

# Logical Queries over Views: Decidability and Expressiveness

JAMES BAILEY,  
The University of Melbourne  
and  
GUOZHU DONG,  
Wright State University  
and  
ANTHONY WIDJAJA TO  
University of Edinburgh

---

We study the problem of deciding satisfiability of first order logic queries over views, our aim being to delimit the boundary between the decidable and the undecidable fragments of this language. Views currently occupy a central place in database research, due to their role in applications such as information integration and data warehousing. Our main result is the identification of a decidable class of first order queries over unary conjunctive views that generalises the decidability of the classical class of first order sentences over unary relations, known as the Löwenheim class. We then demonstrate how various extensions of this class lead to undecidability and also provide some expressivity results. Besides its theoretical interest, our new decidable class is potentially interesting for use in applications such as deciding implication of complex dependencies, analysis of a restricted class of active database rules, and ontology reasoning.

Categories and Subject Descriptors: F4.1 [MATHEMATICAL LOGIC AND FORMAL LANGUAGES]: Mathematical Logic; H2.3 [DATABASE MANAGEMENT]: Languages

General Terms: Theory

Additional Key Words and Phrases: Satisfiability, containment, unary view, decidability, first order logic, database query, database view, conjunctive query, Löwenheim class, monadic logic, unary logic, ontology reasoning

---

## 1. INTRODUCTION

The study of views in relational databases has attracted much attention over the years. Views are an indispensable component for activities such as data integration

---

Author's address: James Bailey, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.

Guozhu Dong, Department of Computer Science and Engineering, Wright State University, USA. Anthony Widjaja To, School of Informatics, University of Edinburgh, United Kingdom. Supported by ORSAS Awards and EPSRC grant E005039.

A preliminary version of this paper appeared in [Bailey and Dong 1999].

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20 ACM 1529-3785/20/0700-0001 \$5.00

and data warehousing [Widom 1995; Garcia-Molina et al. 1995; Levy et al. 1996], where they can be used as “mediators” for source information that is not directly accessible to users. This is especially helpful in modelling the integration of data from diverse sources, such as legacy systems and/or the world wide web.

Much of the research related to views has addressed fundamental problems such as containment and rewriting/optimisation of queries using views (e.g. see [Ullman 1997; Halevy 2001]). In this paper, we examine the case when views are used, in a somewhat different manner, as the basic unit for writing logical expressions. We provide results on the related decision problem in this paper, for a range of possible view definitions. It is well-known that unions of conjunctive queries are the most frequently asked queries in database [Abiteboul et al. 1995]. For this reason, we shall consider view definitions with such flavour. In particular, for the case where views are monadic/unary conjunctive queries, we show that the corresponding first-order query logic is decidable. This corresponds to an interesting new fragment of first order logic and the fragment extends the well known Löwenheim class. On the application side, this decidable query language also has some interesting potential applications for areas such as implication of complex dependencies [Cosmadakis et al. 1990], ontology reasoning [Horrocks et al. 2003; Horrocks 2005] and termination results for active rules [Bailey et al. 1998].

### 1.1 Informal Statement of the Problem

Consider a relational vocabulary  $R_1, \dots, R_p$  and a set of views  $V_1, \dots, V_n$ . Each view definition corresponds to a first order formula over the vocabulary. Some example views (using horn clause style notation) are

$$\begin{aligned} V_1(x_1, y_1) &\leftarrow R_1(x_1, y_1), R_2(y_1, y_1, z_1), R_3(z_1, z_2, x_1), \neg R_4(z_2, x_1) \\ V_2(z_1) &\leftarrow R_1(z_1, z_1) \end{aligned}$$

Each such view can be expanded into to a first order sentence, e.g.  $V_1(x_1, y_1) \Leftrightarrow \exists z_1, z_2 (R_1(x_1, y_1) \wedge R_2(y_1, y_1, z_1), R_3(z_1, z_2, x_1) \wedge \neg R_4(z_2, x_1))$ . A *first order view query* is a first order formula expressed *solely* in terms of the given views. e.g.  $q_1 = \exists x_1, y_1 ((V_1(x_1, y_1) \vee V_1(y_1, x_1)) \wedge \neg V_2(x_1)) \wedge \forall z_1 (V_2(z_1) \Rightarrow V_1(z_1, z_1))$  is an example first order view query, but  $q_2 = \exists x_1, y_1 (V_1(x_1, y_1) \vee R(y_1, x_1))$  is not. By expanding the view definitions, every first order view query can clearly be rewritten to eliminate the views. Hence, first order view queries can be thought of as a fragment of first order logic, with the exact nature of the fragment varying according to how expressive the views are permitted to be.

From a database perspective, first order view queries are particularly suited to applications where the source data is unavailable, but summary data (in the form of views) is. Since many database and reasoning languages are based on first order logic (or extensions thereof), this makes it a useful choice for manipulating the views.

Our purpose in this paper is to determine, for what types of view definitions, satisfiability (over both finite and infinite models) is decidable for the language. If views can be binary, then this language is clearly as powerful as first order logic over binary base relations, and hence undecidable (see [Boerger et al. 1996]). The situation becomes far more interesting, when we restrict the form that views may take — in particular, when their arity must be unary. Such a restriction has the

effect of constraining which parts of the underlying database can be “seen” by the view formula and also constrains how such parts may be connected.

## 1.2 Contributions

In this paper, we study first-order logic over unary conjunctive views, which we call the *first-order unary conjunctive view language* (UCV). The logic UCV can be construed as an extension of monadic first-order logic (a.k.a. Löwenheim class), which is a well-known decidable fragment of first-order logic [Boerger et al. 1996]. Unlike the Löwenheim class, however, UCV is intimately related to database relations of higher arity as their use is permitted inside the views. Our main results can be summarised as follows:

- (1) We show that satisfiability for UCV is decidable in 2-NEXPTIME, but is NEXPTIME-hard. The proof is done by first showing that UCV has a small model property. More precisely, we show that every satisfiable UCV formula has a model of at most doubly exponential size.  
Like the Löwenheim class, the decidability of UCV logic is related to determining which of  $2^n$  (where  $n$  is the number of unary predicates) atomic types is realizable. However, our decidability result does not follow from the small model property of the Löwenheim class, since view definitions in UCV logic can be interdependent. Views may either intersect, be contained in each other or be mutually exclusive. This means that the  $2^n$  atomic types of a UCV formula can thus be interdependent. The possible interactions between them can be very subtle and attempting to encode such relationships for a model requires the development of new techniques.
- (2) We study four extensions of UCV and show that all of them lead to undecidable logics. We show that allowing inequalities, negations, or recursions inside the conjunctive views leads to undecidability. Also, allowing binary views leads to undecidability.
- (3) We argue that the language is able to express some interesting properties, which might be applied to various kinds of reasoning over ontologies [Horrocks et al. 2003; Horrocks 2005]. It can also be thought of as a powerful generalisation of unary inclusion dependencies [Cosmadakis et al. 1990]. Furthermore, it has an interesting characterisation as a decidable class of rules (triggers) for active databases.
- (4) We give a toolbox for showing inexpressibility in UCV logic, which we use to show that over graphs, first-order definable queries such as transitivity, and symmetry are inexpressible in UCV.

To briefly give a feel for this decidable language, we next provide some example unary conjunctive views and a first order unary conjunctive view query defined over them:

$$\begin{aligned}
 V_1(x) &\leftarrow R_1(x, y), R_2(y, z), R_3(z, x'), R_4(x', x) \\
 V_2(x) &\leftarrow R_1(x, y), R_1(x, z), R_4(y, z) \\
 V_3(x) &\leftarrow R_1(x, y), R_1(x, z), R_4(y, y), R_4(z, x) \\
 V_4(x) &\leftarrow R_1(x, y), R_3(y, z), R_4(z, x'), R_4(x', y'), R_3(y', x) \\
 \exists x(V_2(x) \wedge \neg V_1(x)) \wedge \neg \exists y(V_3(y) \wedge \neg V_4(y))
 \end{aligned}$$

### 1.3 Paper Outline

The paper is structured as follows: Section 2 defines the necessary preliminaries and background concepts. Section 3 presents the definition of the logic UCV. Section 4 is the core section of the paper, where the decidability result for the class UCV is proved. Section 5 shows that extensions to the language, such as allowing negation, inequality or recursion in views, result in undecidability. Section 6 covers applications of the decidability results and then Section 7 provides some results on expressivity. Section 8 discusses related work and section 9 summarises and discusses future work.

## 2. PRELIMINARIES

In this section, we state basic definitions and relevant results. The reader is assumed to be familiar with standard results and notations from mathematical logic (e.g. see [Enderton 2001]). In the following, formulas are always first-order. The symbol  $\mathcal{FO}$  denotes the set of first order formulas over any vocabulary  $\sigma$ . In addition, if  $\mathcal{L} \subseteq \mathcal{FO}$  (i.e.  $\mathcal{L}$  is a fragment of  $\mathcal{FO}$ ), we denote by  $\mathcal{L}(\sigma)$  the set of formulas in  $\mathcal{L}$  over the vocabulary  $\sigma$ .

### 2.1 First-order logic

A (relational) *vocabulary*  $\sigma$  is a tuple  $\langle R_1, \dots, R_n \rangle$  of relation symbols with each  $R_i$  associated with a specified arity  $r_i$ . A (relational)  $\sigma$ -*structure*  $\mathbf{A}$  is the tuple

$$\langle A; R_1^{\mathbf{A}}, \dots, R_n^{\mathbf{A}} \rangle$$

where  $A$  is a non-empty set, called the *universe* (of  $\mathbf{A}$ ), and  $R_i^{\mathbf{A}}$  is an  $r_i$ -ary relation over  $A$  interpreting  $R_i$ . We refer to the elements in the set  $A$  as *the elements in*  $\mathbf{A}$ , or simply *constants*<sup>1</sup> (of  $\mathbf{A}$ ). In the sequel, we write  $R_i$  instead of  $R_i^{\mathbf{A}}$  when the meaning is clear from the context. We also use  $STRUCT(\sigma)$  to denote the set of all  $\sigma$ -structures. We assume a countably infinite set VAR of variables. An *instantiation* (or *valuation*) of a structure  $\mathbf{I}$  is a function  $v : \text{VAR} \rightarrow I$ . Extend this function to free tuples (i.e. tuple of variables) in the obvious way. We use the usual Tarskian notion of satisfaction to define  $\mathbf{I} \models \phi[v]$ , i.e., whether  $\phi$  is true in  $\mathbf{I}$  under  $v$ . If  $\phi$  is a sentence, we simply write  $\mathbf{I} \models \phi$ . The *image* of a structure  $\mathbf{I}$  under a formula  $\phi(x_1, \dots, x_n)$  is

$$\phi(\mathbf{I}) \stackrel{\text{def}}{=} \{v(x_1, \dots, x_n) : v \text{ is an instantiation of } \mathbf{I}, \text{ and } \mathbf{I} \models \phi[v]\}.$$

In particular, if  $n = 0$ , we have that  $\phi(\mathbf{I}) \neq \emptyset$  iff  $\mathbf{I} \models \phi$ . We say that two  $\sigma$ -structures  $\mathbf{A}$  and  $\mathbf{B}$  *agree* on  $\mathcal{L}$  iff for all  $\phi \in \mathcal{L}(\sigma)$  we have  $\mathbf{A} \models \phi \Leftrightarrow \mathbf{B} \models \phi$ .

Following the convention in database theory, the (*tuple*) *database*  $\mathcal{D}(\mathbf{A})$  *corresponding to the structure*  $\mathbf{A}$  (defined above) is the set

$$\{R_i(t) : 1 \leq i \leq n \text{ and } t \in R_i^{\mathbf{A}}\}.$$

<sup>1</sup>Although it is common in mathematical logic to use the term “constants” to mean the interpretation of constant symbols in the structure, no confusion shall arise in this article, as we assume the absence of constant symbols in the vocabulary. Our results, nevertheless, easily extend to vocabularies with constant symbols.

It is easy to see that such a database can be considered a structure with universe  $\text{adom}(\mathbf{A})$ , which is defined to be the set of all elements of  $\mathbf{A}$  occurring in at least one relation  $R_i$ , and relations built appropriately from  $\mathcal{D}(\mathbf{A})$ . Abusing terminologies, we refer to the elements of  $\mathcal{D}(\mathbf{A})$  as *tuples (associated with  $\mathbf{A}$ )*. In addition, when the meaning is clear from the context, we shall also abuse the term *free tuple* to mean an atomic formula  $R(u)$ , where  $R \in \sigma$  and  $u$  is a tuple of variables.

A formula  $\phi$  is said to be *satisfiable* if there exists a structure  $\mathbf{A}$  (either of finite or infinite size) such that  $\phi(\mathbf{A}) \neq \emptyset$ ; such a structure is said to be a *model* for  $\phi$ . We say that  $\phi$  is *finitely satisfiable* if there exists a finite structure  $\mathbf{I}$  such that  $\phi(\mathbf{I}) \neq \emptyset$ . Without loss of generality, we shall focus only on sentences when we are dealing with the satisfiability problem. In fact, if  $\phi$  has some free variables, taking its existential closure preserves satisfiability [Indeed we shall see that the languages we consider are closed under first-order quantification].

Given two  $\sigma$ -structures  $\mathbf{A}, \mathbf{B}$ , recall that  $\mathbf{A}$  is a *substructure* of  $\mathbf{B}$  (written  $\mathbf{A} \subseteq \mathbf{B}$ ) if  $A \subseteq B$  and  $R^{\mathbf{A}} \subseteq R^{\mathbf{B}}$  for every relation symbol  $R$  in  $\sigma$ . We say that  $\mathbf{A}$  is an *induced substructure* of  $\mathbf{B}$  (i.e. *induced* by  $A \subseteq B$ ) if for every relation symbol  $R$  in  $\sigma$ ,  $R^{\mathbf{A}} = R^{\mathbf{B}} \cap A^r$ , where  $r$  is the arity of  $R$ . Now, a *homomorphism* from  $\mathbf{A}$  to  $\mathbf{B}$  is a function  $h : A \rightarrow B$  such that, for every relation symbol  $R$  in  $\sigma$  and  $\mathbf{a} = (a_1, \dots, a_r) \in R^{\mathbf{A}}$ , it is the case that  $h(\mathbf{a}) \stackrel{\text{def}}{=} (h(a_1), \dots, h(a_r)) \in R^{\mathbf{B}}$ . An *isomorphism* is a bijective homomorphism whose inverse is a homomorphism.

The *quantifier rank*  $\text{qr}(\phi)$  of a formula  $\phi$  is the maximum nesting depth of quantifiers in  $\phi$ .

## 2.2 Views

For our purpose, a *view* over  $\sigma$  can be thought of as an arbitrary FO formula over  $\sigma$ . We say that a view  $V$  is *conjunctive* if it can be written as a *conjunctive query*, i.e. of the form

$$\exists x_1, \dots, x_n (R_1(u_1) \wedge \dots \wedge R_k(u_k))$$

where each  $R_i$  is a relation symbol, and each  $u_i$  is a *free tuple* of appropriate arity. We adopt the Horn clause (Datalog) style notation for writing conjunctive views. For example, if  $\{y_1, \dots, y_n\}$  is the set of free variables in the above conjunctive query, then we can rewrite it as

$$V(y_1, \dots, y_n) \leftarrow R_1(u_1), \dots, R_k(u_k)$$

where  $V(y_1, \dots, y_n)$  is called the *head* of  $V$ , and the conjunction  $R_1(u_1), \dots, R_k(u_k)$  the *body* of  $V$ . The *length* of the conjunctive view  $V$  is defined to be the sum of the arities of the relation symbols in the *multiset*  $\{R_1, \dots, R_k\}$ . For example, the lengths of the two views  $V$  and  $V'$  defined as

$$\begin{aligned} V(x) &\leftarrow E(x, y) \\ V'(x) &\leftarrow E(x, y), E(y, z) \end{aligned}$$

are, respectively, two and four. Additionally, if  $n = 1$  (i.e. has a head of arity 1), the view is said to be *unary*. *Unless stated otherwise, we shall say “view” to mean “unary-conjunctive view with neither equality nor negation in its body”.*

### 2.3 Graphs

We use standard definitions from graph theory (e.g. see [Diestel 2005]). A *graph* is a structure  $\mathbf{G} = (G, E)$  where  $E$  is a binary relation. If  $E$  is symmetric,  $\mathbf{G}$  is said to be *undirected*. The *girth* of a graph is the length of its shortest cycle. For two vertices  $x, y \in G$ , we denote their distance by  $d_{\mathbf{G}}(x, y)$  (or just  $d(x, y)$  when  $\mathbf{G}$  is clear from the context). For two sets  $S_1$  and  $S_2$  of vertices in  $\mathbf{G}$ , we define their distance to be

$$d_{\mathbf{G}}(S_1, S_2) := \min\{d_{\mathbf{G}}(a, b) : a \in S_1 \text{ and } b \in S_2\}.$$

In a weighted graph  $\mathbf{G}$  with weight  $w_{\mathbf{G}} : E \rightarrow \mathbb{N}$ , the weight  $w_{\mathbf{G}}(P)$  of a path  $P$  in  $\mathbf{G}$  is just  $\sum_{e \in E(P)} w_{\mathbf{G}}(e)$ . We shall write  $w$  instead of  $w_{\mathbf{G}}$  if the meaning is clear from the context. In the sequel, we shall frequently mention trees and forests. We always assume that any tree has a selected node, which we call a *root* of the tree. Given a tree  $\mathbf{T} = (T, E)$ , we can partition  $T$  according to the distance of the vertices from the root.

An undirected graph  $\mathbf{G} = (G, E)$  is said to be *k-regular* if each vertex in  $\mathbf{G}$  has exactly  $k$  neighbours. The graph  $\mathbf{G}$  is said to be *regular* if it is *k-regular* for some  $k$ .

The *Gaifman graph* (see [Gaifman 1982]) associated with a structure  $\mathbf{A}$  is the weighted undirected multi-graph  $\mathbb{G}(\mathbf{A}) = (G, E)$  such that:

- (1)  $G = A$ .
- (2) The multi-set  $E$  is defined as follows: for each  $x, y \in G$ , we put an  $R(t)$ -labeled edge  $xy$  in  $E$  with weight  $r$  (the arity of  $R$ ) iff  $x$  and  $y$  appear in a tuple  $R(t)$  in  $\mathcal{D}(\mathbf{A})$ . [Notice that the multiplicity of  $xy$  in  $E$  depends on the number of tuples in  $\mathcal{D}(\mathbf{A})$  that contain both  $x$  and  $y$  as their arguments.]

Note also that the subgraph of  $\mathbb{G}(\mathbf{A})$  induced by the set of all elements of  $\mathbf{A}$  in a tuple  $t$  is the complete graph  $K_r$ , and so an  $L$ -labelled edge is adjacent to an edge  $e \in E$  iff all  $L$ -labelled edges are adjacent (i.e. connected) to the edge  $e$ . For any  $a, b \in A$ , we define the *distance*  $d_{\mathbf{A}}(a, b)$  between  $a$  and  $b$  to be their distance in  $\mathbb{G}(\mathbf{A})$ . Also, extend this distance function to tuples and sets of tuples by interpreting them as sets of elements of  $\mathbf{A}$  that appear in them. Any pair of tuples  $R(t)$  and  $R'(t')$  in  $\mathcal{D}(\mathbf{A})$  are said to be *connected* (in  $\mathbf{A}$ ) if in  $\mathbb{G}(\mathbf{A})$  some (and hence all)  $R(t)$ -labeled edge is adjacent to some (and hence all)  $R'(t')$ -labeled edge.

### 2.4 Unary formulas

A *unary formula* is an arbitrary FO formula *without equality* such that each of its relation symbols has arity one. Let  $\sigma$  be a vocabulary whose relation symbols are of arity one. We shall use  $\text{UFO}(\sigma)$  to denote the set of all unary formulas without equality over  $\sigma$ . Also, we define  $\text{UFO} = \cup_{\sigma} \text{UFO}(\sigma)$ . The following lemma will be useful for proving expressiveness results in Section 7.

**LEMMA 2.1.** *For every unary sentence, there exists an equivalent one of quantifier rank 1.*

**PROOF.** By a straightforward manipulation. See the proof of lemma 21.12 in [Boolos et al. 2002]. [Their proof actually gives more than the result they claim.]

In fact, their construction converts an arbitrary unary sentence into one with one unary variable and of quantifier rank 1.]  $\square$

## 2.5 Ehrenfeucht-Fraïsse Games

We shall need a limited form of Ehrenfeucht-Fraïsse games; for a general account, the reader may consult [Libkin 2004]. The games are played by two players, Spoiler and Duplicator, on two  $\sigma$ -structures  $\mathbf{A}$  and  $\mathbf{B}$ . The goal of Spoiler is to show that the structures are different, while Duplicator aims to show that they are the same. The game consists of a single round. Spoiler chooses a structure (say,  $\mathbf{A}$ ) and an element  $a$  in it, after which Duplicator has to respond by choosing an element  $b$  in the other structure  $\mathbf{B}$ . Duplicator wins the game iff the substructure of  $\mathbf{A}$  induced by  $\{a\}$  is isomorphic to the substructure of  $\mathbf{B}$  induced by  $\{b\}$ . Duplicator has a winning strategy iff Duplicator has a winning move, regardless of how Spoiler behaves.

**PROPOSITION 2.2 (EHRENFUCHT-FRAÏSSE GAMES).** *Duplicator has a winning strategy on  $\mathbf{A}$  and  $\mathbf{B}$  iff  $\mathbf{A}$  and  $\mathbf{B}$  agree on first-order formulas over  $\sigma$  of quantifier rank 1.*

## 2.6 Other Notation

Regarding other notation we shall use throughout the rest of the paper: we shall use  $a, b$  for constants,  $x, y, z$  for variables,  $u$  for free tuples,  $U, V$  for views,  $\mathcal{U}, \mathcal{V}$  for sets of views,  $\sigma$  for vocabularies,  $R_1, R_2, \dots$  for relation symbols,  $\mathbf{A}, \mathbf{B}, \dots$  for structures and  $A, B$  for their respective universes. If  $\mathcal{D}$  is a database (a set of tuples), we use  $\text{adom}(\mathcal{D})$  to denote the set of constants in  $\mathcal{D}$ . Finally, given  $a \in \text{adom}(\mathcal{D})$  and a “new” constant  $b \notin \mathcal{D}$ , we define  $\mathcal{D}[b/a]$  to be the database that is obtained from  $\mathcal{D}$  by replacing every occurrence of  $a$  by  $b$ . The notation  $\mathcal{D}[b_1/a_1, \dots, b_n/a_n]$  is defined in the same way.

## 3. DEFINITION OF FIRST ORDER UNARY-CONJUNCTIVE-VIEW LOGIC

Let  $\sigma$  be an arbitrary vocabulary, and  $\mathcal{V}$  be a finite set of (unary conjunctive) views over  $\sigma$ , which we refer to as a  $\sigma$ -view set. We now inductively define the set  $\text{UCV}(\sigma, \mathcal{V})$  of *first order unary-conjunctive-view (UCV) queries/formulas over the vocabulary  $\sigma$  and a  $\sigma$ -view set  $\mathcal{V}$* :

- (1) if  $V \in \mathcal{V}$ , then  $V(x) \in \text{UCV}(\sigma, \mathcal{V})$ ; and
- (2) if  $\phi, \psi \in \text{UCV}(\sigma, \mathcal{V})$ , then the formulas  $\neg\phi, \phi \wedge \psi$  and  $\exists x\phi$  belong to  $\text{UCV}(\sigma, \mathcal{V})$ .

The smallest set of so-constructed formulas defines the set  $\text{UCV}(\sigma, \mathcal{V})$ . We denote the set of all UCV formulas over the vocabulary  $\sigma$  by  $\text{UCV}(\sigma)$ , i.e.  $\text{UCV}(\sigma) \stackrel{\text{def}}{=} \bigcup_{\mathcal{V}} \text{UCV}(\sigma, \mathcal{V})$  where  $\mathcal{V}$  may be any  $\sigma$ -view set. Further, the set of all UCV queries is denoted by  $\text{UCV}$ , i.e.  $\text{UCV} \stackrel{\text{def}}{=} \bigcup_{\sigma} \text{UCV}(\sigma)$ , where  $\sigma$  is any vocabulary. As usual, we use the shorthands  $\phi \vee \psi, \phi \rightarrow \psi, \phi \leftrightarrow \psi$ , and  $\forall x\phi$  for (respectively)  $\neg(\neg\phi \wedge \neg\psi), \neg\phi \vee \psi, (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$ , and  $\neg\exists x\neg\phi$ . Thus, the UCV language is closed under boolean combinations and first-order quantifications. As an example, consider the UCV formula

$$q_1 = \exists x(V(x) \wedge \neg V'(x))$$

where  $V$  and  $V'$  are defined as

$$\begin{aligned} V(x) &\leftarrow E(x, y) \\ V'(x) &\leftarrow E(x, y), E(y, z) \end{aligned}$$

This formula asserts that there exists a vertex from which there is an outgoing arc, but no outgoing directed walk of length 2.

Let us make a few remarks on the expressive power of the logic UCV with respect to other logics. It is easy to see that the UCV language strictly subsumes UFO (the Löwenheim class without equality [Löwenheim 1915; Börger et al. 1997]), as UCV queries can be defined over *any* relational vocabularies (i.e. including ones that include  $k$ -ary relation symbols with  $k > 1$ ). It is also easy to see that allowing any general existential positive formula (i.e. one of the form  $\exists \bar{x} \phi(\bar{x})$  where  $\phi$  is a quantifier-free formula with no negation) with one free variable, does not increase the expressive power of the logic. Indeed, the quantifier-free subformula  $\phi$  can be rewritten in disjunctive normal form without introducing negation, after which we may distribute the existential quantifier across the disjunctions and consequently transform entire formula to a disjunction of conjunctive queries with one or zero free variables. Each such conjunctive query can then be treated as a view.

There are two ways in which we can interpret a UCV formula. The standard way is to think of a UCV query as an FO formula over the underlying vocabulary. Take the afore-mentioned query  $q_1$  as an example. We can interpret this query as the formula

$$\exists x (\exists y (E(x, y)) \wedge \neg \exists y, z (E(x, y) \wedge E(y, z)))$$

over the graph vocabulary. The non-standard way is to regard a UCV query  $\phi$  as a unary formula over the view set. For example, we can think of  $q_1$  as a unary formula over the vocabulary  $\sigma' = \langle V, V' \rangle$ . Now, if  $\phi \in \text{UCV}(\sigma, \mathcal{V})$ , then we denote by  $\phi^{\mathcal{V}}$  the unary formula over  $\mathcal{V}$  corresponding to  $\phi$  in the non-standard interpretation of UCV queries. However, for notational convenience, we shall write  $\phi$  instead of  $\phi^{\mathcal{V}}$  when the meaning is clear from the context. Given a vocabulary  $\sigma$  and a  $\sigma$ -view set  $\mathcal{V} = \{V_1, \dots, V_n\}$ , we may define the function  $\Lambda : \text{STRUCT}(\sigma) \rightarrow \text{STRUCT}(\mathcal{V})$  such that for any  $\mathbf{I} \in \text{STRUCT}(\sigma)$

$$\Lambda(\mathbf{I}) \stackrel{\text{def}}{=} \langle I; V_1^{\Lambda(\mathbf{I})}, \dots, V_n^{\Lambda(\mathbf{I})} \rangle$$

where  $V_i^{\Lambda(\mathbf{I})} \stackrel{\text{def}}{=} V_i(\mathbf{I})$ . For example, let  $\sigma = \langle E \rangle$  and  $\mathcal{V} = \{V, V'\}$  be as above, and let

$$\mathbf{I} = \langle \{1, 2, 3, 4\}; E^{\mathbf{I}} = \{(1, 2), (2, 3), (3, 4)\} \rangle.$$

Then, we have

$$\mathbf{J} \stackrel{\text{def}}{=} \Lambda(\mathbf{I}) = \langle \{1, 2, 3, 4\}, V^{\mathbf{J}} = \{1, 2, 3\}, V'^{\mathbf{J}} = \{1, 2\} \rangle.$$

In the following, we shall reserve the symbol  $\Lambda$  to denote this special function. In addition, if  $\mathbf{J} \in \text{STRUCT}(\mathcal{V})$  and there exists a structure  $\mathbf{I} \in \text{STRUCT}(\sigma)$  such that  $\Lambda(\mathbf{I}) = \mathbf{J}$ , we say that the structure  $\mathbf{J}$  is *realizable* with respect to the

vocabulary  $\sigma$  and the view set  $\mathcal{V}$ , or that  $\mathbf{I}$  realizes  $\mathbf{J}$ . We shall omit mention of  $\sigma$  and  $\mathcal{V}$  if they are understood by context.

A number of remarks about the notion of realizability are in order. First, some unary structures are *not* realizable with respect to a given view set  $\mathcal{V}$ . For example, if using the view definitions for  $V$  and  $V'$  from  $q_1$  above, the formula

$$q_2 = \exists x(V'(x) \wedge \neg V(x))$$

has infinitely many models if treated as a unary formula (encoding  $V$  and  $V'$  as relation symbols rather than views). However none of these models are realizable, since  $V' \subseteq V$ . Second, if  $\phi \in \text{UCV}(\sigma, \mathcal{V})$  has a model  $\mathbf{I}$ , then the structure  $\Lambda(\mathbf{I})$  over  $\mathcal{V}$  is a model for  $\phi^{\mathcal{V}}$ . In other words, if a UCV query is satisfiable, then it is also satisfiable if treated as a unary formula. Conversely, it is also clearly true that a UCV query is satisfiable, if it is satisfiable when treated as a unary formula and that at least one of its models is realizable. More precisely, if  $\Lambda(\mathbf{I})$  is a model for  $\phi^{\mathcal{V}}$ , then  $\mathbf{I}$  is a model for  $\phi$ . So, combining these, we have  $\mathbf{I} \models \phi$  iff  $\Lambda(\mathbf{I}) \models \phi^{\mathcal{V}}$ . So, we immediately have the following lemma:

LEMMA 3.1. *Suppose  $\mathbf{A}, \mathbf{B} \in \text{STRUCT}(\sigma)$  and  $\phi \in \text{UCV}(\sigma, \mathcal{V})$ . Then, for  $\Lambda : \text{STRUCT}(\sigma) \rightarrow \text{STRUCT}(\mathcal{V})$  defined above, the following statements are equivalent:*

- (1)  $\mathbf{A} \models \phi$  iff  $\mathbf{B} \models \phi$ ,
- (2)  $\Lambda(\mathbf{A}) \models \phi^{\Lambda}$  iff  $\Lambda(\mathbf{B}) \models \phi^{\Lambda}$ .

This lemma is useful when combined with Ehrenfeucht-Fraïsse games. For example, suppose that we are given a model  $\mathbf{A}$  for  $\phi$ , and we construct a “nicer” structure  $\mathbf{B}$  that, we wish, satisfies  $\phi$ . If we can prove that the second statement in the lemma (which is often easier to establish as views have arity one), we might deduce that  $\mathbf{B} \models \phi$ .

#### 4. DECIDABILITY OF UCV QUERIES

In this section, we prove our main result that satisfiability is decidable for UCV formulas. Our main theorem stipulates that UCV has the bounded model property.

THEOREM 4.1. *Let  $\phi$  be a formula in UCV. Suppose, further, that  $\phi$  contains precisely the views in the view set  $\mathcal{V}$ , and relation symbols in the vocabulary  $\sigma$ , with  $m$  being the maximum length of the views in  $\mathcal{V}$ , and  $p = |\sigma|$ . If  $\phi$  is satisfiable, then it has a model using at most  $2^{2^{q(p,m)}}$  elements, for some fixed polynomial  $q$  in  $p$  and  $m$ .*

Before we prove this theorem, we first derive some corollaries. Simple algebraic manipulations yield the following corollary.

COROLLARY 4.2. *Continuing from Theorem 4.1, if  $n$  is the size of (the parse tree of) a satisfiable formula  $\phi$ , then  $\phi$  has a model of size  $2^{2^{g(n)}}$  for some fixed polynomial  $g$  in  $n$ .*

Corollary 4.2 immediately leads to the decidability of satisfiability for UCV. We can in fact derive a tighter bound.

THEOREM 4.3. *Satisfiability for the UCV class of formulas is in 2-NEXPTIME.*

This theorem follows immediately from the following proposition and corollary 4.2.

PROPOSITION 4.4. *Let  $s$  be a non-decreasing function with  $s(n) \geq n$ . Then, the problem of determining whether an FO sentence has a model of size at most  $s(n)$ , where  $n$  is the size of the input formula, can be decided nondeterministically in  $2^{O(n \log(s(n)))}$  steps.*

PROOF. We may use any reasonable encoding code( $\mathbf{A}$ ) of a finite structure  $\mathbf{A}$  in bits (e.g. see [Libkin 2004, Chapter 6]). The size of the encoding, denoted  $|\mathbf{A}|$ , is polynomial in  $|A|$ . We first guess a structure  $\mathbf{A}$  of size at most  $s(n)$ . Let  $s' = |A|$ . Since the size  $|\mathbf{A}|$  of the encoding of  $\mathbf{A}$  is polynomial in  $s'$ , the guessing procedure takes  $O(s^k(n))$  time steps for some constant  $k$ . We, then, use the usual procedure for evaluating whether  $\mathbf{A} \models \phi$ . This can be done in  $O(n \times |\mathbf{A}|^n)$  steps (e.g. see [Libkin 2004, Proposition 6.6]). Simple algebraic manipulations give the sought after upper bound.  $\square$

Observe that a lower bound for satisfiability of UCV formulas follows immediately from the NEXPTIME completeness for satisfiability of UFO formulas given in [Börger et al. 1997]

THEOREM 4.5. *Satisfiability for the UCV class of formulas is NEXPTIME hard.*

What remains now is to prove theorem 4.1.

PROOF OF THEOREM 4.1. Let  $\phi, m, p$  be as stated in theorem 4.1. We begin by first enumerating all possible views over  $\sigma$  of length at most  $m$ . As we shall see later in the proof of Subproperty 4.14, doing so will help facilitate the correctness of our construction of a finite model, since enumerating all such views effectively allows us to determine all possible ways the model may be “seen” by views, or parts of views. Let  $\mathcal{U} = \{V_1, \dots, V_N\}$  be the set of all non-equivalent views obtained. By elementary counting, one may easily verify that  $N \leq m(mp)^m$ . Indeed, each view is composed of its head and its body, whose length is bounded by  $m$ . The body is a set of conjuncts that we may fix in some order. There are at most  $m$  variables that the head can take. Each position in the body is a variable ( $m$  choices) that is part of a relation  $R$  ( $p$  choices). The upper bound is then immediate.

Let  $\mathbf{I}_0$  be a (possibly infinite) model for  $\phi$ . [If it is infinite, by the Löwenheim-Skolem theorem, we may assume that it is countable.] Without loss of generality, we may assume that there exists a “universe” relation  $U$  in  $\mathbf{I}_0$  which contains each constant in  $\text{adom}(\mathbf{I}_0)$ . Otherwise, if  $U' \notin \sigma$  is a unary relation symbol, the  $(\sigma \cup \{U'\})$ -structure obtained by adding to  $\mathbf{I}_0$  the relation  $U'$ , which is to be interpreted as  $I_0$ , is also a model for  $\phi$ .

Let us now define  $2^N$  formulas  $C_0, \dots, C_{2^N-1}$  of the form

$$C_i(x) \stackrel{\text{def}}{=} (\neg)V_1(x) \wedge \dots \wedge (\neg)V_N(x),$$

where the conjunct  $V_j(x)$  is negated iff the  $j$ th bit of the binary representation of  $i$ , which we denote by  $\text{bit}_j(i)$ , is 0. For each  $\mathbf{A} \in \text{STRUCT}(\sigma)$ , these formulas induce an equivalence relation on  $A$  with each set  $C_i(\mathbf{A})$  being an equivalence class. When

**A** is clear, we refer to the equivalence class  $C_i(\mathbf{A})$  simply as  $C_i$ . In addition, the existence of the universe relation  $U$  in  $\mathbf{I}_0$  implies that the all-negative equivalence class  $C_0$  is empty.

We next describe a sequence of five satisfaction-preserving procedures for deriving a finite model from  $\mathbf{I}_0$ . This sequence is best described diagrammatically:

$$\mathbf{I}_0 \xrightarrow{\text{makeJF}} \mathbf{I}_1 \xrightarrow{\text{rename1}} \mathbf{I}_2 \xrightarrow{\text{rename2}} \mathbf{I}_3 \xrightarrow{\text{copy}} \mathbf{I}_4 \xrightarrow{\text{prune}} \mathbf{I}_5.$$

The  $i$ th procedure above takes a structure  $\mathbf{I}_i$  as input, and outputs another structure  $\mathbf{I}_{i+1}$ . The structure  $\mathbf{I}_5$  is guaranteed to be finite (and indeed bounded). That each procedure preserves satisfiability immediately follows by subproperties 4.8, 4.10, 4.12, 4.13, and 4.14. While reading the description of the procedures below, it is instructive to keep in mind that the property that  $C_i(\mathbf{I}_j) = \emptyset$  iff  $C_i(\mathbf{I}_{j+1}) = \emptyset$  is sufficient for showing that the  $j$ th procedure preserves satisfiability (see lemma 4.7).

We first give some intuitive description of our construction. Suppose that  $a \in C_i(\mathbf{I}_0)$ . This means that there are some tuples in  $\mathbf{I}_0$  that witness/justify  $a \in V_j(\mathbf{I}_0)$ , for each view  $V_j$  that appears positively in  $C_i$ . Roughly speaking, the procedure **makeJF** will construct a certain forest-like graphical representation of  $\mathbf{I}_0$ , called a “justification forest”. The number of trees in the forest will correspond to the number of nonempty equivalence classes  $\{C_i\}$ , which is at most  $2^N$ . The root node of each tree will contain a set of tuples that justify some chosen element of the associated equivalence class, which we call a “justification set”. The elements of tuples in justification sets are in turn, justified in their children and so on. Notice that the height of the trees might be infinite. Our problem is, then, to somehow make these trees of finite height by pruning (chopping) the trees and justifying the leaves using the tuples that are in other trees. In doing so, one has to be careful that each time we modify the structure, we do not introduce tuples that will make some empty equivalence class nonempty. We shall approach this problem as follows. Observe that two trees might share some common constants and are in some sense interdependent. We first remove this interdependence by assigning distinct colors to constants in different trees, which is the goal of procedure **rename1**. Even then, two distinct levels in a tree might share some common constants and hence be interdependent. The procedure **rename2** shall ensure that a constant can only appear at only two consecutive levels  $j$  and  $j + 1$  in a tree, where level  $j$  is the first time the constant appears and level  $j + 1$  contains the justification set for the constant. The procedure **copy**, then, creates a large number of isomorphic copies of each tree so that we have enough trees to justify each node at a sufficiently deep level (in this case doubly exponentially large in  $N$ ). We finally apply the procedure **prune** to prune each tree, making it of finite height, and then link each leaf node with a number of root nodes of some other trees, so that each constant in the leaf node is properly justified.

*The procedure **makeJF***

We define the structure  $\mathbf{I}_1$  by first defining a sequence  $\mathbf{I}_1^0, \mathbf{I}_1^1, \dots$  of structures such that  $\mathbf{I}_1^k$  is a substructure of  $\mathbf{I}_1^{k+1}$ , and then setting  $\mathbf{I}_1 = \bigcup_{k=0}^{\infty} \mathbf{I}_1^k$ . [Note: we take the normal union, not *disjoint union*.] We first deal with the base case of  $\mathbf{I}_1^0$ . For each non-empty equivalence class  $C_i(\mathbf{I}_0)$ , we choose a witnessing constant  $a_i \in C_i(\mathbf{I}_0)$ .

We define  $I_1^0$  as the collection of all such  $a_i$ s. All relations in  $\mathbf{I}_1^0$  are empty. Each  $a_i$  is said to be *unjustified* in  $\mathbf{I}_1^0$ , meaning that the model is missing tuples that can witness the truth of  $a_i$  being a member of some equivalence class. We now describe how to define  $\mathbf{I}_1^{k+1}$  from  $\mathbf{I}_1^k$ . For each  $a \in I_1^k$ , if  $a \in C_i(\mathbf{I}_0)$  for some  $i$ , it is the case that  $a \in V_j(\mathbf{I}_0)$  iff  $\text{bit}_j(i) = 1$  for  $1 \leq j \leq N$ . For such  $a$ , we may take a minimal witnessing substructure  $\mathbf{S}_a$  of  $\mathbf{I}_0$  such that  $a \in V_j(\mathbf{S}_a)$  iff  $\text{bit}_j(i) = 1$ . As each constant in  $\text{adom}(\mathbf{S}_a)$  appears in at least one relation in  $\mathbf{S}_a$ , we shall often think of these witnessing structures as databases (i.e. sets of tuples), and refer to them as *justification sets*. We define the structure  $\mathbf{I}_1^{k+1}$  to be the union of  $\mathbf{I}_1^k$  and all the witnessing structures  $\mathbf{S}_a$  such that  $a$  is unjustified in  $\mathbf{I}_1^k$ . The elements in  $I_1^k$  become *justified* in  $\mathbf{I}_1^{k+1}$ . The elements in  $I_1^{k+1} - I_1^k$  are then said to be *unjustified* in  $\mathbf{I}_1^{k+1}$ . Observe that the structure  $\mathbf{I}_1^{k+1}$  does not unjustify any elements that were justified in  $\mathbf{I}_1^k$ , since there is no negation in the view definitions. Finally, the structure  $\mathbf{I}_1$  is defined as the union of all  $\mathbf{I}_1^k$ s. Observe that each element in  $I_1$  appears in at least one relation in  $\mathbf{I}_1$ .

The structure  $\mathbf{I}_1$  has an intuitive graphical representation, which we denote by  $\mathcal{H}_1$ . The graph  $\mathcal{H}_1$  is simply a labeled forest in which each tree  $T_i$  (for some  $0 \leq i \leq 2^N - 1$ ) corresponds to exactly one witnessing constant  $a_i$  for each non-empty  $C_i$ . We define  $T_i$  as follows: the root of  $T_i$  is labeled by  $\mathbf{S}_{a_i} \times C_i$ ; and for each  $j = 0, 1, \dots$ , any  $\mathbf{S}_b \times C_k$ -labeled node  $v$  at level  $j$  (for some justification set  $\mathbf{S}_b$  and equivalence class formula  $C_k$ ), and any constant  $c$  in  $\text{adom}(\mathbf{S}_b)$  that is distinct from  $b$ , define a new  $\mathbf{S}_c \times C_{k'}$ -labeled node to be a child of  $v$ , for the unique  $k'$  such that  $c \in C_{k'}(\mathbf{I}_0)$ . In the following, when the meaning is clear, we shall often refer to an  $(\mathbf{S}_a \times C_k)$ -labeled node simply as a  $\mathbf{S}_a$ -labeled node. Also, observe the similarity of the construction of  $\mathcal{H}_1$  and that of  $\mathbf{I}_1$ . In fact, the union of all  $\mathbf{S}_a$ , for which there is an  $\mathbf{S}_a$ -labeled node in  $\mathcal{H}_1$ , is precisely  $\mathbf{I}_1$ . Observe also that each tree  $T_i$  may be infinite. For obvious reasons, we shall refer to  $T_i$  as a *justification tree* (of  $a_i$ ), and to  $\mathcal{H}_1$  as *justification forest*. In the following, for any justification tree  $T$  and any justification forest  $\mathcal{H}$ , their *corresponding structures* (or *databases*), denoted by  $\mathcal{D}(T)$  and  $\mathcal{D}(\mathcal{H})$  respectively, are defined to be the union of all  $\mathbf{S}_a$ , such that there is an  $\mathbf{S}_a$ -labeled node in, respectively,  $T$  and  $\mathcal{H}$ . Furthermore, we shall use  $\text{adom}(T)$  and  $\text{adom}(\mathcal{H})$  to denote  $\text{adom}(\mathcal{D}(T))$  and  $\text{adom}(\mathcal{D}(\mathcal{H}))$ , respectively. The elements in the set  $\text{adom}(T)$  and  $\text{adom}(\mathcal{H})$  are referred to as, respectively, constants in  $T$  and constants in  $\mathcal{H}$ .

We now illustrate this procedure by a small example. Define the UCV formula

$$\phi = \forall x(V_1(x) \wedge \neg V_2(x)),$$

where the views are

$$\begin{aligned} V_1(x) &\leftarrow E(x, y) \\ V_2(x) &\leftarrow E(x, x). \end{aligned}$$

Here, we have  $\mathcal{V} = \{V_1, V_2\}$ ,  $\sigma = \langle E \rangle$ , and  $m = 2$ . Suppose that

$$\mathbf{I}_0 = \langle \mathbb{N}, E = \{(0, 1), (1, 2), (2, 3), (3, 4), \dots\} \rangle$$

is a path extending indefinitely to the right. Then, we have  $\mathbf{I}_0 \models \phi$ . Enumerating

all non-equivalent views over  $\sigma$  of length at most  $m$ , we have  $\mathcal{U} = \{V_1, V_2, V_3\}$  where

$$V_3(x) \leftarrow E(y, x).$$

Now, there are exactly two non-empty equivalence classes:

$$\begin{aligned} C_{100} &= \{0\} \\ C_{101} &= \{1, 2, \dots\}. \end{aligned}$$

Then, we have  $\mathbf{S}_0 = \{E(0, 1)\}$ , and  $\mathbf{S}_i = \{E(i-1, i), E(i, i+1)\}$  for  $i > 0$ . Following the above procedure, we obtain the trees  $T_{100}$  and  $T_{101}$  as depicted in figure 1. Note that  $\mathcal{H}_1$  is the disjoint union of  $T_{100}$  and  $T_{101}$ .

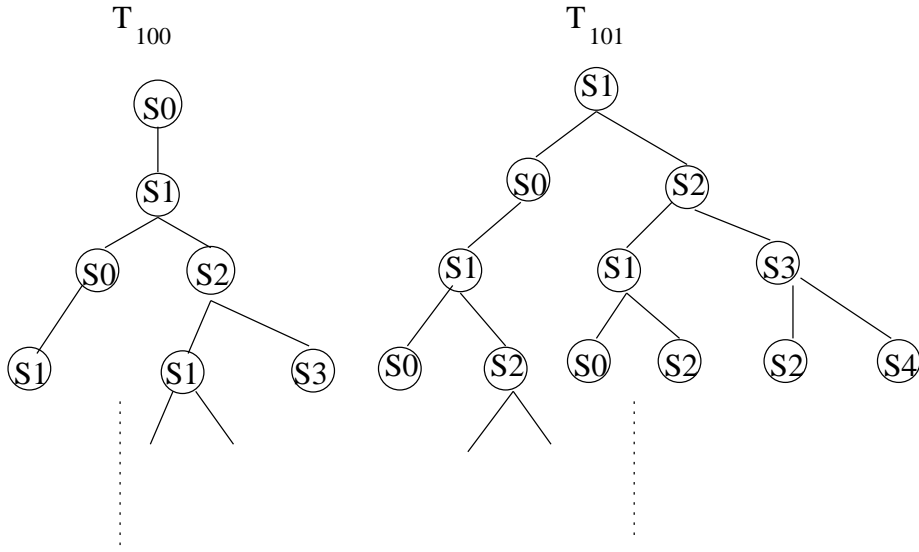


Fig. 1. A depiction of the justification forest  $\mathcal{H}_1$  as an output of `makeJF`. Note that  $\mathbf{S}_0 = \{E(0, 1)\}$ , and  $\mathbf{S}_i = \{E(i-1, i), E(i, i+1)\}$  for  $i > 0$ .

#### The procedure `rename1`

*Proviso: in subsequent procedures (including the present one), we shall not change the second entries (i.e.  $C_i$ ) of each node label (i.e. of the form  $\mathbf{S}_a \times C_i$ ) and frequently omit mention of them.*

The aim of this procedure is to ensure that there are no two justification trees  $T$  and  $T'$  with  $\text{adom}(T) \cap \text{adom}(T') \neq \emptyset$ . It essentially performs renaming of constants in  $\text{adom}(T)$ , for each tree  $T$  in  $\mathcal{H}_1$ . This step will later help us guarantee the correctness of the last step that is used to produce the final model  $\mathbf{I}_5$ , which relies on a kind of “tree disjointness” property. More formally, we define  $\mathbf{I}_2$  to

be the disjoint union<sup>2</sup> of  $\mathcal{D}(T)$  over all trees  $T$  in  $\mathcal{H}_1$ . The justification forest  $\mathcal{H}_2$  corresponding to  $\mathbf{I}_2$  can be obtained from  $\mathcal{H}_1$  by renaming constants of the tuples in each tree  $T$  in  $\mathcal{H}_1$  accordingly.

Let us continue with our previous example of  $\mathcal{H}_1$ . The graph  $\mathcal{H}_2$  in this case will be precisely identical to  $\mathcal{H}_1$ , except that in  $T_{101}$  we use the label, say,  $\mathbf{S}_{0'}$  =  $\{E(0', 1')\}$  (resp.  $\mathbf{S}_{i'} = \{E((i-1)', i'), E(i', (i+1)')\}$  for  $i > 0$ ) instead of  $\mathbf{S}_0$  (resp.  $\mathbf{S}_i$  for  $i > 0$ ).

*The procedure `rename2`*

The aim of this procedure is to transform the model in such a way that each constant  $a$  can appear only at two consecutive levels, say  $j$  and  $j+1$ , within each tree. It appears at level  $j$  as part of an  $S_b$ -labeled node  $v$ , for some constant  $b \neq a$ , and at level  $j+1$  as part of an  $S_a$ -labeled node that is a child of  $v$ . Further, the procedure ensures that any given constant occurs in at most one node's label at each level in a tree. Again, this step will later help us guarantee the correctness of the step that is used to produce the final model  $\mathbf{I}_5$ , which relies on the existence of a kind of internal “disjointness” property within trees.

Let us fix a sibling ordering for the nodes within each tree  $T_i$  in  $\mathcal{H}_2$ . Define a set  $U$  of constants disjoint from  $I_2$  as follows:

$$U = \{a_{j,l} : j, l \in \mathbb{N} \text{ and } a \in I_2\}.$$

For  $a, b \in I_2$ , we require that  $a_{j,l} \neq b_{j',l'}$  whenever either  $j \neq j'$ , or  $l \neq l'$ , or  $a \neq b$ . For each tree  $T_i$  and for each  $j = 1, 2, \dots$ , choose the  $l$ th node  $v$  with respect to the fixed sibling ordering (say,  $\mathbf{S}_a$ -labeled) at level  $j$  in  $T_i$ . Let  $v$ 's children be  $v_1, \dots, v_k$  (labeled by, respectively,  $\mathbf{S}_{b^1}, \dots, \mathbf{S}_{b^k}$  with  $b^h \neq a$ ). Now do the following: change  $v$  to  $\mathbf{S}_a[b_{j,l}^1, \dots, b_{j,l}^k/b^1, \dots, b^k]$ ; and change  $v_h$ , where  $1 \leq h \leq k$ , to  $\mathbf{S}_{b_{j,l}^h} \stackrel{\text{def}}{=} \mathbf{S}_{b^h}[b_{j,l}^h/b^h]$ . Observe that there are two stages in this procedure where each non-root node at level  $j$ , say  $\mathbf{S}_a$ -labeled, undergoes relabeling: first when we are at level  $j-1$  (the constant  $a$  is renamed by  $a_{j,k}$  for some  $k$ ), and second when we are at level  $j$  (constants other than  $a_{j,k}$  are renamed for what is now  $\mathbf{S}_{a_{j,k}}$ ). The output of this procedure on  $\mathcal{H}_2$  is denoted by  $\mathcal{H}_3$ , whose corresponding structure we denote by  $\mathbf{I}_3$ .

Continuing with our previous example. The root node  $u_1$  of  $T_{100}$  in  $\mathcal{H}_2$  is  $\mathbf{S}_0 = \{E(0, 1)\}$ , its child  $u_2$  (sibling zero at level 1) is  $\mathbf{S}_1 = \{E(0, 1), E(1, 2)\}$  and in turn the children of that child are  $u_3 = \mathbf{S}_0 = \{E(0, 1)\}$  (sibling 0 at level 2) and  $u_4 = \mathbf{S}_2 = \{E(1, 2), E(2, 3)\}$  (sibling 1 at level 2). Under the `rename2` procedure, node  $u_1$  is unchanged, since it is at level zero. Node  $u_2$  is changed to  $\mathbf{S}_1 = \{E(0_{1,0}, 1), E(1, 2_{1,0})\}$ . Node  $u_3$  is changed to  $\mathbf{S}_{0_{1,0}} = \{E(0_{1,0}, 1_{2,0})\}$  and  $u_4$  is changed to  $\mathbf{S}_{2_{1,0}} = \{E(1_{2,1}, 2_{1,0}), E(2_{1,0}, 3_{2,1})\}$ .

*The procedure `copy`*

This procedure makes a number of isomorphic copies of the model  $\mathcal{H}_3$  and then unions them together. Duplicating the model in this way facilitates the construction

<sup>2</sup>The disjoint union of two  $\sigma$ -structures  $\mathbf{A}$  and  $\mathbf{B}$  with  $A \cap B = \emptyset$  is the structure with universe  $A \cup B$  and relation  $R$  interpreted as  $R^{\mathbf{A}} \cup R^{\mathbf{B}}$ . If  $A \cap B \neq \emptyset$ , one can simply force disjointness by renaming constants.

of a bounded model by the **prune** procedure, that will be described shortly. Let  $\delta$  be the total number of constants that appear in some tuples from a node label at level  $h := cm$  in  $\mathcal{H}_3$ , for some fixed  $c \in \mathbb{N}$ , independent from  $\phi$ , whose value will later become clear in the proofs that follow. By virtue of procedure **makeJF**, we are guaranteed that each node in  $\mathcal{H}_3$  can have at most  $N \times m$  children, where  $N \times m$  represents an upper bound on the number of constants each justification set might contain. Since there are at most  $2^N$  trees in  $\mathcal{H}_3$ , by elementary counting, we see that  $\delta \leq 2^N \times (N \times m)^h$ . Now, letting  $g := cm$ , make  $\Delta := \delta^g$  (isomorphic) copies of  $\mathcal{H}_3$ , each with a disjoint set of constants. That is, the node labeling of each new copy of  $\mathcal{H}_3$  is isomorphic to that of  $\mathcal{H}_3$ , except that it uses disjoint set of constants. Let us call them the copies  $\mathcal{B}_1, \dots, \mathcal{B}_\Delta$  (the original copy of  $\mathcal{H}_3$  is included). So, we have  $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset$ , for  $i \neq j$ . For each tree  $T_i$  in  $\mathcal{H}_3$ , we denote by  $T_i^k$  the isomorphic copy of  $T_i$  in  $\mathcal{B}_k$ . Now, let

$$\mathcal{H}_4 = \mathcal{B}_1 \cup \dots \cup \mathcal{B}_\Delta.$$

The structure corresponding to  $\mathcal{H}_4$  is denoted by  $\mathbf{I}_4$ . In the sequel, each node at level  $h$  in  $\mathcal{B}_k$  is said to be a (*potential*) *leaf* of  $\mathcal{B}_k$ .

*The procedure **prune***

The purpose of this procedure is to transform  $\mathbf{I}_4$  into a finite model. Intuitively, this is achieved by “pruning” all trees at level  $h$  in  $\mathcal{H}_4$  and then rejustifying the resulting unjustified constants by “linking” them to a justification being used in some other part of the model. This is the most complex step in the entire sequences of procedures, and care will be needed later to prove to ensure that satisfiability is not violated when constants are being rejustified.

We begin first by describing the connections that we wish to construct between the different parts of the model. Roughly speaking, the model we intend to construct somewhat resembles a  $\delta$ -regular graph (recall definition from Section 2.3), whose nodes are the copies  $\mathcal{B}_1 \cup \dots \cup \mathcal{B}_\Delta$  made earlier, and where edges between copies indicate that one copy is being used to make a new justification for a node at level  $h$  in another copy.

Firstly though, we state a proposition from extremal graph theory (see [Bollobas 2004, Theorem 1.4’ Chapter III]) for proof) that can be used to guarantee the existence of the kind of  $\delta$ -regular graph we intend to construct.

PROPOSITION 4.6. *Fix two positive integers  $\delta, g$  and take an integer  $\Delta$  with*

$$\Delta \geq \frac{(\delta - 1)^{g-1} - 1}{\delta - 2}.$$

*Then, there exists a  $\delta$ -regular graph of size  $\Delta$  with girth at least  $g$ .*

Using  $\delta, g$  and  $\Delta$  as defined in the copy procedure, this proposition implies that there exists a  $\delta$ -regular graph  $\mathbf{G}$  with vertices  $\{\mathcal{B}_1, \dots, \mathcal{B}_\Delta\}$  and with girth at least  $g$ . Let us now treat  $\mathbf{G}$  as a directed graph, where each edge in  $\mathbf{G}$  is regarded as two bidirectional arcs.

Observe that, for each vertex  $\mathcal{B}_k$ , there is a bijection  $out_k$  from the set of leafs (nodes at height  $h$ ) of  $\mathcal{B}_k$  to the set of arcs going out from  $\mathcal{B}_k$  in  $\mathbf{G}$ . We next take each leaf of  $\mathcal{B}_k$  in turn. For a leaf  $v$  (say,  $\mathbf{S}_b$ -labeled), suppose that  $out_k(v) =$

$(\mathcal{B}_k, \mathcal{B}_{k'})$ . Choose  $i$  such that  $b \in C_i(\mathbf{I}_4)$ . If the root of  $T_i^{k'}$  is  $\mathbf{S}_c$ -labeled, for some  $c \in I_4$ , then we delete all descendants of  $v$  in  $T_i^k$  and change  $v$  to  $\mathbf{S}_c[b/c]$ . In this way, we “prune” each of the trees in  $\mathcal{H}_4$ , and link each leaf node to the root node of another tree for the purpose of justification. We denote by  $\mathcal{H}_5$  the resulting collection of interlinked models, whose corresponding structure is denoted by  $\mathbf{I}_5$ .  $\mathcal{H}_5$  can be thought of as a collection of interlinked forests, where each forest corresponds to one of the copies  $\{\mathcal{B}_1, \dots, \mathcal{B}_\Delta\}$  and each forest is a collection of trees.

Observe now that each “tree” in  $\mathcal{H}_5$  is of height  $h$ . Since there are at most  $\Delta \times 2^N$  trees in  $\mathcal{H}_5$ , each of which has at most  $(N \times m)^{h+1}$  constants, we see that

$$\begin{aligned} |I_5| &\leq (\Delta \times 2^N) \times (N \times m)^{h+1} \\ &\leq ((2^N \times (Nm)^{cm})^{cm} \times 2^N) \times (N \times m)^{cm+1} \end{aligned}$$

It is easy to calculate now that  $|I_5| \leq 2^{2^{q(p,m)}}$  for some polynomial  $q$  in  $p$  and  $m$ . We have thus managed to construct a bounded model  $\mathbf{I}_5$  which satisfies the original UCV formula  $\phi$ .  $\square$

We now prove the correctness of our construction for theorem 4.1. The proof is divided into a series of subproperties that assert the correctness of each procedure in our construction. First, we prove a simple lemma.

**LEMMA 4.7.** *Let  $\mathcal{V}$  be a set of (unary) views over a vocabulary  $\sigma$ . Suppose  $\mathbf{I}, \mathbf{J}$  are  $\sigma$ -structures such that  $C(\mathbf{I})$  is non-empty iff  $C(\mathbf{J})$  is non-empty for each equivalence class formula  $C$  constructed with respect to  $\mathcal{V}$ . Then,  $\mathbf{I}$  and  $\mathbf{J}$  agree on  $\text{UCV}(\sigma, \mathcal{V})$ .*

**PROOF.** By standard Ehrenfeucht-Fraïsse argument, we see that  $\Lambda(\mathbf{I})$  and  $\Lambda(\mathbf{J})$  agree on  $\text{UFO}(\mathcal{V})$ . Then, by lemma 3.1, we have that  $\mathbf{I}$  and  $\mathbf{J}$  agree on  $\text{UCV}(\sigma, \mathcal{V})$ .  $\square$

**SUBPROPERTY 4.8 (CORRECTNESS OF MAKEJF).** (1) *For each node  $v$  (say,  $(\mathbf{S}_a \times C_i)$ -labeled) of  $\mathcal{H}_1$ ,  $a \in C_i(\mathbf{I}_1)$  with witnessing structure  $\mathbf{S}_a$ .*

(2) *If  $\mathbf{I}_0 \models \phi$ , then  $\mathbf{I}_1 \models \phi$ .*

**PROOF.** First, note that  $\mathbf{I}_1 \subseteq \mathbf{I}_0$ . Since conjunctive queries are monotonic, we have  $V(\mathbf{I}_1) \subseteq V(\mathbf{I}_0)$  for each view  $V \in \mathcal{U}$ . So, we have that  $a \in V(\mathbf{I}_1)$  implies that  $a \in V(\mathbf{I}_0)$ . In addition, for each constant  $a \in I_1$ , if  $a \in V(\mathbf{I}_0)$ , then  $a \in V(\mathbf{I}_1)$ , which is witnessed at some  $\mathbf{S}_a$ -labeled node. In turn, this implies that for  $a \in I_1$ , it is the case that  $a \in C(\mathbf{I}_1)$  iff  $a \in C(\mathbf{I}_0)$ . This proves the first statement. Also, by construction, if  $C_i(\mathbf{I}_0)$  is non-empty, where  $i \in \{0, \dots, 2^N - 1\}$ , we know that one of its members belongs to  $\mathbf{I}_1$ , witnessed at the root of  $T_i$ . Therefore, we also have that  $C(\mathbf{I}_0)$  is non-empty iff  $C(\mathbf{I}_1)$  is non-empty. In view of lemma 4.7, we conclude the second statement.  $\square$

At this stage, it is worth noting that, once  $\mathcal{H}_1$  has been constructed, the subsequent procedures might modify the label  $(\mathbf{S}_a \times C_i)$  — its name (e.g. from  $\mathbf{S}_a \times C_i$  to  $\mathbf{S}_{a'} \times C_i$  for some new constant  $a'$ ) as well as its contents (e.g. replacing each occurrence of a tuple  $R(a, a)$  by  $R(a', a')$  for some new constant  $a'$ ). Despite this, we wish to highlight that one invariant is preserved by each of these procedures that have been described:

INVARIANT 4.9 (JUSTIFICATION SET). *Suppose  $\mathcal{H}$  is a justification forest of a structure  $\mathbf{I}$ , and there exists a  $(\mathbf{S}_a \times C_i)$ -labeled node of  $\mathcal{H}$ . Then, we have  $a \in C_i(\mathbf{I})$  with witnessing structure  $\mathbf{S}_a$ .*

Subproperty 4.8 shows that this is satisfied by  $\mathcal{H}_1$ . In fact, that this invariant is preserved by the later procedures will be almost immediate from the proof of correctness of the procedure. Hence, we leave it to the reader to verify.

SUBPROPERTY 4.10 (CORRECTNESS OF **RENAME1**). *If  $\mathbf{I}_1 \models \phi$ , then  $\mathbf{I}_2 \models \phi$ .*

PROOF. In this procedure, we perform constant renaming for each tree  $T_i$  in  $\mathcal{H}_1$ . For the purpose of this proof, let us denote the tree so obtained by  $T'_i$ . Such a renaming induces a bijection  $f_i : \text{adom}(T_i) \rightarrow \text{adom}(T'_i)$ . Extend  $f_i$  to tuples, structures, and trees in the obvious way. Observe that the structures corresponding to the trees  $f_i(T_i)$  and  $T_i$  are isomorphic. Now, in view of lemma 4.11, it is easy to check that for each tree  $T_i$  in  $\mathcal{H}_1$  and a constant  $a$  in the structure corresponding to  $T_i$ ,  $a \in C_j(\mathbf{I}_1)$  iff  $f_i(a) \in C_j(\mathbf{I}_2)$ . By virtue of lemma 4.7, we conclude our proof.  $\square$

LEMMA 4.11. *Suppose that  $a$  is a constant in the structure corresponding to  $T_i$  of  $\mathcal{H}_1$ . Then,  $a \in V(\mathbf{I}_1)$  iff  $f_i(a) \in V(\mathbf{I}_2)$ .*

PROOF. ( $\Rightarrow$ ) By subproperty 4.8, it is the case that  $a \in V(\mathbf{S}_a)$ . Since  $f_i$  is a bijection, it is also true that  $f_i(a) \in V(f_i(\mathbf{S}_a))$ .

( $\Leftarrow$ ) Let  $\mathbf{M}$  be a minimal set of tuples in  $\mathbf{I}_2$  such that  $f_i(a) \in V(\mathbf{M})$ . Observe that there is a one-to-one function mapping the set of conjuncts in  $V$  to  $\mathbf{M}$ . For each tree  $T_j$  in  $\mathcal{H}_2$ , let  $\mathbf{M}_j$  denote the members of  $\mathbf{M}$  that can be found in  $T_j$ . Note that  $\text{adom}(\mathbf{M}_j) \cap \text{adom}(\mathbf{M}_{j'}) = \emptyset$  for  $j \neq j'$ . Now, let  $\mathbf{M}' = \bigcup_j f_j^{-1}(\mathbf{M}_j)$ . It is not hard to see that  $a \in V(\mathbf{M}')$ . Since  $\mathbf{M}' \subseteq \mathbf{I}_1$ , we have  $a \in V(\mathbf{I}_1)$ .  $\square$

SUBPROPERTY 4.12 (CORRECTNESS OF **RENAME2**). *If  $\mathbf{I}_2 \models \phi$ , then  $\mathbf{I}_3 \models \phi$ .*

PROOF. Define the function  $\eta : \mathbf{I}_3 \rightarrow \mathbf{I}_2$  such that  $\eta(a_{j,k}) = a$ . Note that  $\eta$  is onto. Extend  $\eta$  to tuples, and sets of tuples in the obvious way. In view of lemma 4.7, it is sufficient to show that, for each  $a \in \mathbf{I}_3$  and  $i \in \{0, \dots, 2^N - 1\}$ ,  $a \in C_i(\mathbf{I}_3)$  iff  $\eta(a) \in C_i(\mathbf{I}_2)$ . In turn, it is enough to show that,  $a \in V(\mathbf{I}_3)$  iff  $\eta(a) \in V(\mathbf{I}_2)$ .

( $\Rightarrow$ ) Take a minimal set  $\mathbf{M}$  of tuples in  $\mathbf{I}_3$  such that  $a \in V(\mathbf{M})$ . Then, we have  $\eta(a) \in V(\eta(\mathbf{M}))$ . Since  $\eta(\mathbf{M}) \subseteq \mathbf{I}_2$ , we have  $\eta(a) \in V(\mathbf{I}_2)$ .

( $\Leftarrow$ ) Since invariant 4.9 holds for  $\mathcal{H}_2$ , the fact that  $\eta(a) \in V(\mathbf{I}_2)$  is witnessed by  $S_{\eta(a)} \in \mathcal{H}_2$ . Since  $S_a$  and  $S_{\eta(a)}$  are isomorphic justification sets, we have that  $a \in V(\mathbf{I}_3)$  is justified by  $S_a \in \mathcal{H}_3$ .  $\square$

SUBPROPERTY 4.13 (CORRECTNESS OF **COPY**). (1) *For each node  $v$  (say,  $(\mathbf{S}_a \times C_i)$ -labeled) of  $\mathcal{H}_4$ ,  $a \in C_i(\mathbf{I}_4)$  with witnessing structure  $\mathbf{S}_a$ .*

(2) *If  $\mathbf{I}_3 \models \phi$ , then  $\mathbf{I}_4 \models \phi$ .*

PROOF. Similar to the proof of subproperty 4.10.  $\square$

SUBPROPERTY 4.14 (CORRECTNESS OF **PRUNE**). *If  $\mathbf{I}_4 \models \phi$ , then  $\mathbf{I}_5 \models \phi$ .*

PROOF. Recall that there are  $N_l \stackrel{\text{def}}{=} \delta \times \Delta$  leaves in  $\mathcal{H}_4$ . Let us order these nodes as  $v_1, \dots, v_{N_l}$ . Suppose also that  $v_i$  is labeled by  $\mathbf{S}_{b_i}$  for some  $b_i \in \mathbf{I}_4$ . By virtue of

`rename2`, we see that  $b_i \neq b_j$  whenever  $i \neq j$ . Next, we may think of the procedure `prune` as consisting of  $N_l$  steps, where at step  $i$ , the node  $v_i$  has all its descendants removed (pruned) and  $v_i$  is changed to  $\mathbf{S}'_{b_i} \stackrel{\text{def}}{=} \mathbf{S}_{c_i}[b_i/c_i]$  for some  $c_i \in I_4$ . Letting  $\mathcal{K}_0 \stackrel{\text{def}}{=} \mathcal{H}_4$ , we denote by  $\mathcal{K}_i$  ( $i = 1, \dots, N_l$ ) the resulting model after executing  $i$  steps on  $\mathcal{K}_0$ . The structure corresponding to  $\mathcal{K}_i$  is denoted by  $\mathbf{J}_i$ .

We wish to prove by induction on  $0 \leq i < N_l$  that

- (I). For each  $a \in J_{i+1}$  and  $V \in \mathcal{U}$ ,  $a \in V(\mathbf{J}_{i+1})$  iff  $a \in V(\mathbf{J}_i)$ .
- (II). Invariant 4.9 holds for  $\mathbf{J}_{i+1}$ .
- (III). For each  $a \in J_{i+1}$ , we have  $a \in C_i(\mathbf{J}_{i+1})$  iff  $a \in C_i(\mathbf{J}_i)$ .

Note that  $J_{i+1} \subseteq J_i$ . So, by lemma 4.7 and the fact that invariant 4.9 holds for the initial case  $\mathbf{J}_0$  (from proofs of previous subproperties), statement (III) will imply what we wish to prove. It is easy to see that statement (III) is a direct consequence of statement (I). It is also easy to show that statement (I) implies statement (II). This follows since firstly, at step  $i + 1$ , we replace the content of  $\mathbf{S}_{b_i}$  by that of  $\mathbf{S}_{c_i}$ , except for substituting  $b_i$  for  $c_i$ . Second, the elements  $b_i$  and  $c_i$  belong to the same equivalence class in  $\mathbf{J}_i$ , and invariant 4.9 holds for  $\mathbf{J}_i$  by induction. Therefore, it remains only to prove statement (I).

Let us now fix  $i < N_l$ ,  $a \in J_{i+1}$ , and  $V \in \mathcal{U}$ . It is simple to prove that  $a \in V(\mathbf{J}_i)$  implies  $a \in V(\mathbf{J}_{i+1})$ . This is witnessed by tuples in the  $\mathbf{S}_a$ -labeled (or  $\mathbf{S}'_{b_i}$ -labeled if  $a = b_i$ ) node in  $\mathcal{K}_{i+1}$ , which exists by construction.

Conversely, we take a minimal set  $\mathbf{M}$  of tuples in  $\mathbf{J}_{i+1}$  with  $a \in V(\mathbf{J}_{i+1})$ , witnessed by the valuation  $\nu$ . Our aim is to find a set  $\mathbf{M}'$  of tuples in  $\mathbf{J}_i$  with  $a \in V(\mathbf{M}')$ . Let  $\mathbf{M}_{b_i} \stackrel{\text{def}}{=} \mathbf{M} - \mathcal{D}(\mathbf{J}_i)$ . Intuitively,  $\mathbf{M}_{b_i}$  contains the set of *new* tuples. These are tuples which did not exist in the structure  $\mathbf{J}_i$  and have been created specifically to justify the node whose descendants (justifications) have just been pruned. By construction, we have  $\mathbf{M}_{b_i} \subseteq \mathbf{S}'_{b_i}$ , which implies that  $\text{adom}(\mathbf{M}_{b_i}) \subseteq \text{adom}(\mathbf{S}'_{b_i})$ . Observe also that  $b_i \in \text{adom}(t)$  for each tuple  $t$  in  $\mathbf{M}_{b_i}$ ; otherwise,  $t$  would be a tuple in  $\mathbf{S}_{c_i} \subseteq \mathbf{J}_i$  (i.e. it would not be a new tuple). Define

$$\mathbf{L} := \{t \in \mathbf{M} - \mathbf{M}_{b_i} : t \text{ is connected to some } t' \in \mathbf{M}_{b_i} \text{ in } \mathbf{M}\}.$$

$\mathbf{L}$  consists of tuples that are connected to new tuples. Also, let  $\mathbf{L}' \stackrel{\text{def}}{=} \mathbf{M} - \mathbf{M}_{b_i} - \mathbf{L}$ , i.e., the set of all tuples of  $\mathbf{M}$  that are *not* connected to any (new) tuples in  $\mathbf{M}_{b_i}$ . Note that  $\mathbf{L} \cup \mathbf{L}' \subseteq \mathbf{J}_i$ , and that the sets  $\mathbf{M}_{b_i}$ ,  $\mathbf{L}$ , and  $\mathbf{L}'$  form a partition on  $\mathbf{M}$ . Also, by definition, we have  $\text{adom}(\mathbf{L}') \cap \text{adom}(\mathbf{M}_{b_i} \cup \mathbf{L}) = \emptyset$ . In the following, we define  $\mathbf{M}_{c_i} \stackrel{\text{def}}{=} \mathbf{M}_{b_i}[c_i/b_i]$ . Note that  $\mathbf{M}_{c_i} \subseteq \mathbf{S}_{c_i} \subseteq \mathbf{J}_i$ .

Before we proceed further, it is helpful to see how we partition  $\mathbf{M}$  on a simple example. Suppose that the view  $V$  is defined as

$$V(x_0) \leftarrow E(x_0, x_1), E(x_1, x_2), R(x_3, x_4), R(x_4, x_5).$$

Furthermore, suppose that we take the valuation  $\nu$  defined as  $\nu(x_i) = i$ . In this case,  $\mathbf{M}$  can be described diagrammatically as follows

$$V(0) \leftarrow E(0, 1), E(1, 2), R(3, 4), R(4, 5).$$

Assume now that the only tuple in  $\mathbf{M}$  that doesn't belong to  $\mathcal{D}(\mathbf{J}_i)$  is  $E(0, 1)$ . Then, we have  $\mathbf{M}_{b_i} = \{E(0, 1)\}$ . It is easy to show that  $\mathbf{L} = \{E(1, 2)\}$  and

$\mathbf{L}' = \{R(3, 4), R(4, 5)\}$ .

We next state a result regarding  $\mathbf{L}$  that will shortly be needed. It clarifies the nature of a partition that exists for  $\mathbf{L}$  and the relationships which hold between the elements of the partition.

PROPOSITION 4.15. *We can find tuple-sets  $\mathbf{A}, \mathbf{B} \subseteq \mathbf{L}$  such that:*

- (1)  $\mathbf{A} \cap \mathbf{B} = \emptyset$ ,
- (2)  $\mathbf{A} \cup \mathbf{B} = \mathbf{L}$ ,
- (3)  $\text{adom}(\mathbf{A}) \cap \text{adom}(\mathbf{B}) = \emptyset$ ,
- (4)  $b_i \notin \text{adom}(\mathbf{B})$ , and
- (5)  $\text{adom}(\mathbf{M}_{b_i}) \cap \text{adom}(\mathbf{A}) \subseteq \{b_i\}$ .

The proof of this proposition can be found at the end of this section. We now shall construct  $\mathbf{M}' \subseteq \mathcal{D}(\mathbf{J}_i)$  such that  $a \in V(\mathbf{M}')$ . First, we put  $\mathbf{L}'$  in  $\mathbf{M}'$ . This does not affect our choice of tuple-sets that replace  $\mathbf{M}_{b_i}$ ,  $\mathbf{A}$ , and  $\mathbf{B}$  as  $\text{adom}(\mathbf{L}') \cap \text{adom}(\mathbf{M}_{b_i} \cup \mathbf{L}) = \emptyset$  (i.e. the set of free tuples instantiated by  $\mathbf{L}'$  and the set of free tuples instantiated by  $\mathbf{M}_{b_i} \cup \mathbf{L}$  share no common variables), as we have noted earlier. There are two cases to consider:

*case 1.  $a = b_i$ .* Let  $F$  be the set of all free tuples in the body of  $V$  such that  $\{\nu(u) : u \in F\} = \mathbf{M}_{b_i} \cup \mathbf{B}$ . Suppose  $X$  is the set of all variables in  $V$ . Let  $\{y_1, \dots, y_r\} \subseteq X$  be the set of variables in  $F$  such that  $\nu(y_j) = b_i$ . With  $y$  as a new variable, let  $F' := F[y/y_1, \dots, y_r]$ . Define the new view  $V'(y)$  whose conjuncts are exactly  $F'$ :

$$V'(y) \leftarrow \bigwedge F'$$

Trivially, we have  $b_i \in V'(\mathbf{M}_{b_i} \cup \mathbf{B})$ . Then, as  $b_i \notin \text{adom}(\mathbf{B})$  by proposition 4.15, we have  $c_i \in V'(\mathbf{M}_{c_i} \cup \mathbf{B})$ . Note that  $\mathbf{M}_{c_i} \cup \mathbf{B} \subseteq \mathcal{D}(\mathbf{J}_i)$  and  $V' \in \mathcal{U}$  since  $\text{length}(V') \leq m$ . So, since by induction  $b_i$  and  $c_i$  belong to the same equivalence class in  $\mathbf{J}_i$ , there exist tuple-sets  $\mathbf{P}_{b_i}$  and  $\mathbf{B}'$  with  $\mathbf{P}_{b_i} \cup \mathbf{B}' \subseteq \mathbf{J}_i$  such that  $b_i \in V'(\mathbf{P}_{b_i} \cup \mathbf{B}')$ . [ $\mathbf{P}_{b_i}$  and  $\mathbf{B}'$ , respectively, replace the role of  $\mathbf{M}_{c_i}$  and  $\mathbf{B}$ .] Observe now that  $a \notin \text{adom}(\mathbf{B})$  as  $a = b_i$ . Since  $\text{adom}(\mathbf{M}_{b_i}) \cap \text{adom}(\mathbf{A}) \subseteq \{b_i\}$  and  $\text{adom}(\mathbf{A}) \cap \text{adom}(\mathbf{B}) = \emptyset$  from proposition 4.15, it is easy to verify that

$$a \in V(\mathbf{P}_{b_i} \cup \mathbf{A} \cup \mathbf{B}' \cup \mathbf{L}').$$

*case 2.  $a \neq b_i$ .* This is divided into two further cases:

(a).  $b_i \in \text{adom}(\mathbf{A})$ . This is divided into two further cases:

(i).  $a \in \text{adom}(\mathbf{A})$ . In this case, note that  $a \notin \text{adom}(\mathbf{M}_{b_i})$  (using Proposition 4.15(5)) and  $a \notin \text{adom}(\mathbf{B})$  (using Proposition 4.15(3)). We can then continue in the same fashion as in the case 1.

(ii).  $a \notin \text{adom}(\mathbf{A})$ . Let  $F$  be the set of all free tuples in the body of  $V$  such that  $\{\nu(u) : u \in F\} = \mathbf{A}$ . Let  $\{y_1, \dots, y_r\} \subseteq X$  be the set of variables in  $F$  such that  $\nu(y_j) = b_i$ . Let  $y$  be a new variable (i.e.  $y \notin X$ ) and  $F' := F[y/y_1, \dots, y_r]$ , i.e., we replace each occurrence of the variables  $y_1, \dots, y_r$  in  $F$  by  $y$ . Then, let  $V'(y)$  be the view whose conjuncts are exactly  $F'$ :

$$V'(y) \leftarrow \bigwedge F'.$$

Then,  $V' \in \mathcal{U}$  and  $b_i \in V'(\mathbf{A})$ . Since  $\mathbf{A} \subseteq \mathcal{D}(\mathbf{J}_i)$  and because  $b_i$  and  $c_i$  belong to the same equivalence class in  $\mathbf{J}_i$  (by the induction hypothesis), there exists a set  $\mathbf{A}' \subseteq \mathcal{D}(\mathbf{J}_i)$  such that  $c_i \in V'(\mathbf{A}')$ . Since  $\text{adom}(\mathbf{M}_{b_i}) \cap \text{adom}(\mathbf{A}) \subseteq \{b_i\}$  and  $\text{adom}(\mathbf{A}) \cap \text{adom}(\mathbf{B}) = \emptyset$  from proposition 4.15, it is easy to check that  $a \in V'(\mathbf{M}_{c_i} \cup \mathbf{A}' \cup \mathbf{B} \cup \mathbf{L}')$ .

(b).  $b_i \notin \text{adom}(A)$ . Let  $\mathbf{M}_{c_i} \stackrel{\text{def}}{=} \mathbf{M}_{b_i}[c_i/b_i]$ . By construction, we see that  $\mathbf{M}_{c_i} \subseteq \mathbf{S}_{c_i} \subseteq \mathcal{D}(\mathbf{J}_i)$ . By proposition 4.15 (items 4 and 5), it is the case that

$$a \in V(\mathbf{M}_{c_i} \cup \mathbf{A} \cup \mathbf{B} \cup \mathbf{L}').$$

In any case, we have  $a \in V(\mathbf{J}_i)$ . This completes the proof.

□

It remains to prove proposition 4.15.

PROOF OF PROPOSITION 4.15. The present situation is depicted in figure 2. This is a snapshot of the moment just before we apply step  $i + 1$ . Step  $i$  of **prune**

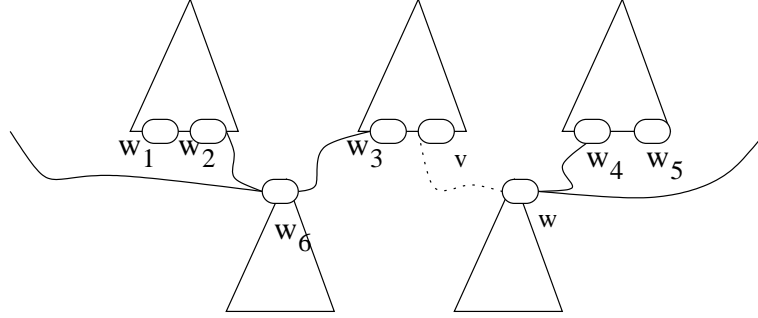


Fig. 2.  $v$  is the  $\mathbf{S}_{b_i}$ -labeled node whose contents are to be changed by  $\mathbf{S}_{c_i}[b_i/c_i]$ . The node  $w$  is  $\mathbf{S}_{c_i}$ -labeled, and will be “linked” to node  $v$  after step  $i + 1$  of **prune** procedure is finished — signified by the dotted line. Solid lines represent links that have been established in step  $j < i + 1$  of the procedure.

procedure simply prunes the subtree rooted at the  $\mathbf{S}_{b_i}$ -labeled node  $v$ , and links (rejustifies)  $v$  using the  $\mathbf{S}_{c_i}$ -labeled node  $w$ , where  $b_i$  and  $c_i$  belong to the same equivalence class in  $\mathbf{J}_i$ . It is important to note that some cousin<sup>3</sup>  $w_3$  of  $v$  might also be linked to a root  $w_6$  of another tree, which in turn might be linked to a leaf node  $w_2$  of another tree, which in turn might have a cousin  $w_1$  that satisfies the same property as  $w_3$  and so on. Furthermore, the node  $w$  might have also been linked to some other leaf  $w_4$  that has a cousin  $w_5$  that is connected to a root of some other tree, and so on. Note that it is impossible for two leafs of a tree to be linked to the same root node of a tree by construction. Hence, the three trees in the middle (i.e. where  $v, w$ , and  $w_6$  are located) are necessarily distinct. The leftmost and rightmost tree might be the same tree depending on the value of the girth  $g$  that we defined earlier.

<sup>3</sup>node of the same tree and level

Let us now define

$$\begin{aligned} \mathbf{A} &\stackrel{\text{def}}{=} \{t \in \mathbf{L} : d_{\mathbb{G}(\mathbf{J}_i)}(t, b_i) \leq m\} \\ \mathbf{B} &\stackrel{\text{def}}{=} \{t \in \mathbf{L} : d_{\mathbb{G}(\mathbf{J}_i)}(t, \mathbf{S}_{c_i}) \leq m\}. \end{aligned}$$

Intuitively, the set  $\mathbf{A}$  contains tuples up to distance  $m$  from the label  $\mathbf{S}_{b_i}$  of node  $v$  in  $\mathbf{J}_i$ , while the set  $\mathbf{B}$  contains tuples up to distance  $m$  from the label  $\mathbf{S}_{c_i}$  of  $w$  in  $\mathbf{J}_i$ . Note that this is distance in the structure  $\mathbf{J}_i$ , not  $\mathbf{J}_{i+1}$ . It is immediate that we have property (2)  $\mathbf{A} \cup \mathbf{B} = \mathbf{L}$ , as the length of the view  $V$  is at most  $m$  and that  $\text{adom}(\mathbf{M}_{b_i}) \subseteq \{b_i\} \cup \text{adom}(\mathbf{S}_{c_i})$ . So, it is sufficient to show that properties 3 and 5 are satisfied, as they obviously imply properties 1 and 4. Note that our construction has ensured that:

- (1) Two nodes in any given tree in  $\mathcal{K}_i$  that are at least distance two apart cannot share a constant.
- (2) Two trees  $T$  and  $T'$  in  $\mathcal{K}_i$  cannot share a constant except on: (i) a unique leaf of  $T$  and the root of  $T'$ , as is the case for  $v$  and  $w$  in Figure 2 or alternatively (ii) a unique leaf of  $T$  and a unique leaf of  $T'$ . This case can happen when both leaves are connected to the root of a different tree  $T''$ , as is the situation for  $w_2$  and  $w_3$  in Figure 2.

Therefore, for some sufficiently large constant  $c' \in \mathbb{N}$ , two nodes  $v'$  and  $v''$  in  $\mathcal{K}_i$  of distance  $c'm$  cannot have two elements of  $\mathbf{J}_i$  that are of distance  $\leq m$  in  $\mathbb{G}(\mathbf{J}_i)$ . [In fact, a careful analysis will show that  $c' = 1$  is sufficient.] Therefore, the locations of the constants in  $\mathbf{A}$  (resp.  $\mathbf{B}$ ) cannot be “very far away” from the tuple  $v$  (resp.  $w$ ). In fact, if we set  $c \geq c'$  (recall that  $g = cm$ ) and consider the path  $P$  between a tuple  $t \in \mathbf{A}$  and the constant  $b_i$  (which belongs to  $v$  and its parent), it cannot connect a root and a leaf of the same tree (i.e. through the body of the tree). So, either it is completely contained in the tree of which  $v$  is a leaf, or it has to alternate between leafs and root several times, and then end in some tree. In figure 2, we may pick the following example

$$v \rightarrow^* w_3 \rightarrow w_6 \rightarrow w_2 \rightarrow^* w_1 \rightarrow \dots,$$

where we use the notation  $\rightarrow^*$  to mean “path in the same tree”. The same analysis can be applied to determine the locations of the tuples of  $\mathbf{B}$ . Therefore, in order to ensure that properties 3 and 5 are satisfied, we just need to ensure that the height of each tree and the girth of  $\mathcal{K}_i$  be large enough, which can be done by taking a sufficiently large  $c$ . When the girth (as ensured in the `copy` and `prune` procedures) is sufficiently large, we can be sure that no paths of length  $\leq m$  exist between  $v$  and  $w$  in  $\mathcal{K}_i$  [In fact, a careful but tedious analysis shows that  $c = 1$  is sufficient.]  $\square$

Theorem 4.1 also holds for infinite models, since even if the initial justification hierarchies are infinite, the proof method used is unchanged. We thus also obtain finite controllability (every satisfiable formula is finitely satisfiable) for UCV.

PROPOSITION 4.16. *The UCV class of formulas is finitely controllable.*

## 5. EXTENDING THE VIEW DEFINITIONS

The previous section showed that the first order language using unary conjunctive view definitions is decidable. A natural way to increase the power of the language

is to make view bodies more expressive (but retain unary arity for the views). We saw earlier that allowing unary views to use disjunction in their definition does not actually increase expressiveness of the UCV language and hence this case is decidable. Unfortunately, as we will show, employing other ways of extending the views results in satisfiability becoming undecidable.

The first extension we consider is allowing inequality in the views, e.g.,

$$V(x) \leftarrow R(x, y), S(x, x), x \neq y$$

Call the first order language over such views the *first order unary conjunctive<sup>≠</sup> view language*. In fact, this language allows us to check whether a two counter machine computation is valid and terminates, which thus leads to the following result:

**THEOREM 5.1.** *Satisfiability is undecidable for the first order unary conjunctive<sup>≠</sup> view query language.*

**PROOF.** The proof is by a reduction from the halting problem of two counter machines (2CM's) starting with zero in the counters. Given any description of a 2CM and its computation, we can show how to a) encode this description in database relations and b) define queries to check this description. We construct a query which is satisfiable iff the 2CM halts. The basic idea of the simulation is similar to one in [Levy et al. 1993], but with the major difference that *cycles are allowed* in the successor relation, though there must be at least one good chain.

A two-counter machine is a deterministic finite state machine with two non-negative counters. The machine can test whether a particular counter is empty or non-empty. The transition function has the form

$$\delta : S \times \{=, >\} \times \{=, >\} \rightarrow S \times \{pop, push\} \times \{pop, push\}$$

For example, the statement  $\delta(4, =, >) = (2, push, pop)$  means that if we are in state 4 with counter 1 equal to 0 and counter 2 greater than 0, then go to state 2 and add one to counter 1 and subtract one from counter 2.

The computation of the machine is stored in the relation  $config(t, s, c_1, c_2)$ , where  $t$  is the time,  $s$  is the state and  $c_1$  and  $c_2$  are values of the counters. Each of the values that  $t$  can take will be timestamps. The states of the machine can be described by integers  $0, 1, \dots, h$  where 0 is the initial state and  $h$  the halting (accepting) state. The first configuration of the machine is  $config(0, 0, 0, 0)$  and thereafter, for each move, the time is increased by one and the state and counter values changed in correspondence with the transition function.

We will use some relations to encode the computation of 2CMs starting with zero in the counters. These are:

- $S_0, \dots, S_h$ : each contains a constant which represents that particular state.
- $succ$ : the successor relation. We will make sure it contains one chain of timestamps starting from *zero* and ending at *last* (but it may in addition contain unrelated cycles of timestamps).
- $config$ : contains computation of the 2CM.
- $zero$ : contains the first constant in the chain in  $succ$ . This constant is also used as the number zero.
- $last$ : contains the last constant in the chain in  $succ$ .

Note that we sometimes blur the distinction between unary relations and unary views, since a view  $V$  can simulate a unary relation  $U$  if it is defined by  $V(x) \leftarrow U(x)$ .

The unary and nullary views (the latter can be eliminated using quantified unary views) are:

- halt*: true if the machine halts.
- bad*: true if the database doesn't correctly describe the computation of the 2CM.
- dsucc*: contains all constants in *succ*.
- dT*: contains all timestamps in *config*.
- dP*: contains all constants in *succ* with predecessors.
- dCol<sub>1</sub>*, *dCol<sub>2</sub>*: are projections of the first and second columns of *succ*.

When defining the views, we also state some formulas (such as *hasPred*) over the views which will be used to form our first order sentence over the views.

- The “domain” views (those starting with the letter *d*) are easy to define, e.g.

$$\begin{aligned} dP(x) &\leftarrow succ(z, x) \\ dCol_1(x) &\leftarrow succ(x, y) \\ dCol_2(x) &\leftarrow succ(y, x) \end{aligned}$$

- hasPred* says “each nonzero constant in *succ* has a predecessor:”

$$hasPred : \forall x(dsucc(x) \Rightarrow (zero(x) \vee dP(x)))$$

- sameDom* says “the constants used in *succ* and the timestamps in *config* are the same set”:

$$sameDom : \forall x(dsucc(x) \Rightarrow dT(x)) \wedge \forall y(dT(y) \Rightarrow dsucc(y))$$

- goodzero* says “the zero occurs in *succ*”:

$$goodzero : \forall x(zero(x) \Rightarrow dsucc(x))$$

- nempty* : each of the domains and unary base relations is not empty

$$nempty : \exists x(dsucc(x))$$

- Check that each constant in *succ* has at most one successor and at most one predecessor and that it has no cycles of length 1.

$$\begin{aligned} bad &\leftarrow succ(x, y), succ(x, z), y \neq z \\ bad &\leftarrow succ(y, x), succ(z, x), y \neq z \\ bad &\leftarrow succ(x, x) \end{aligned}$$

Note that the first two of these rules could be enforced by database style functional dependencies  $x \rightarrow y$  and  $y \rightarrow x$  on *succ*. These are just key dependencies, one of the simplest kinds of functional dependencies.

- Check that every constant in the chain in *succ* which isn't the last one must have a successor

$$hassuccnext : \forall y(dCol_2(y) \Rightarrow (last(y) \vee dCol_1(y)))$$

—Check that the last constant has no successor and zero (the first constant) has no predecessor.

$$\begin{aligned} bad &\leftarrow last(x), succ(x, y) \\ bad &\leftarrow zero(x), succ(y, x) \end{aligned}$$

—Check that every constant eligible to be in last and zero must be so.

$$\begin{aligned} eligiblezero &: \forall y(dCol_1(y) \Rightarrow (dCol_2(y) \vee zero(y))) \\ eligiblelast &: \forall y(dCol_2(y) \Rightarrow (dCol_1(y) \vee last(y))) \end{aligned}$$

—Each  $S_i$  and  $zero$  and  $last$  contain  $\leq 1$  element.

$$\begin{aligned} bad &\leftarrow S_i(x), S_i(y), x \neq y \\ bad &\leftarrow zero(x), zero(y), x \neq y \\ bad &\leftarrow last(x), last(y), x \neq y \end{aligned}$$

—Check that  $S_i, S_j, last, zero$  are disjoint ( $0 \leq i < j \leq h$ ):

$$\begin{aligned} bad &\leftarrow zero(x), last(x) \\ bad &\leftarrow S_i(x), S_j(x) \\ bad &\leftarrow zero(x), S_i(x) \\ bad &\leftarrow last(x), S_i(x) \end{aligned}$$

—Check that the timestamp is the key for  $config$ . There are three rules, one for the state and two for the two counters; the one for the state is:

$$bad \leftarrow config(t, s, c_1, c_2), config(t, s', c'_1, c'_2), s \neq s'$$

—Check the configuration of the 2CM at time zero.  $config$  must have a tuple at  $(0, 0, 0, 0)$  and there must not be any tuples in config with a zero state and non zero times or counters.

$$\begin{aligned} V_{z_s}(s) &\leftarrow zero(t), config(t, s, x, y) \\ V_{z_{c_1}}(c) &\leftarrow zero(t), config(t, x, c, y) \\ V_{z_{c_2}}(c) &\leftarrow zero(t), config(t, x, y, c) \\ V_{y_s}(t) &\leftarrow zero(s), config(t, s, x, y) \\ V_{y_{c_1}}(c_1) &\leftarrow zero(s), config(t, s, c_1, x) \\ V_{y_{c_2}}(c_2) &\leftarrow zero(s), config(t, s, x, c_2) \\ goodconfigzero &: \forall x(V_{z_s}(x) \Rightarrow S_0(x) \wedge \\ &(V_{z_{c_1}}(x) \vee V_{z_{c_2}}(x) \vee V_{y_s}(x) \vee V_{y_{c_1}}(x) \vee V_{y_{c_2}}(x)) \Rightarrow zero(x)) \end{aligned}$$

—For each tuple in  $config$  at time  $t$  which isn't the halt state, there must also be a tuple at time  $t + 1$  in  $config$ .

$$\begin{aligned} V_1(t) &\leftarrow config(t, s, c_1, c_2), S_h(s) \\ V_2(t) &\leftarrow succ(t, t2), config(t2, s', c'_1, c'_2) \\ hasconfignext &: \forall t((dt(t) \wedge \neg V_1(t)) \Rightarrow V_2(t)) \end{aligned}$$

—Check that the transitions of the 2CM are followed. For each transition  $\delta(j, >, =) = (k, pop, push)$ , we include three rules, one for checking the state, one for checking the first counter and one for checking the second counter. For the transition in question we have for checking the state

$$\begin{aligned}
 V_\delta(t') &\leftarrow \text{config}(t, s, c_1, c_2), \text{succ}(t, t'), S_j(s), \text{succ}(x, c_1), \text{zero}(c_2) \\
 V_{\delta_s}(s) &\leftarrow V_\delta(t), \text{config}(t, s, c_1, c_2) \\
 \text{goodstate}_\delta &: \forall s (V_{\delta_s}(s) \Leftrightarrow S_k(s))
 \end{aligned}$$

and for the first counter, we (i) find all the times where the transition is definitely correct for the first counter

$$\begin{aligned}
 Q_{1_\delta}(t') &\leftarrow \text{config}(t, s, c_1, c_2), \\
 &\quad \text{succ}(t, t'), S_j(s), \text{succ}(x, c_1), \\
 &\quad \text{zero}(c_2), \text{succ}(c'_1, c_1), \text{config}(t', s', c'_1, c'_2)
 \end{aligned}$$

(ii) find all the times where the transition may or may not be correct for the first counter

$$Q_{2_\delta}(t') \leftarrow \text{config}(t, s, c_1, c_2), \text{succ}(t, t'), S_j(s), \text{succ}(x, c_1), \text{zero}(c_2)$$

and make sure  $Q_{1_\delta}$  and  $Q_{2_\delta}$  are the same

$$\text{goodtrans}_{\delta_{c_1}} : \forall t (Q_{1_\delta}(t) \Leftrightarrow Q_{2_\delta}(t))$$

Rules for second counter are similar.

For transitions  $\delta_1, \delta_2, \dots, \delta_k$ , the combination can be expressed thus:

$$\begin{aligned}
 \text{goodstate} &: \text{goodstate}_{\delta_1} \wedge \text{goodstate}_{\delta_2} \wedge \dots \wedge \text{goodstate}_{\delta_k} \\
 \text{goodtrans}_{c_1} &: \text{goodtrans}_{\delta_{1c_1}} \wedge \text{goodtrans}_{\delta_{2c_1}} \wedge \dots \wedge \text{goodtrans}_{\delta_{kc_1}} \\
 \text{goodtrans}_{c_2} &: \text{goodtrans}_{\delta_{1c_2}} \wedge \text{goodtrans}_{\delta_{2c_2}} \wedge \dots \wedge \text{goodtrans}_{\delta_{kc_2}}
 \end{aligned}$$

—Check that halting state is in *config*.

$$\begin{aligned}
 \text{hlt}(t) &\leftarrow \text{config}(t, s, c_1, c_2), S_h(s) \\
 \text{halt} &: \exists x \text{hlt}(x)
 \end{aligned}$$

Given these views, we claim that satisfiability is undecidable for the query  $\psi = \neg \text{bad} \wedge \text{hasPred} \wedge \text{sameDom} \wedge \text{halt} \wedge \text{goodzero} \wedge \text{goodconfigzero} \wedge \wedge \text{nempty} \wedge \text{hassuccnext} \wedge \text{eligiblezero} \wedge \text{eligiblelast} \wedge \text{goodstate} \wedge \text{goodtrans}_{c_1} \wedge \text{goodtrans}_{c_2} \wedge \text{hasconfignext}$   $\square$

The second extension we consider is to allow “safe” negation in the conjunctive views, e.g.

$$V(x) \leftarrow R(x, y), R(y, z), \neg R(x, z)$$

Call the first order language over such views the *first order unary conjunctive<sup>¬</sup> view language*. It is also undecidable, by a result in [Bailey et al. 1998].

**THEOREM 5.2.** [Bailey et al. 1998] *Satisfiability is undecidable for the first order unary conjunctive<sup>¬</sup> view query language.*  $\blacksquare$

A third possibility for increasing the expressiveness of views would be to keep the body as a pure conjunctive query, but allow views to have *binary* arity, e.g.

$$V(x, y) \leftarrow R(x, y)$$

This doesn’t yield a decidable language either, since this language has the same expressiveness as first order logic over binary relations, which is known to be undecidable [Börger et al. 1997].

PROPOSITION 5.3. *Satisfiability is undecidable for the first order binary conjunctive view language.* ■

A fourth possibility is to allow linear recursion (the easiest kind of recursion) over unary views. e.g.

$$\begin{aligned} V(x) &\leftarrow \text{edge}(x, y) \\ V(x) &\leftarrow V(y) \wedge \text{edge}(y, x) \end{aligned}$$

$V$  is interpreted as the smallest set such that the 2 rules are satisfied. Call this the first order unary conjunctive<sup>rec</sup> language. This language is undecidable also.

THEOREM 5.4. *Satisfiability is undecidable for the first order unary conjunctive<sup>rec</sup> view language.*

PROOF. (sketch): The proof of theorem 5.1 can be adapted by removing inequality and instead using recursion to ensure there exists a connected chain in *succ*. It then becomes more complicated, but the main property needed is that *zero* is connected to *last* via the constants in *succ*. This can be expressed by

$$\begin{aligned} \text{conn\_zero}(x) &\leftarrow \text{zero}(x) \\ \text{conn\_zero}(x) &\leftarrow \text{conn\_zero}(y), \text{succ}(y, x) \\ \exists x(\text{last}(x) \wedge \text{conn\_zero}(x)) \end{aligned}$$

□

## 6. APPLICATIONS

### 6.1 Reasoning Over Ontologies

A currently active area of research is that of reasoning over ontologies (see e.g. [Horrocks 2005]). The aim here is to use decidable query languages used for accessing and reasoning about information and structure for the Semantic Web. In particular, ontologies provide vocabularies which can define relationships or associations between various concepts (classes) and also properties that link different classes together. Description logics are a key tool for reasoning over schemas and ontologies and to this end, a considerable number of different description logics have been developed [Baader et al. 2003]. To illustrate some reasoning over a simple ontology, we adopt an example from [Horrocks et al. 2003], describing people, countries and some relationships. This example can be encoded in a description logic such as *SHIQ* and also in the UCV query language. We show how to accomplish the latter.

- Define classes such as *Country*, *Person*, *Student* and *Canadian*. These are just unary views defined over unary relations, e.g.  $\text{Country}(x) \leftarrow \text{country}(x)$ . Observe that we can blur the distinction between unary views and unary relations and use them interchangeably.
- State that *student* is a subclass of *Person*.

$$\forall x \text{Student}(x) \Rightarrow \text{Person}(x)$$

- State that *Canada* and *England* are both instances of the class *Country*. To accomplish this in the UCV language, we could define *Canada* and *England* as

unary views and ensure that they are contained in the *Country* relation and are disjoint with all other classes/instances.

- Declare *Nationality* as a property relating the classes *Person* (its domain) and *Country* (its range). In the UCV language, we could model this as a binary relation  $Nationality(x, y)$  and impose constraints on its domain and range. e.g.

$$\begin{aligned} dom\_Nationality(x) &\leftarrow Nationality(x, y) \\ range\_Nationality(y) &\leftarrow Nationality(x, y) \\ \forall x(dom\_Nationality(x) &\Rightarrow Person(x)) \\ \forall x(range\_Nationality(x) &\Rightarrow Country(x)) \end{aligned}$$

- State that *Country* and *Person* are disjoint classes.  $\forall x(Country(x) \Rightarrow \neg Person(x))$ .
- Assert that the class *Stateless* is defined precisely as those members of the class *Person* that have no values for the property *Nationality*.

$$\begin{aligned} has\_Nationality(x) &\leftarrow Nationality(x, y) \\ Stateless(x) &\Leftrightarrow Person(x) \wedge \neg has\_Nationality(x) \end{aligned}$$

The above types of statements are reasonably simple to express. In order to achieve more expressiveness, property chaining and property composition have been identified as important reasoning features. To this end, integration of rule-based KR and DL-based KR is an active area of research. The UCV query language has the advantage of being able to express certain types of join, that involve triangular or polygonal patterns, which would not be expressible in description logics. Adding constructs to description logics to do such joins (e.g. role value maps) destroys their tree model property and leads to undecidability [Baader et al. 2003].

We consequently believe the UCV query language has some intriguing potential to be used as a reasoning component for ontologies, possibly to supplement description logics for some specialized applications. We leave this as an open area for future investigation.

## 6.2 Containment and Equivalence

We now briefly examine the application of our results to query containment. Theorem 4.1 implies we can test whether  $Q_1(x) \subseteq Q_2(x)$  under the constraints  $C_1 \wedge C_2 \dots \wedge C_n$  where  $Q_1, Q_2, C_1, \dots, C_n$  are all first order unary conjunctive view queries in 2-NEXPTIME. This just amounts to testing whether the sentence  $\exists x(Q_1(x) \wedge \neg Q_2(x)) \wedge C_1 \wedge \dots \wedge C_n$  is unsatisfiable. Equivalence of  $Q_1(x)$  and  $Q_2(x)$  can be tested with containment tests in both directions.

Of course, we can also show that testing the containment  $Q_1 \subseteq Q_2$  is undecidable if  $Q_1$  and  $Q_2$  are first order unary conjunctive view<sup>≠</sup> queries, first order unary conjunctive view<sup>∩</sup> queries and first order unary conjunctive<sup>rec</sup> view queries.

Containment of queries with negation was first considered in [Sagiv and Yannakakis 1980]. There it was essentially shown that the problem is decidable for queries which do not apply projection to subexpressions with difference. Such a language is disjoint from ours, since it cannot express a sentence such as  $\exists y V_4(y) \wedge \neg \exists x(V_1(x) \wedge \neg V_2(x))$  where  $V_1$  and  $V_2$  are views defined over several variables.

### 6.3 Inclusion Dependencies

Unary inclusion dependencies were identified as useful in [Cosmadakis et al. 1990]. They take the form  $R[x] \subseteq S[y]$ . If we allow  $R$  and  $S$  above to be unary conjunctive view queries, we could obtain *unary conjunctive view containment dependencies*. Observe that the unary views are actually unary projections of the join of one or more relations.

We can also define a special type of dependency called a *proper* first order unary conjunctive inclusion dependency, having the form  $Q_1(x) \subset Q_2(x)$ , where  $Q_1$  and  $Q_2$  are first order unary conjunctive view queries with one free variable. If  $\{d_1, \dots, d_k\}$  is a set of such dependencies, then it is straightforward to test whether they imply another dependency  $d_x$ , by testing the satisfiability of an appropriate first order unary conjunctive view query.

**THEOREM 6.1.** *Implication for the class of unary conjunctive view containment dependencies with subset and proper subset operators is i) decidable in 2-NEXPTIME and ii) finitely controllable. ■*

The results from [Cosmadakis et al. 1990] show that implication is decidable in polynomial time, but not finitely controllable, for either of the combinations i) functional dependencies plus unary inclusion dependencies, ii) full implication dependencies plus unary inclusion dependencies. In contrast, the stated complexity in the above theorem is much higher, due to the increased expressiveness of the dependencies, yet interestingly the class is finitely controllable.

We might also consider unary conjunctive<sup>≠</sup> containment dependencies. The tests in the proof of theorem 5.1 for the 2CM can be written in the form  $Q_1(x) \subseteq Q_2(x)$ , with the exception of the non-emptiness constraints, which must use the proper subset operator. Interestingly also, we can see from the proof of theorem 5.1, that adding the ability to express functional dependencies would also result in undecidability. We can summarise these observations in the following theorem and its corollary.

**THEOREM 6.2.** *Implication is undecidable for unary conjunctive<sup>≠</sup> (or conjunctive<sup>¬</sup>) view containment dependencies with the subset and the proper subset operators. ■*

**COROLLARY 6.3.** *Implication is undecidable for the combination of unary conjunctive view containment dependencies plus functional dependencies.*

### 6.4 Active Rule Termination

The languages in this paper have their origins in [Bailey et al. 1998], where active database rule languages based on views were studied. The decidability result for first order unary conjunctive views can be used to positively answer an open question raised in [Bailey et al. 1998], which essentially asked whether termination is decidable for active database rules expressed using unary conjunctive views.

## 7. EXPRESSIVE POWER OF THE UCV LANGUAGE

As we have seen in the previous sections, the logic UCV is quite suitable to reason about hereditary information such as “ $x$  is a grandchild of  $y$ ” over family trees. This is due to the fact that UCV can express the existence of a directed walk of

length  $k$  in the graph, for any fixed positive integer  $k$ . Therefore, it is natural to also ask what is inexpressible in the logic. In this section, we describe a game-theoretic technique for proving inexpressibility results for UCV. First, we show an easy adaptation of Ehrenfeucht-Fraïssé games for proving that a boolean query is inexpressible in  $UCV(\sigma, \mathcal{V})$  for a signature  $\sigma$  and a finite view set  $\mathcal{V}$  over  $\sigma$ . Second, we extend this result for proving that a boolean query is inexpressible in  $UCV(\sigma)$ . An inexpressibility result of the second kind is clearly more interesting, as it is independent of our choice of the view set  $\mathcal{V}$  over  $\sigma$ . Moreover, such a result places an ultimate limit of what can be expressed by UCV queries. Although it can be adapted to any class  $\mathcal{C}$  of structures, we shall only state our theorem for proving inexpressibility results in UCV *over all finite structures*. For this section only, we shall use  $STRUCT(\sigma)$  to denote the set of all *finite*  $\sigma$ -structures.

Our first goal is quite easy to achieve. Recall that each view set  $\mathcal{V}$  over  $\sigma$  induces a mapping  $\Lambda : STRUCT(\sigma) \rightarrow STRUCT(\mathcal{V})$  as defined in section 2.

**THEOREM 7.1.** *Let  $\mathbf{A}, \mathbf{B} \in STRUCT(\sigma)$ . Define the function  $\Lambda : STRUCT(\sigma) \rightarrow STRUCT(\mathcal{V})$ . Then, the following statements are equivalent:*

- (1)  $\mathbf{A}$  and  $\mathbf{B}$  agree on  $UCV(\sigma, \mathcal{V})$ .
- (2)  $\Lambda(\mathbf{A}) \equiv_1^{UFO(\mathcal{V})} \Lambda(\mathbf{B})$  (i.e. they agree on  $UFO(\mathcal{V})$  formulas of quantifier rank 1).

**PROOF.** Immediate from lemma 2.1, and lemma 3.1.  $\square$

So, to prove that a boolean query  $\mathcal{Q}$  is not expressible in  $UCV(\sigma, \mathcal{V})$ , it suffices to find two  $\sigma$ -structures such that  $\Lambda(\mathbf{A}) \equiv_1^{UFO(\mathcal{V})} \Lambda(\mathbf{B})$ , but  $\mathbf{A}$  and  $\mathbf{B}$  do not agree on  $\mathcal{Q}$ . In turn, to show that  $\Lambda(\mathbf{A}) \equiv_1^{UFO(\mathcal{V})} \Lambda(\mathbf{B})$ , we can use Ehrenfeucht-Fraïssé games.

We now turn to the second task. Let us begin by stating an obvious corollary of the preceding theorem.

**COROLLARY 7.2.** *Let  $\mathbf{A}, \mathbf{B} \in STRUCT(\sigma)$ . For any view set  $\mathcal{V}$ , define the function  $\Lambda^\mathcal{V} : STRUCT(\sigma) \rightarrow STRUCT(\mathcal{V})$ . Then, the following statements are equivalent:*

- (1)  $\mathbf{A}$  and  $\mathbf{B}$  agree on  $UCV(\sigma)$ .
- (2) For any view set  $\mathcal{V}$  over  $\sigma$ , we have  $\Lambda^\mathcal{V}(\mathbf{A}) \equiv_1^{UFO(\mathcal{V})} \Lambda^\mathcal{V}(\mathbf{B})$

This corollary is not of immediate use. Namely, checking the second statement is a daunting task, as there are infinitely many possible view sets  $\mathcal{V}$  over  $\sigma$ . Instead, we shall propose a sufficient condition for this, which employs the easy direction of the well-known homomorphism preservation theorem (see [Hodges 1997]).

**Definition 7.3.** A formula  $\phi$  over a vocabulary  $\sigma$  is said to be *preserved under homomorphisms*, if for any  $\mathbf{A}, \mathbf{B} \in STRUCT(\sigma)$  the following statement holds: whenever  $\mathbf{a} \stackrel{\text{def}}{=} (a_1, \dots, a_m) \in \phi(\mathbf{A})$  and  $h$  is a homomorphism from  $\mathbf{A}$  to  $\mathbf{B}$ , it is the case that  $h(\mathbf{a}) \stackrel{\text{def}}{=} (h(a_1), \dots, h(a_m)) \in \phi(\mathbf{B})$ .

**LEMMA 7.4.** *Conjunctive queries are preserved under homomorphisms.*

**THEOREM 7.5.** *Let  $\mathbf{A}, \mathbf{B} \in \text{STRUCT}(\sigma)$ . To prove that  $\Lambda(\mathbf{A}) \equiv_1^{\text{UFO}(\mathcal{V})} \Lambda(\mathbf{B})$  for all  $\sigma$ -view sets  $\mathcal{V}$ , it is sufficient to show that*

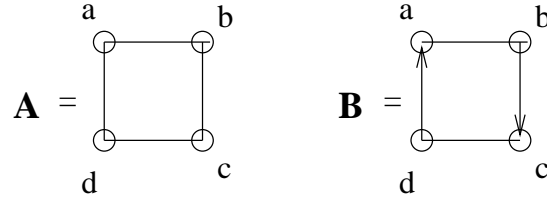
- (1) *For every  $a \in A$ , there exists a homomorphism  $h$  from  $\mathbf{A}$  to  $\mathbf{B}$  and a homomorphism  $g$  from  $\mathbf{B}$  to  $\mathbf{A}$  such that  $g(h(a)) = a$ .*
- (2) *For every  $b \in B$ , there exists a homomorphism  $h$  from  $\mathbf{A}$  to  $\mathbf{B}$  and a homomorphism  $g$  from  $\mathbf{B}$  to  $\mathbf{A}$  such that  $h(g(b)) = b$ .*

**PROOF.** Take an arbitrary  $\sigma$ -view set  $\mathcal{V}$ . We use Ehrenfeucht-Fraïsse game argument. Suppose Spoiler places a pebble on an element  $a$  of  $\Lambda(\mathbf{A})$ , whose domain is  $A$ . Then, the first assumption tells us that there exist homomorphisms  $h : A \rightarrow B$  and  $g : B \rightarrow A$  such that  $g(h(a)) = a$ . Duplicator may respond by placing the other pebble from the same pair on the element  $h(a)$  of  $\Lambda(\mathbf{B})$ . To show this, we need to prove that  $a \mapsto h(a)$  defines an isomorphism between the substructures of  $\Lambda(\mathbf{A})$  and  $\Lambda(\mathbf{B})$  induced by, respectively, the sets  $\{a\}$  and  $\{h(a)\}$ . Let  $V \in \mathcal{V}$ . It is enough to show that  $a \in V(\mathbf{A})$  iff  $h(a) \in V(\mathbf{B})$ . If  $a \in V(\mathbf{A})$ , then we have  $h(a) \in V(\mathbf{B})$  by lemma 7.4. Similarly, if  $h(a) \in V(\mathbf{B})$ , theorem 7.4 implies that  $a = g(h(a)) \in V(\mathbf{A})$ .

For the case where Spoiler plays an element of  $\mathbf{B}$ , we can use the same argument with the aid of the second assumption above. In either case, we have  $\Lambda(\mathbf{A}) \equiv_1 \Lambda(\mathbf{B})$ .  $\square$

This theorem allows us to give easy inexpressibility proofs for a variety of first-order queries. We now give three easy inexpressibility proofs for first-order queries over directed graphs (i.e. structures with one binary relation  $E$ ).

**EXAMPLE 7.1.** *We show that the formula  $\text{SYM} \equiv \forall x, y (E(x, y) \leftrightarrow E(y, x))$  accepting graphs with symmetric  $E$  is not expressible in  $\text{UCV}(\sigma)$ . To do this, consider the graphs  $\mathbf{A}$  and  $\mathbf{B}$  defined as follows*



*Obviously, the graph  $E^{\mathbf{A}}$  is symmetric, while  $E^{\mathbf{B}}$  is not. Consider the functions  $h_1, h_2 : A \rightarrow B$  and  $g : B \rightarrow A$  defined as*

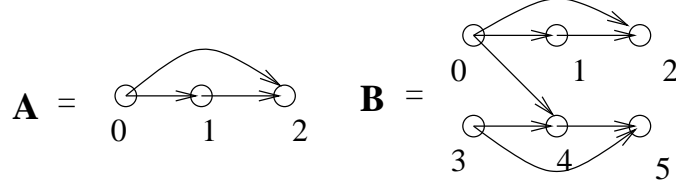
- $h_1(a) = h_1(c) = a$  and  $h_1(b) = h_1(d) = b$ ,
- $h_2(a) = h_2(c) = c$  and  $h_2(b) = h_2(d) = d$ , and
- for  $i \in B$ ,  $g(i) = i$ .

*It is easy to verify that  $h_1$  and  $h_2$  are homomorphisms from  $\mathbf{A}$  to  $\mathbf{B}$ , whereas  $g$  a homomorphism from  $\mathbf{B}$  to  $\mathbf{A}$ . Now, for  $x \in \{a, b\}$ , we have  $g(h_1(x)) = x$  and  $h_1(g(x)) = x$ . For  $x \in \{c, d\}$ , we have  $g(h_2(x)) = x$  and  $h_2(g(x)) = x$ . So, by theorem 7.5 and corollary 7.2, we conclude that  $\text{SYM}$  is not expressible in  $\text{UCV}(\sigma)$  over all finite directed graphs.*

EXAMPLE 7.2. We now show that the transitivity query

$$TRANS \equiv \forall x, y, z (E(x, y) \wedge E(y, z) \rightarrow E(x, z))$$

is not expressible in  $UCV(\sigma)$ . To do this, consider the graphs  $\mathbf{A}$  and  $\mathbf{B}$  defined as

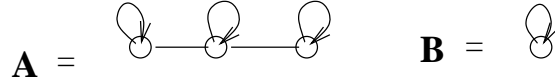


It is obvious that  $\mathbf{A} \models TRANS$ , and it is not the case that  $\mathbf{B} \models TRANS$ . Consider the homomorphisms  $h_1, h_2$  from  $\mathbf{A}$  to  $\mathbf{B}$ , and the homomorphism  $g$  from  $\mathbf{B}$  to  $\mathbf{A}$  defined as

- for  $i \in A$ ,  $h_1(i) = i$ ;
- for  $i \in A$ ,  $h_2(i) = i + 3$ ; and
- for  $i \in B$ ,  $g(i) = i \bmod 3$ .

Then, for  $i \in A$ , we have  $g(h_1(i)) = i$ . Conversely, suppose that  $i \in B$ . If  $i = 0, 1, 2$ , then  $h_1(g(i)) = i$ . Similarly, if  $i = 3, 4, 5$ , then  $h_2(g(i)) = i$ . So, by theorem 7.5 and corollary 7.2, transitivity is not expressible in  $UCV(\sigma)$  over finite directed graphs.

EXAMPLE 7.3. The query  $\forall x, y E(x, y)$  is also not expressible in  $UCV(\sigma)$ . It is easy to apply theorem 7.5 and corollary 7.2 on the following graphs to verify this fact.



## 8. RELATED WORK

Satisfiability of first order logic has been thoroughly investigated in the context of the classical decision problem [Börger et al. 1997]. The main thrust there has been determining for which quantifier prefixes first order languages are decidable. We are not aware of any result of this type which could be used to demonstrate decidability of the first order unary conjunctive view language. Instead, our result is best classified as a new decidable class generalising the traditional decidable unary first-order language (the Löwenheim class [Löwenheim 1915]). Use of the Löwenheim class itself for reasoning about schemas is described in [Theodoratos 1996], where applications towards checking intersection and disjointness of object oriented classes are given.

As observed earlier, description logics are important logics for expressing constraints on desired models. In [Calvanese et al. 1998], the query containment problem is studied in the context of the description logic  $\mathcal{DLR}_{reg}$ . There are certain similarities between this and the first order (unary) view languages we have studied in this paper. The key difference appears to be that although  $\mathcal{DLR}_{reg}$  can be used to define view constraints, these constraints cannot express unary conjunctive

views (since assertions do not allow arbitrary projection). Furthermore,  $\mathcal{DLR}_{reg}$  can express functional dependencies on a single attribute, a feature which would make the UCV language undecidable (see proof of theorem 5.1). There is a result in [Calvanese et al. 1998], however, showing undecidability for a fragment of  $\mathcal{DLR}_{reg}$  with inequality, which could be adapted to give an alternative proof of theorem 5.1 (although inequality is used there in a slightly more powerful way).

Another interesting family of decidable logics are guarded logics. The Guarded Fragment [Andreka et al. 1998] and the Loosely Guarded Fragment [Van Ben- tham 1997] are both logics that have the finite model property [Hodkinson 2002]. The philosophy of UCV is somewhat similar to these guarded logics, since the decidability of UCV also arises from certain restrictions on quantifier use. In terms of expressiveness though, guarded logics seem distinct from UCV formulas, not being able to express cyclic views, such as  $\exists x(V(x)), \text{ where } V(x) \leftarrow R(x, y), R(y, z), R(z, z'), R(z', x)$ .

Another area of work that deals with complexity of views is the view consistency problem, with results given in [Abiteboul and Duschka 1998]. This involves determining whether there exists an underlying database instance that realises a *specific* (bounded) view instance. The problem we have focused on in this paper is slightly more complicated; testing satisfiability of a first order view query asks the question whether there exists an (*unbounded*) view instance that makes the query true. This explains how satisfiability can be undecidable for first order unary conjunctive<sup>≠</sup> view queries, but view consistency for non recursive datalog<sup>≠</sup> views is in *NP*. Monadic views have been recently examined in [Nash et al. 2007], where they were shown to exhibit nice properties in the context of answering and rewriting conjunctive queries using only a set of views. This is an interesting counterpoint to the result of this paper, which demonstrates how monadic views can form the basis of a decidable fragment of first order logic.

## 9. SUMMARY AND FURTHER WORK

In this paper, we have introduced a new decidable language based on the use of unary conjunctive views embedded within first order logic. This is a powerful generalisation of the well known fragment of first order logic using only unary relations (the Löwenheim class). We also showed that our new class is maximal, in the sense that increasing the expressivity of views is not possible without undecidability resulting. Table 1 provides a summary of our decidability results. Note that the Unary Conjunctive<sup>∪</sup> View language corresponds to the extension of UCV by allowing disjunction in the view definition.

We feel that the decidable case we have identified, is sufficiently natural and interesting to be of practical, as well as theoretical interest.

An interesting open problem for future work is to investigate the decidability of an extension to the first order unary conjunctive view language, when equality is allowed to be used outside of the unary views (i.e. included in the first order part). An example formula in this new language is

$$\forall X, Y (V_1(X) \wedge V_2(Y) \Rightarrow X \neq Y)$$

Table 1:  
Summary of Decidability Results for First Order View Languages

Unary Conjunctive View	Decidable
Unary Conjunctive <sup>∪</sup> View	Decidable
Unary Conjunctive <sup>≠</sup> View	Undecidable
Unary Conjunctive <sup>rec</sup> View	Undecidable
Unary Conjunctive <sup>∩</sup> View	Undecidable [Bailey et al. 1998]
Binary Conjunctive View	Undecidable

We conjecture this extended language is decidable, but do not currently have a proof.

For other future work, we believe it would be worthwhile to investigate relationships with description logics and also examine alternative ways of introducing negation into the UCV language. One possibility might be to allow views of arity zero to specify description logic like constraints, such as  $R_1(x, y) \subseteq R_2(x, y)$ .

Finally, there is still an exponential gap between the upper bound complexity of 2-NEXPTIME and lower bound complexity of NEXPTIME-hardness that we derived. The primary reason for this exponential blow-up is the enumeration of all subviews of the views that are present in the formula, which we need for the proof.

#### ACKNOWLEDGMENTS

We thank Sanming Zhou for pointing out useful references on extremal graph theory. We are grateful to Leonid Libkin for helpful comments on a draft of the paper.

#### REFERENCES

- ABITEBOUL, S. AND DUSCHKA, O. 1998. Complexity of answering queries using materialized views. In *Proceedings of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, Seattle, Washington, 254–263.
- ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley Longman Publishing Co., Inc.
- ANDREKA, H., VAN BENTHAM, J., AND NEMETI, I. 1998. Modal logics and bounded fragments of predicate logics. *J. Philosophical Logic* 27, 217–274.
- BAADER, F., CALVANESE, D., MCGUINNESS, D. L., NARDI, D., AND PATEL-SCHNEIDER, P. F. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.
- BAILEY, J. AND DONG, G. 1999. Decidability of first-order logic queries over views. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer, 83–99.
- BAILEY, J., DONG, G., AND RAMAMOHANARAO, K. 1998. Decidability and undecidability results for the termination problem of active database rules. In *Proceedings of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, Seattle, Washington, 264–273.
- BOERGER, E., GRAEDEL, E., AND GUREVICH, Y. 1996. *The Classical Decision Problem*. Springer-Verlag.
- BOLLOBAS, B. 2004. *Extremal Graph Theory*. Dover Publications.
- BOLOS, G. S., BURGESS, J. P., AND JEFFREY, R. C. 2002. *Computability and Logic*. Cambridge University Press.
- BÖRGER, E., GRÄDEL, E., AND GUREVICH, Y. 1997. *The Classical Decision Problem*. Springer-Verlag.

- CALVANESE, D., DE GIACOMO, G., AND LENZERINI, M. 1998. On the decidability of query containment under constraints. In *Proceedings of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, Seattle, Washington, 149–158.
- COSMADAKIS, S., KANELLAKIS, P., AND VARDI, M. 1990. Polynomial time implication problems for unary inclusion dependencies. *Journal of the ACM* 37, 1, 15–46.
- DIESTEL, R. 2005. *Graph Theory*. Springer-Verlag.
- ENDERTON, H. B. 2001. *A Mathematical Introduction To Logic*. A Harcourt Science and Technology Company.
- GAIFMAN, H. 1982. On local and nonlocal properties. In *Logic Colloquium '81*, J. Stern, Ed. North Holland, 105–135.
- GARCIA-MOLINA, H., QUASS, D., PAKONSTANTINOY, Y., RAJARAMAN, A., AND SAGIV, Y. 1995. The TSIMMIS approach to mediation: Data models and language. In *The Second International Workshop on Next Generation Information Technologies and Systems*. Naharia, Israel.
- HALEVY, A. Y. 2001. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases* 10, 4, 270–294.
- HODGES, W. 1997. *A Shorter Model Theory*. Cambridge University Press.
- HODKINSON, I. M. 2002. Loosely guarded fragment of first-order logic has the finite model property. *Studia Logica* 70, 2, 205–240.
- HORROCKS, I. 2005. Applications of description logics: State of the art and research challenges. In *Proc. of 13th International Conference on Conceptual Structures (ICCS)*. 78–90.
- HORROCKS, I., PATEL-SCHNEIDER, P. F., AND VAN HARMELEN, F. 2003. From SHIQ and RDF to OWL: the making of a web ontology language. *Journal of Web Semantics* 1, 1, 7–26.
- LEVY, A., MUMICK, I. S., SAGIV, Y., AND SHMUELI, O. 1993. Equivalence, query reachability, and satisfiability in datalog extensions. In *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Washington D.C., 109–122.
- LEVY, A., RAJARAMAN, A., AND ORDILLE, J. 1996. Querying heterogeneous information sources using source descriptions. In *Proceedings of 22th International Conference on Very Large Data Bases*. Mumbai, India, 251–262.
- LIBKIN, L. 2004. *Elements of Finite Model Theory*. Springer-Verlag.
- LÖWENHEIM, L. 1915. Über möglichkeiten im relativkalkül. *Math. Annalen* 76, 447–470.
- NASH, A., SEGOUFIN, L., AND VIANU, V. 2007. Determinacy and rewriting of conjunctive queries using views: A progress report. In *Proceedings of the International Conference on Database Theory*. 59–73.
- SAGIV, Y. AND YANNAKAKIS, M. 1980. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM* 27, 4, 633–655.
- THEODORATOS, D. 1996. Deductive object oriented schemas. In *Proceedings of ER'96, 15th International Conference on Conceptual Modeling*. Springer, 58–72.
- ULLMAN, J. D. 1997. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory, LNCS 1186*. Springer, Delphi, Greece, 19–40.
- VAN BENTHAM, J. 1997. Dynamic bits and pieces. Tech. Rep. ILLC Research Report LP-97-01, University of Amsterdam.
- WIDOM, J. 1995. Research problems in data warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management*. ACM, Baltimore, Maryland, 25–30.