Under consideration for publication in Formal Aspects of Computing

Constraint Logic Programming with a Relational Machine [1]

Emilio Jesús Gallego Arias[1], James Lipton[2], Julio Mariño[3]

[1] CRI MINES ParisTech, 35 rue St Honoré, Fontainebleau, Seine-et-Marne, France
[2] Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT 06459, USA
[3] ETSI Informáticos, Universidad Politécnica de Madrid, Campus de Montegancedo S/N, 28660 Boadilla del Monte, Spain

Abstract. We present a declarative framework for the compilation of constraint logic programs into variable-free relational theories which are then executed by rewriting. This translation provides an algebraic formulation of the abstract syntax of logic programs. Logic variables, unification, and renaming apart are completely elided in favor of manipulation of variable-free relation expressions.

In this setting, term rewriting not only provides an operational semantics for logic programs, but also a simple framework for reasoning about program execution.

We prove the translation sound, and the rewriting system complete with respect to traditional SLD semantics.

Keywords: logic programming, constraint programming, relation algebra, rewriting, semantics

1. Introduction

Logic programming is a paradigm based on proof search and programming with logical theories. The main goal is *declarative transparency*: guaranteeing that execution respects the mathematical meaning of the program. The power that such a paradigm offers comes at a cost for formal language research and implementation. Logic variables, unification, renaming variables apart and proof search are cumbersome to handle formally. Consequently, it is often the case that the treatment of these aspects is left outside the semantics of programs, complicating reasoning about them and the introduction of new declarative features.

We address this problem here by proposing a new mathematical framework for compilation – based on ideas of Tarski [TG87] and Freyd [FS91] – that encodes logic programming syntax into a variable-free algebraic formalism: relation algebra. Relation algebras are pure equational theories of structures containing the operations of composition, intersection and converse. An important class of relation algebras is that of the so-called *distributive relation algebras with quasi-projections*, which also incorporate union and projections.

We present the translation of constraint logic programs to such algebras in three steps. First, for a CLP

[1] The final publication is available at Springer; 10.1007/s00165-016-0369-z
Correspondence and offprint requests to: E. J. Gallego Arias, J. Lipton, J. Mariño

```
edge(a,b).          connected(X,X).
edge(b,c).          connected(X,Y) ← edge(X,Z), connected(Z,Y).
edge(a,e).
edge(e,f).
```

Figure 1. A logic program to decide on node reachability in a graph.

program P with signature Σ, we define its associated relation algebra $\mathbf{QRA}_\Sigma$, which provides both the target object language for program translation and formal axiomatization of constraints and logic variables. Second, we introduce a constraint compilation procedure that maps constraints to variable-free relation terms in $\mathbf{QRA}_\Sigma$. Third, a program translation procedure compiles constraint logic programs to an equational theory over $\mathbf{QRA}_\Sigma$.

The key feature of the semantics and translation is its variable-free nature. Queries and resolvents, which may contain *logical* variables, are represented as ground terms in our setting, which makes it possible to express program execution as term rewriting, without the need of extra mechanisms for unification or renaming of logic variables. The resulting system is sound and complete with respect to SLD resolution. Our compilation provides a solution to the following problems:

- Underspecification of abstract syntax and logic variable management in logic programs: solved by the inclusion of metalogical operations directly into the compilation process.
- Interdependence of compilation and execution strategies: solved by making target code completely orthogonal to execution.
- Lack of transparency in compilation (for subsequent optimization and abstract interpretation): solved by making target code a low-level yet *fully declarative* translation of the original program.

The second of these problems highlights what we feel is an important contribution to practical implementations of logic programming. The relational translation described in this paper yields target code that is evaluated via rewriting. Different rewriting strategies will permit the programmer to control search strategies (depth/breadth-first search, iterative deepening, etc.). We think this paper is an important step towards efficient algebraic execution and compilation of proof search for CLP. However, the rewriting system given here has not been designed aiming at performance, which will depend on the efficiency of the chosen rewriting engine and strategies.

In this paper we consider several criteria of correctness of the translation. We prove an *adequacy theorem* (Cor. 3.14) which compares a fixed point treatment of relational semantics with a comparable treatment of constraint logic programming, and we establish *equivalence of a specific rewriting strategy with SLD resolution* (Thm. 5.23). The latter result is aimed at showing that rewriting of translated code can execute as least as correctly as SLD resolution. But it is not exclusively for the sake of reproducing the depth-first execution of SLD resolution that this work is undertaken. One of the aims of the separation of translation and rewriting is to make execution independent of declarative content.

We illustrate the spirit of translation, and in particular the variable elimination procedure by considering a simple case, namely the transitive closure of a graph (Fig. 1). In this carefully chosen example the elimination of variables and the translation to binary relation symbols is immediate:

$$\begin{aligned} \mathbf{edge} &= (a,b) \cup (b,c) \cup (a,e) \cup (e,f) \\ \mathbf{connected} &= \mathbf{id} \cup \mathbf{edge}; \mathbf{connected} \end{aligned}$$

The key feature of the resulting term is the composition $\mathbf{edge}; \mathbf{connected}$. The logical variable Z is eliminated by the composition of relations allowing the use of variable-free object code. A query $\mathbf{connected}(a, X)$ is then modeled by the relation $\mathbf{connected} \cap (a, a)\mathbf{1}$ where $\mathbf{1}$ is the (maximal) universal relation and $\mathbf{id}$ by the identity relation. Computation can proceed by rewriting the query using a suitable orientation of the relation algebra equations and unfolding pertinent recursive definitions. Handling actual arbitrary constraint logic programs is more involved. First, we use sequences and projection relations to handle predicates involving an arbitrary number of arguments and an unbounded number of logic variables; second, we formalize constraints in a relational way. Projections and permutations algebraically encode all the operations of logical variables, disjunctive and conjunctive clauses are handled with the help of the standard relational operators $\cap$, $\cup$.

1.1. Background: Constraint Logic Programming

Constraint Logic Programming is logic programming with two different classes of predicates. Computation is the process of finding proofs for *goals* supplied by the programmer, consisting of sequences of such predicates. *Defined predicates* are solved by conventional resolution theorem proving. *Primitive predicates* are defined over an external logical theory. They are handled with the help of domain-specific inference engines called constraint solvers. These solvers are external: they are a black-box addition to the pure logic programming core.

There are many variants of logic programming depending on the fragment of logic used. In the rest of this work, we will focus on Logic Programming with first order Horn clauses and constraints. In this setting, a program is a set of Horn clauses with exactly one atomic consequence and a query is an existentially quantified conjunction. In pure logic programming, the information returned by an answer is a set of witnesses for the existentially quantified variables. In Constraint Logic Programming, answers are constraints restricting possible values of the existentially quantified variables. Conventional logic programming can be recovered from CLP by taking the constraint domain to be the Herbrand Universe. We assume the reader familiar with CLP (an in-depth treatment can be found in [Llo84, JM94]). In this section we give a quick overview of the main notions used in the paper.

Syntax Assume a permutative convention on symbols, i.e., unless otherwise stated explicitly, distinct names f, g stand for different entities (e.g. function symbols) and the same for distinct names i, j, for indices. A first-order language consists of a signature $\Sigma = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}_c \cup \mathcal{P}_d$, given by $\mathcal{C}$, the set of constant symbols, $\mathcal{F}$, the set of term formers or function symbols, and $\mathcal{P}_c, \mathcal{P}_d$ the set of primitive and defined predicates. We usually write $f, g, \ldots$ for elements of $\mathcal{F}$, a, b for elements of $\mathcal{C}$, r, s for elements of $\mathcal{P}_c$, and $p_i, p_j, q_i, q_j, \ldots$ for elements of $\mathcal{P}_d$. The function $\alpha : \Sigma \to \mathbb{N}$ returns the arity of its argument. We assume given a set $\mathcal{X}$ of so-called *logic variables* whose members are denoted $x_i, y_i, z_i, \ldots$.

We write $\mathcal{T}_\Sigma$ for the set of closed terms over $\mathcal{C} \cup \mathcal{F}$. We write $\mathcal{T}_\Sigma(\mathcal{X})$ for the set of open terms (in the variables in $\mathcal{X}$) over $\mathcal{C} \cup \mathcal{F}$. We drop Σ when understood from context. We write t, u, v for terms in $\mathcal{T}_\Sigma$.

Given $p \in \mathcal{P}_d$ of arity $\alpha(p) = n$ and n terms $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X})$, $p(t_1, \ldots, t_n)$ is an *atom*. An atom is *pure* iff all $t_1, \ldots, t_n$ are distinct variables. Similarly, for $r \in \mathcal{P}_c$, $r(t_1, \ldots, t_n)$ is an *atomic constraint*. The set $\mathcal{L}_\mathcal{D}$ of constraint formulas is the conjunctive and existential closure of the set of atomic constraints. We write φ, ϕ, ψ for constraints. A *literal* is an atom or a constraint. We slightly abuse terminology and write $p_i(\vec{x})$ for literals using variables from $\vec{x}$. A *Horn Clause* is a named expression of the following form:

$$cl : p(\vec{t}) \leftarrow q_1(\vec{u}_1), \ldots, q_n(\vec{u}_n)$$

with $p(\vec{t})$ an atom, called the *head* and $q_1(\vec{u}_1), \ldots, q_n(\vec{u}_n)$ a sequence of literals, with $n \geq 0$, called the *tail* of the cl. A *constraint logic program* is a finite set of Horn clauses.

We use vector notation extensively in the paper: $\vec{x} = x_1, \ldots, x_m$, $\vec{t} = t_1, \ldots, t_n$, $\vec{p} = p_1, \ldots, p_k$. The length of a sequence is written $|\cdot|$, thus $|\vec{x}| = m$, etc. $t[\vec{x}]$ denotes a term t from $\mathcal{T}_\Sigma(\mathcal{X})$ using variables in $\vec{x}$, and $\vec{t}[\vec{x}]$ denotes a sequence of terms using variables in $\vec{x}$. $p(\vec{t}[\vec{x}]))$ denotes an atom composed of a predicate p and arguments $t_1[\vec{x}], \ldots, t_n[\vec{x}]$. $\vec{p}(\vec{t}[\vec{x}])$ denotes a sequence of atoms $p_1(\vec{t}_1[\vec{x}]), \ldots, p_n(\vec{t}_n[\vec{x}])$. We may drop $[\vec{x}]$ or even $\vec{t}[\vec{x}]$ when the context allows and just write $\vec{p}$ for $\vec{p}(\vec{t}[\vec{x}])$. Given a sequence $\vec{p}$ of n terms, variables or atoms, we write $\vec{p}_{|k}$ for the sequence of $n - k + 1$ elements starting with the one at position k.

Constraints and the Interpretation of Logic Programs Given a signature Σ, a Σ-structure gives meaning to terms and primitive predicates:

Definition 1.1 (Σ-structure). A Σ-structure $\mathcal{D}$ consists of a set D and an assignment for elements of $\mathcal{C} \cup \mathcal{F} \cup \mathcal{P}_c$ in D in the usual way: constant symbols are mapped to individuals in D, function symbols of arity n to n-ary functions over D and predicate symbols of arity n to n-ary relations on D. We write $a^\mathcal{D}$, $f^\mathcal{D}$, $r^\mathcal{D}$ for the D-interpretation of constants, function and predicate symbols in D.

Definition 1.2 (Constraint Domain). A constraint domain is given by a pair $(\mathcal{D}, \mathcal{L}_\mathcal{D})$ consisting of a Σ-structure $\mathcal{D}$ and the set of constraint formulas generated from Σ.

A Σ-structure $\mathcal{D}$ induces a mapping or interpretation from elements of $\mathcal{L}_\mathcal{D}$ to the two-point lattice:

Definition 1.3 (Constraint Interpretation). The interpretation function for closed formulas $[\![\cdot]\!]^\mathcal{D} : \mathcal{L}_\mathcal{D} \to$

$\{\bot, \top\}$ is defined by induction on the structure of the formulas:

$$\begin{array}{lcl} [\![p(t_1,\ldots,t_n)]\!]^{\mathcal{D}} & = & \begin{cases} \bot & \text{if } (t_1^{\mathcal{D}},\ldots,t_n^{\mathcal{D}}) \notin p^{\mathcal{D}} \\ \top & \text{if } (t_1^{\mathcal{D}},\ldots,t_n^{\mathcal{D}}) \in p^{\mathcal{D}} \end{cases} \\ [\![\varphi \wedge \psi]\!]^{\mathcal{D}} & = & min([\![\varphi]\!]^{\mathcal{D}}, [\![\psi]\!]^{\mathcal{D}}) \\ [\![\exists x.\varphi]\!]^{\mathcal{D}} & = & max([\![\varphi[a/x]]\!]^{\mathcal{D}} \mid a \in D) \end{array}$$

The notion of interpretation is used to define constraint satisfaction:

Definition 1.4 (Constraint Satisfaction). A closed constraint formula φ is satisfied by $\mathcal{D}$, written $\mathcal{D} \models \varphi$, if $[\![\varphi]\!]^{\mathcal{D}} = \top$. If φ is open, we write $\mathcal{D} \models \varphi$ for the satisfiability of its *existential closure* $\mathcal{D} \models \exists(\varphi)$.

Operational Semantics We follow [JM94]: a *resolution computation with constraints* is a sequence of derivations induced by a transition system over program states that represent pending proof obligations and accumulated (satisfiable) constraints:

Definition 1.5 (Program State). A *program state* is an ordered pair $\langle \vec{p} \,|\, \varphi \rangle$ where $\vec{p}$ is a sequence of literals or *resolvent* and φ is a constraint formula or *constraint store*. We write $\Box$ for the empty resolvent.

The standard operational semantics for SLD resolution is defined as the following transition system over program states:

Definition 1.6 (SLD operational semantics).

$$\begin{array}{lcll} \langle \varphi, \vec{p} \,|\, \psi \rangle & \xrightarrow{\mathsf{cs}}_l & \langle \vec{p} \,|\, \psi \wedge \varphi \rangle & \text{iff } \mathcal{D} \models \psi \wedge \varphi \\ \langle p(\vec{t}[\vec{x}]), \vec{p} \,|\, \varphi \rangle & \xrightarrow{\mathsf{res_{cl}}}_l & \langle \vec{q}(\vec{v}[\sigma(\vec{z})]), \vec{p} \,|\, \varphi \wedge (\vec{u}[\sigma(\vec{y})] = \vec{t}[\vec{x}]) \rangle & \text{where: } cl : p(\vec{u}[\vec{y}]) \leftarrow \vec{q}(\vec{v}[\vec{z}]) \\ & & & \mathcal{D} \models \varphi \wedge (\vec{u}[\sigma(\vec{y})] = \vec{t}[\vec{x}]) \\ & & & \sigma \text{ a renaming apart for } \vec{y}, \vec{z}, \vec{x} \end{array}$$

This captures *ideal* CLP systems, where every new constraint is immediately checked for satisfiability. This means that the constraint store is consistent for every program state reached in execution. The reliance on an external constraint solver means that we lack any inference rules for constraint formulas, thus, the Σ-structure $\mathcal{D}$ contains the only and complete specification of validity for constraint formulas.

A query Q is a sequence of literals $p_1, \ldots, p_n$ over $\vec{x}$ variables with $n \geq 1$, logically interpreted as $\exists \vec{x}.(p_1 \wedge \cdots \wedge p_n)$. A query Q is usually embedded into a program state $\langle Q \,|\, \top \rangle$ to be executed with the above transition rules. A program state $\langle \vec{p} \,|\, \varphi \rangle$ *succeeds* iff it has a derivation that leads to an empty $\langle \Box \,|\, \psi' \rangle$ state. A state $\langle \vec{p} \,|\, \varphi \rangle$ is final iff no derivation exists starting from this state (denoted $\langle \vec{p} \,|\, \varphi \rangle \nrightarrow$). If $\vec{p}$ is not the empty resolvent, we say that the state *fails*.

Denotational Semantics The denotational semantics for a logic program P is its set of atomic consequences. In the presence of constraints, that amounts to the set of elements of $\mathcal{D}$ that satisfy P's predicates.

A constructive description of this set is usually done in terms of a sequence of approximations, using the Apt-van Emden-Kowalski interpretation operator T_P, see [Llo84, JM94].

We assume given in the rest of the section a program P with signature Σ, constraint domain $(\mathcal{D}, \mathcal{L}_{\mathcal{D}})$, where $\mathcal{D}$ is a sigma structure.

A valuation θ is a mapping from variables to D and the natural extension that maps terms to D and constraint formulas to closed $\mathcal{L}_{\mathcal{D}}$ formulas. A $\mathcal{D}$-interpretation $\mathcal{I}$ of a formula is an interpretation of the formula with the same domain as $\mathcal{D}$ and the same interpretation for the symbols in Σ as $\mathcal{D}$. It can be represented as a subset of $\mathcal{I} = \{p_i(\vec{a}) \mid p_i \in \mathcal{P}_d, \vec{a} \in D^{\alpha(p_i)}\}$.

Definition 1.7 (Interpretation transformer). Let $\mathcal{I}$ be the directed complete partial order of interpretations. The interpretation transformer $T_P^{\mathcal{D}} : \mathcal{I} \longrightarrow \mathcal{I}$ is defined as:

$$T_P^{\mathcal{D}}(I) = \{p(\vec{a}) \mid p(\vec{x}) \leftarrow \phi, \vec{q} \in P, \theta(\vec{x}) = \vec{a}, \mathcal{D} \models \theta(\phi), \forall i.\theta(q_i) \in \mathcal{I}\}$$

Without loss of generality, we have assumed that constraints ϕ are placed at the head of the clauses' bodies.

$T_P^{\mathcal{D}}$ is continuous and by the Knaster-Tarski theorem it has a *least fixed point* $[\![P]\!]$which is the minimal model of P: $[\![P]\!] = \bigcup_n (T_P^{\mathcal{D}})^{(n)}(\emptyset)$.

```
add(o, X, X).                          nat(o).
add(s(X), Y, s(Z)) ← add(X, Y, Z).     nat(s(N)) ← nat(N).

even(E) ← add(H,H,E).                  leq(o,_).
odd(O) ← nat(O), \+ even(O).           leq(s(N),s(M)) ← leq(N,M).
```

Figure 2. A logic program defining several predicates on Peano naturals.

Example 1.8 (Herbrand Constraint Domain). Suppose $\mathcal{H}$ is the Herbrand universe for a signature Σ, that is to say, the free $\Sigma_{\mathcal{H}}$-structure, and let $\mathcal{L}_{\mathcal{H}}$ stand for all open formulas over the signature of $\mathcal{H}$ built up from atoms using conjunction, existential quantification and equality of terms using variables from $\mathcal{X}$, the set of so-called logic variables. Then the Herbrand Constraint Domain is the pair $(\mathcal{H}, \mathcal{L}_{\mathcal{H}})$.

Example 1.9 (Peano Naturals). A standard example is the $\Sigma_{\mathbb{N}}$-structure given by the set $\mathbb{N}$, constant o interpreted as 0 and the unary function symbol s interpreted as $x \mapsto x + 1$. Figure 2 shows a small example program.

Some executions of logical queries to the program above follow:

$$\begin{array}{lcl}
\langle add(s(s(o)), s(s(o)), R) \mid \top\rangle & \to & \langle \square \mid R = s(s(s(s(o))))\rangle \\
\langle add(X, Y, s(s(o))) \mid \top\rangle & \to & \langle \square \mid X = o \wedge Y = s(s(o))\rangle, \langle \square \mid X = s(o) \wedge Y = s(o)\rangle, \langle \square \mid X = s(s(o)) \wedge Y = o\rangle \\
\langle even(s(s(o))) \mid \top\rangle & \to & \langle \square \mid \top\rangle \\
\langle even((s(o)) \mid \top\rangle & \nrightarrow & \\
\langle even(E) \mid \top\rangle & \to & \langle \square \mid E = o\rangle, \langle \square \mid E = s(s(o))\rangle, \langle \square \mid E = s(s(s(s(o))))\rangle, \ldots
\end{array}$$

2. Relation Algebras and Signatures

In this section, we define $\mathbf{QRA}_{\Sigma}$, a relation algebra in the style of [TG87, FS91]. The algebra contains special constants formalizing a CLP signature Σ and the program predicates $\mathcal{P}_d$, and the relational semantics is built around a constraint domain $\mathcal{D}$, a first order model. In this section we will define its language, its equational theory and semantics. For the entire section, we work with a fixed first-order model $\mathcal{D}$.

The reader should keep in mind, as the relational structure is defined below, that we are trying to capture the following fundamental syntactic components of logic programs in a variable-free manner.

Constants $a, b, c, \ldots$ specified by the signature. These are formalized using relation terms of the form e.g. (a, a) whose intended interpretation is the singleton relation $\{(a, a)\}$. For technical reasons (to fit into the translation of terms and predicates), constants a are interpreted as the following relation expression $(a, a)\mathbf{1}$, where $\mathbf{1}$ is the *universal relation* consisting of all ordered pairs.
In the standard set-theoretic relational semantics defined below, this relation denotes the set of all pairs (a, u) where a is fixed and u runs through all individuals of the relational model.

Variables from a master list $x_1, x_2, \ldots, x_i, \ldots$. These are interpreted using the converse of relational projections $P_1, P_2, \ldots$ to be defined below. In the semantics, the relation P_i associated with the ith variable x_i is the binary relation between n-tuples $\langle u_1, \ldots, u_n\rangle$ and the ith component u_i. In the translation from logical syntax to terms, variables x_i are represented as the *converse* P_i° of the ith projection relation.

Compound terms e.g. $f(x_3, g(a, x_2))$ built using the function symbols present in the signature of differing arities, in this case f of arity 2, and g also of arity 2. This means defining relation constants R_f, R_g that capture function symbols. Thus, unlike first-order logic programming languages (the relational counterparts of) function symbols are constants in the language and hence first-class citizens. For our example of arity 2, the semantics of the binary relation symbol R_f is the set of all pairs $(f(u_1, u_2), \langle u_1, u_2\rangle)$.
The compound term $f(x_3, g(a, x_2))$ can be viewed as $f(y, z)$ for fresh variables y and z, where $y = x_3$ and $z = g(a, x_2)$. This last equation in turn can be written $z = g(u, w)$ and $u = a$ and $w = x_2$. Thus we *flatten* compound terms into conjunctions of basic equations of the form $x =$ a variable or constant. Then the corresponding relational translation is

$$R_f[P_1 P_3^{\circ} \cap P_2 R_g[P_1(a, a)\mathbf{1} \cap P_2 P_2^{\circ}]]$$

which describes *the term starting with f whose first component is the variable x_3 and whose second component is the term starting with g whose first component is a and whose second component is x_2.*

Atomic constraint formulas e.g. $r(f(x_3, g(a, x_2), x_1, b)$ built using relation symbols r of different arities (also included in the signature). Here too we incorporate predicate symbols r as relation constants r. The interpretation of relations is *coreflexive*: it is a set of pairs of the form (u, u), that is to say a subset of the identity relation. In the specific case of a relation r of arity n the interpretation of the associated relational constant r is the set of pairs $(\langle u_1, \ldots, u_{n+k}\rangle, \langle u_1, \ldots, u_{n+k}\rangle)$ such that $r(u_1, \ldots, u_n)$ holds in the associated constraint domain. The atomic formula will also be flattened, converted to a conjunction of basic equations and then interpreted using intersections, compositions and converses of basic relation constants, just like the compound term above.

Vectors of terms Given the presence of projections hd and tl we are able to formalize pairing and hence the formation of sequences of terms in the relation calculus. We have already invoked this tacitly above by using n-ary projections P_i that are built using repeated compositions of hd and tl as spelled out below. A simple example is the relational formalization of a two element sequence $\langle a, b\rangle$ where a and b are constants in Σ. The corresponding relational term is $hd(a, a)hd^\circ \cap tl(b, b)tl^\circ$, which in a set-theoretic interpretation yields the set consisting of the ordered pair $\{(\langle a, b\rangle, \langle a, b\rangle)\}$.

Taking stock of what we will need to capture logic with relational terms, we see that at the very least we require constants to capture a signature, a universal relation $\mathbf{1}$ and its dual $\mathbf{0}$, converses, projections, unions, intersections and compositions. A few more components will be useful as well, as we see in the next subsection, which lays out the grammar and theory we will need.

2.1. Relational Language and Theory

The relation language R_Σ is built from a set $\mathsf{R}_\mathcal{C}$ of relation constants for constant symbols, a set $\mathsf{R}_\mathcal{F}$ of relation constants for function symbols from Σ, and a set of relation constants $\mathsf{R}_{\mathcal{P}_c}$for primitive predicates, as well as a fixed set of relation constants and operators detailed below. Let us begin with $\mathsf{R}_\mathcal{C}$. Each constant symbol $a \in \mathcal{C}_\Sigma$ defines a constant symbol $(a, a) \in \mathsf{R}_\mathcal{C}$, each function symbol $f \in \mathcal{F}_\Sigma$ defines a constant symbol R_f in $\mathsf{R}_\mathcal{F}$. Each predicate symbol $r \in \mathcal{P}_{c\Sigma}$ (the set of primitive, or constraint predicates) defines a constant symbol r in $\mathsf{R}_{\mathcal{P}_c}$. We write R_Σ for the full relation language:

$$\begin{array}{rcl} \mathsf{R}_\mathcal{C} = \{(a, a) \mid a \in \mathcal{C}_\Sigma\} \quad \mathsf{R}_\mathcal{F} = \{\mathsf{R}_f \mid f \in \mathcal{F}_\Sigma\} \quad \mathsf{R}_{\mathcal{P}_c} = \{\mathsf{r} \mid r \in \mathcal{P}_{c\Sigma}\} \\ \mathsf{R}_{atom} & ::= & \mathsf{R}_\mathcal{C} \mid \mathsf{R}_\mathcal{F} \mid \mathsf{R}_{\mathcal{P}_c} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl \\ \mathsf{R}_\Sigma & ::= & \mathsf{R}_{atom} \mid \mathsf{R}_\Sigma{}^\circ \mid \mathsf{R}_\Sigma \cup \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \cap \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma\mathsf{R}_\Sigma \end{array}$$

The constants $\mathbf{1}, \mathbf{0}, id, di$ respectively denote the universal relation (whose standard semantics is the set of all ordered pairs on a certain set), the empty relation, the identity (diagonal) relation, and identity's complement. Juxtaposition RR represents relation composition (often written R;R) and R° is the converse of R. We write hd and tl for the head and tail relations. The projection of an m-tuple onto its i-th element is written P_i and defined by $P_1 = hd, P_2 = tl; hd, \ldots, P_n = tl^{n-1}; hd$.

Relation Variables Later in this paper we will have need of relation expressions in which a finite set of *relation variables* $\{\overline{p}_1, \ldots, \overline{p}_n\}$ occur. We will call such expressions *polynomials* in the given set of relation variables. The relational language $\mathsf{R}_\Sigma(\overline{p}_1, \ldots, \overline{p}_n)$, obtained by freely adding such variables to the relation calculus R_Σ, is defined the same way R_Σ is, but with the relation variables added to the atomic case above.

$$\mathsf{R}_{atom} \quad ::= \quad \overline{p}_1 \mid \cdots \mid \overline{p}_n \mid \mathsf{R}_\mathcal{C} \mid \mathsf{R}_\mathcal{F} \mid \mathsf{R}_{\mathcal{P}_c} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl$$

$\mathbf{QRA}_\Sigma$ (Fig. 3) is the standard theory of distributive relation algebras, plus Tarski's quasiprojections [TG87], and equations axiomatizing the new relations of R_Σ.

Most of the equations will be familiar to the reader. They include the commutativity, associativity of union and intersection, the distributive laws, and basic properties of composition and inversion. Perhaps less familiar is the right *modular law*, $RS \cap T \subseteq (R \cap TS^\circ)S$, which can be thought of as expressing the right-factoring of S in $RS \cap T$. It is here written as containment, but can also be written in the equivalent (and more useful) equational form $RS \cap T = (R \cap TS^\circ)S \cap T$

The *left* modular law $RS \cap T \subseteq R(S \cap R^\circ T)$ is equivalent to the right modular law as can be shown by

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

$$hd(hd)^\circ \cap tl(tl)^\circ \subseteq id \qquad (hd)^\circ hd \subseteq id,\ (tl)^\circ tl \subseteq id \qquad (hd)^\circ tl = \mathbf{1}$$

Figure 3. $\mathbf{QRA}_\Sigma$

taking converses.

$$\begin{aligned}(RS \cap T)^\circ &= S^\circ R^\circ \cap T^\circ && (1)\\ &\subseteq (S^\circ \cap T^\circ R)R^\circ && (2)\\ &= (R(S \cap R^\circ T))^\circ && (3)\end{aligned}$$

Taking converses again we obtain $(RS \cap T) \subseteq R(S \cap R^\circ T)$ as we wanted to show. In special cases the modular law takes on a particularly simple form.

Lemma 2.1. In $\mathbf{QRA}_\Sigma$, $SS^\circ \subset id$ implies $A \cap SR = S(S^\circ A \cap R)$. $S^\circ S \subset id$ implies $A \cap RS = (AS^\circ \cap R)S$.

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

Note that products and their projections are axiomatized in a relational, variable-free manner. The equations for hd, the left projection and tl, the right projection guarantee that they are *functional* relations (every pair has a unique head and a unique tail) and the equation $hd^\circ tl = \mathbf{1}$, when given a set-theoretic interpretation, ensures that any element of the underlying set can be a head or a tail of a pair. $hd(hd)^\circ \cap tl(tl)^\circ \subseteq id$ asserts that two ordered elements uniquely specify a pair with the first element as head and the second as tail.

2.2. Semantics

Fix a signature Σ and a constraint domain $\mathcal{D}$. We now define an *interpretation* to be a set-valued mapping $[\![\_]\!]$ on R_Σ in a way that will satisfy every equation in $\mathbf{QRA}_\Sigma$ and every atomic formula true in $\mathcal{D}$ in the following sense

$$[\![(\langle a_1, \ldots, a_n\rangle, \langle a_1, \ldots, a_n\rangle)]\!] \subseteq [\![\mathsf{r}]\!] \text{ iff } \mathcal{D} \models r(a_1, \ldots, a_n)$$

where $\langle a_1, \ldots, a_n\rangle$ is a relational representation of a vector $a_1, \ldots, a_n$ of Σ terms, and r is a predicate symbol in $\mathcal{P}_c$.

It will be sufficient for the purposes of this paper to fix a canonical interpretation of hd and tl, and use $\mathcal{D}$ itself to interpret the relations R_f and r taken from the constraint signature. Relations will be interpreted in a specific power set as sets of ordered pairs over a structure that contains sequences of members of $\mathcal{D}$. Our sole degree of freedom will be the interpretation of program predicates $\overline{p}_1, \ldots, \overline{p}_n$, as members of this power set. We will then establish the existence, for each program P, of interpretations of the $\overline{p}_i$ that satisfy certain relation equations derived from P.

The canonical semantics $[\![\_]\!]^{\mathcal{D}^\dagger}$ We define $\mathcal{D}^\dagger$ to be the union of $\mathcal{D}^0 = \{\langle\rangle\}$ (the empty sequence), $\mathcal{D}$ and $\mathcal{D}$-finite products, for example: $\mathcal{D}^2, \mathcal{D}^2 \times \mathcal{D}, \mathcal{D} \times \mathcal{D}^2, \ldots$ We write $\langle a_1, \ldots, a_n\rangle$ for members of the n-fold

$$
\begin{array}{lcllcl}
[\![\mathbf{1}]\!]^{\mathcal{D}^\dagger} & = & \mathsf{R}_A & [\![tl]\!]^{\mathcal{D}^\dagger} & = & \{(\langle a, b\rangle, b) \mid a, b \in \mathcal{D}^\dagger\} \\
[\![\mathbf{0}]\!]^{\mathcal{D}^\dagger} & = & \emptyset & [\![R^\circ]\!]^{\mathcal{D}^\dagger} & = & ([\![R]\!]^{\mathcal{D}^\dagger})^\circ \\
[\![id]\!]^{\mathcal{D}^\dagger} & = & \{(u, u) \mid u \in \mathcal{D}^\dagger\} & [\![R \cup S]\!]^{\mathcal{D}^\dagger} & = & [\![R]\!]^{\mathcal{D}^\dagger} \cup [\![S]\!]^{\mathcal{D}^\dagger} \\
[\![di]\!]^{\mathcal{D}^\dagger} & = & \{(u, v) \mid u \neq v\} & [\![R \cap S]\!]^{\mathcal{D}^\dagger} & = & [\![R]\!]^{\mathcal{D}^\dagger} \cap [\![S]\!]^{\mathcal{D}^\dagger} \\
[\![hd]\!]^{\mathcal{D}^\dagger} & = & \{(\langle a, b\rangle, a) \mid a, b \in \mathcal{D}^\dagger\} & [\![(c, c)]\!]^{\mathcal{D}^\dagger} & = & \{(c^{\mathcal{D}}, c^{\mathcal{D}})\} \\
[\![RS]\!]^{\mathcal{D}^\dagger} & = & [\![R]\!]^{\mathcal{D}^\dagger}; [\![S]\!]^{\mathcal{D}^\dagger} & & & \\
[\![\mathsf{R}_f]\!]^{\mathcal{D}^\dagger} & = & \{(x, \vec{y}\vec{u}) \mid x = f^{\mathcal{D}}(a_1, \ldots, a_n) \wedge\ \vec{y} = \langle a_1, \ldots, a_n\rangle, a_i \in \mathcal{D}, \vec{u} \in \mathcal{D}^\dagger, n = \alpha(f)\} & & & \\
[\![\mathsf{r}]\!]^{\mathcal{D}^\dagger} & = & \{(\vec{x}\vec{u}, \vec{x}\vec{u}) \mid \vec{x} = \langle a_1, \ldots, a_n\rangle \wedge r^{\mathcal{D}}(a_1, \ldots, a_n), a_i \in \mathcal{D}, \vec{u} \in \mathcal{D}^\dagger, n = \alpha(r)\} & & &
\end{array}
$$

Figure 4. Standard interpretation of binary relations.

product associating to the right, that is to say, $\langle a_1, \langle a_2, \ldots, \langle a_{n-1}, a_n\rangle \cdots\rangle\rangle$. We assume right-association of products wherever parentheses are absent. Note that the 1 element sequence does not exist in the domain, so we write $\langle a\rangle$ for a as a convenience.

Let $\mathsf{R}_{\mathcal{D}} = \mathcal{D}^\dagger \times \mathcal{D}^\dagger$. We make the power set of $\mathsf{R}_{\mathcal{D}}$ into a model of the relation calculus by interpreting atomic relation terms in a certain canonical way, and the operators in their standard set-theoretic interpretation. We interpret *hd* and *tl* as projections in the model.

Definition 2.2. Given a structure $\mathcal{D}$, a relational $\mathcal{D}$-**interpretation** is a mapping $[\![\_]\!]^{\mathcal{D}^\dagger}$ of relational terms into $\mathsf{R}_{\mathcal{D}}$ satisfying the identities in Fig. 4. The function α used in this table and elsewhere in this paper refers to the arity of its argument, whether a relation or function symbol from the underlying signature.

Theorem 2.3. Equational reasoning in $\mathbf{QRA}_\Sigma$ is sound for any interpretation:

$$\mathbf{QRA}_\Sigma \vdash R = S \Longrightarrow [\![R]\!]^{\mathcal{D}^\dagger} = [\![S]\!]^{\mathcal{D}^\dagger}$$

Proof. The proof is straightforward. The rules of equational reasoning (substituting equal terms for a given variable, applying transitivity, symmetry, identity and congruence laws) obviously preserve equality in a set-theoretic interpretation, so all one has to check is soundness of the axioms of $\mathbf{QRA}_\Sigma$. He we illustrate by showing that the modular law (in its "left-factored" form) holds in any interpretation and leave the remaining cases to the reader.

Suppose $(u, v) \in [\![R \cap ST]\!]^{\mathcal{D}^\dagger} = [\![R]\!]^{\mathcal{D}^\dagger} \cap [\![S]\!]^{\mathcal{D}^\dagger}[\![T]\!]^{\mathcal{D}^\dagger}$. Then for some $w \in \mathcal{D}^\dagger$, we have $(u, w) \in [\![S]\!]^{\mathcal{D}^\dagger}$ and $(w, v) \in [\![T]\!]^{\mathcal{D}^\dagger}$. But then $(w, u) \in ([\![S]\!]^{\mathcal{D}^\dagger})^\circ$ hence (w, v) is in both $([\![S]\!]^{\mathcal{D}^\dagger})^\circ[\![R]\!]^{\mathcal{D}^\dagger}$ and $[\![T]\!]^{\mathcal{D}^\dagger}$, so $(u, v) \in [\![S(S^\circ R \cap T)]\!]^{\mathcal{D}^\dagger}$ as was to be shown
□

2.3. Adding equations to the $\mathbf{QRA}_\Sigma$

We will now discuss how to build interpretations of the relation calculus satisfying a finite set of equations in a certain canonical form. Let P be a program and $\overline{p}_1, \ldots, \overline{p}_n$ be a sequence of *relation variables*. Recall (Subsection 2.1) that the extended relation calculus $\mathsf{R}_\Sigma(\overline{p}_1, \ldots, \overline{p}_n)$ is the set of relation terms, or *polynomials* generated by $\overline{p}_1, \ldots, \overline{p}_n$ and the symbols in R_Σ.

Definition 2.4. A finite set of equations in n relation variables $\overline{p}_1, \ldots, \overline{p}_n$ is said to be *canonical* or *definitional* if it is in the form

$$\overline{p}_i = R_i(\overline{p}_1, \ldots, \overline{p}_n) \quad (1 \leq i \leq n)$$

where each R_i is a polynomial in $\mathsf{R}_\Sigma(\overline{p}_1, \ldots, \overline{p}_n)$.

2.4. The Least Relational Interpretation Satisfying Definitional Equations

Let $\mathcal{F}$ be a finite set of n definitional equations in the relation variables $\overline{p}_1, \ldots, \overline{p}_n$. Given a structure $\mathcal{D}$ we now lift the definition of $\mathcal{D}$-*interpretation* given in Def. 2.2 to the extended relation calculus. An *extended*

interpretation $[\![]\!] : \mathsf{R}_\Sigma(\overline{p}_1, \ldots, \overline{p}_n) \longrightarrow \mathsf{R}_\mathcal{D}$ is a function satisfying the identities in Fig. 4 as well as mapping each relation variable $\overline{p}_i$ to an arbitrary member $[\![\overline{p}_i]\!]$ of $\mathsf{R}_\mathcal{D}$. Given a structure $\mathcal{D}$ for a language its action is completely determined by its values at the $\overline{p}_i$. Note that the set $\mathcal{I}$ of all such interpretations forms a DCPO, a directed-complete partial order with a least element, under pointwise operations. That is to say, any *directed* set $\{[\![\ ]\!]_d : d \in \Lambda\}$ of interpretations has a supremum $\bigvee_{d\in\Lambda}[\![\ ]\!]_d$ given by $T \mapsto \bigcup_{d\in\Lambda}[\![T]\!]_d$. The directedness assumption is necessary. For example, to show that a pointwise supremum of interpretations $\bigvee_{d\in\Lambda}[\![\ ]\!]_d$ preserves composition (one of the 13 identities of Fig. 4), we must show that for any relation terms R and S we have $\bigcup_{d\in\Lambda}[\![RS]\!]_d = \bigcup_{d\in\Lambda}[\![R]\!]_d; \bigcup_{d\in\Lambda}[\![S]\!]_d$. However the right hand side of this identity is equal to $\bigcup_{d,e\in\Lambda\times\Lambda}[\![R]\!]_d; [\![S]\!]_e$. But since the family of interpretations is directed, for every pair d, e of indices in Λ there is an $m \in \Lambda$ with $[\![\ ]\!]_d, [\![\ ]\!]_e \leq [\![\ ]\!]_m$, hence $\bigcup_{d,e\in\Lambda\times\Lambda}[\![R]\!]_d; [\![S]\!]_e \leq \bigcup_{m\in\Lambda}[\![R]\!]_m[\![S]\!]_m$. The reverse inequality is immediate and we obtain $\bigcup_{d\in\Lambda}[\![R]\!]_d; \bigcup_{d\in\Lambda}[\![S]\!]_d = \bigcup_{d\in\Lambda}[\![RS]\!]_d$.

The least element of the collection $\mathcal{I}$ is the interpretation $[\![\ ]\!]_0$ given by $[\![\overline{p}_i]\!]_0 = \emptyset$ for $0 \leq i \leq n$. In the remainder of this section, the word *interpretation* will refer to an extended $\mathcal{D}$-interpretation.

Lemma 2.5. Let $[\![\ ]\!]$ and $[\![\ ]\!]'$ be interpretations. If for all i $[\![\overline{p}_i]\!] \subseteq [\![\overline{p}_i]\!]'$ then $[\![\ ]\!] \leq [\![\ ]\!]'$.

Proof. By induction on the structure of extended relations. For all relational constants c we have $[\![c]\!] = [\![c]\!]'$ We will consider one of the inductive cases, namely that of composition. Suppose $[\![R]\!] \subseteq [\![R]\!]'$ and $[\![S]\!] \subseteq [\![S]\!]'$. Then we must show that $[\![RS]\!] \subseteq [\![RS]\!]'$. But this follows immediately by a set-theoretic argument, since $(x, u) \in [\![R]\!]$ and $(u, y) \in [\![S]\!]$ imply, by inductive hypothesis, that $(x, u) \in [\![R]\!]'$ and $(u, y) \in [\![S]\!]'$. It can also be proved using the axioms of $\mathbf{QRA}_\Sigma$ by showing that $A \cup A' = A'$ and $B \cup B' = B'$ imply $AB \cup A'B' = A'B'$. We leave the remaining cases to the reader. □

We will now define a operator $\Phi_\mathcal{F}$ from interpretations to interpretations, show it continuous and define the *interpretation generated by* $\mathcal{F}$ as its least fixed point. This interpretation will be the least extension of a given relational $\mathcal{D}$-interpretation satisfying the equations in $\mathcal{F}$.

Definition 2.6. Let $\mathcal{F}$ be a set of definitional equations $\{\overline{p}_i \stackrel{\circ}{=} R_i(\overline{p}_1, \ldots, \overline{p}_n) : 1 \leq i \leq n\}$ and let $\mathcal{I}$ be the set of extended $\mathcal{D}$-interpretations, with poset structure induced pointwise. Then we define the operator $\Phi_\mathcal{F} : \mathcal{I} \longrightarrow \mathcal{I}$ as follows

$$\Phi_\mathcal{F}([\![\ ]\!])(\overline{p}_i) = [\![R_i(\overline{p}_1, \ldots, \overline{p}_n)]\!].$$

Theorem 2.7. $\Phi_\mathcal{F}$ is a continuous operator, that is to say it preserves suprema of directed sets.

Proof. Let $\{[\![\ ]\!]_d : d \in \Lambda\}$ be a directed set of interpretations. By Lem. 2.5 it suffices to show that for all p_i

$$\Phi_\mathcal{F}(\bigvee_{d\in\Lambda} [\![\ ]\!]_d)(\overline{p}_i) = (\bigvee_{d\in\Lambda} \Phi_\mathcal{F}([\![\ ]\!]_d))(\overline{p}_i).$$

Let $[\![\ ]\!]^* = \bigvee_{d\in\Lambda}[\![\ ]\!]_d$. Then $\Phi_\mathcal{F}(\bigvee_{d\in\Lambda}[\![\ ]\!]_d)(\overline{p}_i) = [\![R_i(\overline{p}_1, \ldots, \overline{p}_n)]\!]^*$, which in turn is the union $\bigcup_{d\in\Lambda}[\![R_i(\overline{p}_1, \ldots, \overline{p}_n)]\!]_d$. But this is equal to $\bigcup_{d\in\Lambda}\Phi_\mathcal{F}([\![\ ]\!]_d)(\overline{p}_i)$. Therefore $\Phi_\mathcal{F}(\bigvee_{d\in\Lambda}[\![\ ]\!]_d) = \bigvee_{d\in\Lambda}\Phi_\mathcal{F}([\![\ ]\!]_d)$.
□

By Kleene's fixed point theorem $\Phi_\mathcal{F}$ has a least fixed point $[\![\ ]\!]^\dagger$ in $\mathcal{I}$. This fixed point is, in fact, the union of all $\Phi^{(n)}_\mathcal{F}([\![\ ]\!]_0), (n \in \mathbb{N})$. By virtue of its being fixed by $\Phi_\mathcal{F}$ we have $[\![\overline{p}_i]\!]^\dagger = [\![R_i(\overline{p}_1, \ldots, \overline{p}_n)]\!]^\dagger$. That is to say, all equations in $\mathcal{F}$ are true in $[\![\ ]\!]^\dagger$, which is the least interpretation with this property under the pointwise order.

3. Program Translation

We define constraint and program translation to relation terms. To this end, we define a function $\dot{K}$ from constraint formulas with — possibly free — logic variables to a variable-free relational term. $\dot{K}$ is the core of the variable elimination mechanism and will appear throughout the rest of the paper.

Second, we translate defined predicates — and CLP programs — to equations $\overline{p} \stackrel{\circ}{=} R$, where $\overline{p}$ will be drawn from a set of definitional variables standing for program predicate names p, and R is a relation term. The set of definitional equations can be both seen as an executable specification, by understanding it in terms of the rewriting rules given in this paper; or as a declarative one, by unfolding the definitions and using the standard set-theoretic interpretation of binary relations.

3.1. Constraint Translation

We fix a canonical list $x_1, \ldots, x_n$ of variables occurring in all terms, so as to translate them to variable-free relations in a systematic way. There is no loss of generality as later, we transform programs into this canonical form.

Definition 3.1 (Term Translation). Define a translation function $K : \mathcal{T}_\Sigma(\mathcal{X}) \to \mathsf{R}_\Sigma$ from first-order terms to relation expressions as follows:

$$\begin{array}{lcl} K(c) & = & (c,c)\mathbf{1} \\ K(x_i) & = & P_i^\circ \\ K(f(t_1,\ldots,t_n)) & = & \mathsf{R}_f; \bigcap_{i\leq n} P_i; K(t_i) \\ K(\langle t_1,\ldots,t_n\rangle) & = & \bigcap_{i\leq n} P_i; K(t_i) \end{array}$$

The semantics of the relational translation of a term is the set of all of the instances of that term, paired with the corresponding instances of its variables. For instance, the term $x_1 + s(s(x_2))$ is translated to the relation $+; (P_1; P_1^\circ \cap P_2; \mathsf{s}; P_1; \mathsf{s}; P_1; P_2^\circ)$. Thus $(\vec{a}, \vec{b}) \in [\![K(x_1 + s(s(x_2)))]\!]^{\mathcal{D}^\dagger}$ implies that $a = +\langle b_1, s(s(b_2))\rangle$. In the following lemma we see that this pattern holds for $[\![K(\vec{t})]\!]^{\mathcal{D}^\dagger}$ in general. It will be convenient below to use the notation $t^{\mathcal{D}}$ for the interpretation, in $\mathcal{D}$, of a ground term$t \in \mathcal{T}_\Sigma$.

Lemma 3.2. Let $t[\vec{x}]$ be a term of $\mathcal{T}_\Sigma(\mathcal{X})$ whose free variables are among those in the sequence $\vec{x} = x_1, \ldots, x_m$. Then, for any sequences $\vec{a} = a_1, \ldots, a_m \in \mathcal{D}^\dagger, \vec{u} \in \mathcal{D}^\dagger$ and any $b \in \mathcal{D}$ we have

$$(b, \vec{a}\vec{u}) \in [\![K(t[\vec{x}])]\!]^{\mathcal{D}^\dagger} \iff b = (t[\vec{a}/\vec{x}])^{\mathcal{D}}$$

Proof. By induction on term structure. The first base case is $t \equiv c$ where c is a constant in Σ. Then $(b, \vec{a}\vec{u}) \in [\![K(c)]\!]^{\mathcal{D}^\dagger}$ holds if and only if $(b, \vec{a}\vec{u})$ is in $[\![(c,c); \mathbf{1}]\!]^{\mathcal{D}^\dagger}$, or equivalently, if $b = c^{\mathcal{D}}$. But this is equivalent to saying $b = (c[\vec{a}/\vec{x}])^{\mathcal{D}}$.

The second base case is $t \equiv x_i$. Then, the pair $(b, \vec{a}\vec{u})$ is in $[\![K(x_i)]\!]^{\mathcal{D}^\dagger}$, i.e. in $[\![P_i^\circ]\!]^{\mathcal{D}^\dagger}$ if and only if $a_i = b$, or, equivalently, $b = (x_i[\vec{a}/\vec{x}])^{\mathcal{D}}$ as we wanted to show.

For the first inductive case, observe that $(b, \vec{a}\vec{u}) \in [\![K(f(t_1,\ldots,f_n))]\!]^{\mathcal{D}^\dagger}$ if and only if $(b, \vec{a}\vec{u}) \in [\![R_f; \cap_{i\leq n} P_i; K(t_i)]\!]^{\mathcal{D}^\dagger} = [\![R_f]\!]^{\mathcal{D}^\dagger}; \cap_{i\leq n} [\![P_i]\!]^{\mathcal{D}^\dagger}; [\![K(t_i)]\!]^{\mathcal{D}^\dagger}$. This is equivalent to saying that there are elements $\vec{b} = b_1, \ldots, b_n$ with $(b, \vec{b}) \in [\![R_f]\!]^{\mathcal{D}^\dagger}$ and for all $i \leq n$ we have $(\vec{b}, \vec{a}\vec{u}) \in [\![P_i]\!]^{\mathcal{D}^\dagger}; [\![K(t_i)]\!]^{\mathcal{D}^\dagger}$, Equivalently, $b = f^{\mathcal{D}}(b_1, \ldots, b_n)$ and for all i we have $(b_i, \vec{a}\vec{u}) \in [\![K(t_i)]\!]^{\mathcal{D}^\dagger}$. By the induction hypothesis, this is equivalent to $b_i = (t_i[\vec{a}/\vec{x}])^{\mathcal{D}}$, so by definition $b = (f(t_1,\ldots,t_n)[\vec{a}/\vec{x}])^{\mathcal{D}}$ as we wanted to show.

The argument for the remaining inductive case is almost identical. □

We will translate constraints over m variables to partially coreflexive relations over the elements that satisfy them. A binary relation R is *coreflexive* if it is contained in the identity relation, and it is *i-coreflexive* if its i-th projection is contained in the *identity relation*: $P_i^\circ; R; P_i \subseteq id$. Thus, for a variable x_i *free* in a constraint, the translation will be i-coreflexive.

We now formally define two *partial identity relation expressions* I_m, Q_i for the translation of existentially quantified formulas, in such a way that if a constraint $\varphi[\vec{x}]$ over m variables is translated to an m-coreflexive relation the formula $\exists x_i.\ \varphi[\vec{x}]$ corresponds to a coreflexive relation in all the positions but the i-th one, as x_i is no longer free. In this sense Q_i may be seen as a *hiding* relation.

Definition 3.3. The partial identity relation expressions I_m, Q_i for $m, i > 0$ are defined as follows.

$$I_m := \bigcap_{1\leq i\leq m} P_i(P_i)^\circ \qquad Q_i = I_{i-1} \cap J_{i+1} \qquad J_i = tl^i; (tl^\circ)^i$$

I_m is the identity on sequences up to the first m elements. Q_i is the identity on all but the i-th element, with the i-th position relating arbitrary pairs of elements.

Definition 3.4 (Constraint Translation). The $\dot{K} : \mathcal{L}_{\mathcal{D}} \to \mathsf{R}_\Sigma$ translation function for constraint formulas

is:

$$\begin{array}{lcl} \dot{K}(p(t_1,\ldots,t_n)) & = & (\bigcap_{i\leq n} K(t_i)^\circ; P_i^\circ); \mathsf{p}; (\bigcap_{i\leq n} P_i; K(t_i)) \cap id \\ \dot{K}(\varphi \wedge \theta) & = & \dot{K}(\varphi) \cap \dot{K}(\theta) \\ \dot{K}(\exists x_i.\ \varphi) & = & Q_i; \dot{K}(\varphi); Q_i \cap id \end{array}$$

As an example, the translation of the constraint $\exists x_1, x_2.s(x_1) \leq x_2$ is

$$Q_1; ([Q_2; ([(P_1^\circ; \mathsf{s}^\circ; P_1 \cap P_2^\circ; P_2); \leq; (P_1; \mathsf{s}; P_1^\circ \cap P_2; P_2^\circ)] \cap id); Q_2] \cap id); Q_1 \cap id$$

Observe that for every φ we have that $\dot{K}(\varphi)$ is *coreflexive*: $(\vec{u}, \vec{v}) \in [\![\dot{K}(\varphi)]\!]$ implies that $\vec{u} = \vec{v}$.

The following lemma establishes that the semantics of the relational translation of a constraint is faithful to the semantics of the constraint in the underlying model $\mathcal{D}$.

Lemma 3.5. Let $\varphi[\vec{x}]$ be a constraint formula with free variables among $\vec{x} = x_1, \ldots, x_m$. Then, for any sequences $\vec{a} = a_1, \ldots, a_m$ and $\vec{u}$ of members of $\mathcal{D}$

$$(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![\dot{K}(\varphi[\vec{x}])]\!]^{\mathcal{D}^\dagger} \iff \mathcal{D} \models \varphi[\vec{a}/\vec{x}]$$

Proof. By induction on the structure of the formulas:

- We consider the case of a unary constraint predicate p, where our atomic formula is just $p(t[\vec{x}])$ (the argument extends easily to higher arities). Observe that $(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![\dot{K}(p(t))]\!]^{\mathcal{D}^\dagger}$, i.e. $(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![K(t)^\circ P_1^\circ; \mathsf{p}; P_1; K(t)]\!]^{\mathcal{D}^\dagger}$ is equivalent to the assertion that for some $b \in \mathcal{D}, \vec{b}, \vec{v} \in \mathcal{D}^\dagger$

  $$(\vec{a}\vec{u}, b) \in [\![K(t)^\circ]\!]^{\mathcal{D}^\dagger} \text{ and } (b, \vec{b}) \in [\![P_1^\circ]\!]^{\mathcal{D}^\dagger} \text{ and } (b_1\vec{v}, b_1\vec{v}) \in [\![\mathsf{p}]\!]^{\mathcal{D}^\dagger}.$$

  Equivalently, we have

  $$b = b_1, \quad (b\vec{v}, b\vec{v}) \in [\![\mathsf{p}]\!]^{\mathcal{D}^\dagger} \text{ and } b = (t[\vec{a}/\vec{x}])^{\mathcal{D}}$$

  the latter equation by Lem. 3.2. By definition of $[\![\mathsf{p}]\!]^{\mathcal{D}^\dagger}$ this implies that $p^{\mathcal{D}}((t[\vec{a}/\vec{x}])^{\mathcal{D}})$, that is to say, that $\mathcal{D} \models p(t)[\vec{a}/\vec{x}]$.
  Conversely, if $p^{\mathcal{D}}((t[\vec{a}/\vec{x}])^{\mathcal{D}})$ then for some $\vec{v} \in \mathcal{D}^\dagger$ we have

  $$((t[\vec{a}/\vec{x}])^{\mathcal{D}}\vec{v}, (t[\vec{a}/\vec{x}])^{\mathcal{D}}\vec{v}) \in [\![\mathsf{p}]\!]^{\mathcal{D}^\dagger}.$$

  By the equivalences stated above, we obtain $(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![K(t)^\circ P_1^\circ; \mathsf{p}; P_1; K(t)]\!]^{\mathcal{D}^\dagger}$ for any $\vec{u} \in \mathcal{D}^\dagger$.
- For the case $\varphi[\vec{x}] \wedge \theta[\vec{x}]$,

  $$(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![\dot{K}(\varphi[\vec{x}]) \cap \dot{K}(\theta[\vec{x}])]\!] \iff (\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![\dot{K}(\varphi[\vec{x}])]\!] \text{ and } (\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![\dot{K}(\theta[\vec{x}])]\!]$$

  By the induction hypothesis this is equivalent to $\mathcal{D} \models \varphi[\vec{a}/\vec{x}]$ and $\mathcal{D} \models \theta[\vec{a}/\vec{x}]$, i.e. $\mathcal{D} \models \varphi[\vec{a}/\vec{x}] \wedge \theta[\vec{a}/\vec{x}]$.
- For the case $\exists x_i.\varphi[\vec{x}]$, let $\vec{a}$ be an arbitrary sequence in $\mathcal{D}^\dagger$ of the same length m as $\vec{x}$, $a_{i-1}^{\rightarrow} \equiv a_1, \ldots, a_{i-1}$ and $a_{i+1}^{\rightarrow} \equiv a_{i+1}, \ldots, a_m$. For arbitrary sequences $\vec{u}$ we have that

  $$(\vec{a}\vec{u}, \vec{a}\vec{u}) \in Q_i[\![\dot{K}(\varphi[\vec{x}])]\!]Q_i \cap id \iff \exists \vec{b}\vec{r}\ (\vec{a}\vec{u}, \vec{b}) \in Q_i \wedge (\vec{b}, \vec{r}) \in [\![\dot{K}(\varphi[\vec{x}])]\!] \wedge (\vec{r}, \vec{a}\vec{u}) \in Q_i.$$

  Equivalently, by the definition of Q_i

  $$(\vec{a}\vec{u}, \vec{a}\vec{u}) \in Q_i[\![\dot{K}(\varphi[\vec{x}])]\!]Q_i \iff \exists b_i\ (a_{i-1}^{\rightarrow} b_i a_{i+1}^{\rightarrow}\vec{u}, a_{i-1}^{\rightarrow} b_i a_{i+1}^{\rightarrow}\vec{u}) \in [\![\dot{K}(\varphi[\vec{x}])]\!]^{\mathcal{D}^\dagger}$$

  and by the induction hypothesis for φ

  $$(\vec{a}\vec{u}, \vec{a}\vec{u}) \in Q_i[\![\dot{K}(\varphi[\vec{x}])]\!]Q_i \iff \exists b_i\ \mathcal{D} \models \varphi[a_{i-1}^{\rightarrow} b_i a_{i+1}^{\rightarrow}/\vec{x}]$$

  Equivalently, we have

  $$\mathcal{D} \models (\exists x_i \varphi)[\vec{a}/\vec{x}].$$

  as we wanted to show.

□

With the translation of terms and constraints defined, we now proceed to give the translation of complete programs.

3.2. Translation of Constraint Logic Programs

To motivate the technical definitions below, we illustrate the program translation procedure using the Peano addition example in Fig. 2. This program is first *purified:* the variables in the head of the clauses defining each predicate are chosen to be a sequence of fresh variables x_1, x_2, x_3, with all bindings stated as equations in the tail.

$$add(x_1, x_2, x_3) \longleftarrow x_1 = 0, x_2 = x_3.$$
$$add(x_1, x_2, x_3) \longleftarrow \exists x_4 x_5.\ x_1 = s(x_4), x_3 = s(x_5), add(x_4, x_2, x_5))$$

The clauses are combined into a single definition similar to the Clark completion of a program. We also use the variable permutation π sending $x_1, x_2, x_3, x_4, x_5 \mapsto x_4, x_2, x_5, x_1, x_3$ to rewrite the occurrence of the predicate *add* in the tail so that its arguments coincide with those in the head.

$$\begin{aligned} add(x_1, x_2, x_3) \leftrightarrow\ & (x_1 = 0, x_2 = x_3) \\ & \vee\ \exists x_4 x_5.\ x_1 = s(x_4), x_3 = s(x_5), w_\pi\, add(x_1, x_2, x_3). \end{aligned}$$

Now we apply relational translation $\dot{K}$ defined above to all relation equations, and eliminate the existential quantifier using the *partial identity operator* I_3 defined above. We represent the permutation π using the relation expression W_π that simulates its behavior in a variable-free manner and replace the predicate *add* with a corresponding *relation* variable $\overline{add}$. (A formal definition of W_π and its connection with function w_π is given below, see Def. 3.10 and Lemma 3.11.)

$$\overline{add} \triangleq \dot{K}(x_1 = o \wedge x_2 = x_3) \cup I_3((\dot{K}(x_1 = s(x_4) \wedge x_3 = s(x_5)) \cap W_\pi\, \overline{add}\, W_\pi^\circ)))$$

Now we give a description of the general translation procedure. We first process programs to their complete database form as defined in [Cla77], which given the executable nature of our semantics reflects the choice to work within the minimal semantics. The main difference in our processing of a program P to its completed form P' is that a strict policy on variable naming is enforced, so that the resulting completed form is suitable for translation to relational terms.

Definition 3.6 (General Purified Form for Clauses). For a clause $p(\vec{t}[\vec{y}]) \leftarrow \vec{q}(\vec{v}[\vec{y}])$, let $h = \alpha(p)$, $y = |\vec{y}|$, $v = |\vec{v}|$, and $m = h + y + v$. Assume given the following vectors.

$$\begin{array}{lllllll} \vec{x} & = & \vec{x}_h\vec{x}_t & = & \vec{x}_h\vec{x}_y\vec{x}_v & = & x_1, \ldots, x_h, x_{h+1}, \ldots, x_{h+y}, x_{h+y+1}, \ldots, x_m \\ \vec{x}_h & & & & & = & x_1, \ldots, x_h \\ \vec{x}_t & & & = & \vec{x}_y\vec{x}_v & = & x_{h+1}, \ldots, x_{h+y}, x_{h+y+1}, \ldots, x_m \\ \vec{x}_y & & & & & = & x_{h+1}, \ldots, x_{h+y} \\ \vec{x}_v & & & & & = & x_{h+y+1}, \ldots, x_m \end{array}$$

the clause's GPF form is:

$$p(\vec{x}_h) \leftarrow \exists^{h\uparrow}.((\vec{x}_h = \vec{t}[\vec{x}_y] \wedge \vec{x}_v = \vec{v}[\vec{x}_y]), \vec{q}(\vec{x}_v))$$

$\exists^{n\uparrow}$ denotes existential closure with respect to all variables whose index is greater than n. $\vec{x}_h$ and $\vec{x}_t$ stand for head and tail variables. A program is in GPF form iff every one of its clauses is. After the GPF step, we perform Clark's completion.

Definition 3.7 (Completion of a Predicate). We define Clark's completed form for a defined predicate p with clauses $cl_1, \ldots, cl_n$ in GPF form:

$$\left.\begin{array}{l} p(\vec{x}_h) \leftarrow_{cl_1} tl_1 \\ \ldots \\ p(\vec{x}_h) \leftarrow_{cl_n} tl_k \end{array}\right\} \xRightarrow{\text{Clark's comp.}} p(\vec{x}_h) \leftrightarrow tl_1 \vee \cdots \vee tl_k$$

The above definition easily extends to programs. Completed forms are translated to relations by using $\dot{K}$ for the constraints, mapping conjunction to $\cap$ and $\vee$ to $\cup$. Existential quantification, recursive definitions and parameter passing are handled in a special way which we proceed to detail next.

3.2.1. Existential Quantification: Binding Local Variables

Variables *local* to the tail of a clause are existentially quantified. For technical reasons — simpler rewrite rules — we use the *partial identity* relation I_n, rather than the Q_n relation defined in the previous sections. I_n acts as an existential quantifier for all variables of index greater than a given number.

Lemma 3.8.

$$[\![I_n]\!]^{\mathcal{D}^\dagger} \quad = \quad \{(\vec{z}\vec{u}, \vec{z}\vec{v}) | \quad |\vec{z}| = n, \vec{z}, \vec{u}, \vec{v} \in \mathcal{D}^\dagger\}$$

Proof. Immediate: just observe that for each i $[\![P_i(P_i)^\circ]\!]^{\mathcal{D}^\dagger} = \{(\vec{u}a\vec{v}, \vec{u}'a\vec{v}') || \vec{u}| = |\vec{u}'| = i-1$ and $\vec{u}, \vec{u}', \vec{v}, \vec{v}', a \in \mathcal{D}^\dagger\}$ that is to say, that $P_i(P_i)^\circ$ relates arbitrary sequences except for the position i, where it is the identity. □

Lemma 3.9. Let $\vec{a} = a_1, \ldots, a_n \in \mathcal{D}$, $\vec{x} = x_1, \ldots, x_n$, let φ be a constraint over m free variables, with $m > n$, $\vec{y}$ a vector of length k such that $n + k = m$, and $\vec{u}, \vec{v} \in \mathcal{D}^\dagger$, then:

$$(\vec{a}\vec{u}, \vec{a}\vec{v}) \in [\![I_n; \dot{K}(\varphi[\vec{x}\vec{y}]); I_n]\!]^{\mathcal{D}^\dagger} \iff \mathcal{D} \models (\exists^{n\uparrow}.\varphi[\vec{x}\vec{y}])[\vec{a}/\vec{x}]$$

Proof. $(\vec{a}\vec{u}, \vec{a}\vec{v}) \in [\![I_n; \dot{K}(\varphi[\vec{x}\vec{y}]); I_n]\!]^{\mathcal{D}^\dagger} \iff$ for some $b_{n+1}, \ldots, b_m, \vec{u}', \in \mathcal{D}^\dagger$
$(\vec{a}\vec{b}\vec{u}', \vec{a}\vec{b}\vec{u}') \in [\![\dot{K}(\varphi[\vec{x}\vec{y}])]\!]^{\mathcal{D}^\dagger}$.

By Lem. 3.5, we know that $(\vec{a}\vec{b}\vec{u}', \vec{a}\vec{b}\vec{u}') \in [\![\dot{K}(\varphi[\vec{x}\vec{y}])]\!]^{\mathcal{D}^\dagger} \iff \mathcal{D} \models \varphi[\vec{a}\vec{b}/\vec{x}\vec{y}]$.

So $(\vec{a}\vec{u}, \vec{a}\vec{v}) \in [\![I_n; \dot{K}(\varphi[\vec{x}\vec{y}]); I_n]\!]^{\mathcal{D}^\dagger}$ is equivalent to $\mathcal{D} \models (\exists^{n\uparrow}.\varphi[\vec{x}\vec{y}])[\vec{a}/\vec{x}]$.

□

3.2.2. Parameter Passing

The information about the order of parameters in each pure atomic formula $p(x_{i_1}, \ldots, x_{i_r})$ is captured using permutations. Given a permutation $\pi : \{1..n\} \to \{1..n\}$, the function w_π on formulas and terms is defined in the standard way by its action over variables. We write W_π for the corresponding relation:

Definition 3.10 (Switching Relations). Let $\pi : \{1..n\} \to \{1..n\}$ be a permutation. The *switching relation expression* W_π, associated to π is:

$$W_\pi = \bigcap_{j=1}^{n} P_{\pi(j)}(P_j)^\circ.$$

We call n the *order* of π, and write it $o(\pi)$.

Lemma 3.11. Fix a permutation π and its corresponding w_π and W_π. Then:

$$[\![\dot{K}(w_\pi(p(x_1, \ldots, x_n)))]\!] = [\![W_\pi \dot{K}(p) W_\pi^\circ]\!]$$

Proof. Straightforward, This is just a restatement of the easy claim:

$$(\vec{a}', \vec{a}') \in [\![R]\!] \iff (\vec{a}, \vec{a}) \in [\![W_\pi R W_\pi^\circ]\!]$$

where $\vec{a} = a_1, \ldots, a_n$ and $\vec{a}' = a_{\pi(1)}, \ldots, a_{\pi(n)}$. □

3.2.3. The Translation Function

Now we may define the translation for defined predicates. We remind the reader (see Def. 3.6) that after being put in GPF, each clause is in the form $p(\vec{x}_h) \leftrightarrow tl_1 \vee \cdots \vee tl_k$ where each tail tl is in the form $\exists^{h\uparrow}.\vec{b}$ with $\vec{b}$ a conjunction $b_1, \ldots, b_k$ where each b_i is either an atomic formula $p_i(\vec{x}_i)$ (with p_i a defined predicate) or a constraint formula φ.

Definition 3.12 (Relational Translation of Predicates). Let $h, p(\vec{x}_h)$ be as in Def. 3.6. The translation function Tr from completed predicates to relational equations is defined by:

$$\begin{array}{lcl} Tr(p(\vec{x}_h) \leftrightarrow tl_1 \vee \cdots \vee tl_k) & = & (\overline{p} \stackrel{\circ}{=} Tr_{tl}(tl_1) \cup \cdots \cup Tr_{tl}(tl_k)) \\ Tr_{tl}(\exists^{h\uparrow}.\vec{b}) & = & I_h; (Tr_l(b_1) \cap \cdots \cap Tr_l(b_n)); I_h \\ Tr_l(\varphi) & = & \dot{K}(\varphi) \qquad\qquad\qquad\qquad \varphi \text{ a constraint} \\ Tr_l(p_i(\vec{x}_i)) & = & W_\pi; \overline{p_i}; W_\pi^\circ \quad \text{such that } \pi(x_1, \ldots, x_{\alpha(p_i)}) = \vec{x}_i \end{array}$$

where $\vec{x}_i$ is the original sequence of variables in p_i in the Clark completion of the program, and π a permutation on the variables in the clausethat transforms the ordered sequence of length $\alpha(p)$ starting at x_1 to $\vec{x}_i$.

We will sometimes write $I_n(R)$ for $I_n R I_n$ and $W_\pi(R)$ for $W_\pi R W_i^\circ$.

3.3. Adequacy of the Translation

In this subsection we establish the adequacy of the program translation introduced in the preceding subsection in the following sense. The canonical least fixed point semantics for a constraint logic program P agrees with the least fixed point semantics of its relational translation, as defined in subsection 2.4.

The $T_P^{\mathcal{D}}$ operator We shall be interested in the fixed-point semantics induced by the continuous operator $T_P^{\mathcal{D}}$ of Def. 1.7. We recapitulate the relevant definitions here in an equivalent form that will be useful in our context. It will also be convenient to adopt here the conventional notation for program clauses that places all constraints φ in front of the remaining atoms b_i: $p_i(\vec{x}_i) \longleftarrow \varphi, b_1, \ldots, b_r$, since *in the denotational semantics* the order does not matter. The clauses are assumed to be in GPF.

$$T_P^{\mathcal{D}}(I) = \{p_i(\vec{a}) : p_i(\vec{x}_i) \longleftarrow \varphi, b_1, \ldots, b_r \ \in \ P \wedge \exists \vec{u}\, \mathcal{D} \models \varphi(\vec{a}\vec{u}) \wedge \forall k\, b_k[\vec{a}\vec{u}/\vec{x}] \in I\}$$

where the defined predicate symbols in the program are $p_1, \ldots, p_m$, and $\vec{a}, \vec{u}$ are in $\mathcal{D}^\dagger$. The notation $\mathcal{I}_0$ will be used for the empty $\mathcal{D}$-interpretation and for each n we will denote by $\mathcal{I}_n$ the n-fold iteration $(T_P^{\mathcal{D}})^{(n)}(\mathcal{I}_0)$ of $T_P^{\mathcal{D}}$ on $\mathcal{I}_0$. We adopt a parallel notation for relational interpretations. Let $tr(P) = \mathcal{F}$ be the set of definitional equations for program P, and $\Phi_{\mathcal{F}}$ the operator it induces on relational interpretations (Subsec. 2.4). Then, letting $[\![\ ]\!]_0$ be the interpretation sending all definitional relation variables $\overline{p}_i$ to $\mathbf{0}$, we put $[\![\ ]\!]_{n+1} = \Phi_{\mathcal{F}}([\![\ ]\!]_n)$.

Theorem 3.13. Let p_i be a defined predicate symbol in program P, and let $\vec{a} \in \mathcal{D}^\dagger$ be such that $|\vec{a}| = |p_i|$, the arity of p_i. Then for each n and $\vec{v} \in \mathcal{D}^\dagger$

$$(\vec{a}\vec{v}, \vec{a}\vec{v}) \in [\![\overline{p}_i]\!]_n \iff p_i(\vec{a}) \in \mathcal{I}_n.$$

Proof. By induction on n. The base case is immediate. Suppose $(\vec{a}, \vec{a}) \in [\![\overline{p}_i]\!]_{n+1}$. Since $[\![\overline{p}_i]\!]_{n+1} = [\![R_i(\overline{p}_1, \ldots, \overline{p}_m)]\!]_n$ we have $(\vec{a}\vec{v}, \vec{a}\vec{v}) \in \bigcup tr(tl_j)$ where $p \longleftrightarrow \bigvee tl_j$ is the completed form for predicate p_i. But then for some tail $tl_j = \varphi, b_1, \ldots, b_r$ with each $b_k = w_{\pi_k} p_{i_k}(\vec{x})$,

$$(\vec{a}\vec{v}, \vec{a}\vec{v}) \in [\![I_h(\dot{K}(\varphi) \cap \bigcap_{k \leq r} W_{\pi_k} \overline{p}_{i_k} W_{\pi_k}^\circ) I_h]\!]_n$$

where $h = |p_i|$. By coreflexivity of $[\![\dot{K}(\varphi)]\!]$ (i.e. $[\![\dot{K}(\varphi)]\!] \subseteq id$) there exists $\vec{u} \in \mathcal{D}^\dagger$ with $(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![\dot{K}(\varphi) \cap \bigcap W_{\pi_k} \overline{p}_{i_k} W_{\pi_k}^\circ]\!]_n$. Thus we have $(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![\dot{K}(\varphi)]\!]_n$, which by Lem. 3.5 implies that $\mathcal{D} \models \varphi[\vec{a}\vec{u}/\vec{x}]$. Also for every $k \quad (1 \leq k \leq r)$ we have that $(\vec{a}\vec{u}, \vec{a}\vec{u}) \in [\![W_{\pi_k} \overline{p}_{i_k} W_{\pi_k}^\circ]\!]$ which implies that $(w_{\pi_k}\vec{a}\vec{u}, w_{\pi_k}\vec{a}\vec{u}) \in [\![\overline{p}_{i_k}]\!]_n$.

By the induction hypothesis there is a $\vec{u} \in \mathcal{D}^\dagger$ such that for every k we have $p_{i_k}[w_{\pi_k}\vec{a}\vec{u}/w_{p_k}\vec{x}] \in \mathcal{I}_n$. Thus $\forall k\, b_k[\vec{a}\vec{u}/\vec{x}] \in \mathcal{I}_n$. By the definition of $T_P^{\mathcal{D}}$ we have $p_i(\vec{a}) \in \mathcal{I}_{n+1}$, which is what we were trying to show.

The converse is shown by a symmetric argument and is left to the reader. □

Recall that $[\![\ ]\!]^\dagger$ is the least fixed point of $\Phi_{\mathcal{F}}$ and $[\![P]\!]$ the least fixed point of $T_P^{\mathcal{D}}$. We have the following corollary, which asserts that the least relational interpretation of program predicates agrees with its meaning in the program.

Corollary 3.14. Let $\vec{a}, \vec{v} \in \mathcal{D}^\dagger$ with $|\vec{a}| = |p_i|$. Then $(\vec{a}\vec{v}, \vec{a}\vec{v}) \in [\![\overline{p}_i]\!]^\dagger$ if and only if $p_i(\vec{a}) \in [\![P]\!]$.

m_1	$: I_m(\dot{K}(\psi))$	$\overset{P}{\longmapsto}$	$\dot{K}(\exists^{m\uparrow}.\psi)$	Hiding meta-reduction
m_1*	$: I_m(\mathbf{0})$	$\overset{P}{\longmapsto}$	$\mathbf{0}$	
m_2	$: W_\pi(\dot{K}(\psi))$	$\overset{P}{\longmapsto}$	$\dot{K}(w_\pi(\psi))$	Permutation meta-reduction
m_2*	$: W_\pi(\mathbf{0})$	$\overset{P}{\longmapsto}$	$\mathbf{0}$	
m_3	$: \dot{K}(\psi_1) \cap \dot{K}(\psi_2)$	$\overset{P}{\longmapsto}$	$\dot{K}(\psi_1 \wedge \psi_2)$	$\mathcal{D} \models \psi_1 \wedge \psi_2$
m_3	$: \dot{K}(\psi_1) \cap \dot{K}(\psi_2)$	$\overset{P}{\longmapsto}$	$\mathbf{0}$	$\mathcal{D} \not\models \psi_1 \wedge \psi_2$
m_4	$: \dot{K}(\psi) \cap \overline{q}$	$\overset{P}{\longmapsto}$	$\dot{K}(\psi) \cap (\Theta)$	where $\overline{q} \triangleq \Theta \in Tr(P)$

Figure 5. Constraint meta-reductions

Since $[\![\overline{p}_i]\!]^\dagger$ is the union of the $[\![\overline{p}_i]\!]_n$ and $[\![P]\!]$ the union of the $\mathcal{I}_n$ the conclusion follows immediately from the preceding theorem.

4. A Rewriting System for Resolution

In this section we present a rewriting system for proof search. The system is derived from equational theory $\mathbf{QRA}_\Sigma$, which makes soundness of execution immediate. In Sec. 5 we will show that answers obtained by SLD-resolution correspond to answers yielded by our rewriting system and conversely, thus establishing operational completeness.

The translation of logic programs to a ground signature allows us to use rewriting for programs with logic variables, overcoming the practical and theoretical difficulties that the existence of such variables entails. Additionally, we may speak of *executable* semantics: we use the same function to compile and denote CLP programs.

For practical reasons, we do not define our rewriting system over the full relational language, but over the image of the translation function. That is to say, the system makes sense for terms in the image of Tr. In principle, there would be no problem to drop this restriction, at the cost of using significantly more rules for no gain.

Formally, the signature of our rewriting system is given by the following term-forming operations over the sort $\mathcal{T}_R$: $\mathsf{I} : (\mathbb{N} \times \mathcal{T}_R) \to \mathcal{T}_R$, $\mathsf{W} : (\mathsf{Perm} \times \mathcal{T}_R) \to \mathcal{T}_R$, $\mathsf{K} : \mathcal{L}_\mathcal{D} \to \mathcal{T}_R$, $\cup : (\mathcal{T}_R \times \mathcal{T}_R) \to \mathcal{T}_R$ and $\cap : (\mathcal{T}_R \times \mathcal{T}_R) \to \mathcal{T}_R$. Thus, for instance, the relation $I_n; R; I_n$ is formally represented in the rewriting system as $\mathsf{I}(n, \mathsf{R})$, provided R represents R. In practice we make use of the conventional relational notation I_n, W_π when no confusion can arise. Rewrite rules are of the form $\rho : l \to r$ where ρ is the rule's name (optional), l and r patterns, and l not a variable.

For a given term, it may happen that more than one reducible position or *redex* exists. We call a rewriting strategy non-deterministic when the redex can be freely chosen. We say a strategy is parallel-outermost when the redex chosen is not a subterm of any other redex and if there is more than one such redex, all of them are reduced. We say a strategy is left-outermost if it tries to reduce the outermost term, and if this is not possible then selects the leftmost redex of the tree of reduction candidates for the subterms.

4.1. Meta-reductions

We formalize the interface between the rewrite system and the constraint solver as meta-reductions (Fig. 5). Every meta-reduction uses the constraint solver in a black-box manner to perform constraint manipulation and satisfiability checking.

Lemma 4.1. All meta-reductions are sound: if $m_i : l \overset{P}{\longmapsto} r$ then $[\![l]\!]^{\mathcal{D}^\dagger} = [\![r]\!]^{\mathcal{D}^\dagger}$.

Proof. Follows from Lem. 3.5. Let us consider rule m_1, whose left hand side abbreviates the term $I_m\dot{K}(\psi))I_m$ and whose right hand side is $\dot{K}(\exists^{m\uparrow}.\ \psi)$. Suppose the free variables of ψ are among $\vec{x}$, where $\vec{x}$ is chosen to be of length greater then m.

Given $\vec{a}, \vec{u}, \vec{u}' \in \mathcal{D}^\dagger$ with $\mid \vec{a} \mid$ equal to m, we have $(\vec{a}\vec{u}, \vec{a}\vec{u}') \in [\![I_m\dot{K}(\psi))I_m]\!]^{\mathcal{D}^\dagger}$ if there are $\vec{v}, \vec{w}, \vec{w}' \in \mathcal{D}^\dagger$ with $\mid \vec{a}\vec{v} \mid = \mid \vec{x} \mid$ such that $(\vec{a}\vec{v}\vec{w}, \vec{a}\vec{v}\vec{w}') \in [\![\dot{K}(\psi)]\!]^{\mathcal{D}^\dagger}$. By Lem. 3.5, this is the case if and only if there is a $\vec{v}$ such that

$$
\begin{array}{lll}
p_1 & : \mathbf{0} \cup R & \overset{P}{\longmapsto} \quad R \\
p_2 & : \mathbf{0} \cap R & \overset{P}{\longmapsto} \quad \mathbf{0} \\
p_3 & : W_\pi(R \cup S) & \overset{P}{\longmapsto} \quad W_\pi(R) \cup W_\pi(S) \\
p_4 & : I_n(R \cup S) & \overset{P}{\longmapsto} \quad I_n(R) \cup I_n(S) \\
p_5 & : (R \cup S) \cap T & \overset{P}{\longmapsto} \quad (R \cap T) \cup (S \cap T) \\
p_6 & : \dot{K}(\psi) \cap (R \cup S) & \overset{P}{\longmapsto} \quad (\dot{K}(\psi) \cap R) \cup (\dot{K}(\psi) \cap S) \\
p_7 & : \dot{K}(\psi) \cap (R \cap W_\pi(\overline{q_i})) & \overset{P}{\longmapsto} \quad (\dot{K}(\psi) \cap R) \cap W_\pi(\overline{q_i}) \\
p_8 & : \dot{K}(\psi) \cap W_\pi(\overline{q}) & \overset{P}{\longmapsto} \quad W^\circ_\pi(W_\pi(\dot{K}(\psi)) \cap \overline{q}) \\
p_9 & : \dot{K}(\psi) \cap I_m(R) & \overset{P}{\longmapsto} \quad I_m(I_m(\dot{K}(\psi)) \cap R) \cap \dot{K}(\psi)
\end{array}
$$

Figure 6. Rewriting system for SLD.

$\mathcal{D} \models \psi[\vec{a}\vec{v}/\vec{x}]$, i.e. iff $\mathcal{D} \models \exists^{m\uparrow}\ \psi[a_1, \ldots, a_m/x_1, \ldots, x_m]$, which in turn, implies $(\vec{a}\vec{u}, \vec{a}\vec{u}') \in [\![\dot{K}(\exists^{m\uparrow}\ \psi[\vec{x}])]\!]^{\mathcal{D}^\dagger}$. The argument for the converse is symmetric. Soundness for m_2 and m_3 is similar. □

4.2. A Rewriting System for SLD Resolution

The rewriting system for proof search is in Fig. 6. We prove local confluence.

Lemma 4.2. $\overset{P}{\longmapsto}$ is sound: if $p_i : l \overset{P}{\longmapsto} r$ then $[\![l]\!]^{\mathcal{D}^\dagger} = [\![r]\!]^{\mathcal{D}^\dagger}$.

Proof. All of the rules are consequences of relation algebra, except p_9. For p_9, we apply the modular law to obtain the derivation:

$$
\begin{array}{ll}
\dot{K} \cap IRI & =_{[I\dot{K}I \supseteq \dot{K}]} \\
\dot{K} \cap I\dot{K}I \cap IRI & \subseteq_{[RS \cap T \subseteq (R \cap TS^\circ)S]} \\
\dot{K} \cap (IR \cap I\dot{K}II^\circ)I & \subseteq_{[RS \cap T \subseteq R(R^\circ T \cap S)]} \\
\dot{K} \cap I(R \cap I^\circ I\dot{K}II^\circ)I & =_{[I^\circ I = I]} \\
\dot{K} \cap I(R \cap I\dot{K}I)I &
\end{array}
$$

The opposite direction $\dot{K} \cap IRI \supseteq \dot{K} \cap I(I\dot{K}I \cap R)I$ is immediate. □

A *left outermost strategy* gives priority to p_7 over p_8. Our system is confluent under left outermost rewriting.

Lemma 4.3. If we give higher priority to p_7 over p_8, $\overset{P}{\longmapsto}$ is locally confluent.

Proof. We prove that all the critical pairs join. We have three cases:

- m_1 overlaps with p_8, so using p_8: $\dot{K}(\psi_1) \cap I_m(\dot{K}(\psi_2)) \overset{P}{\longmapsto} I_m(I_m(\dot{K}(\psi_1)) \cap \dot{K}(\psi_2)) \cap \dot{K}(\psi_1) \overset{P}{\longmapsto} I_m(\dot{K}(\exists^{m\uparrow}.\psi_1) \cap \dot{K}(\psi_2)) \cap \dot{K}(\psi_1) \overset{P}{\longmapsto} I_m(\dot{K}(\exists^{m\uparrow}.\psi_1 \wedge \psi_2)) \cap \dot{K}(\psi_1) \overset{P}{\longmapsto} \dot{K}(\exists^{m\uparrow}.(\exists^{m\uparrow}.\psi_1 \wedge \psi_2)) \cap \dot{K}(\psi_1) \overset{P}{\longmapsto} \dot{K}(\exists^{m\uparrow}.(\exists^{m\uparrow}.\psi_1 \wedge \psi_2) \wedge \psi_1)$ which is logically equivalent to $\dot{K}(\psi_1 \wedge \exists^{m\uparrow}.\psi_2)$, that we obtain reducing with m_1.
- p_1 overlaps with p_5, so using p_5: $\dot{K}(\psi) \cap (\mathbf{0} \cup R) \overset{P}{\longmapsto} (\dot{K}(\psi) \cap \mathbf{0}) \cup (\dot{K}(\psi) \cap R) \overset{P}{\longmapsto} \mathbf{0} \cup (\dot{K}(\psi) \cap R) \overset{P}{\longmapsto} \dot{K}(\psi) \cap R$, which is what we get using p_1 directly.
- p_7 overlaps with p_8, and indeed this overlapping is not solvable without assigning a priority to some of the rules. The overlapping term is of the form $\dot{K}(\psi_1) \cap (\dot{K}(\psi_2) \cap W(\overline{q}))$, and as p_7 has higher priority than p_8 this is rewritten to $(\dot{K}(\psi_1) \cap \dot{K}(\psi_2)) \cap W(\overline{q})$ which leads to a non-problematic term $\dot{K}(\psi_1 \wedge \psi_2) \cap W(\overline{q})$.

□

5. Operational Equivalence

Theorem 3.14 establishes the correspondence of the relational and transformer-based interpretations. In this section, we aim to emulate that result for the computational interpretation of CLP programs. Viewed as

computational devices, constraint logic programs take the role of a theory, and the computational procedure will perform proof search for user-provided *queries*.

Several proof search strategies are common; we restrict our attention to widely-used SLD resolution; its popularity steas from a good efficiency versus completeness compromise, making it practical for a wide range of applications.

Additionally, SLD is a good representative for a strategy not particularly well suited to be captured in our algebraic approach: clause selection and the particular form of backtracking used yield control-flavored rewriting rules, obstructing profits coming from usual algebraic uniformity; think of an expression $R \cap S$, under SLD, R and S don't fully stand on equal footing.

Given a query, rewriting with the system of Sec. 4, will reach a certain normal form or fail to terminate. Then, our goal is to establish that the rewriting will precisely reach a normal form iff SLD proof search does find a proof.

Formally, we prove that given a program P and query Q, rewriting its relational translation will return a term $K(\varphi)$ iff SLD resolution for $P \vdash Q$ reaches a program state $\langle \square \mid \varphi' \rangle$, with answer constraints logically equivalent, $\varphi \iff \varphi'$. The proof proceeds in two main steps using an intermediate transition system:

- First, we prove "traditional" SLD equivalent to a carefully-crafted transition system that internalizes the renaming apart and search performed by the strategy.
  The states of the new transition system include an internal notion of *scope*, a natural number which represents the number of global variables in the state; *substate*, with parameter passing captured by a *permutation* of variables; *failure* and *parallel states*, which model alternatives in the search tree.
- The internal system can be then directly related to a transition system between relations, which captures the rewriting system at a higher level of granularity than its individual rules. Proving that the rewriting system indeed rewrites in synchrony with the transitions completes the proof.

5.1. Operational Semantics in Logic Style for SLD-resolution

We recall the standard SLD semantics and extend the notion of General Purified Form to program states. We prove several technical results, mainly the equivalence with the system in General Purified Form so we can work exclusively with them without loss of generality in the next subsections.

A *program state* is an ordered pair $\langle A_1, \ldots, A_n \mid \varphi \rangle$ where $A_1, \ldots, A_n$ is a sequence of atomic formulas or constraints known as the *resolvent*, and φ is a constraint formula known as the constraint store. We write $\square$ for the null resolvent, i.e. the empty sequence of formulas. We assume free variables in the constraint store to be existentially quantified.

Definition 5.1. The *standard transition system* capturing SLD resolution is:

$$\begin{array}{lll}
\langle \varphi, \vec{p} \mid \psi \rangle & \xrightarrow{\mathsf{cs}}_l & \langle \vec{p} \mid \psi \wedge \varphi \rangle \qquad \text{iff } \mathcal{D} \models \psi \wedge \varphi \\
\langle p(\vec{t}[\vec{x}]), \vec{p} \mid \varphi \rangle & \xrightarrow{\mathsf{res_{cl}}}_l & \langle \vec{q}(\vec{v}[\sigma(\vec{z})]), \vec{p} \mid \varphi \wedge (\vec{u}[\sigma(\vec{y})] = \vec{t}[\vec{x}]) \rangle \qquad \text{iff } \mathcal{D} \models \varphi \wedge (\vec{u}[\sigma(\vec{y})] = \vec{t}[\vec{x}]) \\
& \text{where:} & cl : p(\vec{u}[\vec{y}]) \leftarrow \vec{q}(\vec{v}[\vec{z}]) \qquad \sigma \text{ a renaming apart for } \vec{y}, \vec{z}, \vec{x}
\end{array}$$

We write GPF for general purified form, that is, the form where atoms in the resolvent contain only variables. For a state Q, we write Q' for its GPF form, and for a program P, we write P' for its GPF form as defined in Sec. 3.

Definition 5.2. The *GPF form* of state $\langle \vec{p}(\vec{u}[\vec{x}]) \mid \varphi[\vec{x}] \rangle$ is $\langle \vec{p}(\vec{x}') \mid \varphi[\vec{x}] \wedge \vec{x}' = \vec{u}[\vec{x}] \rangle$, with $\vec{x} = x_1, \ldots, x_m$, $k = |\vec{u}|$, and $\vec{x}' = x_{m+1}, \ldots, x_{m+k}$.

Lemma 5.3. Let φ be the constraint store of a state Q, and φ' the store of Q'. Then, $\mathcal{D} \models \varphi$ iff $\mathcal{D} \models \varphi'$.

Lem. 5.3 A consequence of soundness. Take a formula $\exists \vec{x}.\varphi$, then, for $\vec{x}'$ fresh, and any sequence of terms $\vec{t}$ from $\mathcal{T}_\Sigma(\mathcal{X})$, $\exists \vec{x}.\varphi \iff \exists \vec{x}.\varphi \wedge \vec{t} = \vec{t} \iff (\exists \vec{x}.\varphi \wedge \vec{x}' = \vec{t})[\vec{x}'/\vec{t}] \iff \exists \vec{x}\vec{x}'.\varphi \wedge \vec{x}' = \vec{t}$. $\square$

Definition 5.4. We define an equivalence relation $\approx_{\mathcal{D}}$ on states:

$$\langle \vec{p}(\vec{t}[\vec{x}_1]) \mid \psi_1[\vec{x}_1] \rangle \approx_{\mathcal{D}} \langle \vec{p}(\vec{t}[\vec{x}_2]) \mid \psi_2[\vec{x}_2] \rangle \quad \text{iff} \quad \mathcal{D} \models \exists \vec{x}_1 \ \psi_1[\vec{x}_1] \iff \mathcal{D} \models \exists \vec{x}_2 \ \psi_2[\vec{x}_2]$$

Lemma 5.5. Let $Q_1 \approx_{\mathcal{D}} R_1$. $Q_1 \to_l Q_2$ iff for some state R_2, $R_1 \to_l R_2$ and $Q_2 \approx_{\mathcal{D}} R_2$.

Lem. 5.5 Immediate consequence of the soundness of the constraint solver. The same resolvent guarantees that the choice of every step is identical. Then, for every step, either a resolution or a constraint one, we have $\psi_1 \iff \psi_2$, thus for a newly added constraint φ arising from either a resolution or a constraint step, it is the case that $\psi_1 \wedge \varphi \iff \psi_2 \wedge \varphi$. □

Lemma 5.6 (GPF Equivalence). For a state Q_1 and its GPF form Q_1':

- A derivation $Q_1 \xrightarrow{\mathsf{cs}}_l Q_2$ exists iff $Q_1' \xrightarrow{\mathsf{cs}}_l C_2$ does and $C_2 \approx_{\mathcal{D}} Q_2'$.
- A derivation $Q_1 \xrightarrow{\mathsf{res}}_l Q_2$ exists iff $Q_1' \xrightarrow{\mathsf{res}}_l C_1 \xrightarrow{\mathsf{cs}}_l C_2$ does and $C_2 \approx_{\mathcal{D}} Q_2'$.

Lem. 5.6 We annotate the number of variables in use in each constraint store to make the proof more readable. Let $|\vec{x}| = m$, $|\vec{u}| = |\vec{x}'| = k$. Recall that $\vec{x}' = x_{m+1}, \ldots, x_{m+k}$ then

$$\begin{array}{lcll} Q_1 & = & \langle \vec{p}(\vec{u}[\vec{x}]) \mid \varphi[\vec{x}] \rangle & \text{[m]} \\ Q_1' & = & \langle \vec{p}(\vec{x}') \mid \varphi[\vec{x}] \wedge \vec{x}' = \vec{u}[\vec{x}] \rangle & \text{[m+k]} \end{array}$$

We know that $Q_1 \approx_{\mathcal{D}} Q_1'$, so a derivation will always exist for Q_1 iff it exists for Q_1'. Now we check that $Q_2' \approx C_2$.

- If $p_1 \equiv \psi$, then we have $Q_1 \xrightarrow{\mathsf{cs}}_l Q_2$ and $Q_1' \xrightarrow{\mathsf{cs}}_l C_2$. The new states are:

$$\begin{array}{lcll} Q_2 & = & \langle \vec{p}_{|2}(\vec{u}_{|2}[\vec{x}]) \mid \varphi[\vec{x}] \wedge \psi \rangle & \text{[m]} \\ Q_2' & = & \langle \vec{p}_{|2}(\vec{x}') \mid \varphi[\vec{x}] \wedge \psi \wedge \vec{x}' = \vec{u}_{|2}[\vec{x}] \rangle & \text{[m+k]} \\ C_2 & = & \langle \vec{p}_{|2}(\vec{x}') \mid \varphi[\vec{x}] \wedge \psi \wedge \vec{x}' = \vec{u}_{|2}[\vec{x}] \rangle & \text{[m+k]} \end{array}$$

  They are the same identical state given that we do not purify constraints.
- If p_1 is a defined predicate with clause:

$$\begin{array}{lll} cl: & p_1(\vec{t}[\vec{y}]) & \leftarrow \vec{q}(\vec{v}[\vec{y}]) \\ cl': & p_1(\vec{x}_h) & \leftarrow \exists_{h+1}^{m'}.((\vec{x}_h = \vec{t}[\vec{x}_y] \wedge \vec{x}_v = \vec{v}[\vec{x}_y]), \vec{q}(\vec{x}_v)) \end{array}$$

  Let $j = |\vec{y}|$, $\vec{x}_\sigma = x_{m+1}, \ldots, x_{m+j}$, $j' = j_1' + j_2'$, $j_1' = |\vec{v}|$ $j_2' = |\vec{u}_{|2}|$, $\vec{x}_q' = x_{m+j+1}, \ldots, x_{m+j+j_1'}$, $\vec{x}_p' = x_{m+j+j_1'+1}, \ldots, x_{m+j+j'}$. The states Q_2 and Q_2' arising from the derivation rules are:

$$\begin{array}{lcll} Q_2 & = & \langle \vec{q}(\vec{v}[\vec{x}_\sigma]), \vec{p}_{|2}(\vec{u}_{|2}[\vec{x}]) \mid \varphi[\vec{x}] \wedge \vec{u}_1[\vec{x}] = \vec{t}[\vec{x}_\sigma] \rangle & \text{[m+j]} \\ Q_2' & = & \langle \vec{q}(\vec{x}_q'')), \vec{p}_{|2}(\vec{x}_p'') \mid \varphi[\vec{x}] \wedge \vec{u}_1[\vec{x}] = \vec{t}[\vec{x}_\sigma] \wedge \vec{x}_q' = \vec{v}[\vec{x}_\sigma] \wedge \vec{x}_p' = \vec{u}_{|2}[\vec{x}] \rangle & \text{[m+j+j']} \end{array}$$

  Let $\vec{x}_{h\sigma}$, etc…, be the $m+k$ shifted vectors of variables arising from renaming them apart from variables in Q_1'. The states C_1, C_2 are:

$$\begin{array}{lcll} C_1 & = & \langle (\vec{x}_{h\sigma} = \vec{t}[\vec{x}_{y\sigma}] \wedge \vec{x}_{v\sigma} = \vec{v}[\vec{x}_{y\sigma}]), \vec{q}(\vec{x}_{v\sigma}), \vec{p}_{|2}(\vec{x}'_{|2}) & \\ & & \mid \varphi[\vec{x}] \wedge \vec{x}' = \vec{u}[\vec{x}] \wedge \vec{x}_{h\sigma} = \vec{x}_1' \rangle & \text{[m+k+m']} \\ C_2 & = & \langle \vec{q}(\vec{x}_{v\sigma}), \vec{p}_{|2}(\vec{x}'_{|2}) & \\ & & \mid \varphi[\vec{x}] \wedge \vec{x}' = \vec{u}[\vec{x}] \wedge \vec{x}_{h\sigma} = \vec{x}_1' \wedge \vec{x}_{h\sigma} = \vec{t}[\vec{x}_{y\sigma}] \wedge \vec{x}_{v\sigma} = \vec{v}[\vec{x}_{y\sigma}] \rangle & \text{[m+k+m']} \end{array}$$

  We will apply vector splitting and variable renaming to go from the constraint store of C_2 to the one belonging to Q_2'. We omit the number of variables used but the reader can easily check that the elimination preserves it.

$$\begin{array}{ll} \varphi[\vec{x}] \wedge \vec{x}' = \vec{u}[\vec{x}] \wedge \vec{x}_{h\sigma} = \vec{x}_1' \wedge \vec{x}_{h\sigma} = \vec{t}[\vec{x}_{y\sigma}] \wedge \vec{x}_{v\sigma} = \vec{v}[\vec{x}_{y\sigma}] & \Leftrightarrow \\ \qquad \{\vec{x}' = \vec{x}_1'\vec{x}'_{|2}, \vec{u}[\vec{x}] = \vec{u}_1[\vec{x}]\vec{u}_{|2}[\vec{x}]\} & \\ \varphi[\vec{x}] \wedge \vec{x}_1' = \vec{u}_1[\vec{x}] \wedge \vec{x}'_{|2} = \vec{u}_{|2}[\vec{x}] \wedge \vec{x}_{h\sigma} = \vec{x}_1' \wedge \vec{x}_{h\sigma} = \vec{t}[\vec{x}_{y\sigma}] \wedge \vec{x}_{v\sigma} = \vec{v}[\vec{x}_{y\sigma}] & \Leftrightarrow \\ \qquad \{\vec{x}_1', \vec{x}_{h\sigma} \text{ elimination}\} & \\ \varphi[\vec{x}] \wedge \vec{u}_1[\vec{x}] = \vec{t}[\vec{x}_{y\sigma}] \wedge \vec{x}_{v\sigma} = \vec{v}[\vec{x}_{y\sigma}] \wedge \vec{x}'_{|2} = \vec{u}_{|2}[\vec{x}] & \Leftrightarrow \\ \qquad \{\text{renaming}\} & \\ \varphi[\vec{x}] \wedge \vec{u}_1[\vec{x}] = \vec{t}[\vec{x}_\sigma] \wedge \vec{x}_q' = \vec{v}[\vec{x}_\sigma] \wedge \vec{x}_p' = \vec{u}_{|2}[\vec{x}] & \end{array}$$

  by soundness, $C_2 \approx_{\mathcal{D}} Q_2'$.

□

The derivation set of a state in GPF form is in direct correspondence to the original one, and reachable answers coincide up to logical equivalence:

Theorem 5.7. $Q \to_l \ldots \to_l \langle \square \,|\, \varphi \rangle$ iff $Q' \to_l \ldots \to_l \langle \square \,|\, \varphi' \rangle$ and $\langle \square \,|\, \varphi \rangle \approx_{\mathcal{D}} \langle \square \,|\, \varphi' \rangle$.

5.1.1. Call-Return Transition System

A resolution step instantiating a predicate definition requires renaming apart of the fresh logical variables to avoid name clashes, given the default global scope of variables in CLP. Adopting a "logical" point of view, renaming in an instantiation step is equivalent to the problem of rewriting two constraints with local quantifiers to a constraint with a single one: $\exists \vec{x}.\varphi[\vec{z}\vec{x}]$, $\exists \vec{y}.\psi[\vec{z}\vec{y}]$ where $\vec{z}$ is shared. Renaming apart allows to obtain a logically-equivalent new constraint $\exists \vec{x}\vec{y}_\sigma.(\varphi[\vec{z}\vec{x}] \wedge \psi[\vec{z}\vec{y}_\sigma])$ where $\vec{y}_\sigma = \sigma_{\vec{x}}(\vec{y})$ a renaming apart of $\vec{y}$ for $\vec{x}$. However, if we relax the top-level quantifier condition, we can combine the two original constraints in different ways, such as $(\exists \vec{x}.\varphi[\vec{x}]) \wedge (\exists \vec{y}.\psi[\vec{y}])$. In the relational approach, this last form is natural, allowing us to avoid an explicit renaming apart process during proof search. Implementing this choice however effectively requires a carefully chosen canonical naming scheme and the use of the common variables $\vec{z}$ to propagate constraints outside the scope of the quantifiers.

This choice gives rise to a notion of *sub-state* with local variables, which we keep track of using a cut-off index n. Variables $x_1, \ldots, x_n$ are "global", whereas variables greater than n are considered "local" to the particular sub-state. Communication between a sub-state and its parent state requires the use of a permutation of variables that permutes the parent's variables such that the needed global variables appear in the correct index position.

Definition 5.8. The set $\mathcal{CS}$ of *call-return states* is defined inductively as:

- $\langle \vec{p} \,|\, \varphi[\vec{x}] \rangle_n$, where $p_i \equiv p_i(\vec{x}_i)$ is an atom or $p_i \equiv \psi$ a constraint, $\vec{x}_i$ a vector of variables, n a natural number, and $\varphi[\vec{x}]$ a constraint store.
- $\left\langle {}^{\pi} CS, \vec{p} \,|\, \varphi[\vec{x}] \right\rangle_n$, where CS is a call-return state, π is a permutation for sequences of natural numbers, $\vec{p}$ a vector of atoms similar to the previous case, n a natural number and $\varphi[\vec{x}]$ a constraint store.

As written, n captures the number of arguments involved in a predicate call and π is a permutation of variables local to the state; it will be undone upon return. Thanks to the canonical naming scheme, the head of every clause is of the form $p(x_1, \ldots, x_n)$. Then, the *call* transition will appropriately permute the current constraint store so that the right constraints are placed on p's variables. Such manipulations of the constraint store are straightforward but tedious, thus we define notations Δ and ∇ for the manipulations performed at call and return time.

$$\Delta_n^{\pi}(\varphi) \equiv \exists^{n\uparrow}.\pi(\varphi) \qquad \nabla_n^{\pi}(\varphi, \psi) \equiv \psi \wedge \pi^{-1}(\exists^{n\uparrow}.\ \varphi)$$

$\Delta_n^{\pi}(\varphi)$ may be read as "modify φ to be placed in a context with n open variables and permutation π". $\nabla_n^{\pi}(\varphi, \psi)$ may be read as "merge constraint φ with scope (π, n) with ψ". With this notation in place, we can proceed to define a new transition system which logically formalizes a resolution step for a defined predicate; this system could be understood as a formalization of the notion of call-frame present in common Prolog implementations:

Definition 5.9. The *Call-Return transition system* is:

$$\begin{array}{lcll}
\langle \psi, \vec{p} \,|\, \varphi \rangle_n & \xrightarrow{\text{constraint}}_{cr} & \langle \vec{p} \,|\, \varphi \wedge \psi \rangle_n & \boxed{\text{if } p_1 \equiv \psi \text{ and } \varphi \wedge \psi \text{ satisfiable}} \\
\langle p(\vec{x}_1), \vec{p} \,|\, \varphi \rangle_n & \xrightarrow{\text{call/cl}_i}_{cr} & \left\langle {}^{\pi} \langle \vec{q} \,|\, \Delta_h^{\pi}(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n & \boxed{\text{if } cl_i : p(\vec{x}_h) \leftarrow \exists^{h\uparrow}.\vec{q} \in P' \text{ and } \pi(\vec{x}_1) = \vec{x}_h} \\
\left\langle {}^{\pi} \langle \square \,|\, \psi \rangle_m, \vec{p} \,|\, \varphi \right\rangle_n & \xrightarrow{\text{return}}_{cr} & \langle \vec{p} \,|\, \nabla_m^{\pi}(\psi, \varphi) \rangle_n & \\
\left\langle {}^{\pi} PS, \vec{p} \,|\, \varphi \right\rangle_n & \xrightarrow{\text{sub}}_{cr} & \left\langle {}^{\pi} PS', \vec{p} \,|\, \varphi \right\rangle_n & \boxed{\text{if } PS \neq \langle \square \,|\, \varphi' \rangle_h, \text{ and } PS \to_p PS'}
\end{array}$$

The call-return transition system is equivalent to the standard one.

Lemma 5.10. Write $\vec{p}_{|2}$ for the subsequence of $\vec{p}$ starting at the second element. Then, given a GPF state $Q_1 = \langle p_1(\vec{x}_1), \vec{p}_{|2}(\vec{x}) \,|\, \varphi \rangle_n$ and program P:

$$\begin{array}{ll}
 & Q_1 \to_l \ldots \to_l \langle \vec{p}_{|2}(\vec{x}) \,|\, \varphi' \rangle_n \iff Q_1 \to_{cr} \ldots \to_{cr} \langle \vec{p}_{|2}(\vec{x}) \,|\, \varphi'' \rangle_n \\
\text{with} & \langle \vec{p}_{|2}(\vec{x}) \,|\, \varphi' \rangle_n \approx_{\mathcal{D}} \langle \vec{p}_{|2}(\vec{x}) \,|\, \varphi'' \rangle_n
\end{array}$$

which implies:

$$Q_1 \to_l \ldots \to_l \langle \Box \mid \varphi' \rangle_n \iff Q_1 \to_{cr} \ldots \to_{cr} \langle \Box \mid \varphi'' \rangle_n \quad \varphi' \iff \varphi''$$

Lem. 5.10 By induction over the length of the first derivation, and unfolding the sub-state transitions.

Base Case: A derivation of length 1, may correspond to either a constraint step or an *empty* clause $p_1(\vec{x}_h) \leftarrow .$

- If p is a constraint, the proof is immediate as the constraint transition is the same in both systems.
- If p is a defined predicate with empty clause, the proof is direct, the transition for the first system is:

  $$\langle p(\vec{x}_p), \vec{p}(\vec{x}) \mid \varphi \rangle_n \xrightarrow{\mathsf{res}}_l \langle \vec{p}(\vec{x}) \mid \varphi \wedge \vec{x}_{h\sigma} = \vec{x}_p \rangle_n$$

  and for the call-return one is:

  $$\langle p(\vec{x}_p), \vec{p}(\vec{x}) \mid \varphi \rangle_n \xrightarrow{\mathsf{call}}_{cr} \left\langle {}^{\pi}\langle \Box \mid \exists^{h\uparrow}.\pi(\varphi) \rangle_h, \vec{p} \mid \varphi \right\rangle_n \xrightarrow{\mathsf{return}}_{cr} \langle \vec{p} \mid \varphi \wedge \pi^{-1}(\exists^{h\uparrow}.\exists^{h\uparrow}.\pi(\varphi)) \rangle_n$$

  $\varphi \wedge \pi^{-1}(\exists^{h\uparrow}.\exists^{h\uparrow}.\pi(\varphi)) \Leftrightarrow \varphi$ and $(\varphi \wedge \vec{x}_{h\sigma} = \vec{x}_p) \Leftrightarrow \varphi$, completing the proof.

Inductive Case: In inductive case, we necessarily have p_1 a defined predicate with a non-empty clause:

$p_1(\vec{x}_h) \leftarrow \exists^n_m.\vec{q}(\vec{x}')$.

Note that the $\vec{x}$ occurring in the states and in the clause are different, we will use $\vec{x}'$ for the one coming from the clause, but it is also a sequence $x_1, \ldots, x_n$. $\vec{x}$ and $\vec{x}'$ only differ in length. We have a derivation of length $i+1$. The derivations for both transition systems are:

$$\langle \vec{p}(\vec{x}) \mid \varphi[\vec{x}] \rangle \to_r \langle \vec{q}(\vec{x}_\sigma), \vec{p}_{|2}(\vec{x}) \mid \varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma} \rangle \overbrace{\to \cdots \to}^{i} \langle \vec{p}_{|2}(\vec{x}) \mid \varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma} \wedge \varphi'[\vec{x}_\sigma] \rangle$$

$$\langle \vec{p}(\vec{x}) \mid \varphi[\vec{x}] \rangle_m \xrightarrow{\mathsf{call}}_{cr} \left\langle {}^{\pi}\langle \vec{q}(\vec{x}') \mid \exists^{h\uparrow}.\pi(\varphi[\vec{x}]) \rangle_h, \vec{p}_{|2} \mid \varphi[\vec{x}] \right\rangle_m \overbrace{\to \cdots \to}^{i'}$$
$$\left\langle {}^{\pi}\langle \Box \mid \exists^{h\uparrow}.\pi(\varphi[\vec{x}]) \wedge \varphi'[\vec{x}'] \rangle_h, \vec{p}_{|2} \mid \varphi[\vec{x}] \right\rangle_m \xrightarrow{\mathsf{return}}_{cr} \langle \vec{p}_{|2}(\vec{x}) \mid \varphi[\vec{x}] \wedge \pi^{-1}(\exists^{h\uparrow}.(\exists^{h\uparrow}.\pi(\varphi[\vec{x}]) \wedge \varphi'(\vec{x}'))) \rangle_m$$

with $\pi(\vec{x}_1)$. We must be able to apply the induction hypothesis for the derivations of length i and i', which amounts to checking equivalence of the substate with a restricted notion of the second one. Then, we must check logical equivalence of the resulting constraint store after return.

Derivations for the first atom or constraint of a resolvent do not depend on the rest of it:

$$\begin{array}{lll} \langle \vec{p}(\vec{x}) \mid \varphi[\vec{x}] \rangle & \to_l \ldots \to_l \langle \Box, \vec{p}_{|2}(\vec{x}) \mid \varphi[\vec{x}] \wedge \varphi'[\vec{x}_1] \rangle & \text{iff} \\ \langle p_1(\vec{x}_1) \mid \varphi[\vec{x}] \rangle & \to_l \ldots \to_l \langle \Box \mid \varphi[\vec{x}] \wedge \varphi'[\vec{x}_1] \rangle & \end{array}$$

Then, we check the equivalence of the two states:

$$\langle \vec{q}(\vec{x}') \mid \exists^{h\uparrow}.\pi(\varphi[\vec{x}]) \rangle \approx_{\mathcal{D}} \langle \vec{q}(\vec{x}_\sigma) \mid \varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma} \rangle$$

Thus, the precise statement needed to prove state equivalence is:

$$\exists \vec{x}'.\exists^{h\uparrow}.\pi(\varphi[\vec{x}]) \iff \exists \vec{x}\vec{x}_\sigma.(\varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma})$$

Let $m = |\vec{x}|$ and $k = |\vec{x}'|$. Thus $\vec{x} = x_1, \ldots, x_m$, $\vec{x}' = x_1, \ldots, x_k$ and $\vec{x}_\sigma = x_{m+1}, \ldots, x_{m+k}$. The captured variables inside the $\exists^{h\uparrow}$ quantifier are $x_{h+1}, \ldots, x_m$. Let $\vec{x}_r = \vec{x}/\vec{x}_1$. Then, $\pi(\vec{x}) = x_1, \ldots, x_h, \pi(\vec{x}_r)$. Renaming apart $\pi(\vec{x}_r)$ to $\vec{x}_{r'} = x_{k+1}, \ldots, x_{k+m}$ we can eliminate the inner quantifier:

$$\exists \vec{x}'\vec{x}_{r'}.\varphi[\vec{x}_h\vec{x}_{r'}] \iff \exists \vec{x}\vec{x}_\sigma.(\varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma})$$

This will match $\vec{x}'$ to $\vec{x}_\sigma$, but $\vec{x}_{r'}$ is missing h variables. If we add h new variables $\vec{x}_{h'}$ and add the equation $\vec{x}_{h'} = \vec{x}_h$ we get the desired equivalence:

$$\exists \vec{x}'\vec{x}_{r'}\vec{x}_{h'}.(\varphi[\vec{x}_h\vec{x}_{r'}] \wedge \vec{x}_{h'} = \vec{x}_h) \iff \exists \vec{x}\vec{x}_\sigma.(\varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma})$$

We apply the induction hypothesis. Actually, we are applying induction as many times as elements or

constraints $\vec{q}$ has. We could recast this lemma to make this fact more explicit:

$$\langle \vec{p}(\vec{x}) \,|\, \varphi \rangle \overbrace{\rightarrow \cdots \rightarrow}^{i} \langle \square, \vec{p}_{|2}(\vec{x}) \,|\, \varphi' \rangle \overbrace{\rightarrow \cdots \rightarrow}^{j} \langle \square \,|\, \varphi'' \rangle$$

but we think the current presentation is clearer.

After applying the induction hypothesis, the following equivalence remains to be proven:

$$\exists \vec{x} \vec{x}_\sigma . (\varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma} \wedge \varphi'[\vec{x}_\sigma]) \iff \exists \vec{x} . (\varphi[\vec{x}] \wedge \pi^{-1}(\exists^{h\uparrow} . (\exists^{h\uparrow} . \pi(\varphi[\vec{x}]) \wedge \varphi'(\vec{x}'))))$$

We focus on the formula on the right. Similarly to the previous case, we apply the permutation using the knowledge of the variables involved:

$$\exists \vec{x} . (\varphi[\vec{x}] \wedge \exists \vec{x}' / \vec{x}_1 . (\varphi'(\pi^{-1}(\vec{x}')) \wedge \exists \vec{x}' / \vec{x}_1 (\varphi[\vec{x}])))$$

Renaming apart $\vec{x}'$ and adding the new variables needed with their corresponding equations, we get:

$$\exists \vec{x} \vec{x}_\sigma . (\varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma} \wedge \varphi'[\vec{x}_\sigma] \wedge \exists \vec{x}' / \vec{x}_1 (\varphi[\vec{x}]))$$

which is clearly equivalent to:

$$\exists \vec{x} \vec{x}_\sigma . (\varphi[\vec{x}] \wedge \vec{x}_1 = \vec{x}_{h\sigma} \wedge \varphi'[\vec{x}_\sigma])$$

This concludes the proof. □

5.1.2. Folding of SLD derivations

We now extend our notion of *call-return state* to internalize disjunction in the proof-search tree. We fold the set of possible derivations from a CS state into a single one between *resolution states*, an extension of our previous states with a parallel constructor $(PS_1 \blacksquare PS_2)$. Making failure explicit is necessary, so we also introduce a new $\langle fail \rangle$ state. The left-bias of SLD resolution needs additional treatment, thus we split the previous *resolution transition* in two: clause selection and parameter passing. *Resolution states* capture the theory of naming and proof search of constraint logic programming except recursion, which operates meta-logically by grafting predicate symbols onto their definitions.

Definition 5.11. The set $\mathcal{PS}$ of *resolution states* is inductively defined as:

- $\langle fail \rangle$.
- $\langle \vec{p} \,|\, \varphi \rangle_n$, where $p_i \equiv P_i(\vec{x}_i)$ is an atom, or a constraint $p_i \equiv \psi$, $\vec{x}_i$ a vector of variables, φ a constraint store and n a natural number.
- $\left\langle {}^{\pi} PS, \vec{p} \,|\, \varphi \right\rangle_n$, where PS is a resolution state, and π a permutation.
- $\left\langle {}^{\pi}\blacktriangleright PS, \vec{p} \,|\, \varphi \right\rangle_n$, the "select state". It represents the state just before selecting a clause to proceed with proof search.
- $(PS_1 \blacksquare PS_2)$. Parallel composition: captures choice in the proof search tree.

Definition 5.12. The *resolution transition system* $\rightarrow_P \subseteq (\mathcal{PS} \times \mathcal{PS})$ is shown in Fig. 7.

The two first transitions deal with the case where a constraint is first in the resolvent, failing or adding it to the constraint store in case it is satisfiable. When the head of the resolvent is a defined predicate, the *call* transition will replace it by its definition, properly encapsulated by a select state equipped with the permutation capturing argument order.

The *select* transition performs two tasks: first, it modifies the current constraint store adding the appropriate permutation and scoping (n, π); second, it selects the first clause for proof search. The *return* transitions will either propagate failure or undo the permutation and scoping performed at call time. *sub*, *backtrack*, and *seq* are structural transitions with a straightforward interpretation from a proof search perspective.

The main property of the system is the internalization of the SLD search strategy:

Lemma 5.13 (Clause Selection). Suppose we are given a set of clauses:

$$cl_1 : p(\vec{x}_h) \leftarrow \exists^{h\uparrow} . \vec{q} \quad cl_2 : p(\vec{x}_h) \leftarrow \exists^{h\uparrow} . \vec{r}$$

and a state $\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle$. The derivation set using the call-return system is:

$$\left\{ \begin{array}{l} \langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n \xrightarrow{\mathsf{call/cl_1}}_{cr} \left\langle {}^{\pi} \langle \vec{q} \,|\, \Delta_h^{\pi}(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \twoheadrightarrow_{cr} \\ \langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n \xrightarrow{\mathsf{call/cl_2}}_{cr} \left\langle {}^{\pi} \langle \vec{r} \,|\, \Delta_h^{\pi}(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \xrightarrow{\mathsf{constraint}}_{cr} \left\langle {}^{\pi} \langle \vec{r}_{|2} \,|\, r_1 \wedge \Delta_h^{\pi}(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \end{array} \right\}$$

$$
\begin{array}{rcll}
\langle \psi, \vec{p} \,|\, \varphi \rangle_n & \xrightarrow{\mathsf{constraint}}_p & \langle \vec{p} \,|\, \varphi \wedge \psi \rangle_n & \\
\langle \psi, \vec{p} \,|\, \varphi \rangle_n & \xrightarrow{\mathsf{fail}}_p & \langle \mathit{fail} \rangle & \boxed{\text{if } \varphi \wedge \psi \text{ is not satisfiable}} \\
\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n & \xrightarrow{\mathsf{call}}_p & \left\langle {}^{\pi}\blacktriangleright \left(\langle \vec{q}_1 \,|\, \top \rangle_h \,\middle\|\, \ldots \,\middle\|\, \langle \vec{q}_k \,|\, \top \rangle_h\right), \vec{p} \,|\, \varphi \right\rangle_n & \\
 & & & \boxed{\text{if } p(\vec{x}_h) \leftarrow \exists^{h\uparrow}.(\vec{q}_1 \vee \ldots \vee \vec{q}_k) \in P',\ \pi(\vec{x}) = \vec{x}_h} \\
\left\langle {}^{\pi}\blacktriangleright \left(\langle \vec{q} \,|\, \psi \rangle_h \,\middle\|\, PS\right), \vec{p} \,|\, \varphi \right\rangle_n & \xrightarrow{\mathsf{select}}_p & \left(\left\langle {}^{\pi}\langle \vec{q} \,|\, \psi \wedge \Delta^{\pi}_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \,\middle\|\, \left\langle {}^{\pi}\blacktriangleright PS, \vec{p} \,|\, \varphi \right\rangle_n\right) & \\
\left\langle {}^{\pi}\langle \square \,|\, \psi \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n & \xrightarrow{\mathsf{return}}_p & \langle \vec{p} \,|\, \nabla^{\pi}_h(\psi, \varphi) \rangle_n & \\
\left\langle {}^{\pi}\langle \mathit{fail} \rangle, \vec{p} \,|\, \varphi \right\rangle_n & \xrightarrow{\mathsf{return}}_p & \langle \mathit{fail} \rangle & \\
\left\langle {}^{\pi} PS, \vec{p} \,|\, \varphi \right\rangle_n & \xrightarrow{\mathsf{sub}}_p & \left\langle {}^{\pi} PS', \vec{p} \,|\, \varphi \right\rangle_n & \boxed{\text{if } PS \neq \langle \square \,|\, \psi \rangle_n,\ PS \neq \langle \mathit{fail} \rangle, \text{ and } PS \rightarrow_p PS'} \\
(\langle \mathit{fail} \rangle \,\|\, PS) & \xrightarrow{\mathsf{backtrack}}_p & PS & \\
(PS_1 \,\|\, PS_2) & \xrightarrow{\mathsf{seq}}_p & (PS_1' \,\|\, PS_2) & \boxed{\text{if } PS \neq \langle \mathit{fail} \rangle, \text{ and } PS_1 \rightarrow_p PS_1'}
\end{array}
$$

(We omit the case in *select* where the left side has no PS component which happens when the number of clauses for a given predicate is one ($k = 1$))

Figure 7. Resolution Transition System

iff the derivation in the resolution system is:

$$\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n \rightarrow_p \ldots \rightarrow_p \left\langle {}^{\pi}\langle \vec{r}_{|2} \,|\, r_1 \wedge \Delta^{\pi}_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n$$

Lem. 5.13 The derivation set is only possible if $q_1 \wedge \Delta^{\pi}_h(\varphi)$ is not satisfiable. We check the transitions using $\rightarrow_p$ (we label *sub* and *seq* transitions with the actual atomic ones):

$$
\begin{array}{l}
\langle p(\vec{x}) \,|\, \varphi \rangle_n \xrightarrow{\mathsf{call}}_p \left\langle {}^{\pi}\blacktriangleright \left(\langle \vec{q} \,|\, \top \rangle_h \,\middle\|\, \langle \vec{r} \,|\, \top \rangle_h\right), \vec{p} \,|\, \varphi \right\rangle_n \xrightarrow{\mathsf{select}}_p \\
\left(\left\langle {}^{\pi}\langle \vec{q} \,|\, \Delta^{\pi}_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \,\middle\|\, \left\langle {}^{\pi}\blacktriangleright \langle \vec{r} \,|\, \top \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n\right) \xrightarrow{\mathsf{fail}}_p \\
\left(\left\langle {}^{\pi}\langle \mathit{fail} \rangle, \vec{p} \,|\, \varphi \right\rangle \,\middle\|\, \left\langle {}^{\pi}\blacktriangleright \langle \vec{r} \,|\, \top \rangle, \vec{p} \,|\, \varphi \right\rangle\right) \xrightarrow{\mathsf{return}}_p \left(\langle \mathit{fail} \rangle \,\middle\|\, \left\langle {}^{\pi}\blacktriangleright \langle \vec{r} \,|\, \top \rangle, \vec{p} \,|\, \varphi \right\rangle\right) \xrightarrow{\mathsf{backtrack}}_p \\
\left\langle {}^{\pi}\blacktriangleright \langle \vec{r} \,|\, \top \rangle, \vec{p} \,|\, \varphi \right\rangle \xrightarrow{\mathsf{select}}_p \left\langle {}^{\pi}\langle \vec{r} \,|\, \Delta^{\pi}_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \xrightarrow{\mathsf{constraint}}_{cr} \left\langle {}^{\pi}\langle \vec{r}_{|2} \,|\, r_1 \wedge \Delta^{\pi}_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n
\end{array}
$$

If $\rightarrow_p$ could carry out any other transition, the derivation set would be different. □

Lemma 5.14 (Backtracking). For a set of clauses:

$$
\begin{array}{ll}
cl_1 : p(\vec{x}_h) \leftarrow \exists^{h\uparrow}.q(\vec{x}_1), \vec{q} & cl_3 : q(\vec{x}_i) \leftarrow \exists^{i\uparrow}.\vec{s} \\
cl_2 : p(\vec{x}_h) \leftarrow \exists^{h\uparrow}.r(\vec{x}_2), \vec{r} & cl_4 : r(\vec{x}_j) \leftarrow \exists^{j\uparrow}.\vec{t}
\end{array}
$$

and a state $\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle$, the derivation set using the call-return system is:

$$
\left\{
\begin{array}{ll}
\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n \xrightarrow{\mathsf{call/cl}_1}_{cr} & \left\langle {}^{\pi}\langle q(\vec{x}_1), \vec{q} \,|\, \Delta^{\pi}_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \xrightarrow{\mathsf{call/cl}_3}_{cr} \\
 & \left\langle {}^{\pi}\left\langle {}^{\pi_1}\langle \vec{s} \,|\, \Delta^{\pi_1}_i(\Delta^{\pi}_h(\varphi)) \rangle_h, \vec{q} \,|\, \Delta^{\pi}_h(\varphi) \right\rangle, \vec{p} \,|\, \varphi \right\rangle_n \twoheadrightarrow_{cr} \\
\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n \xrightarrow{\mathsf{call/cl}_2}_{cr} & \left\langle {}^{\pi}\langle r(\vec{x}_2), \vec{r} \,|\, \Delta^{\pi}_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \xrightarrow{\mathsf{call/cl}_4}_{cr} \\
 & \left\langle {}^{\pi}\left\langle {}^{\pi_2}\langle \vec{t} \,|\, \Delta^{\pi_2}_j(\Delta^{\pi}_h(\varphi)) \rangle_h, \vec{r} \,|\, \Delta^{\pi}_h(\varphi) \right\rangle, \vec{p} \,|\, \varphi \right\rangle_n \xrightarrow{\mathsf{constraint}}_{cr} \\
 & \left\langle {}^{\pi}\left\langle {}^{\pi_2}\langle \vec{t}_{|2} \,|\, t_1 \wedge \Delta^{\pi_2}_j(\Delta^{\pi}_h(\varphi)) \rangle_h, \vec{r} \,|\, \Delta^{\pi}_h(\varphi) \right\rangle, \vec{p} \,|\, \varphi \right\rangle_n
\end{array}
\right\}
$$

iff the derivation using the resolution system is:

$$\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n \rightarrow_p \ldots \rightarrow_p \left\langle {}^{\pi}\left\langle {}^{\pi_2}\langle \vec{t}_{|2} \,|\, t_1 \wedge \Delta^{\pi_2}_j(\Delta^{\pi}_h(\varphi)) \rangle_h, \vec{r} \,|\, \Delta^{\pi}_h(\varphi) \right\rangle, \vec{p} \,|\, \varphi \right\rangle_n$$

Lem. 5.14 We check the transitions as in the previous lemma.

$$\langle p(\vec{x}), \vec{p} \,|\, \varphi\rangle_n \xrightarrow{\mathsf{call}}_p \left\langle^{\pi}\blacktriangleright (\langle q(\vec{x}_1), \vec{q} \,|\, \top\rangle_h \blacktriangleright \langle r(\vec{x}_2), \vec{r} \,|\, \top\rangle_h), \vec{p} \,|\, \varphi\right\rangle_n \xrightarrow{\mathsf{select}}_p$$
$$\left(\left\langle^{\pi}\langle q(\vec{x}_1), \vec{q} \,|\, \Delta_h^{\pi}(\varphi)\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n \blacktriangleright \left\langle^{\pi}\blacktriangleright \langle r(\vec{x}_2), \vec{r} \,|\, \top\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n\right) \xrightarrow{\mathsf{call}}_p$$
$$\left(\left\langle^{\pi}\left\langle^{\pi_1}\langle \vec{s} \,|\, \Delta_i^{\pi_1}(\Delta_h^{\pi}(\varphi))\rangle_h, \vec{q} \,|\, \Delta_h^{\pi}(\varphi)\right\rangle, \vec{p} \,|\, \varphi\right\rangle_n \blacktriangleright \left\langle^{\pi}\blacktriangleright \langle r(\vec{x}_2), \vec{r} \,|\, \top\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n\right) \xrightarrow{\mathsf{fail}}_p$$
$$\left(\left\langle^{\pi}\left\langle^{\pi_1}\langle \mathit{fail}\rangle, \vec{q} \,|\, \Delta_h^{\pi}(\varphi)\right\rangle, \vec{p} \,|\, \varphi\right\rangle_n \blacktriangleright \left\langle^{\pi}\blacktriangleright \langle r(\vec{x}_2), \vec{r} \,|\, \top\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n\right) \xrightarrow{\mathsf{return}}_p$$
$$\left(\left\langle^{\pi}\langle \mathit{fail}\rangle, \vec{p} \,|\, \varphi\right\rangle_n \blacktriangleright \left\langle^{\pi}\blacktriangleright \langle r(\vec{x}_2), \vec{r} \,|\, \top\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n\right) \xrightarrow{\mathsf{return}}_p \left(\langle \mathit{fail}\rangle \blacktriangleright \left\langle^{\pi}\blacktriangleright \langle r(\vec{x}_2), \vec{r} \,|\, \top\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n\right)$$
$$\xrightarrow{\mathsf{backtrack}}_p \left\langle^{\pi}\blacktriangleright \langle r(\vec{x}_2), \vec{r} \,|\, \top\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n \xrightarrow{\mathsf{select}}_p \left\langle^{\pi}\langle r(\vec{x}_2), \vec{r} \,|\, \Delta_h^{\pi}(\varphi)\rangle_h, \vec{p} \,|\, \varphi\right\rangle_n \xrightarrow{\mathsf{call}}_p$$
$$\left\langle^{\pi}\left\langle^{\pi_2}\langle \vec{t} \,|\, \Delta_j^{\pi_2}(\Delta_h^{\pi}(\varphi))\rangle_h, \vec{r} \,|\, \Delta_h^{\pi}(\varphi)\right\rangle, \vec{p} \,|\, \varphi\right\rangle_n \xrightarrow{\mathsf{constraint}}_p$$
$$\left\langle^{\pi}\left\langle^{\pi_2}\langle \vec{t}_{|2} \,|\, t_1 \wedge \Delta_j^{\pi_2}(\Delta_h^{\pi}(\varphi))\rangle_h, \vec{r} \,|\, \Delta_h^{\pi}(\varphi)\right\rangle, \vec{p} \,|\, \varphi\right\rangle_n$$

□

Lemma 5.15. For all queries $\langle \vec{p} \,|\, \varphi\rangle_n$, the first $\rightarrow_{cr}$ successful derivation using a SLD strategy uniquely corresponds to a $\rightarrow_p$ derivation:

$$\langle \vec{p} \,|\, \varphi\rangle_n \rightarrow_{cr} \ldots \rightarrow_{cr} \langle \square \,|\, \varphi'\rangle_n \quad \Longleftrightarrow \quad \langle \vec{p} \,|\, \varphi\rangle_n \rightarrow_p \ldots \rightarrow_p (\langle \square \,|\, \varphi'\rangle_n \blacktriangleright PS)$$

Proof. By induction over the length of the successful derivation, repeatedly applying Lem. 5.13 and Lem. 5.14. □

Theorem 5.16. The transition systems of Def. 5.1 and Fig. 7 are answer-equivalent: for any query they return the same answer constraint.

Thm. 5.16 The standard transition system is equivalent to the call-return system by Lem. 5.6 and Lem. 5.10. The call-return system is equivalent to the resolution transition system by Cor. 5.15. □

5.2. Relational Operational Semantics for SLD-resolution

Thm. 5.16 provides a convenient formal framework for the algebraic study of SLD, however, the rewriting system in Sec. 4.2 is too fine-grained to be directly related to the resolution transition system. In order to overcome this problem, we introduce a transition system over carefully chosen relations, which can in turn be related to the rewriting system. We use the helper notation $\overrightarrow{W(\overline{p})_{\cap}} \equiv R_1 \cap \ldots \cap R_n$, where $R_i \equiv \dot{K}(\varphi_i)$ or $R_i \equiv W_{\pi_i}(\overline{p}_i)$.

Definition 5.17. The set $\mathcal{RS}$ of *relational states* is inductively defined as:

- $\mathbf{0}$, failure.
- $I_n(\dot{K}(\varphi) \cap \overrightarrow{W(\overline{p})_{\cap}})$, base query, n, φ, $\overline{p}$ parameters.
- $I_n(W_{\pi}^{\circ}(\dot{K}(\varphi) \cap RS) \cap \overrightarrow{W(\overline{p})_{\cap}})$, selection, n, π, φ, $\overline{p}$, RS parameters.
- $I_n(W_{\pi}^{\circ}(RS \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_{\cap}})$, subquery, n, π, φ, $\overline{p}$, RS parameters.
- $(RS_1 \cup RS_2)$, parallel, RS_1, RS_2 parameters.

Recall that a predicate p is translated to an equation $\overline{p} \stackrel{\circ}{=} \Theta_1 \cup \cdots \cup \Theta_k$, and $\Theta_i = I_{\alpha(p)}(\dot{K}(\varphi_i) \cap \overrightarrow{W(\overline{q})_{\cap}})$.

Definition 5.18. The *transition system for relational states* is defined by the rules of Fig. 8.

5.3. The Equivalence

We define an isomorphism between logical and relational states. It is straightforward to check that both transition systems are equivalent, that is to say, the isomorphism is a simulation between them. Then, we check that the rewriting system of Sec. 4.2 implements the relational transition system, which gives a proof of completeness.

$$
\begin{array}{lcll}
I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathsf{constraint}}_r & I_n(\dot{K}(\varphi \wedge \psi) \cap \overrightarrow{W(\overline{p})_\cap}) & \boxed{\text{if } \varphi \wedge \psi \text{ satisfiable}} \\
I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathit{fail}}_r & \mathbf{0} & \boxed{\text{if } \varphi \wedge \psi \text{ not satisfiable}} \\
I_n((\dot{K}(\varphi) \cap W_\pi(\overline{p})) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathsf{call}}_r & I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) & \boxed{\text{with } \overline{p} \stackrel{\circ}{=} \Theta} \\
I_n(W^\circ_\pi(\dot{K}(\varphi) \cap (\Theta_1 \cup \Theta)) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathsf{select}}_r & I_n(W^\circ_\pi(\Theta'_1 \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \cup & \\
 & & I_n(W^\circ_\pi(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) & \\
 & & \boxed{\Theta_1 \equiv I_m(\overrightarrow{W(\overline{q})_\cap})} \quad \boxed{\Theta'_1 \equiv I_m(\dot{K}(\exists^{m\uparrow}.\varphi) \cap \overrightarrow{W(\overline{q})_\cap})} & \\
I_n(W^\circ_\pi(I_m(\dot{K}(\varphi_1)) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathsf{return}}_r & I_n(\dot{K}(\pi^{\text{-}1}(\varphi \wedge (\exists^{m\uparrow}.\ \varphi_1))) \cap \overrightarrow{W(\overline{p})_\cap}) & \\
I_n(W^\circ_\pi(\mathbf{0} \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathsf{return}}_r & \mathbf{0} & \\
I_n(W^\circ_\pi(RS \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathsf{sub}}_r & I_n(W^\circ_\pi(RS' \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \boxed{\text{if } RS \to_r RS'} \\
(\mathbf{0} \cup RS) & \xrightarrow{\mathsf{backtrack}}_r & RS & \\
(RS_1 \cup RS_2) & \xrightarrow{\mathsf{seq}}_r & (RS'_1 \cup RS_2) & \boxed{\text{if } RS_1 \to_r RS'_1}
\end{array}
$$

Figure 8. Relational Transition Rules

Definition 5.19. We define functions $\mathsf{R} : \mathcal{PS} \to \mathcal{RS}$ and $\mathsf{R}^{\text{-}1} : \mathcal{RS} \to \mathcal{PS}$ by induction over the structure of the states:

$$
\begin{array}{ll}
\mathsf{R}(\langle \mathit{fail} \rangle) & = \mathbf{0} \\
\mathsf{R}(\langle \vec{p} \,|\, \varphi \rangle_n) & = I_n(\dot{K}(\varphi) \cap \overrightarrow{W(\overline{p})_\cap}) \\
\mathsf{R}(\left\langle {}^{\pi}\blacktriangleright PS, \vec{p} \,|\, \varphi \right\rangle_n) & = I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \mathsf{R}(PS)) \cap \overrightarrow{W(\overline{p})_\cap}) \\
\mathsf{R}(\left\langle {}^{\pi} PS, \vec{p} \,|\, \varphi \right\rangle_n) & = I_n(W^\circ_\pi(\mathsf{R}(PS) \cap \dot{K}(\pi(\varphi))) \cap \overrightarrow{W(\overline{p})_\cap}) \\
\mathsf{R}((PS_1 \blacktriangleright PS_2)) & = (\mathsf{R}(PS_1) \cup \mathsf{R}(PS_2))
\end{array}
$$

$\vec{p}$ is in purified form, so each element p_i of $\vec{p}$ corresponds to a relational term $\dot{K}(\varphi_i)$ or $W_{\pi_i}(\overline{P_i})$. $\mathsf{R}^{\text{-}1}$ is defined as:

$$
\begin{array}{ll}
\mathsf{R}^{\text{-}1}(\mathbf{0}) & = \langle \mathit{fail} \rangle \\
\mathsf{R}^{\text{-}1}(I_n(\dot{K}(\varphi) \cap \overrightarrow{W(\overline{p})_\cap})) & = \langle \vec{p} \,|\, \varphi \rangle_n \\
\mathsf{R}^{\text{-}1}(I_n(W^\circ_\pi(\dot{K}(\varphi) \cap RS) \cap \overrightarrow{W(\overline{p})_\cap})) & = \left\langle {}^{\pi}\blacktriangleright \mathsf{R}^{\text{-}1}(RS), \vec{p} \,|\, \pi^{\text{-}1}(\varphi) \right\rangle_n \\
\mathsf{R}^{\text{-}1}(I_n(W^\circ_\pi(RS \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap})) & = \left\langle {}^{\pi}\mathsf{R}^{\text{-}1}(RS), \vec{p} \,|\, \pi^{\text{-}1}(\varphi) \right\rangle_n \\
\mathsf{R}^{\text{-}1}((RS_1 \cup RS_2)) & = \left(\mathsf{R}^{\text{-}1}(RS_1) \blacktriangleright \mathsf{R}^{\text{-}1}(RS_2)\right)
\end{array}
$$

Note that R is an extension of the translation function of Sec. 4.2, and an isomorphism.

Lemma 5.20. R is an isomorphism.

Lem. 5.20 By induction over the structure of the states. □

Lemma 5.21. *R is a simulation between the resolution transition system of Fig. 7 and the relational transition system of Fig. 8.*

Lem. 5.21 We check that the relation $R \subseteq (\mathcal{PS} \times \mathcal{RS})$ induced by the isomorphism R is a simulation:

$$\forall RS, PS.\ (PS, RS) \in R \Rightarrow ((PS \to_p PS' \iff RS \to_r RS') \wedge (PS', RS') \in R)$$

Given that R is a bijective map and that the transition systems are deterministic, any of

$$PS \to_p PS' \quad \Rightarrow \quad \mathsf{R}(PS) \to_r \mathsf{R}(PS') \qquad\qquad RS \to_r RS' \quad \Rightarrow \quad \mathsf{R}^{\text{-}1}(PS) \to_p \mathsf{R}^{\text{-}1}(PS')$$

implies that R is a simulation. We check the non-obvious transitions constraint, fail, call, and select.In order to help the reader, we show both transitions, then perform the check outlined above.

• *constraint*:

$$
\begin{array}{lcl}
\langle \psi, \vec{p} \,|\, \varphi \rangle_n & \xrightarrow{\mathsf{constraint}}_p & \langle \vec{p} \,|\, \varphi \wedge \psi \rangle_n \\
I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{\mathsf{constraint}}_r & I_n(\dot{K}(\varphi \wedge \psi) \cap \overrightarrow{W(\overline{p})_\cap})
\end{array}
$$

and the corresponding check:

$$\mathsf{R}(\langle \psi, \vec{p} \,|\, \varphi \rangle_n) = I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{constraint}}_r$$
$$I_n(\dot{K}(\varphi \wedge \psi) \cap \overrightarrow{W(\overline{p})_\cap}) = \mathsf{R}(\langle \vec{p} \,|\, \varphi \wedge \psi \rangle_n)$$

• *fail*:

$$\langle \psi, \vec{p} \,|\, \varphi \rangle_n \xrightarrow{\mathsf{fail}}_p \langle \mathit{fail} \rangle \quad \boxed{\text{if } \varphi \wedge \psi \text{ is not satisfiable}}$$
$$I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{fail}}_r \mathbf{0}$$

the simulation check is:

$$\mathsf{R}(\langle \psi, \vec{p} \,|\, \varphi \rangle_n) = I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{fail}}_r \mathbf{0} = \mathsf{R}(\langle \mathit{fail} \rangle)$$

• *call*:

$$\langle p(\vec{x}), \vec{p} \,|\, \varphi \rangle_n \xrightarrow{\mathsf{call}}_p \left\langle {}^{\pi}\blacktriangleright (\langle \vec{q}_1 \,|\, \top \rangle_h \blacktriangleright \ldots \blacktriangleright \langle \vec{q}_k \,|\, \top \rangle_h), \vec{p} \,|\, \varphi \right\rangle_n$$
$$\boxed{\text{if } p(\vec{x}_h) \leftarrow \exists^{h\uparrow}.(\vec{q}_1 \vee \ldots \vee \vec{q}_k) \in P',\ \pi(\vec{x}) = \vec{x}_h}$$
$$I_n((\dot{K}(\varphi) \cap W_\pi(\overline{p})) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{call}}_r I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \boxed{\text{with } \overline{p} = \Theta}$$

$\mathsf{R}(\langle \vec{q}_i \,|\, \top \rangle_h) = I_h(\overrightarrow{W(\overline{q_i})_\cap}) \equiv \Theta_i$, thus $\mathsf{R}((\langle \vec{q}_1 \,|\, \top \rangle_h \blacktriangleright \ldots \blacktriangleright \langle \vec{q}_k \,|\, \top \rangle_h)) = \Theta_1 \cup \cdots \cup \Theta_k \equiv \Theta$. The check is:

$$\mathsf{R}(\langle \vec{p} \,|\, \varphi \rangle_n) = I_n(\dot{K}(\varphi) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{call}}_r$$
$$I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) = \mathsf{R}(\left\langle {}^{\pi}\blacktriangleright (\langle \vec{q}_1 \,|\, \top \rangle_h \blacktriangleright \ldots \blacktriangleright \langle \vec{q}_k \,|\, \top \rangle_h), \vec{p} \,|\, \varphi \right\rangle_n)$$

• *select*:

$$\left\langle {}^{\pi}\blacktriangleright (\langle \vec{q} \,|\, \top \rangle_h \blacktriangleright PS), \vec{p} \,|\, \varphi \right\rangle_n \xrightarrow{\mathsf{select}}_p \left(\left\langle {}^{\pi}\langle \vec{q} \,|\, \Delta^\pi_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \blacktriangleright \left\langle {}^{\pi}\blacktriangleright PS, \vec{p} \,|\, \varphi \right\rangle_n\right)$$
$$I_n(W^\circ_\pi(\dot{K}(\varphi) \cap (\Theta_1 \cup \Theta)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{select}}_r I_n(W^\circ_\pi(\Theta'_1 \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \cup$$
$$I_n(W^\circ_\pi(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap})$$
$$\boxed{\Theta_1 \equiv I_m(\overrightarrow{W(\overline{q})_\cap})} \quad \boxed{\Theta'_1 \equiv I_m(\dot{K}(\exists^{m\uparrow}.\varphi) \cap \overrightarrow{W(\overline{q})_\cap})}$$

The check is:

$$\mathsf{R}(\left\langle {}^{\pi}\blacktriangleright (\langle \vec{q} \,|\, \top \rangle_h \blacktriangleright PS), \vec{p} \,|\, \varphi \right\rangle_n) = I_n(W^\circ_\pi(\dot{K}(\varphi) \cap (I_h(\overrightarrow{W(\overline{q})_\cap}) \cup \mathsf{R}(PS))) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{select}}_p$$
$$I_n(W^\circ_\pi(I_h(\dot{K}(\exists^{m\uparrow}.\varphi) \cap \overrightarrow{W(\overline{q})_\cap}) \cap \dot{K}(\pi(\varphi))) \cap \overrightarrow{W(\overline{p})_\cap}) \cup$$
$$I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \mathsf{R}(PS)) \cap \overrightarrow{W(\overline{p})_\cap}) = \mathsf{R}(\left(\left\langle {}^{\pi}\langle \vec{q} \,|\, \Delta^\pi_h(\varphi) \rangle_h, \vec{p} \,|\, \varphi \right\rangle_n \blacktriangleright \left\langle {}^{\pi}\blacktriangleright PS, \vec{p} \,|\, \varphi \right\rangle_n\right))$$

□

The last step in the equivalence proof is to check that the transition relation is properly embedded into the rewriting relation.

Lemma 5.22. The relational transition system of Fig. 8 is implemented by the rewriting system of Fig. 6 That is to say, for every transition $(r_1, r_2) \in (\rightarrow_r)$,

$$\exists n.(r_1, r_2) \in (\overset{P}{\longmapsto})^n \wedge \forall r_3.(r_1, r_3) \in (\overset{P}{\longmapsto})^n \Rightarrow r_2 = r_3$$

Lem. 5.22 Given that our rewriting system is locally confluent, we can easily check that the transition system is just a collapsing of a particular rewriting chain, omitting uninteresting states. We show a few relevant transitions: constraint, fail, call, and return.

• *constraint*:

$$I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{constraint}}_r I_n(\dot{K}(\varphi \wedge \psi) \cap \overrightarrow{W(\overline{p})_\cap})$$

This transition is implemented by the rewriting rule m_3.

• *fail*:

$$I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{\mathsf{fail}}_r \mathbf{0}$$

This transition is implemented by the rewriting chain:

$$\begin{array}{llll} I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \overset{P}{\longmapsto} (m_3*) & I_n(\mathbf{0} \cap \overrightarrow{W(\overline{p})_\cap}) & \overset{P}{\longmapsto} (p_2) \\ I_n(\mathbf{0}) & \overset{P}{\longmapsto} (m_1*) & \mathbf{0} & \end{array}$$

• *call*:

$$I_n((\dot{K}(\varphi) \cap W_\pi(\overline{p})) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \xrightarrow{\mathsf{call}}_r \quad I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \boxed{\text{with } \overline{p} = \Theta}$$

the transition is implemented by the rewriting chain:

$$\begin{array}{llll} I_n((\dot{K}(\varphi) \cap W_\pi(\overline{p})) \cap \overrightarrow{W(\overline{p})_\cap}) & \overset{P}{\longmapsto} (p_8) & I_n(W_\pi(W^\circ_\pi(\dot{K}(\varphi)) \cap \overline{p}) \cap \overrightarrow{W(\overline{p})_\cap}) & \overset{P}{\longmapsto} (m_2) \\ I_n(W_\pi(\dot{K}(\pi(\varphi)) \cap \overline{p}) \cap \overrightarrow{W(\overline{p})_\cap}) & \overset{P}{\longmapsto} (m_4) & I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) & \end{array}$$

• *return*:

$$I_n(W^\circ_\pi(I_m(\dot{K}(\psi)) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \xrightarrow{\mathsf{return}}_r \quad I_n(\dot{K}(\pi^{\text{-}1}(\varphi \wedge (\exists^{m\uparrow}.\ \psi))) \cap \overrightarrow{W(\overline{p})_\cap})$$

the transition is implemented by the rewriting chain:

$$\begin{array}{llll} I_n(W^\circ_\pi(I_m(\dot{K}(\psi)) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \overset{P}{\longmapsto} (m_1) & I_n(W^\circ_\pi(\dot{K}(\exists^{m\uparrow}.\ \psi) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(p)_\cap} & \overset{P}{\longmapsto} (m_3) \\ I_n(W^\circ_\pi(\dot{K}(\exists^{m\uparrow}.\ \psi \wedge \varphi)) \cap \overrightarrow{W(p)_\cap} & \overset{P}{\longmapsto} (m_2) & I_n(\dot{K}(\pi^{\text{-}1}(\exists^{m\uparrow}.\ \psi \wedge \varphi)) \cap \overrightarrow{W(p)_\cap} & \end{array}$$

• *return* second case:

$$I_n(W^\circ_\pi(\mathbf{0} \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \xrightarrow{\mathsf{return}}_r \quad \mathbf{0}$$

the transition is implemented by the rewriting chain:

$$\begin{array}{llllll} I_n(W^\circ_\pi(\mathbf{0} \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) & \overset{P}{\longmapsto} (p_2) & I_n(W^\circ_\pi(\mathbf{0}) \cap \overrightarrow{W(p)_\cap}) & \overset{P}{\longmapsto} (m_2*) \\ I_n(\mathbf{0} \cap \overrightarrow{W(p)_\cap}) & \overset{P}{\longmapsto} (p_2) & I_n(\mathbf{0}) & \overset{P}{\longmapsto} (m_1*) \\ \mathbf{0} & & & \end{array}$$

The *sub* and *seq* rules are a consequence of the rewriting strategy used. □

Thus, relation rewriting will return an answer constraint $K(\varphi)$ iff SLD resolution reaches a state $\langle \square \mid \varphi' \rangle$ and $\varphi \iff \varphi'$.

Theorem 5.23. The rewriting system simulates SLD-resolution. That is to say, the rewriting system of Fig. 6 implements the transition system of Def. 1.6. Formally, for every transition $(r_1, r_2) \in (\rightarrow_l)^*$,

$$\exists n.(Tr(r_1), Tr(r_2)) \in (\overset{P}{\longmapsto})^n \qquad \text{and} \qquad \forall r_3.(Tr(r_1), r_3) \in (\overset{P}{\longmapsto})^n \Rightarrow Tr(r_2) = r_3$$

Thm. 5.23 By Lem. 5.22 and Theorem 5.16. Indeed, when SLD-resolution diverges, the relational rewriting system does so in the same way. □

6. Related and Future Work

Previous Work: The paper is the continuation of previous work in [BL94, LC98, GALMN11] extended to constraint logic programming, which requires a new translation procedure, operational semantics, and rewriting system.

In particular, the presence of constraints in this paper permits a different translation of the Clark completion of a program and plays a crucial role in the proof of completeness, which was missing in earlier work.

This paper improves substantially on the results of the preliminary conference version of [GALMn15]. We incorporate a new, unified parametric relational theory embodying definitional recursion and constraints, a new *adequacy* theorem comparing the relational semantics of translated programs with the fixed-point semantics of the original programs, and proofs of correctness and completeness for the relational execution mechanism.

Related Work: A number of solutions have been proposed to the syntactic specification problem. There is an extensive literature treating abstract syntax of logic programming (and other programming paradigms) using encodings in higher-order logic and the lambda calculus [PE88], which has been very successful in formalizing the treatment of substitution, unification and renaming of variables, although it provides no special framework for the management and progressive instantiation of logic variables, and no treatment of constraints. Our approach is essentially orthogonal to this, since it relies on the complete elimination of variables, substitution, renaming and, in particular, existentially quantified variables. Our reduction of management of logic variables to variable free rewriting is new, and provides a complete solution to their formal treatment.

An interesting approach to syntax specification is the use of nominal logic [UPG04, CU04] in logic programming, another, the formalization of logic programming in categorical logic [AM89, RB86, KP96, ALM09, FFL03] which provides a mathematical framework for the treatment of variables, as well as for derivations [KP11]. None of the cited work gives a solution that simultaneously includes logic variables, constraints, proof search strategies and rewriting however.

Bellia and Occhiuto [BO93] have defined a new calculus, the C-expression calculus, to eliminate variables in logic programming. We believe our translation into the well-understood and scalable formalism of binary relations is more applicable to extensions of logic programming.

An interesting combinatory, variable-free approach to logic programming has also been developed by Hamfelt, Nilsson and Vitória in [Nil90, HN98, HNV98] based on higher order predicates. Their execution mechanism is not based on rewriting and the authors do not consider constraints as we do here.

Future Work: A complementary approach to this work is the use of category theory, in particular Freyd's theory of *tabular allegories* [FS91] which extends the binary relation calculus to an abstract category of relations providing native facilities for generation of fresh variables and a categorical treatment of monads. A first attempt in this direction has been published by the authors in [GAL12]. It would be interesting to extend the translation to hereditarily Harrop or higher order logic [MNPS91] by using a stronger relational formalism, such as *division* and *power* allegories. Also, the framework would yield important benefits if it was extended to include relation and set constraints explicitly.

We have also started a mechanized proof using the Coq theorem prover, in order to alleviate the symbolic density of some of the proofs. So far, we have modeled the transition systems and proved a few lemmas such as 5.20. See `https://github.com/ejgallego/clprm-coq` for details.

This paper deals with the mathematical definition of combinatorial proof search; we consider that developing a realistic implementation requires considerable effort and is out of scope for this mainly theoretical contribution. However, we have kept as an important priority for the framework to be amenable to practical implementation, and we discuss our thoughts after some preliminary work in that area.

A first prototype implementation written in Haskell showed the feasibility of the relational compilation of logic programs, and the possibility of executing logic programs with different search strategies — depth- and breadth-first. Also, its limitations revealed that term sharing – which was not supported – is a key point regarding performance. Most of the rewriting rules are straightforward and we believe they should pose little problem to mature rewriting engines; the main challenge is posed by the rule for procedure call, which duplicates a term. We plan to investigate how state-of-the-art rewriting engines can deal efficiently with this issue via term-sharing.

7. Conclusion

We have developed a declarative relational framework for the compilation of Constraint Logic programming that eliminates logic variables and gives an algebraic treatment of program syntax. We have proved operational equivalence to the classical approach. We believe our framework is also a step towards the unified formal treatment of the metatheory and compilation and execution of constraint logic programming. Programs can be analyzed, transformed and optimized entirely within this framework. In these two ways, specification and implementation are brought closer together than in the traditional logic programming formalism.

Acknowledgments: The authors want to thank the anonymous reviewers for their suggestions. This research has been partially funded by Comunidad de Madrid grant S2013/ICE-2731 *(N-Greens Software)* and Spanish MINECO grant TIN2012-39391-C04-03 (*StrongSoft*).

References

[ALM09] Gianluca Amato, James Lipton, and Robert McGrail. On the algebraic structure of declarative programming languages. *Theoretical Computer Science*, 410(46):4626 – 4671, 2009. Abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi.

[AM89] Andrea Asperti and Simone Martini. Projections instead of variables: A category theoretic interpretation of logic programs. In *ICLP*, pages 337–352, 1989.

[BL94] Paul Broome and James Lipton. Combinatory logic programming: computing in relation calculi. In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming*, pages 269–285, Cambridge, MA, USA, 1994. MIT Press.

[BO93] Marco Bellia and M. Eugenia Occhiuto. C-expressions: A variable-free calculus for equational logic programming. *Theor. Comput. Sci.*, 107(2):209–252, 1993.

[Cla77] Keith L. Clark. Negation as failure. In Gallaire and Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1977.

[CU04] James Cheney and Christian Urban. Alpha-prolog: A logic programming language with names, binding, and alpha-equivalence, 2004.

[FFL03] Stacy E. Finkelstein, Peter J. Freyd, and James Lipton. A new framework for declarative programming. *Theor. Comput. Sci.*, 300(1-3):91–160, 2003.

[FS91] P.J. Freyd and A. Scedrov. *Categories, Allegories*. North Holland Publishing Company, 1991.

[GAL12] Emilio Jesús Gallego Arias and James Lipton. Logic programming in tabular allegories. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPIcs*, pages 334–347. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[GALMN11] Emilio Jesús Gallego Arias, James Lipton, Julio Mariño, and Pablo Nogueira. First-order unification using variable-free relational algebra. *Logic Journal of IGPL*, 19(6):790–820, 2011.

[GALMn15] Emilio Jesús Gallego Arias, James Lipton, and Julio Mariño. Declarative compilation for constraint logic programming. In Maurizio Proietti and Hirohisa Seki, editors, *Logic-Based Program Synthesis and Transformation*, volume 8981 of *Lecture Notes in Computer Science*, pages 299–316. Springer International Publishing, 2015.

[HN98] Andreas Hamfelt and Jørgen Fischer Nilsson. Inductive synthesis of logic programs by composition of combinatory program schemes. In P. Flener, editor, *LOPSTR'98, 8th. International Workshop on Logic-Based Program Synthesis and Transformation*, volume 1559 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 1998.

[HNV98] A. Hamfelt, J.F. Nilsson, and A. Vitoria. A combinatory form of pure logic programs and its compositional semantics. Technical Report, 1998.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[KP96] Yoshiki Kinoshita and A. John Power. A fibrational semantics for logic programs. In Roy Dyckhoff, Heinrich Herre, and Peter Schroeder-Heister, editors, *ELP*, volume 1050 of *Lecture Notes in Computer Science*, pages 177–191. Springer, 1996.

[KP11] Ekaterina Komendantskaya and John Power. Coalgebraic derivations in logic programming. In Marc Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 352–366. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[LC98] Jim Lipton and Emily Chapman. Some notes on logic programming with a relational machine. In Ali Jaoua, Peter Kempf, and Gunther Schmidt, editors, *Using Relational Methods in Computer Science*, Technical Report Nr. 1998-03, pages 1–34. Fakultät für Informatik, Universität der Bundeswehr München, July 1998.

[Llo84] John W. Lloyd. *Foundations of logic programming*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1-2):125–157, 1991.

[Nil90] J. F. Nilsson. Combinatory logic programming. In *Procs. of the 2nd Workshop on Meta- programming in Logic, K.U. Leuven, Belgium*, 1990.

[PE88] F. Pfenning and C. Elliot. Higher-order abstract syntax. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 199–208, New York, NY, USA, 1988. ACM.

[RB86] David E. Rydeheard and Rod M. Burstall. A categorical unification algorithm. In *Proceedings of a tutorial and workshop on Category theory and computer programming*, pages 493–505, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[TG87] Alfred Tarski and Steven Givant. *A Formalization of Set Theory Without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 1987.

[UPG04] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3):473–497, 2004.