From Primal Infon Logic with Individual Variables to Datalog

Nikolaj Bjørner[1], Guido de Caso[2], and Yuri Gurevich[1]

[1] Microsoft Research, Redmond, WA, United States
[2] DC, FCEyN, Universidad de Buenos Aires, Buenos Aires, Argentina
{nbjorner, gurevich}@microsoft.com, gdecaso@dc.uba.ar

Abstract. The logic core of Distributed Knowledge Authorization Logic, DKAL, is constructive logic with a quotation construct `said`. This logic is known as the logic of *infons*. The *primal* fragment of infon logic is amenable to linear time decision algorithms when policies and queries are ground. In the presence of policies with variables and implicit universal quantification, but no functions of positive arity, primal infon logic can be reduced to Datalog. We here present a practical reduction of the entailment problem for primal infon logic with individual variables to the entailment problem of Datalog.

Keywords: infon logic, Datalog, PIV, translation

1 Introduction

The entailment problem for a logic L is a decision problem: Given a finite set H of L formulas (the hypotheses) and one additional L formula q (the query), decide whether H entails q. A *reduction* of logic L_1 to logic L_2 is a reduction of the entailment problem for L_1 to that of L_2.

In [1], the entailment problem for the primal infon logic was reduced to the entailment problem for Datalog. Primal infon logic is called there primal infon logic with variables, in short PIV, to distinguish it from propositional primal infon logic introduced earlier in [5]. We will use the abbreviation "PIV" as well.

Here we develop a more practical PIV-to-Datalog reduction. The reduction itself is more efficient, and the resulting Datalog program runs faster; see § 4.5 in this connection. An implementation of the reduction is found at [3].

The definition of infon logic has been refined over time. Propositional primal infon logic was introduced in [4] and investigated in [5]. Its extension with individual variables and the substitution rule, called PIV, was introduced in [1]. These logics employed two quotation constructs, namely `said` and `implied`. In the meantime, `implied` was retired. The reasons for the original introduction of two quotation constructs and subsequent retirement of one of them are related to the use of infon logic in Distributed Knowledge Authorization Language (DKAL) [2] and will be addressed elsewhere. More precisely we retired `said` and then renamed `implied` to `said`. This detail is irrelevant in the context of the present paper, but it is relevant in the context of DKAL. In the rest of the present paper, by default, PIV means the simplified version of the original PIV of [1], without the `implied` construct.

In our main construction we take advantage of the retirement of `implied`. But the work reported here started prior to the retirement of `implied`, and we implemented two PIV-to-Datalog reductions; both of them are found at [3]. One of them reduces PIV with `implied` to Datalog, and the other reduces PIV without `implied` to Datalog.

This paper is self-contained, but familiarity with [5], [1] and Datalog may be useful. In §2, we recall the basic definitions of PIV. We also recall Datalog. In §3, we develop a succinct and economical representation of PIV formulas. Finally, in §4 we reduce PIV to Datalog.

Acknowledgment

We are grateful to Andreas Blass and Thomas Eiter for useful comments.

2 PIV, Axiomless PIV, and Datalog

We recall PIV and narrow the entailment problem for PIV to that of the axiomless PIV, the fragment of PIV without the axioms. We also recall Datalog.

2.1 PIV

PIV is a logic calculus defined in [1]. In this subsection, we recall the definitions of PIV while making the obvious simplifications to reflect the retirement of `implied`. Also, in contrast to [1], this version of PIV is typed.

Terms and formulas. Terms are constants or variables. Each term has a type. One required type is "principal". Atomic formulas are formed as usual from relational symbols and terms. Each argument place of a relational symbol has a type and is supposed to be filled in with a term of that type. Compound formulas are built from atomic formulas and $\top$ by the binary connectives of conjunction ($\wedge$) and implication ($\to$) and unary connectives "p `said`" where p is a term of type "principal". Formulas of the form "p `said` α" are *quotation formulas.*

Example 1. Alice `said` friends(Alice, Bob) is a quotation formula. In this case a principal named Alice is saying that Bob (another principal) is a friend of hers. Quotations can be nested, as in Alice `said` Bob `said` friends(Bob, Chuck).

Quotation prefixes. A *quotation prefix* is a string of the form

$$p_1 \texttt{ said } p_2 \texttt{ said } \ldots p_d \texttt{ said} \tag{1}$$

where every p_i is a term of type principal. The *depth* d of quotation prefix (1) may be zero in which case the quotation prefix is empty. We use `pref` and π, sometimes with subscripts, to denote quotation prefixes.

Every formula β has a unique presentation of the form $\pi\alpha$ where π is the maximal quotation prefix of β and the quotation prefix of α is empty (though quotations may appear in α, for example α may be a conjunction of quotations). The prefix π is denoted $\mathit{\Pi}(\beta)$ and α is called the *body* of β.

Example 2. Following with our example with principals Alice, Bob and Chuck, $\mathit{\Pi}(\text{Alice } \texttt{said} \text{ Bob } \texttt{said} \text{ friends(Bob, Chuck)}) = \text{Alice } \texttt{said} \text{ Bob } \texttt{said}$.

Now we are ready to define the axioms and rules of inference of PIV. Let α and β range over the formulas, `pref` range over the quotation prefixes, and ξ range over the substitutions of terms for variables.

Axioms.

$$\texttt{pref}\ \top$$

Rules of inference.

$$\begin{array}{ll}
(\wedge \text{ elimination}) & \dfrac{\texttt{pref}\ (\alpha \wedge \beta)}{\texttt{pref}\ \alpha} \qquad \dfrac{\texttt{pref}\ (\alpha \wedge \beta)}{\texttt{pref}\ \beta} \\
(\wedge \text{ introduction}) & \dfrac{\texttt{pref}\ \alpha \qquad \texttt{pref}\ \beta}{\texttt{pref}\ (\alpha \wedge \beta)} \\
(\rightarrow \text{ elimination}) & \dfrac{\texttt{pref}\ \alpha \qquad \texttt{pref}\ (\alpha \rightarrow \beta)}{\texttt{pref}\ \beta} \\
(\rightarrow \text{ introduction}) & \dfrac{\texttt{pref}\ \beta}{\texttt{pref}\ (\alpha \rightarrow \beta)} \\
(\text{substitution}) & \dfrac{\alpha}{\xi\, \alpha}
\end{array}$$

A *derivation* D of a formula q from hypotheses H is a sequence $\alpha_1, \alpha_2, \ldots, \alpha_n$ of formulas such that $\alpha_n = q$ and every α_i is an axiom, a hypothesis or the result of applying a derivation rule to one or two preceding formulas. The formulas α_i are the *members* of D. The number n is the length of D. We write $H \vdash q$ and say that H *entails* q if there is a derivation of q from H.

So the entailment problem for PIV is to decide whether a given finite set H of hypotheses entails a formula q. The pair (H, q) forms an instance of the entailment problem. A substitution ξ for variables is *native* to (H, q) if every constant in the range of ξ occurs in (H, q).

Proposition 3 (Theorem 18 in [1]). *A set H of formulas entails a formula q if and only if there is a set H' of native-substitution instances of formulas in H such that q is deducible from H' without using the substitution rule.*

For a proof, see [1, Theorem 18]. Note that the variables of the query q are treated as constants. Without loss of generality we may assume that q is ground.

2.2 Local derivations

We adopt the definition of formula components from [5] (though there it was restricted to propositional formulas).

Definition 4 (Components). The *components* of a formula γ are defined by induction:

- γ itself is a component of γ.
- If $\pi(\alpha * \beta)$ is a component of γ, where $*$ is conjunction or implication, then $\pi\alpha$ and $\pi\beta$ are components of γ.

The components of a set of formulas are the components of its members.

Check that a component of a component of γ is a component of γ.

Example 5. The components of

$$\text{Alice } \texttt{said} \text{ (friends(Alice, Bob)} \wedge \text{friends(Bob, Chuck))}$$

are the formula itself as well as formulas Alice `said` friends(Alice, Bob) and Alice `said` friends(Bob, Chuck).

Proposition 6 (Theorem 5.11 in [5]). *If there is a substitution-free derivation of q from H then there is a derivation of q from H where all members are components of $H \cup \{q\}$.*

In [5], the term "local formula" was used in the connection to the `said`/`implied` interplay. In the absence of `implied`, we are free to use the term for a different purpose.

Definition 7 (Local formulas). A formula α is *local* to a formula γ *via a substitution* ξ if $\alpha = \xi\beta$ for some component β of γ. And α is *local* to a set Γ of formulas *via* ξ if it is local to some member of Γ via ξ.

Example 8. If γ = Alice `said` (friends(Alice, Bob) $\wedge$ friends(Bob, y)), then formula Alice `said` friends(Bob, Alice) is local to γ.

Lemma 9. *A component φ of a formula $\xi\gamma'$ local to a formula γ via a substitution ξ is local to γ via ξ.*

Proof. Since γ' is a component of γ, its components are also components of γ. It suffices to show that $\varphi = \xi\beta$ for a component β of γ'. We do that by induction on φ..

The basic case $\varphi = \gamma'$ is obvious. In the induction step, we have a formula $\pi(\alpha_1 * \alpha_2)$, where the operation $*$ is either conjunction or implication, and $\varphi = \pi\,\alpha_i$ for some i, and (by the induction hypothesis) $\pi(\alpha_1 * \alpha_2) = \xi\delta$ for some component δ of γ'. Then δ has the form $\pi'(\delta_1 * \delta_2)$, so that $\pi = \xi\pi'$, $\alpha_i = \xi\delta_i$, and $\varphi = \pi\,\alpha_i = (\xi\pi')(\xi\delta_i) = \xi(\pi'\delta_i)$. So the desired $\beta = \pi'\delta_i$. $\square$

Definition 10 (Local derivations). A derivation D of q from H is *local* if every member of D is local to (H, q) via a substitution native to (H, q).

Theorem 11. *If $H \vdash q$ then there is a local derivation of q from H.*

Proof. By Proposition 3, there is a set H' of native-substitution instances of the hypotheses such that q is derivable from H' in a substitution-free way. By Proposition 6, there is a derivation D' of q from H' such that all members of D' are components of $H' \cup \{q\}$. By virtue of Lemma 9, every member of D' is local to (H, q) via a substitution native to (H, q). The desired local derivation of q from H is obtained by listing H and then D'. $\square$

2.3 Parse trees and parse forests

For future references, we give a few definitions and introduce some notation. In particular, we will define parse trees of formulas in a way that is convenient for our purposes in this paper. A *labeled tree* is a tree where nodes and edges may have labels. Our trees grow downward, so that the root is the top node. A *forest* is a sequence of disjoint labeled trees. (Making labeled trees disjoint will be no problem as we will be interested in labeled trees only up to isomorphisms.)

The *parse tree* $\mathrm{PT}(\gamma)$ of a formula γ is a labeled tree. We define $\mathrm{PT}(\gamma)$ by induction on γ. If γ is atomic then $\mathrm{PT}(\gamma)$ is a single node labeled with γ. If γ is a quotation formula then $\mathrm{PT}(\gamma)$ is obtained from $\mathrm{PT}(\mathrm{Body}(\gamma))$ by creating an unlabeled parent node u of the root and labeling the new edge with $\Pi(\gamma)$.

Suppose that $\gamma = \alpha * \beta$ where $*$ is conjunction or implication, and let $u = \mathrm{Root}(\mathrm{PT}(\alpha))$, $v = \mathrm{Root}(\mathrm{PT}(\beta))$. The parse tree of γ is obtained by turning the forest

$$(\mathrm{PT}(\alpha),\ \mathrm{PT}(\beta))$$

into a tree by creating a new node w labeled with $*$ (that is with conjunction or with implication) and attaching to it the parse trees for α and β as the left and the right subtrees respectively. The attaching process is as follows.

If α is a quotation formula $\pi\,\alpha'$, then the root u of $\mathrm{PT}(\alpha)$ has a unique child u', and the edge (u, u') is labeled with π. In this case, merge u with w retaining the label $*$ on w and retaining the label π on the edge (w, u'); the node u' becomes the left child of w. Otherwise make u a left child of w and leave the new edge (w, u) unlabeled. The parse tree of β is attached similarly.

That completes the inductive definition of $\mathrm{PT}(\gamma)$. The parse tree of a formula can be constructed in linear time. Note that a node of $\mathrm{PT}(\gamma)$ is unlabeled if only if it is the root and γ is a quotation formula.

Example 12. Figures 1, 2 and 3 contain example parse trees for different PIV formulas.

For every labeled node u on a parse tree $\mathrm{PT}(\gamma)$, we define a quotation prefix $\Pi(u)$ and a formula $\mathbf{F}(u)$. Recall that some edges of $\mathrm{PT}(\gamma)$ are labeled with quotation prefixes and the others are unlabeled. Think of the unlabeled edges as labeled with the empty quotation prefix. To obtain $\Pi(u)$, walk from $\mathrm{Root}(\mathrm{PT}(\gamma))$ down to u and concatenate the labels on your way. To obtain $\mathbf{F}(u)$, let T be the

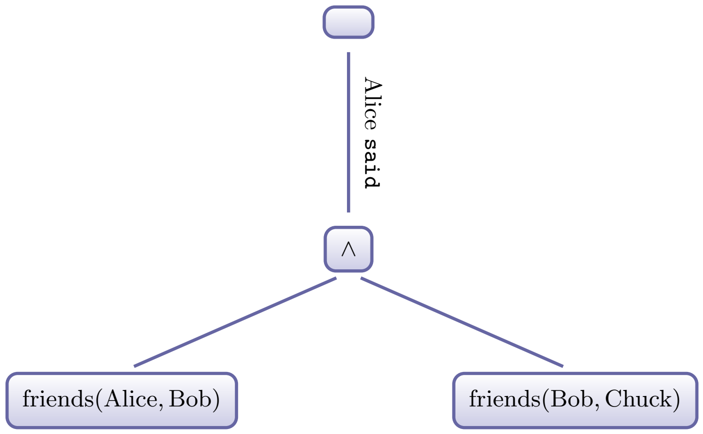

Fig. 1. Parse tree for "Alice `said` (friends(Alice, Bob) ∧ friends(Bob, Chuck))"

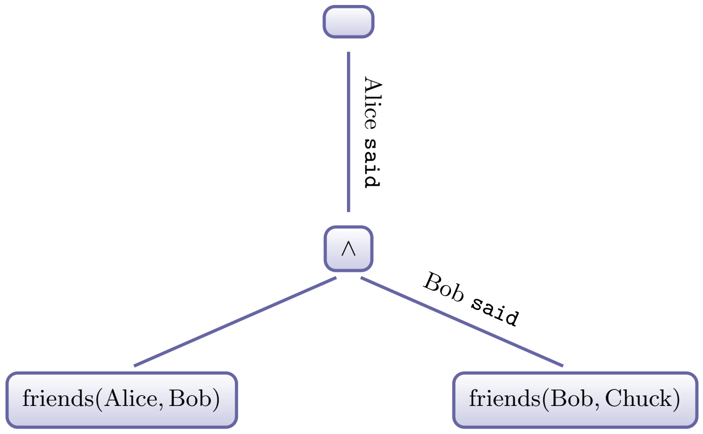

Fig. 2. Parse tree for "Alice `said` (friends(Alice, Bob) ∧ Bob `said` friends(Bob, Chuck))"

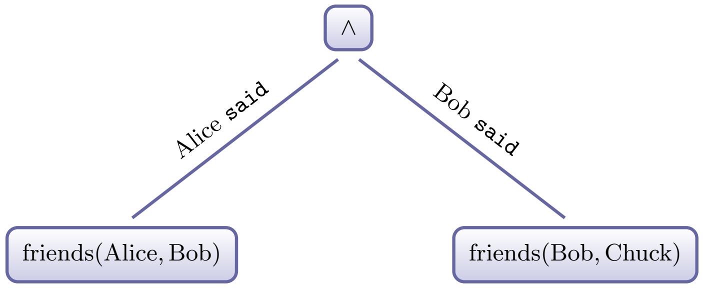

Fig. 3. Parse tree for "(Alice `said` friends(Alice, Bob)) ∧ (Bob `said` friends(Bob, Chuck))"

subtree of $\mathrm{PT}(\gamma)$ rooted at u. It is easy to see that T is the parse tree of some formula α; set $\mathbf{F}(u) = \alpha$.

The *parse forest* of a sequence $\gamma_1, \ldots, \gamma_n$ of formulas is the labeled forest $(\mathrm{PT}(\gamma_1), \ldots, \mathrm{PT}(\gamma_n))$.

2.4 From PIV to Axiomless PIV

Datalog, viewed as a calculus (see the next subsection), has no axioms. In that connection, it is convenient (at least for expository purposes) to dispense with the axioms of PIV first and then reduce the axiomless PIV to Datalog.

Definition 13. *Axiomless* PIV is a fragment of PIV obtained by dropping the axioms.

The entailment problem for PIV reduces to that of the axiomless PIV in linear time. But first let us note that instances (H, q) of the entailment problem for PIV that arise in applications are almost invariably "topless" in the sense that they do not contain $\top$.

Proposition 14. *If H entails q in PIV and if $\top$ does not occur in (H, q) then H entails q already in the axiomless PIV.*

Proof. This follows easily from Theorem 11, but here is a direct proof that gives some additional information. By default we work in the original PIV. Formulas, sets of formulas and derivations are *topless* if they have no occurrences of $\top$; otherwise they are *fancy.*

Lemma 15. *A shortest substitution-free derivation of a topless formula from topless hypotheses is topless.*

Proof (of Lemma 15). Proof by contradiction. Let D be a shortest derivation of a topless formula q from topless hypotheses H. Assume that D is fancy and let M be the set of the fancy members of D of the maximal length. We will prove that any member γ of M is redundant which gives the desired contradiction. More exactly, we will prove this: if a derivation rule R uses γ as a premise to produce a formula δ then δ occurs in D before γ.

So suppose that a member γ of M is used as a premise for a derivation rule R. By the length maximality of γ, we have that
(i) either R is conjunction elimination, in which case γ has the form $\texttt{pref}(\alpha \wedge \beta)$ and the conclusion is either $\texttt{pref}\,\alpha$ or $\texttt{pref}\,\beta$,
(ii) or else R is implication elimination, in which case γ is the major premise of the form $\texttt{pref}\,(\alpha \to \beta)$ and the conclusion is $\texttt{pref}\,\beta$.

We consider only case (ii). Since γ is an implication, it cannot be an axiom. Since all hypotheses are topless, γ cannot be a hypothesis. Thus γ is obtained by an application of a derivation rule Q. By the length maximality, Q is an introduction rule. Given the form of γ, Q is implication introduction obtaining γ from $\texttt{pref}\,\beta$, so that $\texttt{pref}\,\beta$ is an earlier member of D. $\square$

To complete the proof of the proposition, suppose that a topless set H of hypotheses entails a topless formula q. By Proposition 3, there exist a set H' of native-substitution instances of formulas in H and a substitution-free derivation of q from H'. Clearly H' is topless. By Lemma 15, any shortest substitution-free derivation of q from H' is topless, and thus H entails q in Axiomless PIV. $\square$

Theorem 16. *There is a linear-time reduction of the entailment problem for PIV to that of Axiomless PIV.*

Proof. The idea is to view $\top$ as just another nullary relation symbol and treat axioms as additional hypotheses. The obvious problem is that there are infinitely many axioms. Fortunately only few of them are relevant to a given instance (H, q) of the PIV entailment problem. Let A be the set of axioms that are components of $H \cup \{q\}$. Since the total number of the components of (H, q) is $O(n)$, the cardinality $|A| = O(n)$. We show that H entails q in PIV if and only $A \cup H$ entails q in Axiomless PIV. The "if" direction is obvious. We prove the "only if" direction.

Suppose that H entails q. It suffices to show that there is a derivation D of q from H such that A entails every axiom in D. By Proposition 3, there is a set H' of native-substitution instances of formulas in H such that q is deducible from H' without using the substitution rule. By Theorem 11, there is a local derivation D of q from H. Every member α' of D is a substitution instance $\xi\alpha$ of a component α of (H, q). If α' is an axiom, then α is an axiom, and thus A entails α'. $\square$

2.5 Datalog

A Datalog program is a finite set of rules. A Datalog rule has the form

$$\delta_0 \quad :- \quad \delta_1, \delta_2, \ldots, \delta_n \tag{2}$$

where each δ_i is an atomic formula in which every term is a constant or variable. δ_0 is the head of the rule, and the sequence $\delta_1, \delta_2, \ldots, \delta_n$ is the body. The length n of the body can be zero. The vocabulary of a Datalog program P consists of the relation symbols and constants that occur in P.

Entailment. Let P be a Datalog program and let r be an atomic formula (a *query*) in the vocabulary of P possibly extended with additional constants. The vocabulary of (P, r) consists of the relation symbols and constants in (P, r). Call a substitution ξ *native* to (P, r) if every constant in the range of ξ occurs in (P, r). Let ξ range over native substitutions. Construct a sequence

$$\Phi_0 \subseteq \Phi_1 \subseteq \Phi_2 \subseteq \ldots$$

of sets of atomic formulas as follows. $\Phi_0 = \emptyset$. If Φ_i is already constructed then, for every rule (2) of P and every substitution ξ, do this: if atomic formulas $\xi\delta_1, \ldots, \xi\delta_n$ belong to Φ_i then put $\xi\delta_0$ into Φ_{i+1}. The program P *entails* the query r if and only if r belongs to $\bigcup_i \Phi_i$.

Lemma 17. *Every Φ_i is closed under native substitutions. In other words, if $\delta \in \Phi_i$ then $\eta\delta \in \Phi_i$ for every native substitution η.*

Proof. The case of $i = 0$ is trivial. We suppose that the claim has been proven for i, and we prove it for $i+1$. Let $\delta \in \Phi_{i+1}$ and η be a native substitution. Then there is a rule (2) and there is a native substitution ξ such that $\xi\delta_1, \ldots, \xi\delta_n$ belong to Φ_i and $\delta = \xi\delta_0$. By the induction hypothesis, $\eta\xi\delta_1, \ldots, \eta\xi\delta_n$ belong to Φ_i. By the definition of Φ_{i+1}, we have $\eta\delta = \eta\xi\delta_0 \in \Phi_{i+1}$.

3 Succinct representations of PIV formulas

In the rest of this paper, by default, PIV is Axiomless PIV. In this section, we fix an instance (H, q) of the PIV entailment problem and develop a succinct representation of PIV formulas local to (H, q) via substitutions native to (H, q). We presume that H is ordered in some way so that the parse forest for the sequence (H, q) of formulas is well defined. In this section, by default, components are components of (H, q), local formulas are local to (H, q), substitutions are native to (H, q), and nodes are nodes of the parse forest for (H, q).

If σ is a formula or quotation prefix then $\mathrm{Var}(\sigma)$ is the list $x_1, \ldots, x_k$ of different variables of σ in the order they occur in σ, so that if $i < j$ then the first occurrence of x_i precedes the first occurrence of x_j.

Recall that, according to §2.3, for every labeled node u, we have a quotation prefix $\Pi(u)$ and a formula $\mathbf{F}(u)$.

Lemma 18. *For any labeled node u, $\Pi(u)\mathbf{F}(u)$ is a component with $\mathbf{F}(u)$ being the body.*

Proof. Any such node u belongs to the parse tree of some formula γ in $H \cup \{q\}$. Let u_0 be the root of $\mathrm{PT}(\gamma)$. We prove the lemma by induction on the distance d from u_0 to u.

If $d = 0$, then $u = u_0$. In this case, γ is not a quotation formula, $\Pi(u)$ is empty, and $\mathbf{F}(u) = \Pi(u)\mathbf{F}(u) = \gamma$.

Suppose that $d > 0$ and let v be the parent of u. If v is unlabeled then $v = u_0$, $d = 1$, and γ is a quotation formula. In this case, $\Pi(u) = \Pi(\gamma)$ and $\mathbf{F}(u) = \mathrm{Body}(\gamma)$, so that $\Pi(u)\mathbf{F}(u)$ is again γ.

Suppose that the parent node v is labeled. Then $\mathbf{F}(v)$ has the form $\alpha*\beta$ where $*$ is either conjunction or implication. By the induction hypothesis, $\Pi(v)(\alpha * \beta)$ is a component. We consider the case where u is the left child of v. Then $\Pi(u) = \Pi(v)\Pi(\alpha)$, $\mathbf{F}(u) = \mathrm{Body}(\alpha)$, so that $\Pi(u)\mathbf{F}(u) = \Pi(v)\alpha$ which is a component. □

The formula $\Pi(u)\mathbf{F}(u)$ will be called the *component* of u and denoted $\mathrm{Component}(u)$.

Lemma 19. *For every component φ, there is a labeled node u with*

$$\Pi(u) = \Pi(\varphi) \text{ and } \mathbf{F}(u) = \mathrm{Body}(\varphi).$$

Proof. It suffices to fix a formula γ in $H \cup \{q\}$ and restrict attention to the components φ of γ. Recall Definition 4 of the components of γ.

First suppose that $\varphi = \gamma$. If γ is a quotation formula then the desired u is the unique child of the root of $\mathrm{PT}(\gamma)$; otherwise u is the root itself. Second suppose that ψ has the form $\pi(\alpha * \beta)$ where $*$ is conjunction of implication and φ is either $\pi\alpha$ or else $\pi\beta$. We consider the case when $\varphi = \pi\alpha$. By the induction hypothesis, there is a labeled node v on $\mathrm{PT}(\gamma)$ with $\Pi(v) = \pi$ and $\mathbf{F}(v) = \alpha * \beta$. The desired u is a child of v. We have $\Pi(u) = \pi\Pi(\alpha)$ and $\mathbf{F}(u) = \mathrm{Body}(\alpha)$. $\square$

The depth-first traversal of trees naturally extends to the depth first traversal of forests: traverse the first tree, then jump to the root of the second tree, and so on. As a result we have a linear order on our nodes, namely the depth-first order.

Definition 20 (Lead nodes and components).

- A formula α *dominates* a formula β if β is a substitution instance of α.
- The *lead node of a local formula* α is the labeled node u satisfying the following conditions where φ is the component of u.
  - φ dominates α.
  - If ψ is another component that dominates α then the number of the occurrences of variables in φ is greater than or equal to that of ψ.
  - In the depth-first order of nodes, u is the first labeled node whose component dominates α and has the maximal number of the occurrences of variables.
- A *lead node* is the lead node of some local formula (e.g. its own component).
- The *lead component* $\mathrm{Lead}(\alpha)$ of a local formula α is the component of the lead node of α.

Example 21. Figure 4 gives the parse tree for a formula

$$\begin{aligned}&(\text{Alice } \texttt{said} \text{ friends}(\text{Alice}, x) \;\rightarrow\; (\text{Alice } \texttt{said} \text{ friends}(\text{Alice}, \text{Bob})) \wedge \\ &(\text{Alice } \texttt{said} \text{ friends}(\text{Alice}, y) \;\rightarrow\; (\text{Alice } \texttt{said} \text{ friends}(\text{Alice}, \text{Chuck})).\end{aligned}$$

Nodes u, v_1, v_2, w_1 are all lead nodes, and w_1 is the lead nodes of all the leaves.

For every local formula α, the lead node of α and the lead formula are well defined. Indeed α is a substitution instance of and thus dominated by a component. So the set S of nodes whose components dominating α is not empty. A nonempty subset S' of these nodes have components with the maximal possible number of the occurrences of variables. The first node u in S' is the lead node of α, and the component of u is the lead component of α.

Unification As usual, formulas φ_1, φ_2 without common variables are *unifiable* if there is a substitution ξ such that $\xi\varphi_1 = \xi\varphi_2$. Such a substitution ξ is a *unifier* of φ_1, φ_2. It is a *most general unifier* if it is a substitution instance of any other unifier of φ_1, φ_2.

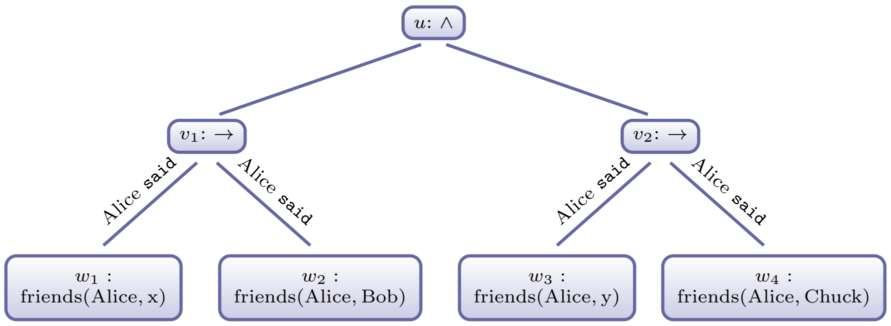

Fig. 4. Domination example

Example 22. Formulas $P(a, y, z_1), P(x, b, z_2)$, where a, b are constants and x, y, z_1, z_2 are variables, are unifiable by means of a substitution ξ such that

$$\xi(x) = a,\ \xi(y) = b,\ \xi(z_1) = z_1,\ \xi(z_2) = z_2.$$

The following proposition is well known.

Proposition 23. *Formulas φ_1, φ_2 are unifiable if they have a common instantiation instance. If they are unifiable, they have a most general unifier.*

Here is a simple construction of a most general unifier for unifiable formulas φ_1, φ_2 with no common variables. Note that φ_1, φ_2 have the same form obtained by replacing all terms with some symbol, e.g. @. Define a graph on the terms (constants and variables) in the two formulas. Two distinct terms form an edge if they occur, in different formulas, at the same position (corresponding to the same occurrence of @ in the form). In Example 22, there are three edges: $(a, x), (b, y), (z_1, z_2)$. Since φ_1, φ_2 are unifiable, every connected component C of the graph contains at most one constant. If C contains a constant c, let $t_C = c$; otherwise let t_C be the lexicographically first variable in C. For every variable z in our formulas, set $\xi(z) = t_C$ where C is the component of z. It is easy to see that $\xi(\varphi_1) = \xi(\varphi_2)$.

4 Reduction to Datalog

Again PIV is by default Axiomless PIV. Given an instance (H, q) of the PIV entailment problem, we construct an instance (P, r) of Datalog such that

– The constants in (P, r) are exactly those in (H, q), so that a substitution is native to (P, r) if and only if it is native to (H, q).
– The set H of hypotheses entails the query q in PIV if and only if the program P entails query r in Datalog using only native substitutions.

Below, by default, components are components of (H, q), local formulas are local to (H, q), substitutions are native to (H, q), and nodes are labeled nodes of the parse forest for (H, q).

4.1 The vocabulary of (P, r)

We describe the vocabulary of (P, r).

– One relation symbol $\mathcal{D}_u$ for every lead node u. The arity of $\mathcal{D}_u$ is the number of different variables in the component $\Pi(u)\mathbf{F}(u)$ of u.
– The constants that occur in (H, q).

4.2 Rendition of local formulas

We translate every PIV formula α local to (H, q) into atomic formula $\mathcal{R}\alpha$, the *rendition* of α, in the vocabulary of (P, r).

Definition 24.

– If α is the component of a lead node u and if $X = \mathrm{Var}(\alpha) = (x_1, \ldots, x_k)$ then $\mathcal{R}\alpha = \mathcal{D}_u(X) = \mathcal{D}_u(x_1, \ldots, x_k)$.
– Let φ be the lead component of α. It follows that $\alpha = \xi\varphi$ for some substitution xi. In this case, $\mathcal{R}\xi\varphi = \xi\mathcal{R}\varphi$.

Lemma 25. *For any local formula α and any native substitution η, if $\mathrm{Lead}(\alpha) = \mathrm{Lead}(\eta\alpha)$ then $\mathcal{R}(\eta\alpha) = \eta\mathcal{R}\alpha$.*

Proof. Let φ be the lead component of α. Then $\alpha = \xi\varphi$ for some ξ. We have $\mathcal{R}(\eta\alpha) = \mathcal{R}(\eta\xi\varphi) = \eta\xi\mathcal{R}(\varphi) = \eta\mathcal{R}\alpha$. □

4.3 Program P and query r

Bodyless rules Every hypothesis φ in H is rendered as a bodyless Datalog rule with head $\mathcal{R}\varphi$.

Rules related to conjunction Each lead component γ of the form $\pi(\alpha' \wedge \beta')$ gives rise to three Datalog rules

$$\begin{array}{lll} \mathcal{R}\gamma & :- & \mathcal{R}\alpha\ \mathcal{R}\beta, \\ \mathcal{R}\alpha & :- & \mathcal{R}\gamma \\ \mathcal{R}\beta & :- & \mathcal{R}\gamma \end{array} \tag{3}$$

where $\alpha = \mathrm{Lead}(\pi\alpha')$ and $\beta = \mathrm{Lead}(\pi\beta')$.

Rules related to implication Each lead component γ of the form $\pi(\alpha' \to \beta')$ gives rise to two Datalog rules

$$\begin{array}{lll} \mathcal{R}\beta & :- & \mathcal{R}\alpha,\ \mathcal{R}\gamma \\ \mathcal{R}\gamma & :- & \mathcal{R}\beta \end{array} \tag{4}$$

where $\alpha = \mathrm{Lead}(\pi\alpha')$ and $\beta = \mathrm{Lead}(\pi\beta')$.

Unification rules In addition to derivation rules related to conjunction and implication, PIV has the substitution rule. Of course Datalog has its own built-in substitution rule. But it does not suffice to establish that $\mathcal{R}(\alpha)$ entails any $\mathcal{R}(\eta\alpha)$ in Datalog. The problem is that α and $\eta\alpha$ may have different lead nodes and so be expressed via different relations. For example, we may have that $\alpha = P(a, y, z_1)$, α is its own lead component with a lead node u, $\eta\alpha = P(a, b, z_1)$, and the lead component of $\eta\alpha$ is $P(x, b, z_2)$ with a lead node v, so that $\mathcal{R}\alpha = \mathcal{D}_u(y, z_1)$ while $\mathcal{R}\eta\alpha = \eta\mathcal{D}_v(x, z_2)$. To this end, we provide program P with additional rules, the unification rules. This would allow us to derive $\mathcal{R}\eta\alpha$ from $\mathcal{R}\alpha$; see Lemma 27.

Every pair of unifiable lead components φ, ψ gives rise to two Datalog rules. We assume without loss of generality that φ and ψ have no common variables. The two rules are

$$\begin{aligned} \xi\mathcal{R}\varphi &\ :-\ \xi\mathcal{R}\psi \\ \xi\mathcal{R}\psi &\ :-\ \xi\mathcal{R}\varphi \end{aligned} \tag{5}$$

where ξ is a most general unifier for φ, ψ.

That completes the construction of P.

Datalog query The desired Datalog query $r = \mathcal{R}(q)$.

Remark 26 (On a single Datalog program). It may seem that by converting $\mathcal{D}_u(X)$ into $D(u, X)$ we get a single Datalog program independent from the given instance of the PIV entailment problem. But note that different relation symbols $\mathcal{D}_u$ may have different types and widths (a.k.a. arities). The trick would allows us, however, to use one relation symbol for all relations $\mathcal{D}_u$ of the same type. Besides, there is a prospect of enabling Datalog to deal with sequences of elements.

4.4 Soundness and completeness

Lemma 27. *By virtue of the unification rules, $\mathcal{R}\alpha$ yields any $\mathcal{R}(\eta\alpha)$ for any local formula α and any native substitution η.*

Proof. If the lead component of $\eta\alpha$ is that of α then, by Lemma 25, $\mathcal{R}\eta\alpha = \eta\mathcal{R}\alpha$, and so $\mathcal{R}\alpha$ yields $\mathcal{R}\eta\alpha$. So suppose that the lead components φ and ψ of α and $\eta\alpha$ respectively are distinct. Without loss of generality, ψ has no variables in common with φ or α.

By the definition of lead components, $\alpha = \eta_0\varphi$ and $\eta\alpha = \eta'\psi$ for some η_0, η'. By the definition of renditions, $\mathcal{R}\alpha = \eta_0\mathcal{R}\varphi$ and $\mathcal{R}(\eta\alpha) = \eta'\mathcal{R}\psi$. Substitutions η and η' have disjoint domains and can be fused into one substitution that we call η. So $\eta\alpha = \eta\psi$. Then $\mathcal{R}(\eta\alpha) = \eta\mathcal{R}\psi$.

Let ξ be a most general unifier for α and ψ, so that $\xi\alpha = \xi\psi$ and $\eta = \chi\xi$ for some χ. We have $\xi\eta_0\varphi = \xi\alpha = \xi\psi$. Clearly $\xi\eta_0$ is a most general unifier for φ and ψ. By the unification rules, $\xi\eta_0\mathcal{R}\varphi$ yields $\xi\eta_0\mathcal{R}\varphi$ which is equal to $\xi\mathcal{R}\psi$. Hence $\xi\mathcal{R}\alpha$ yields $\xi\mathcal{R}\psi$.

We have $\mathcal{R}\alpha \vdash \xi\mathcal{R}\alpha \vdash \xi\mathcal{R}\psi \vdash \chi\xi\mathcal{R}\psi = \eta\mathcal{R}\psi = \mathcal{R}(\eta\alpha)$. $\square$

Theorem 28 (Soundness). *For any local formula ψ,*
if $H \vdash \psi$ in PIV then $P \vdash \mathcal{R}(\psi)$ in Datalog.

The proof the soundness theorem proceeds by induction on the given derivation in PIV. The induction step splits into a number of cases depending on the rule used to derive ψ. The proof is rather tedious but routine, and we skip it. Lemma 27 covers the case of substitution rule in the induction step.

Theorem 29 (Completeness). *For every local formula ψ,*
if $P \vdash \mathcal{R}(\psi)$ in Datalog then $H \vdash \psi$ in PIV.

Again, the proof is rather tedious but routine, and we skip it.

4.5 Complexity considerations

Our purpose in this paper was to reduce primal logic with individual variables (PIV) to the standard Datalog so that any any off-the-shell Datalog tool could be applied to the output. Alternatively one may want to design a specialized Datalog-like tool to work directly on PIV but that is a different direction that we may want to take in future.

The reduction time of our algorithm, that is the time it takes to construct an instance of the Datalog entailment problem from a given instance of the PIV entailment problem, is linear in the output size. The size of the output is $O(N \cdot W)$ where N is the number of rules and W is the maximal width W of the Datalog relations $\mathcal{D}_u$. The number $N = O(n^2)$. There are only linear number of rules related to conjunction and implication but the number of unification rules may be quadratic. (We made an attempt to decrease the number of unification rules; that explains "the maximal number of the occurrences of variables" attribute in Definition 20.) W may be linear, so the output size is $O(n^3)$ in the worst case.

At the end we want to solve a given instance of the PIV entailment problem. So we are really interested in the reduction time plus the time to solve the output instance of the Datalog entailment problem. The latter much depends, in general, on the width of the Datalog relations. In our admittedly limited experience, the width of Datalog relations $\mathcal{D}_u$ has been ≤ 6, and there were few unification rules.

Following [1], we could eliminate unification rules altogether, with a side effect of increasing W. To explain the elimination idea, recall the preamble to the introduction of unification rules in §4.3. There we mentioned an example where PIV formulas $P(a, y, z_1)$, $P(x, b, z_2)$ gave rise to binary Datalog relations $\mathcal{D}_u(y, z_1)$ and $\mathcal{D}_v(x, z_2)$. Instead, we could view formulas $P(a, y, z_1)$, $P(x, b, z_2)$ as substitution instances of one formula $P(x, y, z)$ which would lead us to one ternary Datalog relation. Is the elimination idea good? That depends on your applications. If you are interested in scenarios where the input instances of the PIV entailment problem have few occurrences of constants, the price for the elimination of unification rules may be worth paying. In our applications, typically there are many occurrences of constants and few variables, so that unification rules are beneficial.

Finally let us notice that the second of rules (4) is not safe. Recall, however, that we restrict attention to native substitutions. Only the constants in the original instance of the PIV entailment problem can appear in the head $\mathcal{R}\gamma$ of the rule.

References

1. Blass, A., Gurevich, Y.: Hilbertian deductive systems, infon logic, and Datalog. Bull. of Euro. Assoc. for Theor. Computer Sci. 102, 122–150 (2010), a later version at `http://research.microsoft.com/en-us/um/people/gurevich/Opera/204.pdf`
2. Blass, A., Gurevich, Y., Moskal, M., Neeman, I.: Evidential authorization. In: Nanz, S. (ed.) The Future of Software Engineering. pp. 73–99. Springer (2011)
3. DKAL at CodePlex: `http://dkal.codeplex.com/`
4. Gurevich, Y., Neeman, I.: DKAL 2: A simplified and improved authorization language. Tech. rep., MSR-TR-2009-11, Microsoft Research (2009)
5. Gurevich, Y., Neeman, I.: The infon logic. ACM Trans. on Computational Logic 12:2, article 9 (2009), a later version at `http://research.microsoft.com/en-us/um/people/gurevich/Opera/198.pdf`