Some Notes on Logic Programming with a Relational Machine

EXTENDED ABSTRACT

James Lipton
Dept. of Mathematics
Wesleyan University

Emily Chapman
Dept. of Mathematics
Wesleyan University

Abstract

We study the use of relation calculi for compilation and execution of Horn Clause programs with an extended notion of input and output. We consider various other extensions to the Prolog core.

0 Introduction

Logic programming is programming with predicates in a certain fragment of logic. More broadly speaking, it is programming with executable specifications: code that has independent mathematical meaning consistent with its input-output behavior. Specifications are often formalized as relations. In this paper we explore how Logic Programming itself may be profitably understood, extended and compiled in terms of an underlying equational relational calculus, in which relation variables play a fundamental role, similar in some regards to the role of function variables in the lambda-calculus. The variables in the resulting terms correspond to predicate (second order) variables in the original program, whereas all first order variables are eliminated. Subsequent abstraction provides closed form solutions to all the program predicates.

In this study we use the relation algebra formalism as an executable algebra of logic programs, somewhat in the Backus tradition[Backus]. Our approach here, as in [BroLip, Colp] is to translate logic programs into combinatory relation expressions, which are then executed via rewriting and output-formatting algorithms. It will be useful to illustrate our intentions here with an example.

Consider the following Horn Clause program defining the transitive closure of a graph

```
conn(X,X).
conn(X,Y):-edge(X,Z),conn(Z,Y).
```

```
edge(a,b).
edge(b,c).
edge(a,l).
edge(l,c).
```

and the queries

```
| ?- conn(a,c).
| ?- conn(X,c).
```

This rather carefully chosen example can be easily reformulated in first-order-variable-free relational terms as follows. We introduce the binary relation variables *conn* and *edge* and translate the program into a pair of relation equations. Composition of relations is denoted by " ; ".

$$\begin{aligned} edge &= \{(a,b),(b,c),(a,l),(l,c)\} \\ conn &= id \cup (edge; conn) \end{aligned}$$

where *id* denotes the identity relation. The queries are then represented by the relation expressions:

$$\{(a,c)\} \cap conn \text{ and } (\mathbf{1}; \{(c,c)\}) \cap conn$$

where $\mathbf{1}$ is the universal relation $\mathcal{H} \times \mathcal{H}$ and $\mathcal{H}$ is the set of closed terms in the program signature (hence $(\mathbf{1}; \{(c, c)\})$ represents the set of all pairs whose second component is c).

Some questions that arise naturally at this point are: can such a translation be defined for an arbitrary Horn Clause program, or even for extensions of conventional programs, what kind of relation calculus is suitable for it, and can the resulting relational expressions be easily executed, controlled and optimised?

The aim of these notes is to sketch out in some detail a relation calculus, a translation, a semantics and a rewriting system that give evidence for an affirmative answer to these questions. The relational abstract machine sketched here is described in detail in [Chap, Ruhlen][1].

A translation that will work for all Horn Clause programs (with equality) is necessarily a bit more involved than the one just shown. The one presented here draws on foundational ideas of Tarski, Givant, Freyd, Maddux, and Broome that have appeared at different times in the relational and allegory-theoretic literature although it is different from all of them in that is it aimed at producing *executable* intermediate relational code. Our work thus builds computation into the relational formalism, and shows how the relation calculus offers a new vehicle for proof search and automated deduction as well.

In the earlier paper [BroLip] cited, the authors showed how the existence of a logically correct executable translation of Horn clause programs follows from results of Tarski and others, and explored other "pure" relational programming formalisms. Here that work is considerably refined, and a simple rewriting system explicitly written out and shown correct.

Combinatory approaches to logic programming have appeared elsewhere in the literature: Bellia and Occhiuto develop an algebra of programs that captures unification, rewriting and narrowing in [BelOcc]. Our contribution is to define a more expressive algebra of logic programs, which admits first-order queries, extensions to equational logic, negation and higher order logic, as well as to show that such an algebra can be found within the relation calculus, a well-understood mathematical formalism which easily incorporates other programming paradigms.

A great deal of work has been done in specification refinement using relational specifications based on Hoare's work [Algebra, Naumann, Backh], as well as on relational approaches to hardware design [BroHut, JonShe, BroJon]. In light of this work, the relational translation described here may provide a new formal link between logic programming, hardware specification, and program synthesis.

The work presented here also owes a substantial debt to the logic program transformation ideas of Clark [Lloyd] and Warren [WAM].

1 A Relation Calculus for Horn Clause Programming

In the following sections we formalize a relational theory and a class of relational structures suitable as a target compilation language for logic programs. The relational theory is close in spirit to the positive fragment of the untyped μ-relation calculus MU introduced by deRoever (see e.g. [deRoe, BakRoe, FriasMad]) with additional equations that capture unification requirements over the Herbrand Universe.

1.1 A Language and meta-language for Horn-clause programming

We begin by fixing a first-order signature for a Horn Clause program

$$\Sigma = \mathcal{C}_\Sigma \cup \mathcal{F}_\Sigma,$$

where $\mathcal{C}_\Sigma$ is the set of constant symbols of Σ and $\mathcal{F}_\Sigma$ the set of function symbols. We let $\alpha(f)$ denote the arity of the function symbol f. Let $\mathcal{T}_\Sigma$ be the set of closed terms over Σ. We fix a countable set of variables $\mathcal{X}$(*not* to be confused with relation variables introduced in a later section) and let $\mathcal{T}_\Sigma(\mathcal{X})$ be the set of open terms over $\mathcal{X}$.

Pairing

We will need to formalize an object-level pairing function, in order to account for the coding of vectors of terms in our relational language, that is to say, explicit vectors of terms manipulated by the compiler.

[1]The machine has been implemented in SWI-Prolog. Check `http://www.cs.wesleyan.edu/~lipton` for details.

This is done by adding a second, top-level tier to our concept of term and signature.

Many efficiencies in the translation of n-ary predicates to binary relations are achieved by working with relations that stand for pairs of vectors of length *at least* n, rather than those of length exactly n, since this permits dynamic expansion of arities on demand. By this device we are able to compile programs independently of queries, which may introduce an unpredictable number of new variables. This dictates certain choices of notation and conventions, such as associating pairs to the right (like simple type expressions).

We define the set of [Σ-closed] *extended* terms over Σ by letting Σ^+ be the signature with constants given by all [closed] terms over Σ together with the binary function symbol

$$[\ ,\ ].$$

A Σ-open or Σ-closed extended term is a closed term over this meta-signature, built up from Σ-terms that are open or closed. In other words, we abuse language, and let closed or open refer to the bottom-tier terms that occur in our pairs. We will never need top-tier variables. Spelling it out via an inductive definition:

Definition 1.1 *Let Σ be a signature and $\mathcal{T}_\Sigma$ the set of terms over Σ. Then the set $\mathcal{T}_\Sigma^+$ of extended terms is defined as follows:*

1. *If u is a [closed] term over Σ it is a [Σ-closed] extended term.*
2. *If u_1, u_2 are [Σ-closed] extended terms then so is $[u_1, u_2]$.*

Henceforth open or closed extended term will mean Σ-open or Σ-closed.

Formal vectors and concatenation.

The presence of a formal pairing operator $[\ ,\ ]$ in $\mathcal{T}_\Sigma^+$ allows us to define an internal product of terms within the collection of extended terms. This pairing operator associates to the right, so that we will write, for example

$$[t1, t2, t3, t4] \text{ for } [t_1, [t_2, [t_3, t_4]]]$$

which we can think of as a *formal vector* of terms. If u is the extended term $[u_1, u_2, \dots, u_n]$, and x is an extended term, we write ux to denote the extended term

$$[u_1, u_2, \dots, u_n, x].$$

Thus every extended term u can be assigned a *length* n and components $u_1, \dots, u_n$ as follows:

Definition 1.2 *An extended term t has* **length** *1 if it is a Σ-term, and* **length** $n+1$ *if it is of the form ux, where x is not extended, and u is $[u_1, u_2, \dots, u_n]$. We write $\| u \|$ for the length of u and call $[u_1, u_2, \dots, u_n]$ the formal vector representation of u and u_i the ith component of u.*

Note that, in general, if u is an extended term of the form $[u_1, u_2, \dots, u_n]$, it is a formal vector of length *at least* n (since u_n may be an extended term). If x is a Σ-term and u, v and extended terms $\| x \| = 1$ and $\| uv \| = \| u \| + \| v \|$.

We now associate to each signature Σ a relation calculus $\mathsf{Rel}\Sigma$ into which we will carry out a translation of both language and metalanguage. Among the atomic relation expressions in the calculus will be the n relation symbols f_i^n $\quad (1 \leq i \leq n)$ for each function symbol of arity n in Σ, formalizing the relation of projection of each term of the form $f(u_1, \dots, u_n)$ onto its i-th subterm, and the relation symbol (a, a) for each constant a of Σ.

The relation expression (a, a) is a formal counterpart to the singleton relation $\{(a, a)\}$. The *hd* and *tl* relation expressions are formal projections corresponding to the pairing operator in the meta-signature Σ^+.

1.2 Relational Syntax

Definition 1.3 *Let R_{var} be a countable set of relation variables. The relation calculus* $\mathsf{Rel}\Sigma$ *has the following syntax. The atomic relation expressions are defined by*

$$\langle R_{atom}\rangle \quad ::= \quad \langle R_{var}\rangle \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl \mid \langle R_\Sigma\rangle$$

where R_Σ is the set of relation constants consisting of all expressions $\{f_i^n : f \in \mathcal{F}_\Sigma, \alpha(f) = n, 1 \leq i \leq n\}$ and $\{(a,a) : a \in \mathcal{C}_\Sigma\}$. Compound expressions are given by the following BNF grammar:

$$R \quad ::= \quad \langle R_{atom}\rangle \mid R^o \mid R \cup R \mid R \cap R \mid RR \mid \mathbf{fp}\langle R_{var}\rangle.(R)$$

Juxtaposition RR means composition, which we will sometimes write $R;R$ using the infix operator ";" to enhance legibility. Powers $R^{(n)}$ denote iterations of composition. We denote by $\Re_\Sigma$ the open terms (relation expressions) over the variables R_{var}. When the underlying signature is clear from context we will write $\Re$ for this set of terms. We will say a term is recursion-free *if it contains no occurrences of* **fp**.

Certain compound relation expressions will play an especially important role in the calculus, so we introduce them here.

Definition 1.4 *Define the countable sequence $\{P_i : 1 \leq i\}$ of projection relations as follows:*

$$P_1 \;=\; hd \qquad P_2 \;=\; tl; hd \qquad \cdots \qquad P_n \;=\; (tl)^{(n-1)}; hd \qquad \cdots \tag{1}$$

In the standard semantics to be discussed below, P_i is a relation between a formal vector with at least $i+1$ components and its i-th component. That is to say, P_i's set-theoretic interpretation consists of pairs of formal vectors (u, u_i).

Definition 1.5 *The relation of partial identity up to the first n components on vectors of length at least $n+1$ is defined as follows:*

$$I_n := \bigcap_{1 \leq i \leq n} P_i(P_i)^o \tag{2}$$

We also define the relation of true identity on such vectors by

$$id_n := \left(\bigcap_{1 \leq i \leq n} P_i(P_i)^o\right) \cap tl^n(tl^n)^o. \tag{3}$$

id_n is a subrelation of the identity relation id as a consequence of the equational theory we will define below and, of course, also in the standard semantics.

Definition 1.6 *We also define the relation of* identity on f-terms *for f a function symbol of arity n in the signature Σ by*

$$id_f := \bigcap_{1 \leq i \leq n} f_i^n(f_i^n)^o$$

In the standard interpretation, id_f will denote the set of identical pairs of terms beginning with the function symbol f.

Finally, for each pair of ground terms t_1, t_2 over Σ, define the formal relation expression (t_1, t_2), (whose semantics will be the singleton $\{(t_1, t_2)\}$), as follows:

$$(a,a) \quad := \quad \textit{already defined} \tag{4}$$

$$(f(u_1,\ldots,u_n), f(u_1,\ldots,u_n)) \quad := \quad \bigcap_{1 \leq i \leq n} f_i^n(u_i,u_i)(f_i^n)^o \tag{5}$$

$$(t_1,t_2) := (t_1,t_1)\mathbf{1}(t_2,t_2) \tag{6}$$

It is often convenient to work in a *derived* relational structure where the formal relations (t_1, t_2) are new primitives, satisfying the preceding equations. We will not make use of this variant here.

The reader should also note that unlike id_n, I_n is not a subrelation of the identity relation. In the standard set-theoretic interpretation to be defined below, it will denote all pairs of vectors (ux, uy), with u of length n.

1.3 The Equational Theory $\mathsf{Rel}\Sigma$

We will be using an equational theory which captures the general properties of relations we need as well as the behavior of the encoding of the structure of a Σ-term algebra. We therefore break it down into two components, one we will call DRA, the theory of distributive relation algebras, and the other $R\Sigma$. The latter theory is quite domain specific. It is the part we would expect to change to capture constraint logic programming over other domains, although this matter will not be taken up here.

Note that the containment $A \subseteq B$ abbreviates the equation $A \cap B = B$.

DRA

$$
\begin{gathered}
R \cap R = R \qquad R \cap S = S \cap R \qquad R \cap (S \cap T) = (R \cap S) \cap T \\
R\,id = R \qquad R\mathbf{0} = \mathbf{0} \qquad \mathbf{0} \subseteq R \subseteq \mathbf{1} \\
R \cup R = R \qquad R \cup S = S \cup R \qquad R \cup (S \cup T) = (R \cup S) \cup T \\
R \cup (S \cap R) = R = (R \cup S) \cap R \\
R(S \cup T) = RS \cup RT \qquad\qquad (S \cup T)R = SR \cup TR \\
R \cap (S \cup T) = (R \cap S) \cup (R \cap T) \\
(R \cup S)^o = R^o \cup S^o \qquad (R \cap S)^o = S^o \cap R^o \\
R^{oo} = R \qquad (RS)^o = S^o R^o \\
R(S \cap T) \subseteq RS \cap RT \qquad RS \cap T \subseteq (R \cap TS^o)S \\
id \cup di = \mathbf{1} \qquad id \cap di = \mathbf{0}
\end{gathered}
$$

RΣ

$$
\begin{gathered}
\mathbf{1}(a,a)\mathbf{1} = \mathbf{1} \qquad (a,a)R(a,a) = (a,a) \cap R \qquad (a,a) \subseteq id \\
hd(hd)^o \cap tl(tl)^o \subseteq id \qquad (hd)^o hd = (tl)^o tl = id \qquad (hd)^o tl = \mathbf{1} \\
id_f \equiv \bigcap_{1 \le i \le n} f_i^n (f_i^n)^o \subseteq id \qquad (f_j^n)^o f_i^n = \mathbf{1} \quad (i \neq j) \\
(f_i^n)^o f_i^n = id \qquad (f_i^n)^o g_j^m = \mathbf{0} \\
(f_1)_{i_1}^{n_1} (f_2)_{i_2}^{n_2} \cdots (f_k)_{i_k}^{n_k} \;\cap\; id = \mathbf{0} \\
hd^o f_i^n = 0 = tl^o f_i^n \qquad f_i^n hd = 0 = f_i^n tl \qquad hd \cap id = 0 = tl \cap id \\
id = \bigcup \{(a,a) : a \in \mathcal{C}_\Sigma\} \quad \cup \quad \bigcup \{id_f : f \in \mathcal{F}_\Sigma\}
\end{gathered}
$$

We will also make occasional use of a fixpoint-operator, which will be used in relational structures satisfying axiom **fp**

$$\mathbf{fp}x.\mathcal{E}(x) = \mathcal{E}(\mathbf{fp}x.\mathcal{E}(x)). \tag{7}$$

The so-called *modular law* included in **DRA**

$$RS \cap T \subseteq (R \cap TS^o)S$$

plays an important role in the work discussed here. In the presence of the other **DRA** axioms it is equivalent to its left-modular formulation

$$T \cap RS \subseteq R(R^o T \cap S)$$

and the equational formulation

$$T \cap RS = R(R^o T \cap S) \cap T.$$

The reader should consult [BroLip] for an example of how the modular law is used to improve termination behavior of certain relational rewrite systems, in particular in the computation of transitive closure.

We will have need for the following simplification of the modular law in special contexts.

Lemma 1.7 *In the equational theory DRA, from $SS^o \subset id$ we can infer $A \cap SR = S(S^o A \cap R)$. From $S^o S \subset id$ we can infer $A \cap RS = (AS^o \cap R)S$.*

Proof: By the modular law we have, in the first case, $A \cap SR = S(S^o A \cap R) \cap A$. But $S(S^o A \cap R) \subseteq SS^o A \cap SR \subseteq idA \cap SR = A \cap SR$. Thus $S(S^o A \cap R) \cap A$ reduces to $S(S^o A \cap R)$. The argument for the second claim is symmetric. ▮

In the standard interpretation defined below, the axioms

$$f_i^n hd = 0 = f_i^n tl$$

rule out the occurrence of extended terms as arguments to function symbols from Σ.

Some of the axioms of RΣ are a relational translation of the first-order theory CET: Clark's equality theory [Lloyd]. The last axiom is a relational counterpart to the so-called *domain closure* axiom DC_Σ, satisfied in the Herbrand Universe over a finite signature, stating that every individual is a constant or a term beginning with one of the function symbols in the signature. The axiom scheme

$$(f_1)_{i_1}^{n_1} (f_2)_{i_2}^{n_2} \cdots (f_k)_{i_k}^{n_k} \ \cap \ id = \mathbf{0}$$

for $f_1, \ldots, f_k$ function symbols of Σ of arities $n_1, \ldots, n_k$ respectively, enforces OC, the *occurs-check* axiom scheme $\neg(x = t[x])$ for every term t not identical to x in which x explicitly occurs.

In [Maher], building on earlier results of Mal'cev, Maher shows that CET + OC + DC_Σ is complete, decidable, and admits a partial elimination of quantifiers. This can be exploited to give a compact representation of the set-theoretic content of relations over this equational theory. See [BroLip] for details.

A number of useful identities follow from the equations above, and the definitions (6), (4) and (5).

$$\begin{gathered}
\mathbf{1}^o = \mathbf{1} \qquad \mathbf{0}^o = \mathbf{0} \qquad \mathbf{11} = \mathbf{1} \qquad id^o = id \qquad idR = R \\
A \cap B = B \Rightarrow A \cup B = A \qquad A \subseteq B \Rightarrow A^o \subseteq B^o \\
f_i^n \cap g_j^m = \mathbf{0} \qquad f_i^n \subseteq di \\
(u, v) \cap R = (u, u)R(v, v) \qquad A \cap RS \subseteq R(R^o A \cap R) \\
A \cap RS = R(R^o A \cap S) \cap A = (AS^o \cap R)S \cap A \\
id_n \subseteq id
\end{gathered}$$

as well as the facts that $\cap$ and $\cup$ are the lattice-theoretic infimum and supremum with respect to containment.

1.4 Semantics

1.4.1 Two-tiered structures

Our semantics must interpret formal vectors of terms. It is convenient to do this via a two-tiered notion of model, the lower tier of which is a conventional Σ-algebra, and the upper tier of which interprets the product structure.

Definition 1.8 *A* Σ^+**-algebra** $\mathcal{U}$ *is a pair of structures* $(\mathcal{U}_0, \mathcal{U}_1)$ *with the carrier of the first contained in the second* ($U_0 \subseteq U_1$)*, such that* $\mathcal{U}_0$ *is a* Σ*-algebra and* $\mathcal{U}_1$ *is a set equipped with a binary injective function* $\langle \_, \_ \rangle_{\mathcal{U}} : U_1 \times U_1 \to U_1$. *We will refer to the set* U_1 *as the carrier of the* Σ^+*-algebra* $\mathcal{U}$*, and denote it by* U*, and the function* $\langle \_, \_ \rangle_{\mathcal{U}}$ *as the pairing function supplied by* $\mathcal{U}$.

A Σ^+**-morphism** $\psi : (\mathcal{U}_0, \mathcal{U}_1) \longrightarrow (\mathcal{V}_0, \mathcal{V}_1)$ *is a function from* U_1 *to* V_1 *whose restriction to* U_0 *is a* Σ*-algebra homomorphism from* $\mathcal{U}_0$ *to* $\mathcal{V}_0$ *and which satisfies* $\psi(\langle x, y \rangle_{\mathcal{U}}) = \langle \psi x, \psi y \rangle_{\mathcal{V}}$.

The reader can easily check that $(\mathcal{T}_\Sigma, \mathcal{T}_\Sigma^+)$ is an initial Σ^+-algebra, with pairing function $\langle s, t \rangle = [s, t]$.

1.4.2 Relational structures

We will use the expression *Rel($\mathcal{U}$)* to denote the full relation algebra $\mathcal{P}(U \times U)$.

We will call a structure $\mathfrak{A} = \langle A, \mathbf{0}^{\mathfrak{A}}, \mathbf{1}^{\mathfrak{A}}, id^{\mathfrak{A}}, di^{\mathfrak{A}}, \cap^{\mathfrak{A}}, \cup^{\mathfrak{A}}, (\ )^{o,\mathfrak{A}}, \circ^{\mathfrak{A}} \rangle$ interpreting the relational syntax of the the theory DRA a *distributive relation algebra* or DRA if it satisfies the axioms DRA. It is called a set-theoretic or *standard* DRA if, in addition, A is a subset of the set of binary relations *Rel(U)* $= \mathcal{P}(U \times U)$

on some domain U, with $\mathbf{0}^{\mathfrak{A}}, \mathbf{1}^{\mathfrak{A}}$ the empty and maximal set, and with union, intersection, converse and composition standardly interpreted.

A structure $\mathfrak{A} = \langle A, \mathbf{0}^{\mathfrak{A}}, \mathbf{1}^{\mathfrak{A}}, id^{\mathfrak{A}}, di^{\mathfrak{A}}, (a,a)^{\mathfrak{A}} hd^{\mathfrak{A}}, tl^{\mathfrak{A}}, (f_i^n)^{\mathfrak{A}}, \cap^{\mathfrak{A}}, \cup^{\mathfrak{A}}, (\ )^{o,\mathfrak{A}}, \circ^{\mathfrak{A}} \rangle_{f \in \mathcal{F}_\Sigma; a \in \mathcal{C}_\Sigma}$ is called a $\mathsf{Rel}\Sigma$-algebra if it interprets relational syntax in the obvious way, and satisfies the equational theory $\mathsf{Rel}\Sigma$. It is set-theoretic or *standard* if in addition A is a subset of *Rel*($\mathcal{U}$) for some Σ^+-algebra $\mathcal{U}$, and in addition to the conditions satisfied by standard DRA's, it interprets $R\Sigma$'s relational constant symbols (a,a) and f_i^n in a standard way:

$$(a,a)^{\mathfrak{A}} = \{(a^{\mathcal{U}}, a^{\mathcal{U}})\}$$
$$(f_i^n)^{\mathfrak{A}} = \{(\boldsymbol{x}, \boldsymbol{y}) : (\exists v_1 \cdots v_{n-1})(\boldsymbol{x} = f^{\mathcal{U}}(v_1, \ldots, v_{i-1}, \boldsymbol{y}, v_{i+1}, \ldots, v_{n-1}))\}.$$

Definition 1.9 *Given a Σ^+-algebra $\mathcal{U}$, a $\mathcal{U}$-***interpretation** *is a mapping*

$$[\![\ ]\!]_{\mathcal{U}} : \Re \to Rel(\mathcal{U})$$

satisfying

$$\begin{array}{lcllcl}
[\![(a,a)]\!]_{\mathcal{U}} & = & \{(a^{\mathcal{U}}, a^{\mathcal{U}})\} & [\![tl]\!]_{\mathcal{U}} & = & \{(\langle t_1, t_2\rangle, t_1) : t_1, t_2 \in U\} \\
[\![id]\!]_{\mathcal{U}} & = & \{(u,u) : u \in U\} & [\![di]\!]_{\mathcal{U}} & = & \{(u,v) : u \neq v \in U\} \\
[\![hd]\!]_{\mathcal{U}} & = & \{(\langle t_1, t_2\rangle, t_1) : t_1, t_2 \in U\} & [\![R \cup S]\!]_{\mathcal{U}} & = & [\![R]\!]_{\mathcal{U}} \cup [\![S]\!]_{\mathcal{U}} \\
[\![\mathbf{1}]\!]_{\mathcal{U}} & = & U \times U & [\![R \cap S]\!]_{\mathcal{U}} & = & [\![R]\!]_{\mathcal{U}} \cap [\![S]\!]_{\mathcal{U}} \\
[\![\mathbf{0}]\!]_{\mathcal{U}} & = & \emptyset & [\![RS]\!]_{\mathcal{U}} & = & \{(x,y) : \exists v((x,v) \in [\![R]\!]_{\mathcal{U}} \wedge (v,y) \in [\![S]\!]_{\mathcal{U}})\}
\end{array}$$
$$[\![f_i^n]\!]_{\mathcal{U}} = \{(\boldsymbol{x}, \boldsymbol{y}) : (\exists v_1 \cdots v_{n-1})(\boldsymbol{x} = f^{\mathcal{U}}(v_1, \ldots, v_{i-1}, \boldsymbol{y}, v_{i+1}, \ldots, v_{n-1}))\}$$
$$[\![\mathbf{fp} x.\mathcal{E}(x)]\!]_{\mathcal{U}} = \textstyle\bigcup_{n \geq 0} E^{(n)}$$

where $E^{(0)} = \emptyset$ *and* $E^{(n+1)} = [\![\mathcal{E}(x)]\!]_{\mathcal{U}}[x \leftarrow E^{(n)}]$.

The notation $[\![\_]\!]_{\mathcal{U}}[x \leftarrow R]$ means the function that returns the same value as $[\![\_]\!]_{\mathcal{U}}$ on all inputs save x, for which the output is R.

It is easy to show that any *environment* (function η from relation variables to sets in *Rel*($\mathcal{U}$)) extends uniquely to a $\mathcal{U}$-**interpretation**. We also note that any interpretation is sound for the axioms DRA. We are interested in interpretations which also satisfy the axioms RΣ. We call these $\mathsf{Rel}\Sigma$-*interpretations.*

In addition to the semantics just given, natural categorical models exist: tabular distributive allegories [FreySce] provide a semantics for a considerably more general notion of logic program, over a finite product category [FFL, NFDP, PowKin, CorMont]. We will not have need of this generality to describe relational complilation of conventional Horn Clause programs, although, in the presence of types and other programming features, there are interesting applications of such structures to logic programming, abstract interpretation and compilation.

Lemma 1.10 *If Σ is a signature with at least one function symbol and one constant symbol, and if U is the free Σ^+-algebra (consisting of all closed extended terms over Σ), then Rel($\mathcal{U}$) is a set-theoretic* $\mathsf{Rel}\Sigma$*-algebra in which axiom* **fp** *holds. Thus, the theory* $\mathsf{Rel}\Sigma$ + **fp** *is sound in any $\mathcal{U}$-interpretation.*

The proof is straightforward: every full power set is a complete (Boolean) relation algebra. The axioms RΣ, as mentioned above, translate into the first-order statement of CET +OC+ DC_Σ, which holds in the free extended term algebra $\mathcal{T}_\Sigma^+$.

Definition 1.11 *Let $\mathcal{I}_{\mathcal{U}}$ be the set of interpretations into Rel($\mathcal{U}$).*

We denote by $\sqsubseteq, \sqcap, \sqcup$ the pointwise order and operations on $\mathcal{I}_{\mathcal{U}}$ induced by the corresponding set-theoretic operations on Rel($\mathcal{U}$). $[\![\ ]\!]_{\mathcal{U}}^{\perp}$ and $[\![\ ]\!]_{\mathcal{U}}^{\top}$ denote the maximal and minimal interpretations obtained by setting all relational variables equal to the interpretations of **0** *and* **1** *respectively.*

We now state some useful properties of interpretations, which follow from well known facts about lattices.

Theorem 1.12 *The structure $\langle \mathcal{I}_{\mathcal{U}}, \sqsubseteq, \sqcap, \sqcup, [\![\ ]\!]_{\mathcal{U}}^{\perp}, [\![\ ]\!]_{\mathcal{U}}^{\top} \rangle$ is a complete lattice, with supremum and infimum defined pointwise. Furthermore, each $[\![\ ]\!]_{\mathcal{U}}$ is completely determined by its values on relation variables. If $[\![\ ]\!]_{\mathcal{U}}^1$ and $[\![\ ]\!]_{\mathcal{U}}^2$ are two interpretations with $[\![X]\!]_{\mathcal{U}}^1 \subseteq [\![X]\!]_{\mathcal{U}}^2$ for each relation variable X then $[\![\ ]\!]_{\mathcal{U}}^1 \sqsubseteq [\![\ ]\!]_{\mathcal{U}}^2$.*

Theorem 1.13 *Let $X_1, \ldots, X_n$ be relation variables, and let $\mathcal{F}$ be a corresponding set*

$$\{F_i(X_1, \ldots, X_n) : 1 \leq i \leq n\}$$

of relation expressions with at most the X_i free. Define $\Phi_{\mathcal{F}} : \mathcal{I}_{\mathcal{U}} \to \mathcal{I}_{\mathcal{U}}$ by

$$\Phi_{\mathcal{F}}(\llbracket\,\rrbracket_{\mathcal{U}})(X_i) = \llbracket F_i(X_1, \ldots, X_n)\rrbracket_{\mathcal{U}}.$$

Then

1. *$\Phi_{\mathcal{F}}$ is continuous (hence monotone).*
2. *$\Phi_{\mathcal{F}}$ has a least fixed point, $\llbracket\,\rrbracket^*_{\mathcal{U}}$, which is the least interpretation in $\mathcal{I}_{\mathcal{U}}$ satisfying the equations*

   $$X_i = F_i(X_1, \ldots, X_n).$$

   $\llbracket\,\rrbracket^*_{\mathcal{U}}$ is equal to the supremum of a countable chain $\bigsqcup \llbracket\,\rrbracket^n_{\mathcal{U}}$ of interpretations, where $\llbracket\,\rrbracket^o_{\mathcal{U}} = \llbracket\,\rrbracket^{\perp}_{\mathcal{U}}$ and $\llbracket\,\rrbracket^{n+1}_{\mathcal{U}} = \Phi_{\mathcal{F}}(\llbracket\,\rrbracket^n_{\mathcal{U}})$.

Proof: Straightforward. Proofs of most of these facts occur in one form or another in the literature. The last assertion is the well-known Tarski-Knaster theorem. ▮

For the rest of this paper we fix a finite signature Σ (assumed to have at least one function symbol and one constant), and take the universe $\mathcal{U}$ above to be the set of extended ground terms $\mathcal{T}^+_{\Sigma}$, with pairing function given by

$$\langle\,\rangle(u, v) = [u, v].$$

When $\mathcal{U}$ is so chosen we call the resulting semantics a *standard interpretation* of $\Re$ into *Rel($\mathcal{T}^+_{\Sigma}$)*, and denote it by the unadorned bracket $\llbracket\,\rrbracket$. Since interpretations into the same range can only differ on relation variables and open relation expressions, we sometimes refer to $\llbracket R\rrbracket$, for closed R, as "the" standard interpretation of R.

We close the section by recalling a few elementary definitions from the theory of relations.

Definition 1.14 *A binary relation R is* **reflexive** *if $id \subseteq R$,* **coreflexive** *if $R \subseteq id$,* **functional** *if R^oR is coreflexive, and* **injective** *if RR^o is coreflexive.*

When stating that a certain relation expression is, e.g. injective or coreflexive, we mean that it is so when interpreted in the standard semantics. When we mean that it can be proved so in one of the equational theories defined in this paper, we will so indicate.

Coreflexive relations play a critical role here, since logic programs are translated into such relations, (although some components of logic programs are built up from more general relations). Coreflexive relations are often a good way to code *data* in the calculus of binary relations, that is to say, by embedding *sets* S into the calculus of relations as $S^c = \{(x, x) : x \in S\}$. Diverse approaches to the formalization of data types in the relation calculus has been studied extensively by the Eindhoven group [Backh] and by Bird and de Moor [Algebra].

Three particularly important coreflexive relations associated with any binary relation R are its formal diagonal, domain and range.

$$\begin{aligned}
\Delta(R) &= R \cap id \\
Dom(R) &= R\mathbf{1} \cap id = RR^o \cap id \\
Ran(R) &= R^o\mathbf{1} \cap id = R^oR \cap id
\end{aligned}$$

2 Logic without variables

One of the main results in [TarGiv] due to Tarski, Maddux and Givant (the so-called equipollence theorem), is that every first-order sentence in a theory φ over a theory with a pairing operator has a semantically equivalent equational counterpart $X_\varphi = 1$ in the theory **QRA** of relation algebras with quasi-projections. Tarski and Givant also prove a stronger proof-theoretic version of this result, and exhibit a bijective recursive transformation of sentences φ to their associated relation expressions X_φ and of first-order proofs of the former to equational derivations of the latter. We will not make direct use of the results or proofs here, but the work undertaken in this paper was inspired by it. The main contribution of the paper is to apply this transformation to compilation and evaluation of logic programming by extending it to map proof search in a fragment of first-order logic into rewriting in the appropriate relational theory.

We now sketch a simple proof of a semantic form of the equipollence theorem, for the special case of algebras of relations definable on a term algebra by first-order formulas over a given signature. Let Σ be a finite language. The atomic statements of $\mathcal{T}_\Sigma$ are of the form

$$t_1 = t_2$$

which can be rewritten (after introducing a new variable x) as a conjunction of two *basic* equations $x = t_1$ $x = t_2$. If we continue introducing variables, we can write this as a conjunction of *elementary* or *flat* equations of the form $x = a$ or $x = f(y_1, \ldots, y_n)$ where a is a constant in Σ and the x, y_i are variables. We assume all atomic formulas are of this form.

Now we define a translation $(\ )^r$ from formulas θ in the language of $\mathcal{T}_\Sigma$ to relation expressions as follows. Recall that $\Re_\Sigma$always contains the reserved projection operations *hd* and *tl*[2] Let n be a natural number greater than the largest number of variables occurring in any sentence to be considered below.

Now define, for $(0 \leq i \leq n)$

$$S_1 = P_1 \qquad S_2 = P_2 \qquad \ldots \qquad S_{n-1} = P_{n-1} \qquad S_n = tl^n.$$

where the P_i are given in (1) Observe that, in the standard interpretation for $1 \leq i \leq n$, we have $(u, v) \in [\![S_k]\!]$ iff v is the k^{th} component of u. Now define

$$Q_i^n = \bigcap_{j \neq i \leq n} S_j (S_j)^o \qquad id_n = \bigcap_{j \leq n} S_j (S_j)^o$$

Observe that $(u, v) \in [\![id_n]\!]$ means u and v are vectors of length n and have the same components $(u)_i$ for $(1 \leq i \leq n)$, and that $u [\![Q_i]\!] v$ means all but the i-th component of u and v agree. Let $x_1, \cdots, x_s$ be all the variables, free or bound, that may occur in θ. Recall all atomic formulas θ may be taken *elementary*: either $x_i = a, x_i = x_j$ or $x_{i_0} = f(x_{i_1}, \ldots, x_{i_n})$.

$$\begin{array}{lcllcl}
(x_i = a)^r & = & S_i(a,a)S_i^o \cap id_n & (\varphi \wedge \theta)^r & = & (\varphi)^r \cap (\theta)^r \cap id_n \\
(x_i = x_j)^r & = & S_i S_j^o \cap id_n & (\neg\varphi)^r & = & id_n - (\varphi)^r \\
(x_{i_0} = f(x_{i_1}, \ldots, x_{i_n}))^r & = & \bigcap_j S_{i_0}; f_j^n; S_{i_j}^o \cap id_n & (\exists x_i \varphi)^r & = & Q_i (\varphi)^r Q_i \cap id_n
\end{array}$$

Then we have the following result, where $\mathcal{H}_\Sigma$ denotes the free Σ-algebra, with carrier the set of closed Σ-terms.

Theorem 2.1 (Freyd, Maddux, Tarski) *Let θ be a sentence over the language of $\mathcal{T}_\Sigma$, and let $x_1, \cdots, x_n$ contain all the variables, free or bound, that may occur in θ. Then $\mathcal{H}_\Sigma \models \theta \iff [\![(\theta)^r]\!] = [\![id_n]\!]$. For open formulas φ with free variables among $x_1, \ldots, x_n$*

$$\{(a_1, \ldots, a_n) : ([a_1, \ldots, a_n], [a_1, \ldots, a_n]) \in [\![(\varphi)^r]\!]\} = \{(a_1, \ldots, a_n) : \mathcal{H}_\Sigma \models \varphi[a_1/x_1, \ldots, a_n/x_s]\}$$

The proof is a straightforward induction on the structure of formulas[3].

Proof:

[2] called quasiprojections by Tarski and, in an essentially equivalent form, tabulations of the maximal relation on the product by Freyd[FreySce].

[3] The authors are indebted to Peter Freyd [Freyd92] for (a variant of) this formulation and proof of the variable elimination result.

$\underline{\theta \equiv (x_i = a)}$:
Let $\vec{u}$ be an n-tuple of terms in $\mathcal{H}_\Sigma$, and suppose $(\vec{u}, \vec{u}) \in [\![(x_i = a)^r]\!] = [\![S_i(a, a)S_i^o \cap id_n]\!]$. This implies that $u_i = a$, hence $\mathcal{H}_\Sigma \models (x_i = a)[u_1/x_1, \dots, u_n/x_n]$. Conversely $u_i = a$ forces $(\vec{u}, \vec{u}) \in [\![(x_i = a)^r]\!]$.

$\underline{\theta \equiv (x_i = x_j)}$:
Suppose $(\vec{u}, \vec{u}) \in [\![(x_i = x_j)^r]\!] = [\![S_i S_j^o \cap id_n]\!]$. Then $u_i = u_j$ and $\mathcal{H}_\Sigma \models (x_i = x_j)[\vec{u}/\vec{x}]$. The converse is also immediate.

$\underline{\theta \equiv x_{i_0} = f(x_{i_1}, \dots, x_{i_n})}$:
Suppose $(\vec{u}, \vec{u}) \in [\![(f(x_{i_1}, \dots, x_{i_n}))^r]\!]$, that is to say, in

$$\bigcap\nolimits_j [\![S_{i_0}; f_j^n; S_{i_j}^o]\!] \cap [\![id_n]\!].$$

Then $u_{i_0} = f(u_{i_1}, \dots, u_{i_n})$ and $\mathcal{H}_\Sigma \models \theta[\vec{u}/\vec{x}]$. The converse is immediate.

$\underline{\theta \equiv \varphi \wedge \theta}$:
Suppose $(\vec{u}, \vec{u}) \in [\![(\varphi \wedge \theta)^r]\!]$. Then $(\vec{u}, \vec{u}) \in [\![(\varphi)^r]\!]$ and $(\vec{u}, \vec{u}) \in [\![(\theta)^r]\!]$. By the induction hypothesis, $\mathcal{H}_\Sigma \models \varphi[\vec{u}/\vec{x}]$ and $\mathcal{H}_\Sigma \models \theta[\vec{u}/\vec{x}]$ and hence $\mathcal{H}_\Sigma \models (\varphi \wedge \theta)[\vec{u}/\vec{x}]$

$\underline{\theta \equiv \neg\varphi}$:
Suppose $(\vec{u}, \vec{u}) \in [\![(\neg\varphi)^r]\!]$. Then $(\vec{u}, \vec{u}) \in id_n - [\![(\varphi)^r]\!]$ which is equivalent to saying that $(\vec{u}, \vec{u}) \in [\![(\varphi)^r]\!]$ is not the case. Using the induction hypothesis, this is equivalent to $\mathcal{H}_\Sigma \not\models \varphi[\vec{u}/\vec{x}]$, and hence to $\mathcal{H}_\Sigma \models \neg\varphi[\vec{u}/\vec{x}]$.

$\underline{\theta \equiv \exists x_i \varphi}$:
If $(\vec{u}, \vec{u}) \in [\![(\exists x_i \varphi)^r]\!]$ we must have $(\vec{u}, \vec{u}) \in [\![Q(\varphi)^r Q]\!]$. This means there is an n-tuple of terms $\vec{v}$ with $v_j = u_j$ for every j between 1 and n except i, and $(\vec{v}, \vec{v}) \in [\![(\varphi)^r]\!]$. By the induction hypothesis, this means $\mathcal{H}_\Sigma \models \varphi[\vec{v}/\vec{x}]$. But this is equivalent to $\mathcal{H}_\Sigma \models \exists x_i \varphi[\vec{u}/\vec{x}]$. The converse is left to the reader. ▮

We now turn our attention to the translation of Horn Clause programs. We will want to carry this out in a way that not only preserves meaning, as in the preceding translation, but in a way that is faithful to the operational semantics of the program. We will thus make different, more economical choices, and carry out the translation into $\mathsf{Rel}\Sigma$, defined above, a relational calculus without negation, and with extra constants for capturing the structure of terms in a way that such variable dependent phenomena as unification are easily converted to a variable-free combinatory reduction.

It should be noted that the absence of negation in $\mathsf{Rel}\Sigma$is no handicap, vis-a-vis first-order formulas with negation over the Herbrand Universe, because of the well-known results of Mal'cev [Mal'cev, Maher] that any such formula is equivalent to a two-quantifier formula in which negation occurs only immediately preceding equations between terms, which can be modelled in $\mathsf{Rel}\Sigma$ using di for disequality.

3 Embedded Prolog

3.1 Capturing Horn Clause Reasoning over Term Models

We introduce a family of relational terms that will prove useful in translating logic programs over the Herbrand Universe into relational equations. One component of this translation involves a sequence of intermediate transformations of the constituent predicates $p(t_1, \dots, t_n)$ of the program into relation expressions. We first introduce new variables $\vec{x} = x_1, \dots, x_n$ and rewrite $p(t_1, \dots, t_n)$ as

$$p(x_1, \dots, x_n) \wedge x_1 = t_1 \wedge \dots \wedge x_n = t_n.$$

Conjunctions will be replaced by commas below.

In the relational translation of programs, conjunctions of equations of the form $x_i = t_i$ are translated to relation expressions $\dot{K}(t_1, \ldots, t_n)$ defined by induction on the structure of the terms t_i, as discussed below. We show that the relevant relational translations preserve unification identities derived from Clark's equality theory. This will require a number of ancillary lemmas and definitions, to which the rest of the section is devoted.

Recall that a relation A is *simple* or *functional* if $A^o A \subset id$ and *injective* if $AA^o \subset id$, i.e. if A^o is functional. Injectivity and functionality are preserved under composition and intersection. Thus, in particular, P_j is functional (and hence $(P_j)^o$ injective) for every j, since it is built up from hd and tl by composition.

Definition 3.1 *Let D be the disjoint union of the set open terms over Σ and the set of sequences $\langle t_1, \cdots, t_n\rangle$ of open terms ($m \geq 1$). Define*

$$K : D \to Rel(\mathcal{U})$$

as follows. Let m, n, r be natural numbers greater than 0, and u be a term, or a sequence of n terms over some set $x_1, \cdots, x_m$ of variables. $K_m(u)$ is defined by induction on the structure of u.

$$\begin{aligned}
K(a) &= (a, a)\mathbf{1} \\
K(x_i) &= (P_i)^o \\
K(f(u_1, \ldots, u_r)) &= \bigcap_{i \leq r} f_i^r K(u_i) \\
K(\langle t_1, \cdots, t_n\rangle) &= \bigcap_{i \leq n} P_i K(t_i)
\end{aligned}$$

We also define, for u in D

$$\dot{K}(u) = id \cap K(u)\mathbf{1} \qquad \textit{and} \qquad \Delta(K)(u) = id \cap K(u)$$

Terms t and sequences $\langle t\rangle$ of terms of length 1 are treated differently by K in the preceding definition. Note that the sequences $\langle t_1, \ldots, t_n\rangle$ in D are true members of the cartesian product of copies of the set of terms, and not the formal vectors $[t_1, \ldots, t_n]$ in $\mathcal{T}_\Sigma^+$. Observe also that $K(u)$ is always *injective*, and that for any substitution θ (whose range is among the set of terms over the variables $x_1, \cdots, x_m$ in the universe of discourse), since

$$\theta f(u_1, \ldots, u_n) = f(\theta u_1, \ldots, \theta u_n),$$

we have $K(\theta f(u_1, \ldots, u_n)) = \bigcap_i f_i K(\theta u_i)$

In the standard set-theoretic interpretation, the K terms capture the structure of term equations in the following sense.

Lemma 3.2 *Suppose $\langle t_1, \cdots, t_n\rangle$ is a sequence of terms with free variables among $x_1, \cdots, x_m$*

Let u and y be formal vectors of ground terms of lengths n and m respectively, and u', y' arbitrary extended terms. Let $\vec{u}$ be the sequence $\langle u_1, \ldots, u_n\rangle$. Then

$$(uu', yy') \in [\![K(\langle t_1, \cdots, t_n\rangle)]\!] \iff \mathcal{H}_\Sigma \models \vec{u} = [t_1, \ldots, t_n][y_1/x_1, \cdots, y_m/x_m],$$

where $\mathcal{H}_\Sigma$ is the Herbrand Universe over Σ.

If we think of an m-tuple y of terms as denoting a substitution Θ_y for the variables $x_1, \cdots, x_m$, then $[\![K(\langle t_1, \cdots, t_n\rangle)]\!]$ is the set of instance-substitution pairs

$$\{(\Theta_y[t_1, \ldots, t_n, \_], \Theta_{y\_}) : y \in \mathcal{T}_\Sigma^m\}$$

where the underscores denote arbitrary additional components.

Proof:[of lemma 3.2] We first show by induction on the structure of terms t with free variables among $x_1, \cdots, x_m$ that for a ground term u and and an m-tuple of ground terms $y = \langle y_1, \cdots, y_m\rangle$ we have

$$(u, y) \in [\![K(t)]\!] \quad \text{if and only if} \quad \mathcal{H}_\Sigma \models \vec{u} = t[y_1/x_1, \cdots, y_m/x_m].$$

Suppose t is the constant a. Then $(u, y) \in [\![K(a)]\!] = [\![(a, a)\mathbf{1}]\!] = \{(a, w) : w \in \mathcal{T}_{\Sigma}^{+}\}$ if and only if u is a. But $a = a[y_1/x_1, \cdots, y_m/x_m]$.

If t is the variable x_i then $(u, y) \in [\![K(x_i)]\!]$ if and only if $(y, u) \in [\![P_i]\!]$ which means u is y_i. But this is equivalent to $u = x_i[y_1/x_1, \cdots, y_m/x_m]$.

If t is a term of the form $f(v_1, \ldots, v_n)$ then $(u, y) \in [\![K(t)]\!]$ if and only if $(u, y) \in [\![\bigcap_{i \leq n} f_i^n K(v_i)]\!]$. But this means that for each i between 1 and n $(u, y) \in [\![f_i^n K(v_i)]\!]$, or, equivalently, for some ground terms $w_1, \ldots, w_{n-1}$ and z_i

$$u = f(w_1, \ldots, w_{i-1}, z_i, w_{i+1}, \ldots, w_{n-1})$$

in the Herbrand universe, and $(z_i, y) \in [\![K(v_i)]\!]$. By the induction hypothesis z_i is $v_i[y_1/x_1, \cdots, y_m/x_m]$. Since this is true for every i we obtain

$$u = f(v_1, \ldots, v_n)[y_1/x_1, \cdots, y_m/x_m]$$

which is what we wanted to prove.

If $t = \langle t_1, \cdots, t_n \rangle$ then $(uu', yy') \in [\![K(t)]\!]$ if and only if $(uu', yy') \in [\![\bigcap_{i \leq n} P_i K(t_i)]\!]$. Thus, for each i between 1 and n $(uu', yy') \in [\![P_i K(t_i)]\!]$, hence for each i $(u_i, y) \in K(t_i)$. But then, by the preceding case u_i is $t_i[y_1/x_1, \cdots, y_m/x_m]$, so u is $t[y_1/x_1, \cdots, y_m/x_m]$. ▮

Corollary 3.3 *If $m \geq 0$, u is a formal vector of length m of ground terms in $\mathcal{T}_{\Sigma}$, $\vec{u}$ the corresponding sequence $\langle u_1, \ldots, u_m \rangle$, x is any extended term, and $\langle t_1, \cdots, t_m \rangle$ is an m-tuple of open terms (for the same m) then*

$$(ux, ux) \in [\![id \cap K(\langle t_1, \cdots, t_m \rangle)\mathbf{1}]\!] \iff (\exists \theta)\mathcal{H}_{\Sigma} \models \vec{u} = \langle t_1, \cdots, t_m \rangle \theta$$

where θ is understood to range over substitutions for the variables free in the t_1.

We say two sequences of terms $\langle t_1, \ldots, t_m \rangle$ and $\langle s_1, \ldots, s_m \rangle$ are jointly unifiable by a single substitution θ if for every i, $t_i\theta = s_i\theta$.

Lemma 3.4 *Let $\langle t_1, \ldots, t_n \rangle$ and $\langle s_1, \ldots, s_m \rangle$ be sequences of open terms, with $n \geq m$. Then, if $\langle t_1, \ldots, t_m \rangle$ and $\langle s_1, \ldots, s_m \rangle$ are jointly unifiable, and θ is a most general unifier,*

$$[\![K(\langle t_1, \ldots, t_n \rangle) \cap K(\langle s_1, \ldots, s_m \rangle)]\!]_{\mathcal{U}} = [\![K(\langle t_1\theta, \ldots, t_n\theta \rangle)]\!]_{\mathcal{U}}.$$

If they are not unifiable, the interpretation of the intersection is the empty set.

In addition, if the two sequences are standardized apart

$$[\![\dot{K}(\langle t_1, \ldots, t_n \rangle) \cap \dot{K}(\langle s_1, \ldots, s_m \rangle)]\!]_{\mathcal{U}} = [\![\dot{K}(\langle t_1\theta, \ldots, t_n\theta \rangle)]\!]_{\mathcal{U}} \tag{8}$$

if they are unifiable. The intersection is empty otherwise.

The proof is an immediate consequence of lemma (3.2) and its corollary.

Proof: By lemma (3.2), if u and y are formal vectors of ground terms of lengths n and m respectively, $\vec{u} = \langle u_1, \ldots, u_n \rangle$ and u', y' are arbitrary extended terms, then

$$(uu', yy') \in [\![K(\langle t_1, \ldots, t_n \rangle) \cap K(\langle s_1, \ldots, s_m \rangle)]\!]_{\mathcal{U}}$$
$$\iff$$
$$\mathcal{H}_{\Sigma} \models \vec{u} = \langle t_1, \ldots, t_n \rangle [y_1/x_1, \cdots, y_m/x_m] \wedge \vec{u} = \langle s_1, \ldots, s_n \rangle [y_1/x_1, \cdots, y_m/x_m].$$

Letting θ be the substitution represented by $[y_1/x_1, \cdots, y_m/x_m]$, this implies $\langle t_1, \ldots, t_n \rangle\theta = \langle s_1, \ldots, s_n \rangle\theta$ in the Herbrand universe. Thus $[\![K(\langle t_1, \ldots, t_n \rangle) \cap K(\langle s_1, \ldots, s_m \rangle)]\!]_{\mathcal{U}}$ is precisely the set of instances of $[t_1, \ldots, t_n]\psi$ where ψ is any mgu of $\langle t_1, \ldots, t_n \rangle$ and $\langle s_1, \ldots, s_m \rangle$ whose range is in the set of terms over the variables $x_1, \cdots, x_m$. ▮

This lemma underscores a fundamental feature of variable-elimination in logic programming: *unification is reduced to intersection of combinators*. But one issue remains to be resolved before these results can be fully exploited. The use of $\dot{K}$ expressions in logic program reduction would appear to require standardizing their arguments apart, limiting their usefulness. Fortunately the term sequences that actually arise in our compilation are more robust, as we show in the next, somewhat technical lemma which has surprising consequences for the translation.

Lemma 3.5 (Diagonal Lemma) *Suppose $\langle t_1, \ldots, t_n\rangle$ is a sequence of terms in $\mathcal{T}_\Sigma$ all of whose free variables are among $x_1 \ldots, x_n$ (for the same n). Call a component t_i of such a sequence* **nontrivial** *if $t_i \neq x_i$. Further suppose that for each j between 1 and n if x_j occurs freely in a nontrivial term t_i then $i \neq j$ and t_j is x_j.*

Then for any formal vectors of ground terms u, d in $\mathcal{T}_\Sigma$ of length n and any formal vectors of ground $\mathcal{T}_\Sigma$-terms y, z, if $(uy, dz) \in K(\langle t_1, \ldots, t_n\rangle)$ then $(uy, uy) \in K(\langle t_1, \ldots, t_n\rangle)$ as well. Thus

$$\dot{K}(\langle t_1, \ldots, t_n\rangle) = id \cap K(\langle t_1, \ldots, t_n\rangle). \tag{9}$$

Thus, under these hypotheses, $\dot{K}$ expressions behave like K-terms and (8) holds without the restriction that term arguments be standardized apart. It turns out that the hypotheses of this lemma are always met by the term sequences that will be constructed in the translation of Prolog programs described below. The reader should note, however, that identity (9) does not hold in general.

Proof: Suppose $(uy, dz) \in K(\langle t_1, \ldots, t_n\rangle) = \bigcap_i P_i K(t_i)$. Then, by lemma (3.2), for each i

$$u_i = t_i[d_1/x_1, \ldots, d_n/x_n].$$

Suppose $t_{i_1}, \ldots, t_{i_n}$ are all the nontrivial terms among the t_i, and that $x_{j_1}, \ldots, x_{j_s}$ are variables occurring in these terms. Then, by hypothesis, for all k between 1 and s, $t_{j_k} = x_{j_k}$. Thus

$$\begin{aligned} K(\langle t_1, \ldots, t_n\rangle) &= \bigcap_k P_{j_k} K(x_{j_k}) \cap \bigcap_{i \notin \{j_1, \ldots, j_s\}} P_i K(t_i) \\ &= \bigcap_k P_{j_k} P^o_{j_k} \cap \bigcap_{i \notin \{j_1, \ldots, j_s\}} P_i K(t_i). \end{aligned}$$

Thus for each k between 1 and s, $u_{j_k} = d_{j_k}$. Since the x_{j_k} are only variables free in nontrivial t_i, for each such t_i, $u_i = t_i[d_1/x_1, \ldots, d_n/x_n] = t_i[u_1/x_1, \ldots, u_n/x_n]$. For every other t_i, $t_i = x_i$, so we immediately have $u_i = t_i[u_1/x_1, \ldots, u_n/x_n]$. Therefore

$$\langle u_1, \ldots, u_n\rangle = \langle t_1, \ldots, t_n\rangle[u_1/x_1, \ldots, u_n/x_n],$$

from which $(uy, uy) \in K(\langle t_1, \ldots, t_n\rangle)$ by lemma (3.2). But then every uy in the domain of $K(\langle t_1, \ldots, t_n\rangle)$ is in the domain of $K(\langle t_1, \ldots, t_n\rangle) \cap id$, whence

$$\dot{K}(\langle t_1, \ldots, t_n\rangle) = id \cap K(\langle t_1, \ldots, t_n\rangle).$$

∎

Definition 3.6 *A sequence $\langle t_1, \ldots, t_n\rangle$ of terms in $\mathcal{T}^+_\Sigma$ is called* **clean** *if it satisfies the hypothesis of the preceding lemma.*

Observe that $\langle t_1, \ldots, t_n\rangle$ is clean if and only if the substitution $\varphi_{\vec{t}} = \{t_i/x_i : t_i \text{ non-trivial}\}$ induced by the equations $\{x_i = t_i : 1 \leq i \leq n\}$ is idempotent.

Lemma 3.7 *Suppose $\langle t_1, \ldots, t_n\rangle$ is clean, and $(uy, uy) \in K(\langle t_1, \ldots, t_n\rangle)$. Then the substitution $\theta_u = \{u_1/x_1, \ldots, u_n/x_n\}$ satisfies the equations*

$$x_i = t_i \qquad (1 \leq i \leq n)$$

in the sense that $x_i\theta_u = t_i\theta_u$ for $(1 \leq i \leq n)$.

The proof is just a restatement of the fact, shown in the proof of lemma (3.5) that, for each i,

$$u_i = t_i[u_1/x_1, \ldots, u_n/x_n].$$

3.1.1 Switching Relations

Definition 3.8 *Fix the natural number $n \geq 1$ and let $\vec{\imath} = \langle i_1, \ldots, i_n \rangle$ be a sequence of distinct members of the set $\{1, \ldots, n\}$ natural numbers. Then define the relations $W(\vec{\imath})$ as follows:*

$$W(i_1, \ldots, i_n) = \bigcap_{j=1}^{n} P_{i_j}(P_j)^o.$$

$W(i_1, \ldots, i_n)$ is called a **switching** *relation. We also use the notation $W(\sigma)$ where σ is the permutation of the first n natural numbers into $\vec{\imath}$.*

We now state three useful properties of switching relations, whose simple proofs are left to the reader.

Lemma 3.9 *$W(i_1, \ldots, i_n)$ is functional and injective, provably in the equational theory* $\mathsf{Rel}\Sigma$

Lemma 3.10 *Let $\vec{x}$, and $\vec{y}$ be formal vectors of terms of length at least $n+1$ over the Herbrand Universe. Then*

$$(\vec{x}, \vec{y}) \in [\![W(i_1, \ldots, i_n)]\!] \iff \mathcal{H}_\Sigma \models \bigwedge_{j=1}^{n} x_{i_j} = y_j.$$

Furthermore, if $t_1, \ldots, t_n$ is a tuple of n terms, σ a permutation on $\{1, \ldots, n\}$, and $W = W(\sigma)$

$$[\![W^o \dot{K}(\langle t_1, \ldots, t_n \rangle)\, W]\!] = [\![\dot{K}(\langle t_{\sigma^{-1}(1)}, \ldots, t_{\sigma^{-1}(n)} \rangle)]\!].$$

Lemma 3.11 *If σ is a permutation of the first n nonzero natural numbers*

$$W(\sigma^{-1}) = W(\sigma)^o.$$

provably in the theory $\mathsf{Rel}\Sigma$.

Another useful property of switching relations follows immediately from 1.7.

Lemma 3.12 *One can prove in the equational theory DRA that for any relation expressions R, Q and switching relations W*

$$R \cap WQW^o = W(W^o RW \cap Q)W^o$$

4 Relational Translation of Programs: a sketch

An atomic formula is said to be *linear* or *pure* if its arguments constitute a linear sequence of variables, that is to say, one in which all variables are distinct.

The first step in the Clark completion of a prolog program is the linearization of head predicates. A clause

$$p(t_1, \ldots, t_r) : -Tl$$

is replaced by

$$p(x_1, \ldots, x_r) \colon -x_1 = t_1, \ldots, x_n = t_n, Tl.$$

where the x_i are fresh. The variables free in the tail but *not* in the head are then existentially quantified:

$$p(x_1, \ldots, x_r) : -\exists \vec{y}[x_1 = t_1, \ldots, x_n = t_n, Tl].$$

Now all clauses with head predicate letter p how have precisely the same head, and are thus equivalent to a single clause with this head, but with tail replaced by a disjunction of the newly formed tails:

$$p(x_1, \ldots, x_r) : -\bigvee B_i$$

where each B_i is of the form

$$\exists \vec{y_i}[x_1 = t_{i1}, \ldots, x_n = t_{in}, Tl_i] \tag{10}$$

We are now going to describe a series of additional transformations that have the effect of putting the logic program into a form that maintains its set of ground atomic consequences over the Herbrand Universe, while bringing it closer to the eventual shape of the desired relational equation. The first transformation involves the linearization of the predicates in the tails B_i, with the subsequent generation of equations, some of which then get modified. This is then followed by the introduction of selection operators. We describe it in two steps: linearization and selection.

1. **Linearization:** First we linearize the tails. Proceeding left-to-right, every atomic formula $q(s_1 \ldots, s_m)$ in Tl_i (the non-equational part of B_i) is rewritten as

$$z_1 = s_1, \ldots, z_m = s_m, q(z_1, \ldots, z_m),$$

where

Case 1: Either s_i is a variable that has not occurred earlier in the same atomic predicate, and z_i is identical to s_i (so the equation reads $s_i = s_i$), or

Case 2: z_i is a fresh variable, which is then existentially quantified on the outside. That is to say the existential quantifier $\exists \vec{y_i}$ becomes $\exists \vec{y_i}\vec{z}$, where $\vec{z}$ is the sequence of all distinct fresh variables in the sequence $\vec{z_1} \cdots \vec{z_\ell}$ of variables now occurring in the ℓ predicates $q_1(\vec{z_1}), \ldots, q_\ell(\vec{z_\ell})$ in the non-equational part of the tail.

Starting with the first index greater than r, the arity of the head predicate, which has already required the introduction of variables $x_1, \ldots, x_r$ that may now occur freely in the tail because Case 1, above, obtained, all new distinct existentially quantified variables are renamed in increasing order from left to right, so that *every variable occuring in the entire tail* B_i except for the original bound variables $\vec{y_i}$ is part of a master list $x_1, \ldots x_k$ where k is bounded by the sum of the arities of all the predicate occurrences in the head or tail of the clause.

Finally, we move all generated equations, now of the form $x_i = t_i$ (where the t_i are terms originally ocurring in the clause predicates) to the beginning of the quantifier-free part of the tail. Thus the tail is now of the form

$$\exists \vec{y_i} \exists x_{r+1} \cdots x_w [x_1 = t_1, \ldots, x_w = t_w, q_1(x_{r+1}, \ldots, x_{r+\alpha(1)}), \ldots, q_\ell(x_a, \ldots, x_{a+\alpha(\ell)})],$$

where r is the arity $\alpha(p)$ of the head predicate, $\alpha(i)$ the arity of the i-th tail predicate, and a is $r +$ the sum of the arities of the preceding $\ell - 1$ tail predicates.

Now we rename the remaining existentially bound variables from $\vec{y_i}$ originally occurring in the tail, so they are added to the (end of the) master list of variables x_j.

Finally, for any variable x_j not occurring on the left hand side of one of the equations $x_i = t_i$ the equation $x_j = x_j$ is added to the equational part in its proper place. The equational part is now written in sequence form as $\vec{x} = \vec{t}$. It is a sequence of length bounded by n, the *weight* of the corresponding clause in the original prolog program, which is the sum of the number of variables k occurring in the tail and not in the head, and the arities of the predicate occurrences either in the head or tail of the original clause,

$$wt(C) = k + n + \Sigma_{i=1}^{\ell} r_i.$$

It may be less than the weight because of the possible occurrence of Case 1 above which ensured reuse of variables that would not obstruct linearity (distinctness of variables) of each predicate in the tail. For reference, each clause has the following appearance

$$p(x_1, \ldots, x_n) : - \bigvee B'_i \tag{11}$$

where

$$B'_i = \exists x_{r+1} \cdots x_n [\vec{x} = \vec{t}, q_{i1}(\vec{x_1}), \ldots, q_{i\ell_i}(\vec{x_\ell})] \tag{12}$$

and where each $\vec{x_j}$ is the j-th block of *distinct* variables of length the arity of q_{ij}, starting with the sum of the arities of the preceding predicate occurrences, including that of the head. Letting $\vec{x_0}$ be the sequence of variables now occurring in the head, the entire master sequence of variables $\vec{x} = \vec{x_0}\vec{x_1}, \ldots, \vec{x_\ell}$ may contain repetitions because of Case 1 above, but, we repeat, each block is linear.

2. **Selection:** Each sequence $\vec{x_i}$ of variables occurring in the i-th block in the tail can now be viewed as the result of applying a selection operator ρ_i to the master sequence $\vec{x}$:

$$\rho_i \vec{x} = \vec{x_i}.$$

This yields **normalized** disjuncts of the form

$$D_i = \exists x_{n+1} \cdots x_w [\vec{x} = \vec{t}, q_{i1}(\rho_1 \vec{x}), \ldots, q_{i\ell_i}(\rho_\ell \vec{x})]. \tag{13}$$

We now replace all implications in clauses by bi-implications, as in the Clark completion. We will call the resulting program $\mathcal{P}_n$, consisting of the clauses:

$$p_i(x_1, \ldots, x_n) : -\bigvee D_{ij} \tag{14}$$

the **completed** or **normal form** of $\mathcal{P}$.

Since we will not need this result in our adequacy theorem we do not take the trouble to establish that the preceding program transformations maintain ground atomic consequences in any term model. It is, however, straightforward to prove, using arguments similar to the proof of the fact that the Clark completion of a program has the same atomic (in fact positive) consequences as the program (see e.g. [Shep88, Lloyd]), and repeated applications of the logical equivalence, for fresh $y_1, \ldots, y_n$

$$\exists \vec{x}[A(t_1, \ldots, t_n)] \iff \exists \vec{x} y_1, \ldots, y_n [y_1 = t_1 \wedge \ldots \wedge y_n = t_n \wedge A(y_1, \ldots, y_n)]$$

In the case of Prolog programs with positive and negative equations, treated briefly in a later section, the completed (or *normalized* program) plays a central role, and should be seen as the proper logical formulation of the program captured by our relational translation.

We note following for future reference.

Lemma 4.1 *The sequences $\vec{t}$ that occur in the equational part of each disjunct in the normal form of a program are clean in the sense of definition (3.6).*

This is clear from the way fresh variables are introduced in the normalization process.

Our resulting set of equivalences are now almost ready to be cast in first-order-variable-free relational form. We need to define one more operation on sequences of variables (or on their indices): the permutation σ_i of the indices of $\vec{x}$ *induced* by the selection operator ρ_i.

Definition 4.2 *Let ρ_i be the i-th selection operator defined above. and w' the length of the i-th disjunct's master list of variables. Then the associated permutation of indices*

$$\sigma_i : \{1, 2, \ldots, w'\} \to \{1, 2, \ldots, w'\}$$

is given by letting $x_{\sigma_i(1)}, \ldots, x_{\sigma_i(w')}$ be the sequence $\vec{x_i}$ followed by all remaining variables in $\vec{x}$ in order. σ_i simply shifts the i-th block of variables to the front of the sequence.

4.1 The Relational step

We now transform the normalized program defined above into a finite set of relation equations. For each predicate letter $q \in \Pi$ we introduce a relation *variable* $\overline{q}$. Now consider the normalized clause defining predicate p, of arity m. Each disjunct Θ in its tail of the form (13) is translated into the relation expression $(\Theta)^r$

$$id \cap I_m[\dot{K}(\vec{t}) \cap W(\sigma_1)\overline{p_1}W(\sigma_1)^o \cap \cdots \cap W(\sigma_\ell)\overline{p_\ell}W(\sigma_\ell)^o]I_m. \tag{15}$$

We then translate the whole clause (14) by

$$\overline{p} = (\Theta_1)^r \cup \cdots \cup (\Theta_m)^r.$$

As an immediate consequence of the preceding lemma and lemma (3.5), we have

Lemma 4.3 *If $\dot{K}(\vec{t})$ occurs in a disjunct of a normalized Prolog program, then*

$$\dot{K}(\vec{t}) = K(\vec{t}) \cap id.$$

The result of applying this procedure to all predicate definitions in the canonical completion of a Horn Clause program $\mathcal{P}$ is a set $\mathsf{E}_{\mathcal{P}}$ of n equations $e_1, \ldots, e_n$ in the n predicate letters of the program (now viewed as relation variables). The typical equation e_j being of the form

$$\overline{p}_j = \bigcup_i \Theta_i^j$$

as described above. The Θ_i^j contain the symbols $\overline{p_k}$ as relation variables corresponding to the original predicate letters in the program.

We can write this dependency of relation variables as

$$\overline{p}_i = F_j(\overline{p}_1, \ldots, \overline{p}_n). \tag{16}$$

We call this equational system $\mathsf{E}_{\mathcal{P}}$, the equational translation of $\mathcal{P}$.

"Gaussian elimination"

A final and obvious step is to bind all the variables with the **fp**-operator. We assume that one predicate letter, say p_1 is the one to be queried. Dropping the overbars on relation variables, we rewrite the last equation of system (16) above as

$$p_n = \mathbf{fp} z_n . F_n(p_1, \ldots, p_{n-1}, z_n)$$

and substitute into equation $n-1$, obtaining

$$p_{n-1} = F_{n-1}(p_1, \ldots, p_{n-1}, \mathbf{fp} z_n . F_n(p_1, \ldots, p_{n-1}, z_n))$$

Now we bind the (possibly multiple) occurrences of p_{n-1} and so on, eventually obtaining a single $\Re_\Sigma$-expression

$$p_1 = \mathbf{fp} z_1 . F_1(z_1, \ldots) \tag{17}$$

where the right hand side is a closed (no-free-relation-variable) term. We will call the right-hand expression R_{p_1} and say that R_{p_1} *arises from translation of the predicate p_1 defined by the program $\mathcal{P}$.*

Definition 4.4 *A positive relation expression $R \in \Re_\Sigma$ (or the defining equation $p = R$ for some identifier p) is said to be a (relational)* **logic program** *if R is of the form R_q, i.e. it arises from the translation of a predicate q defined by a logic program $\mathcal{P}$.*

4.1.1 An Example

We illustrate the steps outlined above with an example. We start with the well-known program $\mathcal{P}_{add}$ defining addition of formal numerals over the Herbrand Universe for $\{o, s\}$:

```
add(o,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

Just for reference, the Clark completion (minus the addition of Clark's equality theory) is:

$$\begin{array}{rcl} add(x_1, x_2, x_3) & \longleftrightarrow & \exists x(x_1 = o, x_2 = x, x_3 = x) \ \vee \\ & & \exists x, y, z(x_1 = s(x), x_2 = y, x_3 = s(z), add(x, y, z)). \end{array}$$

After renaming the bound variables x, z as x_4, x_5, (y is already named by x_2) the normal form of $\mathcal{P}$is:

$$\begin{array}{c} add(x_1,x_2,x_3) \quad \longleftrightarrow \\ (\langle x_1,x_2,x_3\rangle = \langle o,x_2,x_2\rangle) \\ \vee \\ \exists x_4,x_5(\langle x_1,x_2,x_3,x_4,x_5\rangle = \langle s(x_4),x_2,s(x_5),x_4,x_5\rangle \\ \wedge\ add(x_4,x_2,x_5)). \end{array}$$

Letting ρ be the selector function $\langle 1,2,3,4,5\rangle \mapsto \langle 4,5,2\rangle$, and σ the associated permutation $\langle 1,2,3,4,5\rangle \mapsto \langle 4,5,2,1,3\rangle$, and writing $\vec{x}$ for $\langle x_1,x_2,x_3,x_4,x_5\rangle$, this can be rewritten:

$$\begin{array}{c} add(x_1,x_2,x_3) \quad \longleftrightarrow \\ \langle x_1,x_2,x_3\rangle = \langle o,x_2,x_2\rangle \\ \vee \\ \exists x_4,x_5(\vec{x} = \langle s(x_4),x_2,s(x_5),x_4,x_5\rangle \ \wedge\ add(\rho\vec{x})). \end{array}$$

The relational translation is:

$$\overline{add} \quad = \quad \dot{K}(o,x_2,x_2)\ \cup I_3[\dot{K}(s(x_4),x_2,s(x_5),x_4,x_5)\ \cap\ W(\sigma)\,\overline{add}\,W(\sigma)^o]I_3.$$

or

$$\overline{add} \quad = \quad \mathbf{fp}.\mathbf{Z}\,\dot{K}(o,x_2,x_2)\ \cup I_3[\dot{K}(s(x_4),x_2,s(x_5),x_4,x_5)\ \cap\ W(\sigma)\,\mathbf{Z}\,W(\sigma)^o]I_3.$$

5 Adequacy of the translation

We now show the translation preserves the intended meaning of the program. To this end we need to define canonical interpretations of both the original program and of the induced equational system (16), or its associated closed form (17).

Let $\mathcal{P}$ be a Prolog program, $\Pi = \{p_1, \ldots, p_n\}$ the predicate symbols occurring in it, Σ its signature, and $T_{\mathcal{P}}$ the Kowalski-VanEmden continuous operator it induces on the power-set of the Herbrand base (see e.g. [Lloyd]). Let $T_{\mathcal{P}}\!\uparrow_0 = \emptyset$, $T_{\mathcal{P}}\!\uparrow_{(n+1)} = T_{\mathcal{P}}(T_{\mathcal{P}}\!\uparrow_n)$ and $T\uparrow_\omega$ the least fixed point $\bigcup T_{\mathcal{P}}\!\uparrow_n$ of $T_{\mathcal{P}}$, that is to say, the least Herbrand model of $\mathcal{P}$.

Define, for each $p_i \in \Pi$ and n in ω

$$\{\![p_i]\!\}^n = \{(u,u) : p_i(u) \in T_p\!\uparrow_n\}$$

and

$$\{\![p_i]\!\}^* = \{(u,u) : p_i(u) \in T_p\!\uparrow_\omega\}$$

Let $\overline{p_i}$ be relation variables associated with the predicate symbols p_i and let

$$\overline{p}_i = F_i(\overline{p}_1, \ldots, \overline{p}_n) \qquad (1 \le i \le n) \tag{18}$$

be the equational translation $\mathsf{E}_{\mathcal{P}}$ of $\mathcal{P}$. Let $\mathcal{F}$ be the set of open relational terms $F_j(\overline{p}_1, \ldots, \overline{p}_n)$, and $R_{p_i} = \mathbf{fp}z_i.F_i(\ldots)$ the associated closed form solutions of this equational translation.

Let $\Phi_{\mathcal{F}}$ be the induced operator on relational interpretations into the standard model $\mathcal{P}(\mathcal{T}_\Sigma^+ \times \mathcal{T}_\Sigma^+)$, as in the statement of theorem (1.13). By the same theorem, $\Phi_{\mathcal{F}}$ has a least fixed point $[\![\,]\!]^*$ which is the supremum of a chain of interpretations $[\![\,]\!]^0 = [\![\,]\!]^\perp, \ldots, [\![\,]\!]^n, \ldots$. We will drop subscripts in the discussion below, when speaking of a typical predicate symbol $p_i \in \Pi$ to simplify notation.

Lemma 5.1 *For each relation variable $\overline{p}$ and any interpretation $[\![\,]\!]$*

$$[\![\overline{p}]\!]^* = [\![R_p]\!]$$

Proof: Straightforward, using the semantics of the **fp** construct and the Tarski-Knaster theorem. ▮

If s is the arity of the program predicate letter p, define the s-th *restriction* $[\![\overline{p}]\!]\big|_s$ of $[\![\overline{p}]\!]$ by

$$[\![\overline{p}]\!]\big|_s = \{([u_1,\ldots,u_s],[u_1,\ldots,u_s]) : \exists x \in \mathcal{T}_\Sigma^+ \, ([u_1,\ldots,u_s,x],[u_1,\ldots,u_s,x]) \in [\![\overline{p}]\!].$$

Then we have the following Soundness and Completeness theorem for our translation:

Theorem 5.2 (Adequacy) *For each $p \in \Pi$*

$$\{\![\overline{p}]\!\}^* = [\![p]\!]^*\big|_s$$

Proof: We establish the two claims below by induction on n, from which the theorem follows immediately.

1. For each natural number n $[\![\overline{p}]\!]^n\big|_s \subseteq \{\![p]\!\}^*$.
2. For each natural number n $\{\![p]\!\}^n \subseteq [\![\overline{p}]\!]^*\big|_s$.

Claim (1): Since $[\![\overline{p}]\!]^0\big|_s$ is empty, the base case is immediate. Suppose the claim holds for n and $(u,u) \in [\![\overline{p}]\!]^n\big|_s$. The clause defining p in the normal form $\mathcal{P}_n$ is of the form

$$p(x_1,\ldots,x_n) : -\bigvee D_i \tag{19}$$

each D_i is of the form

$$D_i = \exists x_{n+1}\cdots x_w[\vec{x} = \vec{t}, q_{i1}(\rho_1\vec{x}),\ldots,q_{i\ell_i}(\rho_\ell\vec{x})]. \tag{20}$$

where the sequence $\vec{t}$, say of length m, is the concatenation $\vec{t_0}\cdots\vec{t_\ell}$ of arguments occurring in the clause

$$p(\vec{t_0}) : -q_{i1}(\vec{t_1}),\ldots,q_{i\ell_i}(\vec{t_\ell}). \tag{21}$$

of the original program $\mathcal{P}$.

The corresponding equation in $\mathsf{E}_\mathcal{P}$ is

$$\overline{p} = \bigcup \Theta_i$$

where

$$\Theta_i = I_s[\dot{K}(\vec{t}) \cap W(\sigma_1)\overline{q_{i1}}W(\sigma_1)^o \cap \cdots \cap W(\sigma_\ell)\overline{q_{i\ell}}W(\sigma_\ell)^o]I_s.$$

So our supposition $(u,u) \in [\![\overline{p}]\!]^n\big|_s$ implies that for some i $(1 \le i \le \ell)$ there is an extended term x, $(ux,ux) \in [\![\Theta_i]\!]^n$. Since $[\![I_s]\!]^n$ consists of formal vectors of length at least $s+1$ sharing their first s components, there is an extended term y such that the length of uy is at least one greater than that of $\vec{t}$, $(uy,uy) \in [\![\dot{K}(\vec{t})]\!]^n$ and for each j between 1 and ℓ, $(uy,uy) \in [\![W(\sigma_j)\overline{q_j}W(\sigma_j)^o]\!]^n$. But then, by corollary (3.3), for some ground substitution θ, $\langle (uy)_1,\ldots,(uy)_m\rangle = \vec{t}\theta$ (where m is the length of $\vec{t}$), from which we have the following consequences:

- $u = \vec{t_o}\theta$, in other words $p(u)$ is a θ-instance of the head $p(\vec{t_0})$ of the original program clause (21).
- for each j between 1 and ℓ, and some extended term z, $(\vec{t}\theta z, \vec{t}\theta z) \in [\![W(\sigma_j)\overline{q_j}W(\sigma_j)^o]\!]^n$, hence (by lemma 3.10)
- $(\vec{t_j}\theta z', \vec{t_j}\theta z') \in [\![\overline{q_j}]\!]^n$, whence $(\vec{t_j}\theta, \vec{t_j}\theta) \in [\![\overline{q_j}]\!]^n\big|_{r_j}$ where r_j is the arity of q_j.

By the induction hypothesis $(\vec{t_j}\theta, \vec{t_j}\theta) \in \{\![q_j]\!\}^*$, whence (by definition of $\{\![\,]\!\}^*$), $q_j(\vec{t_j}\theta) \in T\uparrow_\omega$. But then every θ-instance of the tail of clause (21) holds in the least Herbrand model of $\mathcal{P}$, hence so does $p(\vec{t_0}\theta)$. But recall that $u = \vec{t_0}\theta$ so this means $(u,u) \in \{\![p]\!\}^*$, which is what we wanted to prove.

The other direction, which requires a symmetric argument, is left to the reader. ▮

5.1 Extending the Translation to Equations and Disequations

Enriched Terms

We will have need of a metalanguage of *enriched terms* for discussing certain operations on terms (such as unification and anti-unification). To this end, we introduce operators

$$\sqcap^2, \quad \neg^1$$

of the arities shown, and define an enriched term to be an expression of the form

$$t_1 \sqcap \ldots \sqcap t_n \sqcap \neg s_1 \sqcap \neg s_2 \sqcap \cdots \sqcap \neg s_m \tag{22}$$

where the t_i and s_i are terms (members of the term model of Σ), and where $m \geq 0$ and $n \geq 0$. Simply put: enriched terms are either un-enriched terms or they are built by application of the unary operator $\neg$ to un-enriched terms, and by iterated applications of the binary symbol $\sqcap$ to enriched terms. The binary operator associates to the right so that the typical enriched term may be written in the form (22).

5.1.1 Horn Clauses with Equations

We now consider programs comprised of finite sets of clauses of the form

$$p(t_1, \ldots, t_r) : -\mathbf{e} \,\square\, Tl$$

where $\mathbf{e}$, the *equality component*, is a finite set of *equality formulas*

$$u_1 \sim v_1, \ldots, u_m \sim v_m$$

with u,v terms, and $\sim$ one of the symbols $\{=, \neq\}$, and Tl is a pure Horn clause tail.

We now summarize the normalization procedure for such programs.

Head linearization: We first introduce fresh variables in the head, *à la* Clark, adding new equations to the equality component of the tail, and existentially quantifying all variables "free" in the tail, that is to say, occurring in the tail and not in the head.

$$p(x_1, \ldots, x_r) : -\exists \vec{y}[x_1 = t_1, \ldots, x_r = t_r, \mathbf{e} \,\square\, Tl].$$

The equality component is now the expanded sequence $x_1 = t_1, \ldots, x_r = t_r, \mathbf{e}$.

First processing of equality component: . Modify every equality formula $u \sim v$ in the equality component as follows:

1. If $u \sim v$ is $u = v$ where both u and v are variables, replace v with u everywhere in the tail, and remove v from the sequence of existentially quantified variables.
2. If $u \sim v$ and u is a variable, and the above conditions are not met, do not modify the equation.
3. Otherwise, replace $u \sim v$ with $z = u \wedge z \sim v$, where z is fresh and is added to the sequence of existentially quantified variables.

Linearization of the tail Now we proceed exactly as with pure prolog programs, replacing sequences of terms t_i occurring in atomic predicates in the tail with sequences of variables x_i (which may or may not be fresh, depending on whether Case 1 or Case 2 obtains in the linearization procedure described in section 4), and adding new equations $x_i = t_i$ to the equality component of the clause.

Renaming: finally all variables occurring in the tail are renamed in the same way as in the pure Prolog case, so as to form part of a master list $\vec{x} = x_1, \ldots, x_n$. Each clause now looks like this:

$$p(x_1, \ldots, x_r) : -\exists x_{r+1} \cdots x_n[x_{i_1} \sim t_{i_1}, \ldots, x_{i_k} \sim t_{i_k} \,\square\, Tl'],$$

where the new tail Tl' is the linearized version of the old, and where the equality formulae $x_{i_j} = t_{i_j}$ include processed equality formulae from the original clause as well as equations added during the linearization steps. Unlike the pure Horn clause case, there may be repeated occurrences of variables on the left-hand side of the equality formulas. As before, we force all variables in the clause to occur on the left hand side of equality formulas, via the possible addition of identities $x_i = x_i$ if necessary.

Enrichment: We now replace all equality formulas of the form $x_{i_j} \neq t_{i_j}$ and rewrite then as $x_{i_j} = \neg t_{i_j}$. Finally we gather all equations (now possibly involving enriched terms) which *share a common variable* x on the left hand side

$$x = t_{j_1}, \ldots, x = t_{j_s}$$

and write then as the single equation

$$x = t_{j_1} \sqcap \cdots \sqcap t_{j_s}$$

where some of the t's may be of the form $\neg u$, u a term.

At the expense of adding enriched terms, we have now brought the clause into a form where each variable occurring anywhere in the clause occurs exactly once on the left hand side of an equality formula, and where no occurrences of disequality remain.

$$p(x_1, \ldots, x_r) : -\exists x_{r+1} \cdots x_n[x_1 = s_1, \ldots, x_n = s_n \,\square\, Tl'].$$

Relational step: The clause is now translated to a relation expression in a manner very similar to the pure case

$$\overline{p} = I_s[\Delta(K)(s_1, \ldots, s_n) \cap W(\sigma_1)\overline{q_1}W(\sigma_1)^o \cap \cdots \cap W(\sigma_\ell)\overline{q_\ell}W(\sigma_\ell)^o]I_s$$

where $\Delta(K)(s_1, \ldots, s_n)$ is the *diagonal* $K(s_1, \ldots, s_n) \cap id$, but where we now need to expand the definition of K to include sequences of enriched terms. This is discussed next.

Definition 5.3 *K maps enriched terms, and sequences of such to relation expressions as follows:*

$$\begin{array}{rcl}
K(a) & = & (a, a)\mathbf{1} \\
K(x_i) & = & (P_i)^o \\
K(f(u_1, \ldots, u_n)) & = & \bigcap_{i \le n} f_i^n K(u_i) \\
K(\langle t_1, \cdots, t_n \rangle) & = & \bigcap_{i \le n} P_i K(t_i) \\
K(\neg t) & = & di K(t) \\
K(t_1 \sqcap t_2) & = & K(t_1) \cap K(t_2)
\end{array}$$

We now need to extend lemma 3.7 in the appropriate way. It should be noted that in the enriched case, the analogue of diagonal lemma (3.5) is of no use. We can no longer assume that sequences $\langle t_1, \ldots, t_n \rangle$ of enriched terms appearing in expressions of the form $\Delta(K)(\langle t_1, \ldots, t_n \rangle)$ are *clean* in the sense of that lemma. Thus the diagonal $\Delta(K)(\langle t_1, \ldots, t_n \rangle)$ may be a proper subset of $\dot{K}\langle t_1, \ldots, t_n \rangle)$, which is why we are forced to use the diagonal in the translation. With this choice, we obtain the correct translation of equational component of a logic program clause in the following sense.

Lemma 5.4 *Let $x_{i_1} \sim t_{i_1}, \ldots, x_{i_k} \sim t_{i_k}$ be a sequence of equality formulas, where the t_{i_j} are terms, and $\langle s_0, \ldots, s_n \rangle$ the sequence of enriched terms produced by the enrichement process described above, with $x_1, \ldots, x_n$ the list of all free variables occurring in the t_{i_j} (and hence the s_i). Then:*

$$\begin{array}{rcl}
[\![\Delta(K)(\langle s_0, \ldots, s_n \rangle)]\!] & = & \{(v, v) : v = uz \text{ where} \\
& & u, z \in \mathcal{T}_\Sigma^+, u = [u_1, \ldots, u_n] \text{ and} \\
& & x_{i_1}\theta_u \sim t_{i_1}\theta_u, \ldots, x_{i_k}\theta_u \sim t_{i_k},
\end{array}$$

where θ_u is the substitution $\{u_1/x_1, \ldots, u_n/x_n\}$. in other words, the diagonal of $K(\langle s_0, \ldots, s_n \rangle)$ consists of precisely those ground substitutions that satisfy the original equations and disequations.

With this lemma, it is easy to extend the adequacy theorem (5.2) to the equational Horn clause case. For details the reader is referred to [Chap].

6 Evaluation

The equational theory $\mathsf{E}_{\mathcal{P}}$ induced by a program $\mathcal{P}$ now opens up a new way of computing queries to logic programs of a quite general nature, via directed relational rewriting. In this section we sketch a rewrite system, together with a deterministic rewriting strategy that simulates SLD resolution of conventional prolog queries . It is our first concern to show that the conventional operational interpretation of the original program can be recovered relationally, before addressing alternative evaluations.

Suppose $\mathcal{P}$ is a program with predicate symbols $\Pi = \{p_1, \ldots, p_n\}$ and G is a query, say of the form

$$q_1(\vec{t_1}), \ldots, q_m(\vec{t_m}) \tag{23}$$

where the q_i are members of Π of arity $\alpha(i)$ and the $\vec{t_i}$ are tuples of terms of appropriate arities. Let the corresponding relation variables in the induced equational theory $\mathsf{E}_{\mathcal{P}}$ be $\overline{p_i}$.

Definition 6.1 *The translation A_G of a query G of the form (23) is the relation expression:*

$$\dot{K}(\vec{t_1} \ldots \vec{t_m}) \cap \overline{q_1}' \cap \ldots \cap \overline{q_n}'.$$

where $\overline{q_i}'$ is $W(\sigma_i)\overline{q_i}W(\sigma_i)^o$, σ_i being the permutation on $\{1, \ldots, m\}$ such that

$$\langle t_{\sigma_i(1)}, \ldots, t_{\sigma_i(n)} \rangle = \vec{t_i}\vec{t_1} \cdots \vec{t}_{i-1}\vec{t}_{i+1} \cdots \vec{t_m}$$

in other words, the permutation asociated with the i-th selector ρ_i, as in definition (4.2).

In the canonical semantics, the meaning of the translated query $[\![A_G]\!]^*$ is easily seen to coincide with the set of ground terms satisfying G in the least Herbrand model of $\mathcal{P}$. Evaluation of the query G is simulated by rewriting the relation expression A_G according to certain strategy outlined below.

We have a choice to make in the way we describe evaluation. We may explicitly model recursion at the object level by using the fix-point operator in our syntax, replacing all relation variables by their closed form solutions, or we may handle it metalogically by dynamically replacing relation variables by their definitions when they are evaluated in rewriting. We adopt the latter approach here.

Termination occurs when the resultant term has no free variables (or, when closed forms are used, when the term is *recursion-free*: without occurrences of **fp**). We now describe the rewriting system and the evaluation strategy used, and sketch a proof of its soundness and completeness. It should be remarked that certain reductions in the system below are best called meta-reductions, since they involve rules that carry out in one step some rather intricate rewriting of expressions. For example, we carry out the rewriting

$$\dot{K}(t_1, \ldots, t_n) \cap \dot{K}(s_1, \ldots, s_m) \quad \overset{P}{\longmapsto} \quad \dot{K}(\theta t_1, \ldots, \theta t_n) \tag{24}$$

$$\tag{25}$$

where

- the tuples $\vec{t} = (t_1, \ldots, t_n)$ and $\vec{s} = (s_1, \ldots, s_m)$ are standardized apart
- $m \leq n$ and
- θ is an mgu[4] of $(t_1, \ldots, t_m)$ and $(s_1, \ldots, s_m)$.

and

$$\dot{K}(t_1, \ldots, t_n) \cap \dot{K}(s_1, \ldots, s_m) \quad \overset{P}{\longmapsto} \quad \mathbf{0} \tag{26}$$

if no unifier of $(t_1, \ldots, t_m)$ and $(s_1, \ldots, s_m)$ exists.

These rules are sound with respect to $\mathsf{Rel}\Sigma$-interpretations (lemma 3.4).

Another meta-reduction is

$$W(\sigma)^o\dot{K}(t_1, \ldots, t_n)W(\sigma) \quad \overset{P}{\longmapsto} \quad \dot{K}(t_{\sigma(1)} \ldots, t_{\sigma(n)}) \tag{27}$$

$$I_k\dot{K}(t_1, \ldots, t_n)I_k \quad \overset{P}{\longmapsto} \quad \dot{K}(t_1, \ldots, t_k) \qquad \text{if } k \leq n. \tag{28}$$

[4]whose range is among the set of terms over the master list of variables $x_1, \cdots, x_m$

which is sound with respect to $\mathsf{Rel}\Sigma$-interpretations by lemmas 3.9, 3.10, 3.11 and 3.12.

Each meta-reduction *abbreviates a sequence of term rewritings*. That is to say, is equationally derivable in the theory $\mathsf{Rel}\Sigma$. The chief aim of the translation presented in this paper is to exploit the benefits of variable free rewriting in compilation, so it is essential that, in practice, these reductions be carried out within the relational syntax. The presentation of this rewriting system and a proof of its correctness is straightforward, but lengthy. It will not be discussed further in this sketch.

We also use some bona-fide rewriting rules, which are just directed equations from the theory of Distributive Relation Algebras, or **fp**-algebras, hence obviously sound for $\mathsf{Rel}\Sigma$- and **fp**-interpretations.

Table 1: Nondeterministic Logic Programming Reductions

$$\begin{array}{lll}
(A \cup B) \cap C & \overset{P}{\longmapsto} & (A \cap C) \cup (A \cap C) \\
A \cap (B \cap C) & \overset{P}{\longmapsto} & (A \cap B) \cap C \\
A \cap (Q \cup R) & \overset{P}{\longmapsto} & (A \cap Q) \cup (A \cap R) \\
A \cap \mathbf{fp}x.\mathcal{E}(x) & \overset{P}{\longmapsto} & A \cap \mathcal{E}(\mathbf{fp}x.\mathcal{E}(x)) \\
\overline{p}_i & \overset{P}{\longmapsto} & F_i(\overline{p}_1, \ldots, \overline{p}_n) \qquad (1 \leq i \leq n)
\end{array}$$

Table 2: Meta-reductions

$$\begin{array}{lll}
A \cap I_m Q I_m & \overset{P}{\longmapsto} & I_m[I_m A I_m \cap Q] I_m \cap A \\
A \cap W(\sigma)\, Q W(\sigma)^o & \overset{P}{\longmapsto} & W(\sigma)\,[W(\sigma)^o\, A W(\sigma)\, \cap Q] W(\sigma)^o \\
W(\sigma)^o \dot{K}(t_1, \ldots, t_n) W(\sigma) & \overset{P}{\longmapsto} & \dot{K}(t_{\sigma(1)} \ldots, t_{\sigma(n)}) \\
\dot{K}(u) \cap \dot{K}(v) & \overset{P}{\longmapsto} & \dot{K}(\theta v) \qquad (\theta = \mathsf{mgu}(u,v), \| u \| \leq \| v \|) \\
\dot{K}(v) \cap \dot{K}(u) & \overset{P}{\longmapsto} & \dot{K}(\theta v) \qquad (\theta = \mathsf{mgu}(u,v), \| u \| \leq \| v \|) \\
\dot{K}(v) \cap \dot{K}(u) & \overset{P}{\longmapsto} & \mathbf{0} \qquad (u, v \text{ not unifiable}) \\
I_s \dot{K}(t_1, \ldots, t_n) I_s & \overset{P}{\longmapsto} & \dot{K}(t_1, \ldots, t_s) \qquad (s \leq n)
\end{array}$$

6.1 The evaluation strategy

The typical query

$$\dot{K}(\vec{t_1} \ldots \vec{t_m}) \cap \overline{q_1}' \cap \ldots \cap \overline{q_m}'.$$

is evaluated as follows:

1. Associate to the left (second reduction):

$$(\dot{K}(\vec{t_1} \ldots \vec{t_m}) \cap \overline{q_1}') \cap \ldots \cap \overline{q_m}',$$

   where, since $\overline{q_1}'$ is $W(\sigma_1)\overline{q_1}W(\sigma_1)^o$,

2. apply the modular law in the form of the second meta-reduction:

$$W(\sigma_1)(W(\sigma_1)^o \dot{K}(\vec{t_1} \ldots \vec{t_m}) W(\sigma_1) \cap \overline{q_1}) W(\sigma_1)^o \cap \ldots \cap \overline{q_m}',$$

   apply the third meta-reduction (which permutes the indices of the sequence $\vec{t_1} \ldots \vec{t_m}$)

$$W(\sigma_1)(\dot{K}(\vec{t}_{\sigma_1(1)} \ldots \vec{t}_{\sigma_1(m)}) \cap \overline{q_1}) W(\sigma_1)^o \cap \ldots \cap \overline{q_m}',$$

3. replace $\overline{q_1}$ by its definition $\Theta_1 \cup \overline{\Theta}_2$ where $\overline{\Theta}_2$ is $\bigcup_{2 \leq i \leq r} \Theta_i$ for some r:

$$W(\sigma_1)(\dot{K}(\vec{t}_{\sigma_1(1)} \ldots \vec{t}_{\sigma_1(m)}) \cap [\Theta_1 \cup \overline{\Theta}_2]) W(\sigma_1)^o \cap \overline{q_2}' \ldots \cap \overline{q_m}'.$$

   and

4. Distribute the first $\cap$ accross the union:

$$W(\sigma_1)([(\dot{K}(\vec{t}_{\sigma_1(m)}\dots\vec{t}_{\sigma_1(m)})\cap\Theta_1)\cup(\dot{K}(\vec{t}_{\sigma_1(m)}\dots\vec{t}_{\sigma_1(m)})\cap\overline{\Theta}_2)])W(\sigma_1)^o\cap\overline{q_2}'\cap\dots\cap\overline{q_m}'. \quad (29)$$

Now we concentrate on the evaluation of $(\dot{K}(\vec{t}_{\sigma_1(m)}\dots\vec{t}_{\sigma_1(m)})\cap\Theta_1)$ which is an expression of the form (see (15)):

$$\dot{K}(\vec{t}_{\sigma_1(1)}\dots\vec{t}_{\sigma_1(m)})\cap I_s[\dot{K}(\vec{u})\cap W(\tau_1)\overline{q_{i1}}W(\tau_1)^o\cap\dots\cap W(\tau_\ell)\overline{q_{i\ell}}W(\tau_\ell)^o]I_s.$$

5. Now we apply the first meta-reduction:

$$I_s[I_s\dot{K}(\vec{t}_{\sigma_1(1)}\dots\vec{t}_{\sigma_1(m)})I_s\cap\dot{K}(\vec{u})\cap W(\tau_1)\overline{q_{i1}}W(\tau_1)^o\cap\dots\cap W(\tau_\ell)\overline{q_{i\ell}}W(\tau_\ell)^o]I_s\cap\dot{K}(\vec{t}).$$

Observing that $\vec{t}_{\sigma_1(1)}$ is precisely the truncation of the sequence $\vec{t}_{\sigma_1(1)}\dots\vec{t}_{\sigma_1(m)}$ to its first s components, and that $\vec{t}_{\sigma_1(1)}$ is precisely $\vec{t_1}$ (since the first selector permutation is in fact the identity) and applying the last meta-reduction:

$$I_s[\dot{K}(\vec{t_1})\cap\dot{K}(\vec{u})\cap W(\tau_1)\overline{q_{i1}}W(\tau_1)^o\cap\dots\cap W(\tau_\ell)\overline{q_{i\ell}}W(\tau_\ell)^o]I_s\cap\dot{K}(\vec{t}).$$

Letting $\vec{u'}$ be the result of applying $\mathsf{mgu}(\vec{u},\vec{t_1})$ to $\vec{u}$, we obtain

$$I_s[(\dot{K}(\vec{u'})\cap W(\tau_1)\overline{q_{i1}}W(\tau_1)^o)\cap\dots\cap W(\tau_\ell)\overline{q_{i\ell}}W(\tau_\ell)^o]I_s\cap\dot{K}(\vec{t}). \quad (30)$$

This brings us back to the evaluation of a basic form *term-sequence* $\cap$ *W(relation-variable)W^o*

$$\dot{K}(\vec{u''})\cap W(\tau_1)\overline{q_{i1}}W(\tau_1)^o$$

so we proceed as before, left-to-right.

As we are, in essence, reducing the entire SLD-tree, termination requires a finite SLD-tree. To simulate search along *one* branch, we would have to define *termination* to mean the reduction to a relation term *one of whose* unionands is relation-variable-free. This can be enforced with a different reduction strategy that we will not consider here. See [Ruhlen] for details.

6.1.1 Returning answers

The meta-reduction rules have the effect of reducing every composition and intersection of a K-term $\dot{K}(\vec{t})$ that can occur in evaluation *to another* K-term. Thus a resulting variable-free expression will be a union of such terms. These expressions, which constitute a normal form of the rewriting system, can be returned to the user, or alternatively, they can be *printed* as a normalized constraint-set description of the solution. This notion of printing has been described in [BroLip]. Although this extra strength is not exploited in the results of this paper, the printing algorithm reduces *any* variable-free relation expression using the full relation-algebra connectives (that is to say, including negation) to a two-quantifier formula in which negation occurs only immediately before atomic formulas (equations between terms) using the quantifier elimination algorithm of Mal'cev [Mal'cev], also described in [Maher]. It is therefore able to return readable answers to a considerably stronger relational rewriting system in which arbitrary first-order constraints over the Herbrand Universe are used as queries and outputs. The quantifier elimination process is the basis of Constructive Negation [Chan, Stuckey].

6.2 Soundness and Completeness of Evaluation

Soundness of evaluation is an immediate consequence of the fact that every rewrite rule $R \stackrel{P}{\longmapsto} R'$ is equationally sound: in any interpretation into $\mathcal{P}(\mathcal{T}_\Sigma^+\times\mathcal{T}_\Sigma^+)$

$$[\![R]\!]=[\![R']\!]$$

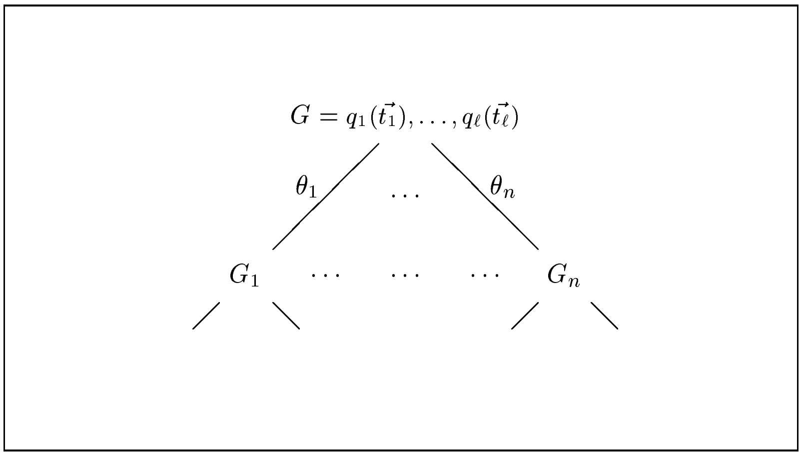
Suppose we are given a program $\mathcal{P}$ and a query G as in (23), with a finite SLD tree with leftmost selection rule. Since our reduction system is equationally sound, and the adequacy theorem shows that a pair of ground terms (u, u) is in $[\![A_G]\!]^*|_s$ precisely when the components of u make the query true in the least Herbrand Model of $\mathcal{P}$, the only way completeness can fail to hold is if evaluation of A_G fails to terminate. If it does terminate, i.e. A_G reduces to a closed term A, then we have the following:

1. $[\![A_G]\!]^* = [\![A]\!]^*$.
2. For s the number of terms in query G, $[\![A_G]\!]^*|_s = \{(u, u) : u \text{ ground and } \mathsf{M}_{\mathcal{P}} \models G(u)\}$ where $\mathsf{M}_{\mathcal{P}}$ is the least Herbrand model of $\mathcal{P}$.

Thus, in order to establish completeness it suffices to prove the following theorem.

Theorem 6.2 *Suppose $G = q_1(\vec{t_1}), \ldots, q_m(\vec{t_m})$ is a query for a logic program $\mathcal{P}$, and has a finite SLD tree with the leftmost selection rule. Let A_G be the relational query associated with G. Then there is a closed term A such that $A_G \stackrel{\mathcal{P}}{\longmapsto} A$.*

Proof: By induction on the depth of the given SLD tree. Suppose G has the following finite SLD tree:



where each G_i is of the form

$$B_i\theta_i, q_2(\vec{t_2}\theta_i), \ldots, q_m(\vec{t_m}\theta_i).$$

and where B_i is the tail of the i-th clause in the program $\mathcal{P}$ whose head predicate is p_1:

$$p_1(\vec{u_{i0}}) : -q_{i1}(\vec{u_{i1}}), \ldots, q_{i\ell}(\vec{u_{i\ell}}).$$

Then G_1 is precisely the goal:

$$(q_{11}(\vec{u_{11}}), \ldots, q_{1\ell}(\vec{u_{1\ell}}))\theta_1, q_2(\vec{t_2}\theta_1), \ldots, q_\ell(\vec{t_\ell}\theta_1),$$

where θ_i is a most general unifier of $\vec{u_{10}}$ and $\vec{t_1}$.

The SLD tree below this goal is finite (and of smaller depth than the original goal), hence so is the SLD tree (with left-most selection rule) below the first subgoal $G_{11} = q_{11}(\vec{u_{11}}\theta_1)$.

Let $A_{G_{11}}$ be the relational translation of this goal, namely

$$\dot{K}(\vec{u_{11}}\theta_1) \cap q_{11}$$

We may apply the induction hypothesis to conclude that our evaluation strategy will reduce $A_{G_{11}}$ to a relation-variable-free A_{11}, which, in fact, must be the K-term corresponding to the computed answer substitution for this goal.

Recall (from 29) that evaluation of the original query A_G yields, in several steps, an expression of the form

$$W(\sigma_1)([(\dot{K}(\vec{t}_{\sigma_1(m)}\ldots\vec{t}_{\sigma_1(m)})\cap\Theta_1)\cup(\dot{K}(\vec{t}_{\sigma_1(m)}\ldots\vec{t}_{\sigma_1(m)})\cap\overline{\Theta}_2)])W(\sigma_1)^o\cap\overline{q_2}'\cap\ldots\cap\overline{q_m}'.$$

the leftmost reducible subexpression of which reduces to (30)

$$I_s[(\dot{K}(\vec{u_{11}}\theta_1)\cap W(\tau_1)\overline{q_{11}}W(\tau_1)^o)\cap W(\tau_2)\overline{q_{12}}W(\tau_2)^o\cap\cdots\cap W(\tau_\ell)\overline{q_{1\ell}}W(\tau_\ell)^o]I_s\cap\dot{K}(\vec{t}).$$

Since the first selector $W(\tau_1)$ is the identity, the leftmost reducible subterm $\dot{K}(\vec{u_{11}}\theta_1)\cap W(\tau_1)\overline{q_{11}}W(\tau_1)^o$ is precisely $A_{G_{11}}$ which reduces to A_{11}. We are left with (using primes to hide the switching relations W):

$$W(\sigma_1)(I_s[A_{11}\cap\overline{q_{12}}'\cap\cdots\cap\overline{q_{1\ell}}']I_s\cap\dot{K}(\vec{t})\cup(\dot{K}(\vec{t}_{\sigma_1(m)}\ldots\vec{t}_{\sigma_1(m)})\cap\overline{\Theta}_2))W(\sigma_1)^o\cap\overline{q_2}'\cap\ldots\cap\overline{q_m}'.$$

The stack of goals (relation-variables) present is bounded by the size of the original SLD tree, and for each one we argue in the same way, eventually producing a union of relation-variable free goals

$$A = A_1'\cup\ldots\cup A_\nu'$$

where ν is the number of success nodes on the fringe of the original SLD tree. ▌

7 Related and Future Work, and Conclusions

There have been other efforts in the literature to remove variables from Prolog, and to make evaluation and compilation more algebraic. Bellia and Occhiuto develop an algebra of programs that captures unification, rewriting and narrowing in [BelOcc]. Our work rests heavily on the fact that such an algebra can be found within the relation calculus, whose semantics is well-understood, and which admits natural extensions to higher-order and linear contexts, as well as a rich representation theory (see e.g. the treatment of Allegories in [FreySce]). The compilation process described here can be viewed as an application of the canonical inclusion of a regular category into its associated category of relations. The greater generality of the categorical framework opens the way to applying this technique to extensions of the logic and to constraints, as well as to a denotational treatment based on relations, which we are currently exploring.

Categorical treatments of logic programming provide alternative ways of algebraicizing the subject, which are, in a sense, variable free (see e.g. [AspMart, Diac, PowKin, FFL, NFDP, Pym]). Corradini and Montanari [CorMont] have given a categorical analysis of logic program execution in terms of transition systems. None of these approaches have as yet been applied to compilation or the definition of an abstract machine for logic programs, although this might be an interesting alternative to the work in this paper.

At this point there are a number of questions raised by this approach which we hope to address. Can a significant portion of our relational machine be captured with a Church-Rosser, strongly normalizing set of rewrite rules? Comon and Jouannaud and Kirchner's work [Comon, CoHaJo] on rewriting systems for unification and disunification suggests that this is quite feasible, as does the work of Bellia and Occhiuto, *op. cit.*

We would also like to exploit the rich semantics of relational formalisms to obtain new notions of observables, and abstract interpretation, as well as to extend the relational compilation to higher-order logic programming.

To some extent this work was a foray into the terrain of relation-based computing as a separate discipline, with logic programming as an extended case-study . Some instances of "pure" relational programming languages, of limited expressive power, were studied in [BroLip]. A useful rewriting system for the full relation calculus seemed a lot to ask for in the absence of some computational paradigm and we thought it would help, at the start, to anchor such a system in logic programming-inspired reductions.

Work by the RUBY group at Oxford on hardware [BroHut, JonShe], and on program synthesis via relations by Bird and de Moor[Algebra], Maddux[Maddux, RelSem], Naumann [Naumann] and Backhouse [Backh], to name a few of the many researchers in this field, suggests that the relational paradigm can provide significant computational insights at almost every level of the field. Exploration –via relations– of possible connections between program synthesis and a more general notion logic of programming seems particularly tempting.

References

[AspMart] A. Asperti and S. Martini. Projections instead of variables, a category theoretic interpretation of logic programs. In *Proceedings of the 6^{th} International Conference on Logic Programming*, pages 337–352. MIT Press, 1989.

[WAM] Hassan Ait-Kaci, *Warren's Abstract Machine: A Tutorial Reconstruction*, MIT Press, Series in Logic Programming. 1991.

[Backh] Backhouse, R, et. al., "A Relational Theory of Data Types," in *Workshop on Constructive Algorithmics: The Role of Relations in Program Development,* Utrecht University, 1990.

[Backus] Backus, J., "Can Programming be liberated from the von Neumann style?" in *ACM Turing Award Lectures*, ACM Press (Addison-Wesley), New York, 1987.

[BakRoe] de Bakker, J. and de Roever, "A Calculus for Recursive Program Schemes" , ed. Nivat, A., in *Automata, Languages and Programming*, 1973.

[BelOcc] Bellia, M., and Occhiuto, M. E., "C-expressions, a variable-free calculus for equational logic programming", in *Theoretical Computer Science* 107, Elsevier, 1993.

[Naumann] David A. Naumann "A recursion theorem for predicate transformers on inductive data types", *Information Processing Letters*, volume 50, 6, pp. 329–336, 1994.

[BirdMoor1] R S Bird, Oege de Moor and P Hogendijk. Generic programming with types and relations. *Journal of Functional Programming*, 6(1), 1996.

[Algebra] Bird, R. S. and De Moor, O. *Algebra of Programming.* Prentice Hall, 1996

[BroHut] Brown, C. and Hutton, G., "categories, Allegories and Circuit Design", in *Proceedings of the 9th Symposium on Logic in Computer Science*, IEEE, 1994.

[Broome86] Broome, P. *Transformations of Parallel Programs with Higher-order Relational Operators*, Ph. D. dissertation, Univ. of Delaware, 1986.

[Broome91] Broome, P. "Applications of Algebraic Logic to Recursive Query Optimization," *8th Army Conference on Applied Mathematics and Computing,* 1991.

[BrLi92] Broome, P., and Lipton, J., "Constructive Relational Programming", *Transactions of the 9th Army Conference on Applied Mathematics and Computing,* ARO-Report 92-1, 1992.

[BroJon] Brown, Carolyn and Jones, G., "Hardware Component Allegories", in Proc. LICS 1994, IEEE.

[Chap] *Compilation of Logic Programs to a Relational Machine*, Honors Thesis, Wesleyan University, 1997.

[Chan] Chan, D. *Constructive Negation Based On the Completed Database,* Logic Programming: Proceedings of the Fifth International Conference and Symposium Eds. R. A. Kowalski and K. A. Bowen, MIT Press, Cambridge, MA, pp 111–125, 1988.

[Clark] Clark, K., "Negation as Failure", in *Logic and Data Bases*, Gallaire and Minker eds., Plenum Press, New York, 1978.

[CorMont] Corradini, A. and Montanari, U. An algebraic semantics for structured transition systems and its application to logic programs. In *Theoretical Computer Science* 103. Elsevier, 1993.

[Comon] Comon, H. "Disunification, A Survey", in *Logic and Automation*, Lassez and Plotkin, eds., MIT, 1992.

[CoHaJo] Comon, H., Haberstrau, M., Jouannaud, J-P, "Decidable problems in shallow equational theories", in *Proc. 7th. Symp. of LICS*, IEEE Computer Society Press, 1992.

[clp94] Joxan Jaffar and Michael Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20, 1994.

[BroLip] Broome, P and Lipton, J., "Combinatory Logic Programming: Computing in Relation Calculi" in *Proceedings of the 1994 International Symposium on Logic Programming*, M. Bruynooghe, ed., M.I.T. Press, pp. 269-285, 1994.

[Colp] Lipton, J., "A Relational Foundation for Logic Programming", to appear.

[Diac] R. Diaconescu. *Category Semantics for Equational and Constraint Logic Programming.* PhD thesis, Oxford University, 1994.

[FriasMad] "Completeness of a Relational Calculus for Program Schemes", proceedings of LICS 98, IEEE, 1998.

[FFL] Finkelstein, S., Freyd, P. and Lipton, J., "Logic Programming in Tau-Categories", in *Computer Science Logic '94*, LNCS 933, Springer, pp. 249-263, 1995.

[Freyd92] e-mail manuscript, Categories, Relations and Computation bulletin board, Feb. 15, 1992.

[FreySce] Freyd,P. and Scedrov, A., *Categories,Allegories,* North-Holland, 1990.

[NFDP] Finkelstein, S., Freyd, P. and Lipton, J., "A New Framework for Declarative Programming", to appear in *Theoretical Computer Science*, Elsevier.

[LakRed] T. K. Lakshman and Uday S. Reddy. Typed Prolog: A semantic reconstruction of the Mycroft-O'Keefe type system. In *Proceedings of the 1991* IEEE Symposium on Logic Programming, pages 202–220, 1991.

[JafLas] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of Symposium on Principles of Programming Languages.* ACM, 1987.

[JonShe] Jones, G. and Sheeran, M., "Circuit Design in RUBY". In *Formal Methods in VLSI Design*, ed. J. Staunstrup, North-Holland, 1990.

[Lloyd] Lloyd, J. W., *Foundations of Logic Programming*, second ed., Springer-Verlag, 1987.

[Maddux] Maddux, R., "Introductory course on relation algebras, finite-dimensional cylindric algebras, and their interconnections", in *Algebraic Logic* (Proc. Conf. Budapest 1988) ed. by H. Andreka, J. D. Monk, and I. Nemeti, Colloq. Math. Soc. J. Bolyai 54 North-Holland Amsterdam, 1991, 361–392.

[RelSem] Maddux R., Relation-Algebraic Semantics, in *Theoretical Computer Science*, Volume 160, June 1996.

[Maher] Maher, M., "A Complete Axiomatization of the Theories of Finite, Rational and Infinite Trees" *Proceedings of the third IEEE Symposium on Logic and Computer Science,* 1988.

[MdM96] R. McPhee and O. de Moor, "Compositional Logic Programming", in *Proceedings of the JICSLP'96 post-conference workshop: Multi-paradigm logic programming*, M. Chakravarty and Y. Guo and T. Ida, eds., Technische Universität Berlin, also PRG Report 96-28, Oxford, 1996.

[Mal'cev] Mal'cev, A.I.,"On The Elementary Theories of Locally Free Universal Algebras", *Soviet Math. Doklady*, 1961, pp.768-771.

[Miller] Miller, M., Nadathur, G., Pfenning, F. and Scedrov, A., "Uniform Proofs as a Foundation for Logic Programming." *Annals of Pure and Applied Logic,* 1990.

[Moor92] de Moor, Oege, *Categories, Relations and Dynamic Programming*, Dissertation, and Technical Monograph PRG-98, Oxford Computing Lab, Oxford, 1992.

[MycOk] A. Mycroft and R. A. O'Keefe. A Polymorphic Type System for Prolog. *Artificial Intelligence*, 23(3):295–307, 1984.

[Peirce] Pierce, C.S., *The Logic of Relatives*, Studies in Logic. Johns Hopkins University, Baltimore, 1882, 235-277.

[PowKin] Power, J. and Kinoshita, Y., A new foundation for logic programming. In *Extensions of Logic Programming '96.* Springer Verlag, 1996.

[Pym] Pym, D., Functorial Kripke models of the $\lambda\pi$-calculus. Lecture at Newton Institute Semantics Programme, Workshop on Category Theory and Logic Programming, Cambridge, September 1995.

[deRoe] de Roever, W.P., "Recursion and Parameter Mechanisms: an Axiomatic Approach", Report IW 20/74, Matematisch Centrum, Amsterdam, 1974.

[Ruhlen] *Development of a Rewrite System for Relational Algebra Equations Modeling Horn Clause Resolution* Master's Thesis, Wesleyan University, 1997.

[Schmidt] Schmidt, G. and Ströhlein, T., *Relations and Graphs*, EATCS Monographs, Springer, 1993.

[Shep88] Sheperdson, J.C., Negation in Logic Programming, in *Deductive Databases and Logic Programming*, Jack Minker, Rd., Morgan Kaufman, Los Altos, CA, 1988.

[Stuckey] Stuckey, P. "Constructive Negation for Constraint Logic Programming" in Proceedings of the 6th IEEE Symposium on Logic and Computer Science, New York, pp.328-339, 1991.

[TarGiv] Tarski, A. and Givant, S., *A formalization of set theory without variables,* Colloquium publications, V. 41, American Mathematical Society, Providence, RI, 1987.