. (2016). Bisimulations on data graphs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR*, pp. 309–318.

Abriola, S., Descotte, M. E., & Figueira, S. (2014). Definability for downward and vertical XPath on data trees. In *21th Workshop on Logic, Language, Information and Computation*, Vol. 6642 of *Lecture Notes in Computer Science*, pp. 20–34.

Abriola, S., Descotte, M. E., & Figueira, S. (2017a). Model theory of XPath on data trees. Part II: Binary bisimulation and definability. *Information and Computation*, *To appear.*

Abriola, S., Figueira, D., & Figueira, S. (2017b). Logics of repeating values on data trees and branching counter systems. In *International Conference on Foundations of Software Science and Computational Structures*, Lecture Notes in Computer Science. Springer.

Angles, R., & Gutiérrez, C. (2008). Survey of graph database models. *ACM Comput. Surv.*, *40*(1).

Areces, C., Koller, A., & Striegnitz, K. (2008). Referring expressions as formulas of description logic. In *Proc. of the 5th INLG*, Salt Fork, OH, USA.

Areces, C., Figueira, S., & Gorín, D. (2011). Using logic in the generation of referring expressions. In *Logical Aspects of Computational Linguistics*, pp. 17–32. Springer.

Balcázar, J., Gabarro, J., & Santha, M. (1992). Deciding bisimilarity is P-complete. *Formal aspects of computing*, *4*(1), 638–648.

Barceló, P. (2013). Querying graph databases. In *PODS*, pp. 175–188.

Belardinelli, F., Lomuscio, A., & Patrizi, F. (2014). Verification of agent-based artifact systems. *J. Artif. Intell. Res. (JAIR)*, *51*, 333–376.

Blackburn, P., de Rijke, M., & Venema, Y. (2001). *Modal Logic*. Cambridge University Press.

Bojańczyk, M., Muscholl, A., Schwentick, T., & Segoufin, L. (2009). Two-variable logic on data trees and XML reasoning. *J. ACM*, *56*(3).

Calvanese, D., De Giacomo, G., Lenzerini, M., & Vardi, M. Y. (2000). Containment of conjunctive regular path queries with inverse. In *KR*, pp. 176–185.

Clarke, E. M., Grumberg, O., & Peled, D. (2001). *Model checking*. MIT Press.

Dalmau, V., Kolaitis, P. G., & Vardi, M. Y. (2002). Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP*, pp. 310–326.

David, C., Gheerbrant, A., Libkin, L., & Martens, W. (2013). Containment of pattern-based queries over data trees. In *ICDT*, pp. 201–212. ACM.

Dawar, A., & Otto, M. (2009). Modal characterisation theorems over special classes of frames. *Annals of Pure and Applied Logic*, *161*(1), 1–42.

Dechter, R. (1992). From local to global consistency. *Artif. Intell.*, *55*(1), 87–108.

Dechter, R. (2003). *Constraint processing*. Elsevier Morgan Kaufmann.

Demri, S., D'Souza, D., & Gascon, R. (2007). Decidable temporal logic with repeating values. In *Symposium on Logical Foundations of Computer Science*, Vol. 4514 of *LNCS*, pp. 180–194. Springer.

Demri, S., Figueira, D., & Praveen, M. (2016). Reasoning about data repetitions with counter systems. *Log. Methods Comput. Sci.*, *12*(3).

Demri, S., & Lazić, R. (2009). LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, *10*(3).

Demri, S., Lazić, R., & Nowak, D. (2005). On the freeze quantifier in constraint LTL: Decidability and complexity. In *International Symposium on Temporal Representation and Reasoning*, pp. 113–121. IEEE Press.

Fan, W., Li, J., Wang, X., & Wu, Y. (2012). Query preserving graph compression. In *SIGMOD*, pp. 157–168.

Figueira, D. (2010). *Reasoning on Words and Trees with Data*. PhD thesis, Laboratoire Spécification et Vérification, ENS Cachan, France.

Figueira, D. (2012). Decidability of downward XPath. *ACM Trans. Comput. Log.*, *13*(4).

Figueira, D. (2013). On XPath with transitive axes and data tests. In *PODS*, pp. 249–260. ACM.

Figueira, D., Figueira, S., & Areces, C. (2014). Basic model theory of xpath on data trees. In *ICDT*, pp. 50–60.

Figueira, D., Figueira, S., & Areces, C. (2015). Model theory of XPath on data trees. Part I: Bisimulation and characterization. *Journal of Artificial Intelligence Research*, *53*, 271–314.

Figueira, S., & Gorín, D. (2010). On the size of shortest modal descriptions.. In *Advances in Modal Logic*, Vol. 8, pp. 114–132.

Fischer, M. J., & Ladner, R. E. (1979). Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, *18*(2), 194–211.

Getoor, L., & Diehl, C. P. (2005). Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, *7*(2), 3–12.

Givan, R., Dean, T. L., & Greig, M. (2003). Equivalence notions and model minimization in markov decision processes. *Artif. Intell.*, *147*(1-2), 163–223.

Hariri, B. B., Calvanese, D., De Giacomo, G., Deutsch, A., & Montali, M. (2013a). Verification of relational data-centric dynamic systems with external services. In *PODS*, pp. 163–174.

Hariri, B. B., Calvanese, D., Montali, M., De Giacomo, G., De Masellis, R., & Felli, P. (2013b). Description logic knowledge and action bases. *J. Artif. Intell. Res. (JAIR)*, *46*, 651–686.

Jurdziński, M., & Lazić, R. (2007). Alternation-free modal mu-calculus for data trees. In *LICS*, pp. 131–140. IEEE Press.

Jurdziński, M., & Lazić, R. (2011). Alternating automata on data trees and xpath satisfiability. *ACM Trans. Comput. Log.*, *12*(3), 19.

Kara, A., Schwentick, T., & Zeume, T. (2010). Temporal logics on words with multiple data values. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*.

Kolaitis, P. G., & Vardi, M. Y. (2000). A game-theoretic approach to constraint satisfaction. In *AAAI*, pp. 175–181.

Kozen, D. (1977). Lower bounds for natural proof systems. In *FOCS*, pp. 254–266.

Krahmer, E., van Erk, S., & Verleg, A. (2003). Graph-based generation of referring expressions. *Computational Linguistics*, *29*(1).

Krahmer, E., & Van Deemter, K. (2012). Computational generation of referring expressions: A survey. *Computational Linguistics*, *38*(1), 173–218.

Kupferman, O., & Vardi, M. Y. (1998). Verification of fair transition systems. *Chicago J. Theor. Comput. Sci.*, *1998*.

Kurtonina, N., & de Rijke, M. (1999). Expressiveness of concept expressions in first-order description logics. *Artif. Intell.*, *107*(2), 303–333.

Libkin, L., Martens, W., & Vrgoč, D. (2013). Querying graph databases with XPath. In *ICDT*, pp. 129–140. ACM.

Libkin, L., Martens, W., & Vrgoc, D. (2016). Querying graphs with data. *J. ACM*, *63*(2), 14.

Libkin, L., & Vrgoč, D. (2012). Regular path queries on graphs with data. In *ICDT*, pp. 74–85.

Luo, Y., Fletcher, G. H. L., Hidders, J., Bra, P. D., & Wu, Y. (2013a). Regularities and dynamics in bisimulation reductions of big graphs. In *GRADES 2013*, p. 13.

Luo, Y., Fletcher, G. H. L., Hidders, J., Wu, Y., & Bra, P. D. (2013b). External memory k-bisimulation reduction of big graphs. In *22nd ACM CIKM'13*, pp. 919–928.

Lutz, C. (2003). Description logics with concrete domains—a survey. In *Advances in Modal Logics Volume 4*. King's College Publications.

Lutz, C., Areces, C., Horrocks, I., & Sattler, U. (2005). Keys, nominals, and concrete domains. *J. Artif. Intell. Res. (JAIR)*, *23*, 667–726.

Meyer, A. R., & Stockmeyer, L. J. (1972). The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT (FOCS)*, pp. 125–129.

Milner, R. (1971). An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pp. 481–489.

Milner, R. (1999). *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press.

Milo, T., & Suciu, D. (1999). Index structures for path expressions. In *ICDT*, pp. 277–295.

Murawski, A. S., Ramsay, S. J., & Tzevelekos, N. (2015). Bisimilarity in fresh-register automata. In *LICS*, pp. 156–167. IEEE Press.

Park, D. M. R. (1981). Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, pp. 167–183.

Robinson, I., Webber, J., & Eifrem, E. (2013). *Graph Databases*. O'Reilly Media, Inc.

Sangiorgi, D. (2009). On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, *31*(4), 1–41.

Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, *4*(2), 177–192.

van Benthem, J. (1976). *Modal Correspondence Theory*. PhD thesis, Universiteit van Amsterdam.