Combinatory Logic Programming: Computing in Relation Calculi *

Paul Broome
1620 Creston Dr.
Forest Hill, Md.21050
`broome@msiadmin.cit.cornell.edu`

James Lipton
Dept. of Mathematics
Wesleyan University
Middletown, CT 06459
`lipton@allegory.cs.wesleyan.edu`

Abstract

We define a combinatory relational calculus based on a variant of the *Relation Algebra* formalism (developed by Tarski-Givant, and in more general form by Freyd and others) which extends traditional database query formalisms in a number of ways. It is a variable-free calculus for rewriting relation expressions built up from ground relations using union, intersection, complementation, composition and a fixed point operator. The language extends conventional Horn-clause programming with constraints (e.g. constructive negation) by permitting queries and outputs to be relation expressions. We also define an associated relational programming language `RAlog` which combines rewriting in the calculus with a quantifier-elimination algorithm due to Mal'cev. The algorithm is used to translate reduced relation expressions in the calculus into two-quantifier formulas, which constitute the printed output of the language.

The relational calculi and programming language discussed in this paper (`RAlog`) arose from an attempt to abstract the *second order* or pure relational content of logic programs as objects that bind relation variables and formally define relations somewhat the way lambda terms formally define functions. There are several benefits to be gained from such an attempt: to combine the declarative transparency and reversibility of logic programming with some of the features of functional programming languages, and to open the way for a new denotational treatment of the semantics of declarative programming. In this paper we use the relation algebra formalism as an executable algebra of programs, somewhat in the Backus tradition[Backus], to "liberate logic programming from the von Neumann style." Another objective has been to make constraint manipulation and some features of control more transparent and declarative by finding a unified calculus which express constraints, program transformation, evaluation and the logic program itself.

1 Combinatory Logic Programming

It will prove helpful to start by considering an example of the process of abstracting the relational content of a simple logic program. We illustrate with the Prolog program for the reflexive transitive closure of a simple graph, and two queries.

*An abbreviated version of the paper appeared in the proceedings of the 1994 International Logic Programmming Symposium, M. Bruynooghe, ed., M.I.T. Press.

```
conn(X,X).                              edge(a,b).
conn(X,Y):-edge(X,Z),conn(Z,Y).         edge(b,c).
                                        edge(a,l).
                                        edge(l,c).

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

Translating the program: We introduce the binary relation symbols *conn* and *edge* (and let the font difference suffice to distinguish between program predicates and the new symbols), and translate the program into a pair of relation equations. *conn* will be a variable standing for a relation defined via an equation. *edge* will stand for an expression which explicitly describes a finite binary relation on the Herbrand universe $\mathcal{H}$ of the program. For readability, we denote composition of relations 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. We rewrite the last equation as $conn = \boldsymbol{fp}x.(id \cup edge; x)$ where the operator $\boldsymbol{fp}$ means "least fixed point of". The first query above is represented by $\{(a,c)\} \cap conn$ and the second by $(\mathbf{1}; \{(c,c)\}) \cap conn$ where $\mathbf{1}$ is the universal relation $\mathcal{H} \times \mathcal{H}$, (hence $(\mathbf{1}; \{(c,c)\})$ represents the set of all pairs whose second component is c). A solution to the first query would have be to be of the form $\{(a,c)\}$, confirming that $(a,c) \in conn$, or $\mathbf{0}$, denoting the empty relation, if not. A solution to the second query should be some more explicit description, such as $\{(a,c),(b,c),(c,c),(l,c)\}$. We proceed to evaluate the second query informally, to motivate the calculus we will present below. A natural first strategy is to unfold the recursive definition of $conn$:

$$\mathbf{1}; \{(c,c)\} \cap conn = (\mathbf{1}; \{(c,c)\} \cap id) \cup (\mathbf{1}; \{(c,c)\} \cap edge; conn)$$

The first term can be reduced to $\{c,c\}$, and the second can be further unfolded.

$$\begin{aligned} \mathbf{1}; \{(c,c)\} \cap edge; conn &= \mathbf{1}; \{(c,c)\} \cap edge; (id \cup edge; conn) \\ &= \mathbf{1}; \{(c,c)\} \cap (edge; id \cup edge; edge; conn) \\ &= \mathbf{1}; \{(c,c)\} \cap (edge \cup edge; edge; conn) \\ &= (\mathbf{1}; \{(c,c)\} \cap edge) \cup (\mathbf{1}; \{(c,c)\} \cap edge; edge; conn) \\ &= \{(b,c),(l,c)\} \cup (\mathbf{1}; \{(c,c)\} \cap edge; edge; conn) \end{aligned}$$

Continuing this way, we will eventually obtain the term $\mathbf{1}; \{(c,c)\} \cap (edge; edge)$ which is equal to $\{(a,c)\}$. The next unfolding gives us $\mathbf{1}; \{(c,c)\} \cap (edge)^{(3)}; conn$. $edge$ is finite: in fact for any $X > 2$, $(edge)^{(X)} = \mathbf{0}$ hence we are done. The query evaluates to $\{(c,c)\} \cup \{(b,c),(l,c)\} \cup \{(a,c)\}$ which is simply $\{(c,c),(b,c),(l,c),(a,c)\}$.

However what happens if we have a relation R defining an infinite acyclic graph and we pick a pair $\{(u,v)\}$ not in R? We run the risk of expanding $\{(u,v)\} \cap R$ forever despite the fact that the chosen pair is not a member. Clearly we need to use other facts about relations if we are to proceed at least as efficiently as Prolog. This means being able to compute $\mathbf{0}$ whenever Prolog would compute "no". By using the so-called *modular law* in one of its many equivalent forms

$$A \cap R; S = A \cap R; (R^o; A \cap S) \ \text{ or } \ A \cap R; S = A \cap (A; S^o \cap R); S$$

and computing the expression $R^o; A$ or $A; S^o$ before proceeding we are able to show (via equational reasoning) that the right hand term vanishes, and the computation terminates. This is illustrated by the following example.

```
le(X,X).
le(X,Y):-pred(X,Z),le(Z,Y).             pred(s(X),X).

 | ?- le(s(a),s(b)).
```

We translate the program by $id \cup pred; le$ and the query by $(s(a), s(b)) \cap le$. First, unfold le, distribute its definition, note that $\{(s(a), s(b))\} \cap id$ is $\mathbf{0}$, and apply the modular law.

$$
\begin{aligned}
&\{(s(a), s(b))\} \cap le \\
&= \{(s(a), s(b))\} \cap (id \cup pred; le) \\
&= \{(s(a), s(b))\} \cap pred; le) \\
&= \{(s(a), s(b))\} \cap pred; (pred^{o}; \{(s(a), s(b))\} \cap le) \text{ by the modular law} \\
&= \{(s(a), s(b))\} \cap pred; (\{(a, s(b))\} \cap le) \\
&= \{(s(a), s(b))\} \cap pred; (\{(a, s(b))\} \cap id) \cup \{(a, s(b))\} \cap pred; le))
\end{aligned}
$$

Since the pair $(a, s(b))$ is not in id then the expression simplifies to

$$
\begin{aligned}
&= \{(s(a), s(b))\} \cap pred; (\mathbf{0} \cup \{(a, s(b))\} \cap pred; le)) \\
&= \{(s(a), s(b))\} \cap pred; (\{(a, s(b))\} \cap pred; (pred^{o}; \{(a, s(b))\} \cap le)) \\
&= \{(s(a), s(b))\} \cap pred; (\{(a, s(b))\} \cap pred; (\mathbf{0} \cap le)) \\
&= \mathbf{0}
\end{aligned}
$$

Since $pred^{o}; \{(a, s(b))\} = \mathbf{0}$ then the entire expression is the empty relation.

These examples motivate the goals of this paper, but serious questions need to be resolved. How carefully handpicked are they? Is there a general calculus and reduction strategy that will always work at least as well as a Prolog computation? Can this help us extend the notion of declarative programming to a broader context?

The next two sections of this paper are devoted to making precise the relational syntax, program translation and notions of reduction motivated by this example. A key aim of this paper is to delineate the main components of a new relational programming paradigm. Computation in this formulation means *relation calculus rewriting* of queries (relation expressions) to members of some distinguished class of relation expressions (usually recursion-free terms), followed by *representation* of such expressions in a human readable form in a way that depends on the data domain, a process here called *printing*.

$$
\text{query} \longrightarrow \boxed{\begin{matrix}\text{Rewriting in a}\\ \text{relational theory}\end{matrix}} \xrightarrow{\text{reduced form}} \boxed{\text{Printing}} \longrightarrow \text{solution}
$$

In the main example studied in this paper, data means terms in a Herbrand Universe, and printing will involve a quantifier elimination algorithm that makes use of special properties of the Herbrand Universe. We will also define a general translation that maps Prolog programs and queries into a variable-free relation expression, and maps Prolog evaluation with a given interpreter into a particular reduction strategy, which we call *embedded* Prolog. We conclude by showing how the relational calculus extends the query and output language of Prolog and incorporates a robust notion of constraint programming in a natural way. Finally, we discuss new approaches to logic programming semantics opened up by the translation to a relational formalism.

2 The Relational Calculus

We first define the syntax of the equational theory we will be using.

Definition 2.1 *Assume given a set Rvar of relation variables. $\Re$, the set of* ***pure relational expressions,*** *is the collection of terms R generated by the following BNF grammar:*

$$
\begin{aligned}
R \quad ::= \quad & \langle Rvar \rangle \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl \mid R^{o} \\
& R \cup R \mid R \cap R \mid \overline{R} \mid RR \mid \mathbf{fp}\langle var \rangle.(R)
\end{aligned}
$$

Let $\mathcal{L}$ be a finite language comprised of a set of constant symbols $\{a_1, a_2, \ldots, b_1, b_2, \ldots\}$ and a set of function symbols each with a given arity. Then $\Re_{\mathcal{L}}$, the set of $\mathcal{L}$-***term relation expressions*** *is the collection of terms generated by the grammar for $\Re$ and the additional rules*

$$
R \quad ::= \quad \{(t_1, t_2)\} \mid f_i^n
$$

where the t_i are terms over $\mathcal{L}$, f is a function symbol of arity n and $1 \le i \le n$.

Definition 2.2 *A relation expression is* ***atomic*** *if it is one of* $\{f_i^n, hd, tl, id, di, \mathbf{1}, \mathbf{0}\}$*, and* ***finitary*** *if it contains no occurrences of* **fp**. *A relation expression is* ***positive*** *if it contains no occurrences of complementation,* ***strictly positive*** *if di does not occur, and* ***simple*** *if no complementation symbol occurs within the scope of an* **fp-** *operator.*

Juxtaposition indicates relation composition as does ";" when needed for clarity. The converse of R is R^o. A pair of constant terms is denoted by $\{(t_1, t_2)\}$. The projection symbol f_i^n is intended to denote the relation between terms $f(u_1, \ldots, u_n)$ with n arguments and the ith argument u_i.

2.1 Extensional Semantics and Correctness

$\Re$ and $\Re_{\mathcal{L}}$ become computation calculi when equipped with a rewrite relation $\overset{e}{\longmapsto}$ on pairs of terms. We will use $\overset{e}{\underset{*}{\longmapsto}}$ to stand for its reflexive transitive closure.

We begin by defining a standard semantics along the lines of the fixed point semantics for Prolog, here called *extensional* because it characterizes the meaning of a relation expression thought of as a logic program by the set of successful queries, i.e. the operational extension of the program predicates.

We will need a standard pairing operator, in order to reduce relations of any arity to binary ones. Any binary function symbol will do, so we introduce the infix $[\_ \mid \_]$ as a "reserved" binary function symbol[1] We assume that $[\_ \mid \_] \in \mathcal{L}$.

Let $\mathcal{U}$ be an $\mathcal{L}$-algebra i.e. a structure for the language $\mathcal{L}$, and $\eta : Rvar \to \mathcal{U}$ a $\mathcal{U}$*-environment.* We define the interpretation

$$[\![\,]\!]_\eta^{\mathcal{U}} : \Re \to \mathcal{R}_{\mathcal{L}}$$

relative to environment η to be the standard association of sets with relation expressions, e.g.

$$\begin{array}{lcllcl}
[\![hd]\!]_\eta^{\mathcal{U}} & \overset{def}{\equiv} & \{([t_1|t_2], t_1) : t_1, t_2 \in \mathcal{U}\} & [\![R \cup S]\!]_\eta^{\mathcal{U}} & \overset{def}{\equiv} & [\![R]\!]_\eta^{\mathcal{U}} \cup [\![S]\!]_\eta^{\mathcal{U}} \\
[\![\mathbf{1}]\!]_\eta^{\mathcal{U}} & \overset{def}{\equiv} & \mathcal{U} \times \mathcal{U} & [\![R \cap S]\!]_\eta^{\mathcal{U}} & \overset{def}{\equiv} & [\![R]\!]_\eta^{\mathcal{U}} \cap [\![S]\!]_\eta^{\mathcal{U}}
\end{array}$$

$$[\![f_i^n]\!]_\eta^{\mathcal{U}} \overset{def}{\equiv} \{(\boldsymbol{x}, \boldsymbol{y}) : (\exists v_1 \cdots v_{n-1})(\boldsymbol{x} = f(v_1, \ldots, v_{i-1}, \boldsymbol{y}, v_i, \ldots, v_{n-1}))$$

$$[\![\mathbf{fp}x.\mathcal{E}\,(x)]\!]_\eta^{\mathcal{U}} \overset{def}{\equiv} \bigcup_{n \geq 0} [\![\mathcal{E}^{\,(n)}(x)]\!]_\eta^{\mathcal{U}}$$

where $\mathcal{E}^{\,(0)}(x) = \mathbf{0}$ and $\mathcal{E}^{\,(n+1)}(x) = \mathcal{E}\,(\mathcal{E}^{\,(n)}(x))$.

For most of this paper we fix a finite language $\mathcal{L}$, and take the universe $\mathcal{U}$ above to be $\mathcal{H}_{\mathcal{L}}$ the *Herbrand universe* or free term algebra made up of all variable-free terms over $\mathcal{L}$. Letting $\mathcal{R}_{\mathcal{L}}$ be the Boolean algebra of binary relations on $\mathcal{H}_{\mathcal{L}}$, and omitting reference to the environment η *we define the* ***standard*** *interpretation* $[\![\,]\!] : \Re \to \mathcal{R}_{\mathcal{L}}$ *to be* $[\![\,]\!]_\eta^{\mathcal{H}_{\mathcal{L}}}$

Definition 2.3 *A relation R on a set S is* ***functional*** *if $R^o R \subseteq id$,* ***reflexive*** *if $id \subseteq R$, and* ***coreflexive*** *if $R \subseteq id$.*

We also define analogous notions for syntactic relation *expressions.*

Definition 2.4 *We will call a relation expression* ***functional*** *if it is built up by compositions or intersections from the functional atomic relations $hd, tl, f_i^n, id, (u, u)$. We call an expression R* ***coreflexive*** *if it is a union or intersection of expressions of the form $A \cap id, (u, u)$.*

Coreflexive relations can be thought of as encoding *data* in a set-theoretic interpretation, e.g. $\mathbb{N}$ can be represented as $\{(n, n) : n \in \mathbb{N}\}$ and a value 3 as the coreflexive $\{(3, 3)\}$. Note that we can easily represent the domain and range of any relation R as coreflexives: $Dom(R) = RR^o \cap id, Ran(R) = R^o R \cap id$.

Definition 2.5 *Let R be a relation expression, and $\overset{e}{\mapsto}$ a rewriting system. Then we define the e-* ***relational success set*** $\boldsymbol{RSS}(e, R)$ *to be*

$$\boldsymbol{RSS}(e, R) = \{(t_1, t_2) : (t_1, t_2) \cap R \overset{e}{\underset{*}{\longmapsto}} (t_1, t_2)\}$$

where the right hand reduction means: there is an e-reduction sequence ending with (t_1, t_2). If $\mathcal{U}$ is an $\mathcal{L}$-algebra then $\boldsymbol{RSS}(e, R)_{\mathcal{U}}$ is the set of interpretations in $\mathcal{U}$ of the pairs of terms (t_1, t_2) such that $(t_1, t_2) \cap R \overset{e}{\underset{*}{\longmapsto}} (t_1, t_2)$.

[1] If we write this binary function symbol as a prefix pr, then hd and tl are just pr_1^2 and pr_2^2.

Definition 2.6 *Let $\mathcal{L}$ be a language, $\stackrel{e}{\longmapsto}$ a rewriting system and $\hat{\Re} \subseteq \Re_{\mathcal{L}}$. Then we say e is* ***extensionally complete*** *with respect to $\hat{\Re}$ (on $\mathcal{U}$) if for every simple relation expression R in $\hat{\Re}$,*

$$\boldsymbol{RSS}(e, R)_{\mathcal{U}} \quad = \quad [\![R]\!]_{\mathcal{U}}$$

in other words for every pair of terms (t_1, t_2) there is a reduction sequence from $(t_1, t_2) \cap R$ to (t_1, t_2) if and only if the interpretation in $\mathcal{U}$ of (t_1, t_2) is in $[\![R]\!]_{\mathcal{U}}$. Let $\equiv_e$ be an equational theory. Then $\equiv_e$ is ***extensionally complete*** *with respect to $\hat{\Re}$ (on $\mathcal{U}$) if for every simple relation expression R in $\hat{\Re}$, and terms t_1 and t_2,*

$$(t_1, t_2) \cap R \ \equiv_e \ (t_1, t_2)$$

if and only if the interpretation in $\mathcal{U}$ of (t_1, t_2) is in $[\![R]\!]_{\mathcal{U}}$.

Without restricting the class of relational terms, we cannot in general expect extensional completeness, since it implies that all definable relations on a universe $\mathcal{U}$ are recursively enumerable. We now show extensional completeness for $\stackrel{r}{\longmapsto} \cup \stackrel{t}{\longmapsto}$, the system in Table 1 and two simple restricted classes of relations.

Theorem 2.7 *Let $\hat{\Re}$ be the set of relation expressions built up from atomic relations and compositions of pairs of strictly positive relations, using only $\cup, \cap, (\_)^o$. Then $\stackrel{r}{\longmapsto} \cup \stackrel{t}{\longmapsto}$ is extensionally complete with respect to $\hat{\Re}$ on the Herbrand universe.*

Proof: The soundness direction is immediate, every reduction rule above is a directed equation that is clearly valid in the $[\![\cdot]\!]$ interpretation. That is to say if $X \mapsto Y$ then $[\![X]\!] = [\![Y]\!]$. We prove completeness by induction on the structure of R. The atomic cases $(R = id, di, hd, tl, (f_i^n), \mathbf{1}, \mathbf{0}, \{(u, v)\})$ are easy.

<u>R is $S \cup T$</u>: Then $(t_1, t_2) \in [\![R]\!]$ implies $(t_1, t_2) \in [\![S]\!]$ or $(t_1, t_2) \in [\![T]\!]$. So, in either case, $(t_1, t_2) \cap R \mapsto (t_1, t_2) \cap (S \cup T) \mapsto ((t_1, t_2) \cap S) \cup ((t_1, t_2) \cap T)$ which reduces, by induction hypothesis (and two more steps) to (t_1, t_2).

<u>R is $S \cap T$</u>: Then $(t_1, t_2) \in [\![R]\!]$ implies $(t_1, t_2) \in [\![S]\!]$ and $(t_1, t_2) \in [\![T]\!]$. So, $(t_1, t_2) \cap R \mapsto (t_1, t_2) \cap (S \cap T) \mapsto ((t_1, t_2) \cap S) \cap T$ which reduces, by induction hypothesis to $(t_1, t_2) \cap T$ and then to (t_1, t_2). The cases R of the form $(S)^o$or ST are straightforward and left to the reader. ∎

Definition 2.8 *A relation expression R is* ***formative*** *if it is the intersection of expressions built up from singletons (u, u) and id and by zero or more iterations of the operation*

$$X \mapsto SXT^o$$

where S, T are one of the f_j^n or hd or tl.

Lemma 2.9 *The rewrite system $\stackrel{r}{\longmapsto} \cup \stackrel{t}{\longmapsto}$ is extensionally complete with respect to formative expressions on the Herbrand universe $\mathcal{H}_{\mathcal{L}}$.*

Proof: First observe that $(u, v) \cap SXT^o \ \mapsto (u, u)SXT^o(v, v)$. Since each term S, T in a formative expression is functional it is easy to check that $(u, u)S$ reduces to some term (u, v) or to $\mathbf{0}$. Since X is either of the form $S'X'(T')^o$ or (w, w), the original term will reduce to a composition of singletons, and hence to a singleton or $\mathbf{0}$. By soundness of the reduction rules, if $(u, v) \in [\![SXT^o]\!]$ it reduces to (u, v). ∎

We will need one more formulation of completeness.

Definition 2.10 *A rewriting system e is* ***relation-complete*** *with respect to a pair of sets of relation expressions $\mathfrak{Q} \subseteq \Re_{\mathcal{L}}$ (for queries) and $\mathfrak{P} \subseteq \Re_{\mathcal{L}}$ (for programs) if for every expression $A \in \mathfrak{Q}$ and every $R \in \mathfrak{P}$ there is a* ***finitary*** *A' with $[\![A']\!] = [\![A \cap R]\!]$ such that $A \cap R \stackrel{e}{\longmapsto}_* A'$ An equational theory $\equiv_e$ is* ***relation-complete*** *with respect to $\mathfrak{Q}, \mathfrak{P}$ if for any $A \in \mathfrak{Q}$and $R \in \mathfrak{P}$ there is a finitary A' such that $A \cap R \equiv_e A'$.*

Table 1: Relational rewrite rules

Pure relation reduction rules

$\mathbf{fp}x.\mathcal{E}\,(x)$	$\xmapsto{r}$	$\mathcal{E}\,(\mathbf{fp}x.\mathcal{E}\,(x))$	$A \cap (B \cap C)$	$\xmapsto{r}$	$(A \cap B) \cap C$
$A \cap BC$	$\xmapsto{r}$	$B(B^{\circ}A \cap C) \cap A$	$A \cap (A \cup B)$	$\xmapsto{r}$	A
$A \cap BC$	$\xmapsto{r}$	$A \cap (AC^{\circ} \cap B)C$	$(A \cup B) \cap A$	$\xmapsto{r}$	A
$A \cap B^{\circ}$	$\xmapsto{r}$	$(A^{\circ} \cap B)^{\circ}$	$B \cap (A \cup B)$	$\xmapsto{r}$	B
$A \cap (B \cup C)$	$\xmapsto{r}$	$(A \cap B) \cup (A \cap C)$	$(A \cup B) \cap B$	$\xmapsto{r}$	B
$A(B \cup C)$	$\xmapsto{r}$	$AB \cup AC$	$(B \cup C)A$	$\xmapsto{r}$	$BA \cup CA$
$A(BC)$	$\xmapsto{r}$	$(AB)C$	$A \cup (A \cap B)$	$\xmapsto{r}$	A
$(AB)^{\circ}$	$\xmapsto{r}$	$(B^{\circ}A^{\circ})$	$(A \cap B) \cup A$	$\xmapsto{r}$	A
AB°	$\xmapsto{r}$	$(BA^{\circ})^{\circ}$	$A0$	$\xmapsto{r}$	0
$0A$	$\xmapsto{r}$	0	$A \cap 0$	$\xmapsto{r}$	0
$A \cap 1$	$\xmapsto{r}$	A	$0 \cap A$	$\xmapsto{r}$	0
$1 \cap A$	$\xmapsto{r}$	A	$A \cap A$	$\xmapsto{r}$	A
$A \cup 0$	$\xmapsto{r}$	A	$A \cup A$	$\xmapsto{r}$	A
$0 \cup A$	$\xmapsto{r}$	A	$A \cup 1$	$\xmapsto{r}$	1
$id \cup di$	$\xmapsto{r}$	1	$1 \cup A$	$\xmapsto{r}$	1
$di \cup id$	$\xmapsto{r}$	1	$A\,id$	$\xmapsto{r}$	A
$\mathbf{1}di$	$\xmapsto{r}$	$\mathbf{1}$	$id\,A$	$\xmapsto{r}$	A
$di\mathbf{1}$	$\xmapsto{r}$	$\mathbf{1}$	$\mathbf{0}^{\circ}$	$\xmapsto{r}$	$\mathbf{0}$
id°	$\xmapsto{r}$	id	$\mathbf{1}^{\circ}$	$\xmapsto{r}$	$\mathbf{1}$
di°	$\xmapsto{r}$	di	$(A^{\circ})^{\circ}$	$\xmapsto{r}$	A

Term-algebra reduction rules

$\{(u,u)\} \cap id$	$\xmapsto{t}$	$\{(u,u)\}$	$\{(u,u)\} \cap di$	$\xmapsto{t}$	0
$\{(u,v)\} \cap id$	$\xmapsto{t}$	0	$\{(u,v)\} \cap di$	$\xmapsto{t}$	$\{(u,v)\}$
$F \cap id,$	$\xmapsto{t}$	0	$F \cap G$	$\xmapsto{t}$	0
$F^{\circ}G$	$\xmapsto{t}$	0	$F(A \cap B)$	$\xmapsto{t}$	$FA \cap FB$
$id \cap di$	$\xmapsto{t}$	$\mathbf{0}$	$((hd)^{\circ})tl$	$\xmapsto{t}$	$\mathbf{1}$
$((tl)^{\circ})hd$	$\xmapsto{t}$	$\mathbf{1}$	$((hd)^{\circ})hd$	$\xmapsto{t}$	id
$((tl)^{\circ})tl$	$\xmapsto{t}$	id	$((f_i^n)^{\circ})f_i^n$	$\xmapsto{t}$	id
$(f_j^n)^{\circ}(f_i^n)$	$\xmapsto{t}$	$\mathbf{1}$	$(f_j^n)^{\circ}\{(\bar{f},v)\}$	$\xmapsto{t}$	0
$\{(u,\bar{f})\}(f_j^n)$	$\xmapsto{t}$	0	$(hd)^{\circ}\{(\bar{\ell},v)\}$	$\xmapsto{t}$	0
$\{(u,\bar{\ell})\}(hd)$	$\xmapsto{t}$	0	$(tl)^{\circ}\{(\bar{\ell},v)\}$	$\xmapsto{t}$	0
$\{(u,\bar{\ell})\}(tl)$	$\xmapsto{t}$	0	$\{(u,v)\}\{(v,w)\}$	$\xmapsto{t}$	$\{(u,w)\}$
$(\{(u,v)\})^{\circ}$	$\xmapsto{t}$	$\{(v,u)\}$	$\{(u,f(v_1,\ldots,v_n))\}(f_j^n)$	$\xmapsto{t}$	$\{(u,v_j)\}$
$\{(u,[v \mid w])\}hd$	$\xmapsto{t}$	$\{(u,v)\}$	$\{(u,f(v_1,\ldots,v_n))\}(g_j^m)$	$\xmapsto{t}$	0
$\{(u,[v \mid w])\}tl$	$\xmapsto{t}$	$\{(u,w)\}$	$\mathbf{1}(u,v)\mathbf{1}$	$\xmapsto{t}$	$\mathbf{1}$
$\mathbf{1}(f_i^n)$	$\xmapsto{t}$	$\mathbf{1}$	$\mathbf{1}(hd)$	$\xmapsto{t}$	$\mathbf{1}$
$\mathbf{1}(tl)$	$\xmapsto{t}$	$\mathbf{1}$	$\{(u,v)\} \cap A$	$\xmapsto{t}$	$\{(u,u)\}A\{(v,v)\}$

$$A \cap \{(u,v)\} \quad \xmapsto{t} \quad \{(u,u)\}A\{(v,v)\}$$

(u, u_i, v, v_i, w distinct ground terms in $\mathcal{L}$, a, b distinct constants and f, g distinct function symbols of arities n and m.)
F and G range over $\{f_i^n \in \mathcal{L}\,\} \cup \{hd, tl\}$.
$\bar{\ell}$ ranges over all terms not beginning with [_ | _].
$\bar{f}$ ranges over terms not beginning with f.

As with previous definitions, relation completeness can be defined relative to an arbitrary $\mathcal{L}$-algebra $\mathcal{U}$.

2.2 Axiomatizing the Herbrand Universe

We review an equational theory which is complete for the Herbrand Universe. While in general completeness for one model fails for recursive first-order theories it so happens that for a given finite language $\mathcal{L}$, the first order theory of the Herbrand Universe $\mathcal{H}_\mathcal{L}$ over $\mathcal{L}$ (i.e. $\{\varphi : \mathcal{H}_\mathcal{L} \models \varphi\}$) is decidable, admitting a partial elimination of quantifiers. This result (and its generalization to any so-called locally free algebra), was originally proven by Mal'cev in 1961 [Mal'cev], and has been further extended case of the algebras of rational and infinite terms and certain quotients of such algebras [Maher, Comon, CoHaJo]. A complete rewriting system for the quantifier elimination procedure is discussed in ,[Comon]. Since we will be making use of these results we restate them here.

Definition 2.11 (The Theory $\mathcal{T}_\mathcal{L}$) *Let $\mathcal{L}$ be a finite language. The theory $\mathcal{T}_\mathcal{L}$ is the following axiom scheme:*

$(TL1)$	$a \neq b$	(a and b distinct constants)
$(TL2)$	$\forall x(t[x] \neq x)$	(for each term $t[x]$ containing x distinct from x)
$(TL3)$	$\forall \vec{x}(f(\xi_1,\ldots,\xi_n) \neq a)$	(a a constant)
$(TL4)$	$\forall \vec{x}(\ f(\xi_1,\ldots,\xi_n) = f(\eta_1,\ldots,\eta_n) \iff \bigwedge_{i=1}^{n} (\xi_i = \eta_i))$	
$(TL5)$	$\forall \vec{x}(f(\xi_1,\ldots,\xi_n) \neq g(\eta_1,\ldots,\eta_m))$	$(f \not\equiv g)$
$(DC_\mathcal{L})$	$\forall x[\bigvee_{a_i \in \mathcal{L}} x = a_i \ \vee \ \bigvee_{f \in \mathcal{L}}(\exists y_1 \cdots y_{\alpha_f})x = f(y_1 \cdots y_{\alpha_f})]$	

For $f \in \mathcal{L}$ the arity of f is denoted α_f above. Axioms (TL1-5) are also known as *Clark's equality theory* in the logic programming community. The so-called **Domain Closure Axiom** $DC_\mathcal{L}$ expresses in a first order manner that all individuals are terms over $\mathcal{L}$. It is not a formula unless $\mathcal{L}$ is finite.

Theorem 2.12 (Maher,Mal'cev) *$\mathcal{T}_\mathcal{L}$ is decidable and complete, i.e. the set of consequences of $\mathcal{T}_\mathcal{L}$ is the theory of the Herbrand model $\mathcal{H}_\mathcal{L}$.*

Definition 2.13 *A set formula is* ***normal*** *or in* ***normal form*** *if it is of the form:*

$$(\exists \vec{v})(\bigvee_k \Theta_k) \tag{1}$$

with exactly two free variables $\boldsymbol{x}$ *and* $\boldsymbol{y}$*, and where $\vec{v} = (v_1,\ldots,v_n)$ for some natural number n and each Θ_j is a* ***canonical expression****: a conjunction of equations and inequations of the following form:*

$$(x = t_1) \wedge (y = t_2) \wedge \bigwedge_i (v_i = s_i) \ \wedge \ \bigwedge_j (\forall \vec{u})(x \neq p_j) \ \wedge \ \bigwedge_j (\forall \vec{w})(y \neq q_j) \ \wedge \ \bigwedge_{ij} (\forall \vec{z})(v_i \neq r_{ij}) \tag{2}$$

where t_1, t_2, s_i, p_j, q_j and r_{ij} are terms in which $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *do not occur, and where the v_i are distinct and* must occur *in either t_1 or t_2.*

Theorem 2.14 (Normal Form Theorem) *Let φ be a first order formula over $\mathcal{L}$. Then there is a* ***normal*** *formula θ_R such that*

$$\mathcal{T}_\mathcal{L} \ \vdash \ \varphi \longleftrightarrow \theta \tag{3}$$

Definition 2.15 *For each finitary relation expression R we define a* ***description*** *$\varphi_R(x,y)$ to be any formula φ_R such that $[\![R]\!] = \{(x,y) : \varphi_R(x,y)\}$ e.g.*

$$\varphi_{R\cup S}(x,y) \stackrel{def}{\equiv} \varphi_R(x,y) \vee \varphi_S(x,y) \qquad \varphi_{\{a;b\}}(x,y) \stackrel{def}{\equiv} (x = a \wedge y = b)$$
$$\varphi_{R;S}(x,y) \stackrel{def}{\equiv} (\exists V)(\varphi_R(x,V) \wedge \varphi_S(V,y)) \qquad \varphi_{id}(x,y) \stackrel{def}{\equiv} x = y$$
$$\varphi_{f_i^k}(x,y) \stackrel{def}{\equiv} (\exists V_1 \cdots V_{k-1})(x = f(V_1,\ldots,V_{i-1},y,V_i,\ldots,V_{k-1}))$$

Corollary 2.16 *Let R be a finitary relation expression, and φ_R an associated description, as defined above. Then there is a* **normal** *formula θ_R such that*

$$\mathcal{T}_{\mathcal{L}} \vdash \varphi_R \longleftrightarrow \theta_R \tag{4}$$

and there is an algorithm for so reducing it.

Proof: See [Maher, Comon]. ▮

The quantifier elimination result of Mal'cev et. al. serves as a basis for a mechanized translation $R \longmapsto \varphi_R \longmapsto \theta_R$.

Thus, over the Herbrand Universe *any finitary relation expression* is easily converted into a readable value, namely a normal form. It is an instance of the printing component of the relational computing paradigm discussed in the first section of the paper. This component of processing of relation expressions is implemented in a relation language prototype called `RAlog`[2]. Here is an example of a query to the `RAlog` interpreter and its response :

```
>> solve (hd;1)*complement((tl;f^1/1;1)*(hd;g^1/1)).

(hd ; 1) * complement((tl ; f ^ 1 / 1 ; 1) * (hd ; g ^ 1 / 1))={X;Y:
   X=[X2|X3] ^ X2 # g(Y)
 V X=[g(Y)|X2] ^ (X3)(X2 # f(X3))
  }.
```

The printed response stands for:

$$\{(x,y) : (\exists x_2, x_3)(x = [x_2|x_3] \wedge (x_2 \neq g(y)) \bigvee x = [g(y)|x_2] \wedge (\forall x_3)(x_2 \neq f(x_3)))\}.$$

3 Logic without variables

One of the main results in [TarGiv] due to Tarski, Maddux and Givant, 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** given in the appendix. Tarski and Givant also prove a much 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 make use of this result for a specific set of equations and signature. But first we sketch a simple proof of a semantic form of this theorem, for the special case of algebras of relations definable on a term algebra by first-order formulas over a given signature. Let $\mathcal{L}$ be a finite language. The atomic statements of $\mathcal{T}_{\mathcal{L}}$ 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* equations of the form $x = a$ or $x = f(y_1, \ldots, y_n)$ where a is a constant in $\mathcal{L}$ 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 $\mathcal{T}_{\mathcal{L}}$ to relation expressions as follows. A pair of relations A and B are called *conjugated quasiprojections* by Tarski, and *tabulations* by Freyd if they satisfy the axioms (17) in the appendix. Recall that $\Re_{\mathcal{L}}$ always contains the reserved conjugated quasiprojections hd and tl. Let n be a natural number greater than the largest number of variables occurring in any sentence to be considered below. Put:

$$P_1^n = hd \qquad P_2^n = tl; hd \qquad \cdots \qquad P_n^n = (tl)^n; hd \tag{5}$$

Observe that, in the Herbrand interpretation for $u, v \in \mathcal{U}$, we have $u [\![P_k]\!] v$ iff v is the k^{th} component[3] of u. Vectors $\boldsymbol{u}$ satisfying $\boldsymbol{u} [\![P_k]\!] x_k$ for each k will be denoted $\boldsymbol{u} = [x_1, x_2, \ldots, x_k, \cdots, x_n]$

[2] available by ftp from `allegory.cs.wesleyan.edu`

[3] Superscripts on the P_k^n will be omitted when they are clear from context

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

$$Q_i^n = \bigcap_{j \neq i < n} P_j(P_j)^\circ \qquad E^n = \bigcap_{j < n} P_j(P_j)^\circ$$

Observe that $u [\![E]\!] v$ means u and v have the same components $(u)_i$ for $(1 \leq i < n)$ and $u [\![Q_i]\!] v$ 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 are *elementary*: either $x_i = a, x_i = x_j$ or $x_{i_0} = f(x_{i_1}, \ldots, x_{i_n})$.

$$\begin{array}{lcllcl}
(x_i = a)^r & = & P_i(a,a)P_i^o \cap E & (\varphi \wedge \theta)^r & = & (\varphi)^r \cap (\theta)^r \cap E \\
(x_i = x_j)^r & = & P_i P_j^o \cap E & (\neg\varphi)^r & = & E - (\varphi)^r \\
(x_{i_0} = f(x_{i_1}, \ldots, x_{i_n}))^r & = & \bigcap_j P_{i_0}; f_j^n; P_{k_j}^o \cap E & (\exists x_i \varphi)^r & = & Q_i(\varphi)^r Q_i \cap E
\end{array}$$

Then we have the following result.

Theorem 3.1 *If θ is a sentence over the language of $\mathcal{T}_{\mathcal{L}}$, $\mathcal{H}_{\mathcal{L}} \models \theta \iff [\![(\theta)^r]\!] = [\![E]\!]$. For open formulas φ with free variables among $x_1, \ldots, x_n$*

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

The atomic case holds by definition of the relations f_i^n. The rest of the proof is a straightforward induction on the structure of formulas[4].

3.0.1 Equipollence

We will now make use of the central result in Tarski-Givant, namely that first-order reasoning in a relation algebra can be faithfully captured with equational reasoning over the theory QRA (see appendix), a result they call equipollence of the two formalisms. First we state this theorem and several corollaries in the form we shall need in the work that follows. The theory $\mathbf{QRA}(\mathcal{L})$ is given in the appendix.

Theorem 3.2 *Let $\mathcal{L}$ be a finite language containing a binary function symbol $p(x,y)$ and Γ a first-order theory over $\mathcal{L}$ from which injectivity of p ($\forall x \forall y \forall z \forall w (p(x,y) = p(z,w) \rightarrow x = z \wedge y = w)$ is derivable. Then*

$$\Gamma \vdash_1 \varphi \qquad \textit{iff} \qquad (\Gamma)^\dagger \vdash_e X_\varphi = 1$$

where $(\Gamma)^\dagger$ is a set of equational translations $X_\gamma = 1$ of the formulas γ in Γ, and where $\vdash_1$ is first-order derivability and $\vdash_e$ is equational derivability in the theory $\boldsymbol{QRA}(\mathcal{L})$ with atomic relation symbols f_i^n corresponding to the atomic function symbols of arity n of $\mathcal{L}$.

The equipollence result is studied at great length in Tarski and Givant *op.cit.*, where it is extended in several ways relevant to the kind of work discussed in this paper. To cite a few such results: the translation $(\;)^\dagger$ is recursive and preserves finiteness and decidability of theories. Also, the language $\mathcal{L}$ may contain finitely many new binary relation symbols. A similar equipollence result is shown for arithmetic, various systems of set theory and real arithmetic vis-à-vis suitable equational formalisms. The application of such results to capturing set, number-theoretic and real constraint logic programming along the lines of the development below remains to be exploited.

By combining the equipollence theorem with the normal form theorems of the preceding section we obtain

Theorem 3.3 *There is an equational theory $\mathfrak{C}$ (extending $\boldsymbol{QRA}(\mathcal{L})$) such that for any finitary relation expressions R, R'*

$$\mathfrak{C} \vdash_e R = R' \quad \textit{iff} \quad \mathcal{H}_{\mathcal{L}} \models \varphi_R \longleftrightarrow \varphi_{R'} \quad \textit{iff} \quad [\![R]\!] = [\![R']\!]$$

Just take Γ to be $\mathcal{T}_{\mathcal{L}}$ in the preceding theorem, with the binary function symbol $[\_ \mid \_] \in \mathcal{L}$.

We introduce a family of relational terms that will prove useful in translating logic programs, below.

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

Definition 3.4 *Let $m \geq n$ be integers.*

$$\begin{array}{rcl} K_m(a) &=& (a,a) \quad K_m(x_i) \;=\; id \\ K_m(f(u_1,\ldots,u_s)) &=& id \cap \left(\bigcap (f_i^s K(u_i)(f_i^s)^o)\right) \\ K_m([t_1,\ldots,t_n]) &=& id \cap \left(\bigcap (P_i^m K(t_i)(P_i^m)^o)\right) \end{array}$$

In the equational theory $\mathfrak{E}$, these relations capture the structure of terms, as the following lemma shows.

Lemma 3.5 *Two term vectors $\vec{t}$ and $\vec{s}$ with all free variables in one distinct from the other are unifiable (with most general unifier θ) if and only if*

$$\mathfrak{E} \vdash_e K(\vec{t}) \cap K(\vec{s}) = K(\theta\vec{t}).$$

If $\vec{t}$ and $\vec{s}$ are not unifiable then $\mathfrak{E} \vdash_e K(\vec{t}) \cap K(\vec{s}) = \mathbf{0}$.

Proof: Observe that since each K-term is formative, i.e. intersections of basic terms SBS^o which are compositions of functional relations on the left and their converses on the right, with $B = id$, a singleton or basic, the intersection of two K-terms of the form $SBS^o \cap SCS^o$ is provably equal in $\mathfrak{E}$ to $S(B \cap C)S^o$. Thus the following equation is derivable in $\mathfrak{E}$:

$$K(t) \cap K(s) \quad = \quad \bigcap_i (f_i^s(K(u_i) \cap K(v_i))(f_i^s)^o)$$

As we keep peeling off the functional relations f_i^s and their converses on either side, we obtain $\mathbf{0}$ if function symbols and/or constants clash, a "collapse" of the form $K(u_i) \cap K(u_i) = K(u_i)$ wherever they are identical, and, when v is a variable, a *merge*

$$S(K(u))S^o \cap S(K(v))S^o = S[K(u) \cap K(v)]S^o = S(K(u) \cap id)S^o = S(K(u))S^o$$

which corresponds, in ordinary unification of terms, to the result of replacing the variable v by the subterm u. ∎

Lemma 3.6 *There is a translation $\widetilde{K}$ of sequences terms in $\mathcal{L}$ to formative terms in $\Re_{\mathcal{L}}$such that two term vectors $\vec{t}$ and $\vec{s}$ are unifiable (with most general unifier θ and with occurs check) if and only if*

$$\mathfrak{E} \vdash_e \widetilde{K}(\vec{t}) \cap \widetilde{K}(\vec{s}) = \widetilde{K}(\theta\vec{t}).$$

If $\vec{t}$ and $\vec{s}$ are not unifiable then $\mathfrak{E} \vdash_e \widetilde{K}(\vec{t}) \cap \widetilde{K}(\vec{s}) = \mathbf{0}$.

The existence of the translations $\widetilde{K}$ and their properties follows from the the existence of the translation $(\_)^\dagger$ in the equipollence theorem applied to formulas expressing unifiability of terms (e.g. $x = t \wedge x = s$). A specific definition of $\widetilde{K}$ and a tedious but straightforward equational derivation of $\widetilde{K}(\theta\vec{t})$ following standard unification algorithms can also be described. We omit the details here.

Switching Relations

We define a special class of relation expressions used to give a combinatory formulation of permutation, i.e. of the operations of switching components of a vector. First fix n and let $P_1, \ldots, P_n$ be defined as in 5 in section 3. As before, we suppress the superscript n on the since it is fixed for this discussion. The expressions B^n_{ij} defined below will also be written without the n superscript when clear from context.

For each i, j between 1 and n let

$$B^n_{ij} = P_i P_j^o \cap P_j P_i^o \cap [\bigcap_{k \notin \{i,j\}}] P_k P_k^o.$$

The intended interpretation is

$$\langle x_1, \ldots, x_n\rangle [\![B_{i,j}]\!] \langle y_1, \ldots, y_n\rangle \iff x_i = y_j \;\&\; x_j = y_i \;\&\; (\forall k)(k \neq i \;\&\; k \neq j \rightarrow x_k = y_k)$$

Definition 3.7 *For each natural number n we define the n-**switching relation expressions** (or n-**permutations**) to be finite compositions of the B^n_{ij}.*

4 Embedded Prolog

4.1 Relational Logic Programming

We now restrict our attention to the translation of Horn Clause Programs as (recursive) relation equations. Via the ***fp*** operator we can then express each relation defined by the program as a single, negation-free relation term. We outline the procedure here. What follows is a description of a series of steps that transform a Horn clause program in which n predicate letters occur, to a set of n (recursive) relational equations consisting only of positive finitary relation expressions in n relation variables $p_1, \ldots, p_n$

$$\begin{array}{ccc} p_1 & = & F_1(p_1, \ldots, p_n) \\ \vdots & \vdots & \vdots \\ p_n & = & F_n(p_1, \ldots, p_n) \end{array}$$

The transformation of this to a single equation over $\Re_{\mathcal{L}}$(with occurrences of ***fp***) takes place essentially by a cascading variable elimination procedure which is reminiscent of Gaussian elimination in linear systems.

The question was raised in the introduction as to whether the sample transformation of a prolog program into a relation expression in the first pages was "handpicked" or fully representative of a general procedure. The answer is that a general procedure exists, but that the case considered on the opening page was deceptively simple. In order to come up with a relatively straightforward general transformation procedure we must do considerable violence to the appearance of the program (while preserving extensional and operational semantics). We will increase the arity of all predicates, and introduce many new variables.

Step 1: transforming the program

We first rewrite the original program $\mathcal{P}$ so that only *distinct* variables occur as arguments to the original predicates, via the introduction of fresh variables x_i and of equations of the form $x_i = t_i$ where the t_i are terms. For each clause

$$p(t_1, \cdots, t_{n_p}) \longleftarrow q(u_1, \ldots, u_{n_q}), \ldots$$

(where n_p is the arity of the predicate letter p) we obtain

$$p(x_1, \cdots, x_{n_p}) \longleftarrow x_1 = t_1, \ldots, x_{n_p} = t_{n_p}, q(y_1, \ldots, n_q), y_1 = u_1, \ldots, y_{n_q} = u_{n_q}, \ldots \tag{6}$$

If there is more than one clause with a common predicate letter p we rewrite them as a single clause with disjunctions embedded in the tail. Let n be the total maximum number of variables occurring in any one clause. We make *every predicate have arity n by adding as many "filler" variables as required.* Thus the clause just shown reads:

$$\begin{array}{c} p(x_1, \cdots, x_{n_p}, y_1, \ldots, y_{r_q}, \ldots) \longleftarrow \\ x_1 = t_1, \ldots, x_{n_p} = t_{n_p}, q(x_1, \cdots, x_{n_p}, y_1, \ldots, y_{n_q}, \ldots), y_1 = u_1, \ldots, y_{r_q} = u_{n_q}, \ldots \end{array}$$

that is to say, every distinct predicate takes as argument the same vector $\vec{x}$ of variables. Repeated occurrences of a predicate p in one clause need not have their n variables in the same order, so when we eliminate variables and introduce binary relations, we will compose on the left and right with *switching relations* (see 3.7) as explained below.

Step 2:translating the program

We now convert each predicate symbol p_i in the Herbrand base of the program into a coreflexive binary relation (also denoted p_i). Although we will be eliminating variables below, we display an intermediate state of the transformation, with all variables present, and using the relations P_i^m introduced in the previous section, with $m = lth(\vec{x})$ and where we have renamed variables so that $\vec{x} = [x_1, \ldots, x_n]$, the

y_j above are now x_{n+j}, and the terms $u_1, \ldots, u_r$ are now $t_{n+1}, \ldots, t_{n+r}$. Also the reverse implication is replaced by equivalence.

$$\vec{x}p\vec{x} \equiv \vec{x}q\vec{x}, \vec{x}\widetilde{K}([t_1, \ldots, t_m])\vec{x}, \ldots$$

where the $\widetilde{K}$ are as in lemma (3.6). Finally, replacing all commas with $\cap$ and embedded disjunctions with $\cup$ we have

$$p = id \cap q \cap \widetilde{K}([t_1, \ldots, t_m]) \cap \cdots$$

We illustrate with two examples.

Example 1

$$\begin{gathered}
p(x,y) \longleftarrow q(y, f(b)) \\
\Downarrow \\
p(x,y,u,v) \equiv q(x,y,u,v), u = y, v = f(b) \\
\Downarrow \\
p(x,y,u,v) \equiv q(x,y,u,v), [x,y,u,v] = [x,y,y,f(b)] \\
\Downarrow \\
[x,y,u,v]p[x,y,u,v] \equiv [x,y,u,v]q[x,y,u,v], \\
[x,y,u,v]P^4_3(P^4_2)^o[x,y,u,v], [x,y,u,v](P^4_4)f(b) \\
\Downarrow \\
[x,y,u,v]p[x,y,u,v] \equiv [x,y,u,v]q[x,y,u,v], \\
[x,y,u,v]P^4_3(P^4_2)^o[x,y,u,v], [x,y,u,v](P^4_4)(f^1_1)(b,b)(f^1_1)^o(P^4_4)^o[x,y,u,v] \\
\Downarrow \\
p = id \cap q \cap P^4_3(P^4_2)^o \cap (P^4_4)(f^1_1)(b,b)(f^1_1)^o(P^4_4)^o
\end{gathered}$$

Example 2 We now consider an example with recursion. Initially only the variables x and y occur in the clause being considered. Two new variables u and v are introduced. The variables are ordered in the following sequence (x,y,u,v). The first occurrence of an atom with predicate letter p, namely $p(x,y)$ is expanded to arity 4 by adding the remaining variables in order: $p(x,y,u,v)$. The second occurrence $p(u,v)$ is expanded to include all remaining variables in order: $p(u,v,x,y)$. Since we want to end up with an equation in which *all relations involve the same vector of variables* we use switching relations to rearrange variables as needed.

$$\begin{gathered}
p(x,y) \longleftarrow p(y, f(b)) \\
\Downarrow \\
p(x,y) \equiv p(u,v), u = y, v = f(b) \\
\Downarrow \\
p(x,y,u,v) \equiv p(u,v,x,y), [x,y,u,v] = [x,y,y,f(b)] \\
\Downarrow \\
[x,y,u,v]p[x,y,u,v] \equiv [u,v,x,y]p[u,v,x,y], \\
[x,y,u,v]P^4_3(P^4_2)^o[x,y,u,v], [x,y,u,v](P^4_4)f(b) \\
\Downarrow \\
[x,y,u,v]p[x,y,u,v] \equiv [x,y,u,v]B_{1,3}B_{2,4}pB_{2,4}B_{1,3}[x,y,u,v], \\
[x,y,u,v]P^4_3(P^4_2)^o[x,y,u,v], [x,y,u,v](P^4_4)(f^1_1)(b,b)(f^1_1)^o(P^4_4)^o[x,y,u,v] \\
\Downarrow \\
p = id \cap B_{1,3}B_{2,4}pB_{2,4}B_{1,3} \cap P^4_3(P^4_2)^o \cap (P^4_4)(f^1_1)(b,b)(f^1_1)^o(P^4_4)^o
\end{gathered}$$

The result of applying this procedure to all the clauses of a Horn Clause program is to obtain a set of n equations $E_1, \cdots, E_n$ in the n predicate letters of the program (now viewed as relation variables). The typical equation E_i being of the form

$$p_i = Q_1 \cup Q_2 \cup \cdots Q_l$$

where each Q_j is of the form $Q_j = id \cap p'_1 \cap S_1 \cap p'_2 \cap S_2 \cap \cdots \cap p'_n \cap S_n$ where each p'_i is of the form Bp_iB^o, with B_i a *switching relation* and where the S_i are *formative*. We write this as

$$p_i = E_i(p_1, \ldots, p_n). \tag{7}$$

Step 3: "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. We rewrite the last equation of system 7 above as

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

and substitute into equation $n-1$, obtaining

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

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

$$p_1 = \mathbf{fp} z_1 . E_1(z_1, \ldots)$$

where the right hand side is a closed (no-free-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 4.1 *A positive relation expression* $R \in \Re_{\mathcal{L}}$ *(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* $\mathcal{P}$ *.*

4.2 Simulating Depth-First Search

Let R_q be a relational logic program, (u, u) a singleton query, and A a finitary query. We can simulate depth-first-search for these queries by evaluating $(u, u) \cap R$ or $A \cap R$ left-to-right. This means expanding expressions of the form $\mathbf{fp}x.\mathcal{E}\,(x)$ to $\mathcal{E}\,(\mathbf{fp}x.\mathcal{E}\,(x))$ as they are encountered. This is, of course, equivalent to working with the system of equations (7) and replacing each relation symbol p_i from the original Horn-clause program by it's definition $E_i(p_1, \ldots, p_n)$ as it is encountered. First we distribute the query over unions, and then evaluate each term in the union by associating intersections to the left. Evaluation amounts to computing intersections of formative relation expressions and unfoldings of recursive definitions. If the query is a singleton, by the extensional completeness of formative expressions and lemma (3.5) the query will evaluate to itself, if it is in the success set of the program, and to zero if it is finitely failed. But we can consider more general queries and outputs.

Theorem 4.2 *Let* $\mathcal{P}$ *be a Horn clause program. Suppose* R_q *is a relation expression arising from translation of the predicate* q *defined by* $\mathcal{P}$ *. Let* $\mathfrak{A}$ *be the set of finitary queries* $A \in \Re_{\mathcal{L}}$ *satisfying:*

- $D[\![A]\!] = \{\vec{a} : (\vec{a}, \vec{a}) \in [\![A]\!]\}$ *is a finite subset of the success set* $\{\vec{a} : \mathcal{P} \vdash_{SLD} q(\vec{a})\}$ *of* $\mathcal{P}$ *corresponding to the query predicate* q
- *For each* $\vec{a} \in D[\![A]\!]$ *the SLD-tree for the query* $\mathcal{P} \vdash q(\vec{a})$ *is finite.*

Then the equational theory $\mathfrak{E}$ *is relation-complete with respect to* $\mathfrak{A}$ *,*$\{R_q\}$*. That is to say, for any* A *in* $\mathfrak{A}$ *,* $A \cap R_q$ *is provably equal to some finitary* A' *in* $\mathfrak{E}$ *.*

This follows from theorem (3.3). We give a brief sketch. Full details will appear in a forthcoming extended treatment. The proof is by induction on the sequence $\langle \ell_1, \ldots, \ell_r \rangle$ of depths of the SLD trees for members of $[\![A]\!]$. The argument establishes a stronger claim, namely that a deterministic rewrite system exists producing the desired A'.

Let $q(x_1, \ldots, x_n)$ be a predicate defined by program $\mathcal{P}$ and R_q its translation into a relation expression. The clause

$$q(t_1, \ldots, t_n) \longleftarrow p_1, \ldots, p_n \vee \cdots$$

in $\mathcal{P}$ defining q, after translation (without the "Gaussian elimination" step) is of the form

$$R_q = (S_1 \cap R_{p_1} \cap S_2 \cap R_{p_2} \cdots \cap R_{p_n}) \cup \cdots.$$

with formative terms S_i derived from the $\widetilde{K}(\vec{t}_i)$ for suitable term vectors $\vec{t}$. Left-to-right evaluation "simulating Prolog" yields

$$A \cap R_q \mapsto (A \cap S_1) \cap R_{p_1} \cap R_{p_2} \cdots \cap R_{p_n} \cdots \cup \cdots.$$

Now $(A \cap S_1)$ satisfies the hypothesis of the theorem ($[\![A \cap S_1]\!]$ is finite and $(A \cap S_1)$ finitary). Furthermore any member $(\vec{b}, \vec{b}) \in [\![A \cap S_1]\!]$ where $\vec{b} = b_1, \ldots, b_n$ is the term resulting from unification of query $q(a_1, \ldots, a_n)$ for $(\vec{a}, \vec{a}) \in [\![A]\!]$ with the head of the clause $q(t_1, \ldots, t_n) \longleftarrow p_1, \ldots, p_n \vee \cdots$. This unification results in a subgoal $\longleftarrow p'_1, \ldots, p'_n \vee \cdots$ every ground instance of which has an SLD tree whose depth is one less than the depth associated with the query $q(a_1, \ldots, a_n)$. By the induction hypothesis we have

$$(A \cap S_1) \cap R_{p_1} \mapsto A_1$$

for some correct finitary A_1. Thus we have

$$A \cap R_q \mapsto (A_1) \cap (R_{p_2} \cdots \cap R_{p_n}) \cup \cdots.$$

Continuing this way through the remaining R_{p_i} we obtain a correct finitary expression A'.

Remarks: The preceding theorem can be considerably strengthened with different classes of finitary, (but not necessarily finite) queries, the subject of a future treatment. Observe that the query in the preceding theorem may contain complementation. The reader familiar with constructive negation of Chan et al. [Stuckey], based on the quantifier elimination algorithm described above, will note that the class of queries described in the relation-completeness theorem (4.2) discussed above can be extended to coincide precisely with the sets describable by normal-form constraints of constructive negation.

5 Conclusions

We have mapped out a relational framework which captures a robust extension of Horn-clause programming, computationally as well as semantically. The formalism admits a natural extension to other constraint programming languages either by adding new equations (or rewrite rules) directly, or by representing relation expressions in other relational structures. Since the presence of complementation plays no special role once outside of the framework of constructive negation and the Herbrand universe, it is of interest to consider more general relational theories, such as Freyd's *distributive* or *division allegories* , (which permit variable elimination for intuitionistic logic), as well as *power allegories* in which higher order relations are expressed via an enriched equational theory[FreySce], which would permit a similar treatment of higher-order logic programming languages [Miller]. Our approach treats all such relational formalisms as *executable* extensions of the declarative paradigm. Thus, much work in relational program synthesis (e.g. [Backh, Moor92]) can be viewed as an extension to logic programming over the appropriate domain. There is also no need to restrict attention to set-theoretic representations of such theories. The representation theory of allegories and domain-theoretic representations open the way to denotational treatment of logic programming languages. This is perhaps the most important potential application of the relational calculus presented here.

Many computational questions remain to be solved in connection with relational rewriting systems. What are the most natural and efficient deterministic calculi associated with this formalism? What are the right notions of normal form? Are there interesting Church-Rosser systems? Are there strongly normalizing subsystems (corresponding to a class of total logic programs) and what typed calculi can be developed along these lines?

Hopefully some of these questions will be addressed in subsequent papers.

Appendix

The Theories RA and QRA($\mathcal{L}$)

$$R \cap R = R \qquad R \cap S = S \cap R \qquad R \cap (S \cap T) = (R \cap S) \cap T \tag{8}$$

$$Rid = R \qquad R^{\circ\circ} = R \qquad (RS)^\circ = S^\circ R^\circ \qquad (R \cap S)^\circ = S^\circ \cap R^\circ \tag{9}$$

$$R(S \cap T) \subseteq RS \cap RT \qquad RS \cap T \subseteq (R \cap TS^\circ)S \tag{10}$$

$$R \cup R = R \qquad R \cup S = S \cup R \qquad R \cup (S \cup T) = (R \cup S) \cup T. \tag{11}$$

$$R \cup (S \cap R) = R = (R \cup S) \cap R \qquad R(S \cup T) = RS \cup RT \quad R \cap (S \cup T) = (R \cap S) \cup (R \cap T). \tag{12}$$

$$(R \cup S)^\circ = R^\circ \cup S^\circ \qquad (S \cup T)R = SR \cup TR \tag{13}$$

$$\overline{R} \cap R = 0 \qquad R \cap 0 = 0 = R0 \qquad R \cup 0 = R \tag{14}$$

$$\overline{\overline{R}} = R \qquad \overline{R \cup S} = \overline{R} \cap \overline{S} \qquad \overline{R \cap S} = \overline{R} \cup \overline{S} \tag{15}$$

$$\overline{(\overline{A} \cup B)} \cup \overline{\overline{A} \cup \overline{B}} = A \qquad A^{\circ}(\overline{AB}) \subseteq \overline{B} \tag{16}$$

$$A^{\circ}A \subseteq id \qquad B^{\circ}B \subseteq id \qquad A^{\circ}B = \mathbf{1} \tag{17}$$

Def: *Axioms 8-16 define the equational theory* ***RA***. ***QRA*** *is* ***RA*** *together with (17). Let $\mathcal{L}$ be a signature. The Theory* ***QRA($\mathcal{L}$)*** *is the equational theory containing as* **terms** *the finitary expressions in $\Re_{\mathcal{L}}$, and as axioms the theory RA together with a set of axioms*

$$f_i^{n\circ} f_i^n \subseteq id \qquad (1 \leq i \leq n) \qquad\qquad f_i^{n\circ} f_j^n = \mathbf{1} \qquad (i \neq j) \tag{18}$$

for each n-ary function symbol f in $\mathcal{L}$. A **Relation Algebra** $\mathfrak{A} = \langle A, \cup, \overline{(\ )}, \circ, (\ )^{\circ}, id\rangle$ is a model of the RA equations. A Q-relation algebra is a relation algebra with a pair A,B satisfying (17), and A Q-$\mathcal{L}$-relation algebra is a relation algebra with elements $a_1, \ldots, a_n$ interpreting the terms f_i^n above, and satisfying axioms 18. A relation algebra is said to be **representable** in [TarGiv] if it is $(\cup, \overline{(\ )}, \circ, ()^{\circ}, id)$- isomorphic to a family of binary relations on some set with the standard interpretation. In [TarGiv] it is shown that all QRA's are representable[5].

Part of this research was carried out by both authors at the Mathematical Sciences Institute, Cornell University. James Lipton was partly supported by ONR grant 4331-001-srp-01.

References

[Backh] Backhouse, R, et. al., "A Relational Theory of Datatypes," 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.

[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.

[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.

[CurryFeys] Curry, H. B. and R. Feys, *Combinatory Logic*, North-Holland, Amsterdam, 1958.

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

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

[JafLas] Jaffar, J. and Lassez, J-L., "Constraint logic programming." in *Proceedings of the 14th ACM Symposium on the Principles of Programming Languages*, 1987.

[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*, Notes, 1988.

[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.

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

---

[5] A remark attributed to Maddux in [TarGiv] shows that this fact implies the semantic version of main equipollence result 3.2

[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* Technical Monograph PRG-98, Oxford Computing Lab, Oxford, 1992.

[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.