# Declarative Compilation for Constraint Logic Programming

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

[1] University of Pennsylvania and Mines ParisTech, PSL Research University
gallego@cri.ensmp.fr
[2] Wesleyan University jlipton@wesleyan.edu
[3] Universidad Politécnica de Madrid jmarino@fi.upm.es

**Abstract.** We present a new declarative compilation of logic programs with constraints into variable-free relational theories which are then executed by rewriting. This translation provides an algebraic formulation of the abstract syntax of logic programs. Management of logic variables, unification, and renaming apart is completely elided in favor of algebraic manipulation of variable-free relation expressions. We prove the translation is 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 directly programming with logical theories. This is done to achieve *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. Management of logic variables, unification, renaming variables apart and proof search are cumbersome to handle formally. Consequently, it is often the case that the formal definition 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 compilation framework – based on ideas of Tarski [22] and Freyd [9] – 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 convolution. An important class of relation algebras is 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 3 steps. First, for a CLP program $P$ with signature $\Sigma$, 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. Programs that contain logical variables are represented as ground terms in our setting, thus all reasoning and execution is reduced to algebraic equality, allowing the use of rewriting. 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 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.

*Variable Elimination and Relation Composition.* 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:

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

In this carefully chosen example, the elimination of variables and the translation to binary relation symbols is immediate:

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

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. 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 $\cap$, $\cup$.

*Constraint Logic Programming Conventions* We refer the reader to [16] for basic definitions of logic programming over Horn clauses, and [12] for background on the syntax and semantics of constraint logic programming. In this paper we fix a signature $\Sigma$, a set of terms $\mathcal{T}_\Sigma(\mathcal{X})$, and a subset $\mathcal{C}$ of all first-order formulas over $\Sigma$ closed under conjunction and existential quantification to be the set of *constraint formulas* as well as a $\Sigma$-structure $\mathcal{D}$, called the *constraint domain*. Constraint logic programs are sets of Horn clauses. We use vector notation extensively in the paper, to abbreviate Horn clauses with constraints $p \leftarrow q_1, \dots, q_n$, where $p$ is an atomic formula and $q_i$ may be an atomic formula or a constraint. For instance, in our vector notation, a clause is written $p(\boldsymbol{t}[\boldsymbol{x}]) \leftarrow \boldsymbol{q}(\boldsymbol{u}[\boldsymbol{x}, \boldsymbol{y}])$, where the boldface symbols indicate vectors of variables $\boldsymbol{x}, \boldsymbol{y}$, terms $\boldsymbol{t}, \boldsymbol{u}$ (depending on variables $\boldsymbol{x}$, etc...) and predicates $\boldsymbol{q}$ (depending on terms $\boldsymbol{u}$).

## 2 Relation Algebras and Signatures

In this section, we define $\mathbf{QRA}_\Sigma$, a relation algebra in the style of [22,9] formalizing a CLP signature $\Sigma$ and a constraint domain $\mathcal{D}$. We define its language, its equational theory and semantics.

### 2.1 Relational Language and Theory

The relation language $\mathsf{R}_\Sigma$ 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 $\Sigma$, and a set of relation constants for primitive predicates $\mathsf{R}_{\mathcal{CP}}$, 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 $\mathsf{R}_f$ in $\mathsf{R}_\mathcal{F}$. Each predicate symbol $r \in \mathcal{CP}_\Sigma$ defines a constant symbol $\mathsf{r}$ in $\mathsf{R}_{\mathcal{CP}}$. We write $\mathsf{R}_\Sigma$ for the full relation language:

$$\mathsf{R}_\mathcal{C} = \{(a,a) \mid a \in \mathcal{C}_\Sigma\} \quad \mathsf{R}_\mathcal{F} = \{\mathsf{R}_f \mid f \in \mathcal{F}_\Sigma,\} \quad \mathsf{R}_{\mathcal{CP}} = \{\mathsf{r} \mid r \in \mathcal{CP}_\Sigma\}$$
$$\mathsf{R}_{atom} ::= \mathsf{R}_\mathcal{C} \mid \mathsf{R}_\mathcal{F} \mid \mathsf{R}_{\mathcal{CP}} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl$$
$$\mathsf{R}_\Sigma \quad ::= \mathsf{R}_{atom} \mid \mathsf{R}_\Sigma{}^\circ \mid \mathsf{R}_\Sigma \cup \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \cap \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \mathsf{R}_\Sigma$$

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 $R^\circ$ is the inverse of $R$. We write $hd$ and $tl$ for the head and tail relations. The projection of an n-tuple onto its $i$-th element is written $P_i$ and defined as $P_1 = hd, P_2 = tl; hd, \dots, P_n = tl^{n-1}; hd$.

$\mathbf{QRA}_\Sigma$ (Fig. 1) is the standard theory of distributive relation algebras, plus Tarski's quasiprojections [22], and equations axiomatizing the new relations of $\mathsf{R}_\Sigma$. Note that products and their projections are axiomatized in a relational, variable-free manner.

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

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

**Fig. 1. QRA$_\Sigma$**

### 2.2 Semantics

Let $\Sigma$ be a constraint signature and $\mathcal{D}$ a $\Sigma$-structure. Write $t^{\mathcal{D}}$ for the interpretation of a term $t \in \mathcal{T}_\Sigma$. 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 product associating to the right, that is to say, $\langle a_1, \langle a_2, \ldots, \langle a_{n-1}, a_n\rangle \cdots\rangle\rangle$. Furthermore, we assume right-association of products when 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 1.** *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. 2. The function $\alpha$ 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 1.** *Equational reasoning in* **QRA**$_\Sigma$ *is sound for any interpretation:*

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

## 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

$$
\begin{array}{llll}
[\![\mathbf{1}]\!]^{\mathcal{D}^\dagger} & = \mathsf{R}_A & [\![tl]\!]^{\mathcal{D}^\dagger} & = \{(\langle a, b\rangle, b) \mid a, b \in \mathcal{D}^\dagger\} \\
[\![\mathbf{0}]\!]^{\mathcal{D}^\dagger} & = \emptyset & [\![R^\circ]\!]^{\mathcal{D}^\dagger} & = ([\![R]\!]^{\mathcal{D}^\dagger})^\circ \\
[\![id]\!]^{\mathcal{D}^\dagger} & = \{(u, u) \mid u \in \mathcal{D}^\dagger\} & [\![R \cup S]\!]^{\mathcal{D}^\dagger} & = [\![R]\!]^{\mathcal{D}^\dagger} \cup [\![S]\!]^{\mathcal{D}^\dagger} \\
[\![di]\!]^{\mathcal{D}^\dagger} & = \{(u, v) \mid u \neq v\} & [\![R \cap S]\!]^{\mathcal{D}^\dagger} & = [\![R]\!]^{\mathcal{D}^\dagger} \cap [\![S]\!]^{\mathcal{D}^\dagger} \\
[\![hd]\!]^{\mathcal{D}^\dagger} & = \{(\langle a, b\rangle, a) \mid a, b \in \mathcal{D}^\dagger\} & [\![(c, c)]\!]^{\mathcal{D}^\dagger} & = \{(c^{\mathcal{D}}, c^{\mathcal{D}})\} \\
[\![RS]\!]^{\mathcal{D}^\dagger} & = [\![R]\!]^{\mathcal{D}^\dagger}; [\![S]\!]^{\mathcal{D}^\dagger} & &
\end{array}
$$

$$
[\![\mathsf{R}_f]\!]^{\mathcal{D}^\dagger} = \{(x, \boldsymbol{yu}) \mid x = f^{\mathcal{D}}(a_1, \ldots, a_n) \wedge \ \boldsymbol{y} = \langle a_1, \ldots, a_n\rangle, a_i \in \mathcal{D}, \boldsymbol{u} \in \mathcal{D}^\dagger, n = \alpha(f)\}
$$

$$
[\![\mathsf{r}]\!]^{\mathcal{D}^\dagger} = \{(\boldsymbol{xu}, \boldsymbol{xu}) \mid \boldsymbol{x} = \langle a_1, \ldots, a_n\rangle \wedge r^{\mathcal{D}}(a_1, \ldots, a_n), a_i \in \mathcal{D}, \boldsymbol{u} \in \mathcal{D}^\dagger, n = \alpha(r)\}
$$

**Fig. 2.** Standard interpretation of binary relations.

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.

The reader should keep in mind that there are two kinds of predicate symbols in a constraint logic program: *constraint predicates* $r$ which are translated by the function $\dot{K}$ above to relation terms $\mathsf{r}$, and *defined* or program predicates.

We translate defined predicates – and CLP programs – to equations $\overline{p} \doteq R$, 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 2 (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:*

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

*This translation is extended to vectors of terms as follows* $K(\langle t_1, \ldots, t_n\rangle) = \bigcap_{i \leq n} P_i; K(t_i)$.

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 $+; (P_1; P_1^\circ \cap P_2; \mathsf{s}; \mathsf{s}; P_2^\circ)$.

**Lemma 1.** *Let $t[\boldsymbol{x}]$ be a term of $\mathcal{T}_\Sigma(\mathcal{X})$ whose free variables are among those in the sequence $\boldsymbol{x} = x_1, \ldots, x_m$. Then, for any sequences $\boldsymbol{a} = a_1, \ldots, a_m \in \mathcal{D}^\dagger, \boldsymbol{u} \in \mathcal{D}^\dagger$ and any $b \in \mathcal{D}$ we have*

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

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 $\varphi[\boldsymbol{x}]$ over $m$ variables is translated to an $m$-coreflexive relation, the formula $\exists x_i.\ \varphi[\boldsymbol{x}]$ 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.** *The partial identity relation expressions $I_m$, $Q_i$ for $m, i > 0$ are defined as:*

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

*$I_m$ 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 4 (Constraint Translation).** *The $\dot{K} : \mathcal{L}_{\mathcal{D}} \to \mathsf{R}_\Sigma$ translation function for constraint formulas is:*

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

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

$$Q_1; Q_2; (P_1^\circ; \mathsf{s}^\circ; P_1 \cap P_2^\circ; P_2); \le; (P_1; \mathsf{s}; P_1^\circ \cap P_2; P_2^\circ); Q_1; Q_2$$

**Lemma 2.** *Let $\varphi[\boldsymbol{x}]$ be a constraint formula with free variables among $\boldsymbol{x} = x_1, \ldots, x_m$. Then, for any sequences $\boldsymbol{a} = a_1, \ldots, a_m$, $\boldsymbol{u}$ and $\boldsymbol{u}'$ of members of $\mathcal{D}$*

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

### 3.2 Translation of Constraint Logic Programs

To motivate the technical definitions below, we illustrate the program translation procedure with an example. Assume a language with constant 0, a unary function symbol $s$, constraint predicate $=$ and program predicate *add*. We can write the traditional Horn clause definition of Peano addition:

```
add(0,X,X).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

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.

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

The clauses are combined into a single definition similar to the Clark completion of a program. We also use the variable permutation $\pi$ 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 $\pi$ using the relation expression $W_\pi$ 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 $W_\pi$ and its connection with function $w_\pi$ is given below, see Def. 7 and Lemma 4.)

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

Now we give a description of the general translation procedure. We first process programs to their complete database form as defined in [6], 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 5 (General Purified Form for Clauses).** *For a clause* $p(\boldsymbol{t}[\boldsymbol{y}]) \leftarrow \boldsymbol{q}(\boldsymbol{v}[\boldsymbol{y}])$*, let* $h = \alpha(p)$*,* $y = |\boldsymbol{y}|$*,* $v = |\boldsymbol{v}|$*, and* $m = h + y + v$*. Assume vectors:*

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

*the clause's GPF form is:*

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

$\exists^{n\uparrow}$ denotes existential closure with respect to all variables whose index is greater than $n$. $\boldsymbol{x}_h$ and $\boldsymbol{x}_t$ 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 6 (Completion of a Predicate).** *We define Clark's completed form for a predicate $p$ with clauses $cl_1, \ldots, cl_n$ in GPF form:*

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

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.

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

$$(\boldsymbol{au}, \boldsymbol{av}) \in [\![I_n; \dot{K}(\varphi[\boldsymbol{xy}]); I_n]\!]^{\mathcal{D}^\dagger} \iff \mathcal{D} \models (\exists^{n\uparrow}.\varphi[\boldsymbol{xy}])[\boldsymbol{a}/\boldsymbol{x}]$$

**Recursive Predicate Definitions** We shall handle recursive predicate definitions by extending the relational language with a set of definitional symbols $\overline{p}, \overline{q}, \overline{r}, \ldots$ for predicates. Then, a recursive predicate $\overline{p}$ is translated to a definitional equation $\overline{p} \mathrel{\dot{=}} R(\overline{p}_1, \ldots, \overline{p}_n)$, spelled out in Def. 8 where the notation $R(\overline{p}_1, \ldots, \overline{p}_n)$ indicates that relation $R$ resulting from the translation may depend on predicate symbols $\overline{p}_1, \ldots, \overline{p}_n$. Note that $R$ is monotone in $\overline{p}_1, \ldots, \overline{p}_n$. Consequently, using a straightforward fixed point construction we can extend the interpretation $[\![\_]\!]^{\mathcal{D}^\dagger}$ to satisfy $[\![\overline{p}]\!]^{\mathcal{D}^\dagger} = [\![R(\overline{p}_1, \ldots, \overline{p}_n)]\!]^{\mathcal{D}^\dagger}$, thus preserving soundness when we adjoin the definitional equations to $\mathbf{QRA}_\Sigma$. The details are given in Subsection 3.3, below.

**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 $w_\pi$ on formulas and terms is defined in the standard way by its action over variables. We write $W_\pi$ for the corresponding relation:

**Definition 7 (Switching Relations).** *Let $\pi : \{1..n\} \to \{1..n\}$ be a permutation. The* switching relation expression $W_\pi$*, associated to $\pi$ is:*

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

**Lemma 4.** *Fix a permutation $\pi$ and its corresponding $w_\pi$ and $W_\pi$. Then:*

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

**The Translation Function** Now we may define the translation for defined predicates.

**Definition 8 (Relational Translation of Predicates).** *Let $h, p(\boldsymbol{x}_h)$ be as in Def. 5. The translation function Tr from completed predicates to relational equations is defined by:*

$$\begin{array}{lll}
Tr(p(\boldsymbol{x}_h) \leftrightarrow cl_1 \vee \cdots \vee cl_k) & = (\overline{p} \mathrel{\mathring{=}} Tr_{cl}(cl_1) \cup \cdots \cup Tr_{cl}(cl_k)) & \\
Tr_{cl}(\exists^{h\uparrow}.\boldsymbol{p}) & = I_h; (Tr_l(p_1) \cap \cdots \cap Tr_l(p_n)); I_h & \\
Tr_l(\varphi) & = \dot{K}(\varphi) & \varphi \textit{ a constraint} \\
Tr_l(p_i(\boldsymbol{x}_i)) & = W_\pi; \overline{p_i}; W_\pi^\circ \quad \textit{such that } \pi(x_1,\ldots,x_{\alpha(p_i)}) = \boldsymbol{x}_i &
\end{array}$$

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

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

*Example 1.* Figure 3 shows a fragment of a constraint logic program to represent a family relations database [21]. Consider the translation of the program predicates mother, parent, sibling and brother. We write the program in general purified form:

$$\begin{aligned}
mother(x_1,x_2) &\iff (x_1 = sarah) \wedge (x_2 = isaac) \\
parent(x_1,x_2) &\iff father(x_1,x_2) \vee mother(x_1,x_2) \\
sibling(x_1,x_2) &\iff \exists x_3.\ x_1 \neq x_2 \wedge parent(x_3,x_1) \wedge parent(x_3,x_2) \\
brother(x_1,x_2) &\iff male(x_1) \wedge sibling(x_1,x_2)
\end{aligned}$$

Letting $\sigma_1$ and $\sigma_2$ be the permutations $\langle 1,2,3\rangle \longrightarrow \langle 2,3,1\rangle$ and $\langle 1,2,3\rangle \longrightarrow \langle 3,2,1\rangle$ respectively we obtain

$$\begin{aligned}
\overline{mother} &= \dot{K}(x_1 = sarah) \cap \dot{K}(x_2 = isaac) \\
\overline{parent} &= \overline{father} \cup \overline{mother} \\
\overline{sibling} &= \dot{K}(x_1 \neq x_2) \cap I_2[W_{\sigma_1}\overline{parent}W_{\sigma_1}^o \cap W_{\sigma_2}\overline{parent}W_{\sigma_2}^o]I_2 \\
\overline{brother} &= \overline{male} \cap \overline{sibling}
\end{aligned}$$

The query $brother(X, milcah)$ leads to the rewriting of the term $\dot{K}(x_2 = milcah) \cap \overline{brother}$ to $\dot{K}(x_2 = milcah) \cap \dot{K}(x_1 = lot)$.

```
male(terach). male(haran). male(isaac). male(lot).

female(sarah). female(milcah). female(yiscah).

father(terach,haran). father(haran,lot). ↩
    father(haran,milcah).

mother(sarah,isaac).

parent(X,Y) ← father(X,Y).
parent(X,Y) ← mother(X,Y).

sibling(S1,S2) ← S1≠S2, parent(Par,S1), parent(Par,S2).

brother(Brother,Sib) ← male(Brother), sibling(Brother,Sib).
```

**Fig. 3.** Biblical family relations in Prolog.

### 3.3 The Least Relational Interpretation Satisfying Definitional Equations

Let $P$ be a program and $\overline{p}_1, \ldots, \overline{p}_n$ be a sequence of *relation variables*, one for each predicate symbol $p_i$ in the language of $P$. We define the extended relation calculus $\mathsf{R}_\Sigma(\overline{p}_1, \ldots, \overline{p}_n)$ to be the set of terms generated by $\overline{p}_1, \ldots, \overline{p}_n$ and the terms of $\mathsf{R}_\Sigma$. More formally

$$\begin{array}{ll} \mathsf{R}_{atom} & ::= \overline{p}_1 \mid \cdots \mid \overline{p}_n \mid \mathsf{R}_\mathcal{C} \mid \mathsf{R}_\mathcal{F} \mid \mathsf{R}_{\mathcal{CP}} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl \\ \mathsf{R}_\Sigma(\overline{p}_1, \ldots, \overline{p}_n) & ::= \mathsf{R}_{atom} \mid \mathsf{R}_\Sigma{}^\circ \mid \mathsf{R}_\Sigma \cup \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \cap \mathsf{R}_\Sigma \mid \mathsf{R}_\Sigma \mathsf{R}_\Sigma \end{array}$$

Observe that the relational translation of Def. 8 maps programs to sets of definitional equations $\overline{p}_i \doteq R_i(\overline{p}_1, \ldots, \overline{p}_n)$ over $\mathsf{R}_\Sigma(\overline{p}_1, \ldots, \overline{p}_n)$. Let $\mathcal{F}$ be the set of all $n$ such definitional equations.

Given a structure $\mathcal{D}$ we now lift the definition of $\mathcal{D}$-*interpretation* given in Def. 1 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. 2 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 the language of a program, 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 CPO, a 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. 2), 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 $\Lambda$ 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 all $i \quad (0 \le i \le n)$.

In the remainder of this section, the word *interpretation* will refer to an extended $\mathcal{D}$-interpretation.

**Lemma 5.** *Let $[\![\ ]\!]$ and $[\![\ ]\!]'$ be interpretations. If for all $i$ $[\![\overline{p}_i]\!] \subseteq [\![\overline{p}_i]\!]'$ then $[\![\ ]\!] \le [\![\ ]\!]'$.*

*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 9.** *Let $P$ be a program, with predicate symbols $\{p_1, \ldots, p_n\}$. Fix a structure $\mathcal{D}$ for the language of $P$. Let $\mathcal{F}$ be the set of definitional equations $\{\overline{p}_i \mathrel{\mathring{=}} R_i(\overline{p}_1, \ldots, \overline{p}_n) : i \in \mathbb{N}\}$ produced by the translation $Tr$ of $P$ of Def. 8. 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.** *$\Phi_\mathcal{F}$ 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. 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 $\Phi_\mathcal{F}^{(n)}([\![\ ]\!]_0), (n \in \mathbb{N})$. 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.

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

**Fig. 4.** Constraint meta-reductions

## 4 A Rewriting System for Resolution

In this section, we develop a rewriting system for proof search based on the equational theory $\mathbf{QRA}_\Sigma$, which will be proven equivalent to the traditional operational semantics for CLP. In Sec. 5 we will show that answers obtained by resolution correspond to answers yielded by our rewriting system and conversely.

The use of ground terms permits the use of rewriting, overcoming the practical and theoretical difficulties that the existence of logic variables causes in equational reasoning. Additionally, we may speak of *executable* semantics: we use the same function to compile and interpret CLP programs in the relational denotation.

For practical reasons, we don't rewrite over the full relational language, but we will use a more compact representation of the relations resulting from the translation.[4]

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 $\mathsf{R}$ can be represented in it. In practice we make use of the conventional relational notation $I_n, W_\pi$ when no confusion can arise.

### 4.1 Meta-reductions

We formalize the interface between the rewrite system and the constraint solver as meta-reductions (Fig. 4). Every meta-reduction uses the constraint solver in a black-box manner to perform constraint manipulation and satisfiability checking.

**Lemma 6.** *All meta-reductions are sound: if* $m_i : l \stackrel{P}{\longmapsto} r$ *then* $[\![l]\!]^{\mathcal{D}^\dagger} = [\![r]\!]^{\mathcal{D}^\dagger}$.

[4] There is no problem in defining the rewriting system using the general relational signature, but we would need considerably more rules for no gain.

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

**Fig. 5.** Rewriting system for $SLD$.

### 4.2 A Rewriting System for SLD Resolution

We present a rewriting system for proof search in Fig. 5. We prove local confluence. Later we will prove that a query rewrites to a term in the canonical form $\dot{K}(\psi) \cup R$ iff the leftmost branch of the associated SLD-tree of the program is finite.

**Lemma 7.** $\overset{P}{\longmapsto}$ *is sound: if* $p_i : l \overset{P}{\longmapsto} r$ *then* $[\![l]\!]^{\mathcal{D}^\dagger} = [\![r]\!]^{\mathcal{D}^\dagger}$.

**Lemma 8.** *If we give higher priority to* $p_7$ *over* $p_8$, $\overset{P}{\longmapsto}$ *is locally confluent.*

A *left outermost strategy* gives priority to $p_7$ over $p_8$.

## 5 Operational Equivalence

We prove that our rewriting system over relational terms simulates "traditional" SLD proof search specified as a transition-based operational semantics (i.e. [12,7]). For reasons of space, we give a high-level overview of the proof. The full details can be found in the online technical report.

Recall a *resolvent* is a sequence of atoms or constraints $\boldsymbol{p}$. We write $\square$ for the empty resolvent. We assume given a constraint domain $\mathcal{D}$ and its satisfaction relation $\mathcal{D} \models \varphi$. A *program state* is an ordered pair $\langle \boldsymbol{p} \mid \varphi \rangle$ where $\boldsymbol{p}$ is a resolvent and $\varphi$ is a constraint (called the *constraint store*). The notation $cl : p(\boldsymbol{u}[\boldsymbol{y}]) \leftarrow \boldsymbol{q}(\boldsymbol{v}[\boldsymbol{z}])$ indicates that $p(\boldsymbol{u}[\boldsymbol{y}]) \leftarrow \boldsymbol{q}(\boldsymbol{v}[\boldsymbol{z}])$ is a program clause with label $cl$. Then, the standard operational semantics for SLD resolution can be defined as the following transition system over program states:

**Definition 10 (Standard SLD Semantics).**

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

Taking the previous system as a reference, the proof proceeds in two steps: we first define a new transition system that internalizes renaming apart and proof search, and we prove it equivalent to the standard one.

Second, we show a simulation relation between the fully internalized transition system and a transition system defined over relations, which is implemented by the rewriting system of Sec. 4.

With these two equivalences in place, the main theorem is:

**Theorem 3.** *The rewriting system of Fig. 5 implements the transition system of Def. 10. Formally, for every transition* $(r_1, r_2) \in (\rightarrow_l)^*$,

$$\exists n.(Tr(r_1), Tr(r_2)) \in (\overset{P}{\longmapsto})^n$$

*and*

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

Thus, given a program $P$, relational rewriting of translation will return an answer constraint $K(\varphi)$ iff SLD resolution from $P$ reaches a program state $\langle \square \mid \varphi' \rangle$, with $\varphi \iff \varphi'$.

In the next section, we briefly describe the main intermediate system used in the proof.

### 5.1 The Resolution Transition System

The crucial part of the SLD-simulation proof is the definition of a new extended transition system over program states that will internalize both renaming apart and the proof-search tree. It is an intermediate system between relation rewriting and traditional proof search.

The first step towards the new system is the definition of an extended notion of state. In the standard system of Def. 10, a state is a resolvent plus a constraint store. Our extended notion of state includes:

- A notion of *scope*, which is captured by a natural number which can be understood as the number of global variables of the state.
- A notion of *substate*, which includes information about parameter passing in the form of a *permutation*.
- A notion of clause *selection*, and
- a notion of *failure* and *parallel state*, which represents failures in the search tree and alternatives.

Such states are enough to capture all the meta-theory of constraint logic programming except recursion, which operates meta-logically by replacing predicate symbols by their definitions. Formally:

**Definition 11.** *The set $\mathcal{PS}$ of* resolution states *is inductively defined as:*
- $\langle \mathit{fail} \rangle$.
- $\langle \boldsymbol{p} \mid \varphi \rangle_n$, *where* $p_i \equiv P_i(\boldsymbol{x}_i)$ *is an atom, or a constraint* $p_i \equiv \psi$, $\boldsymbol{x}_i$ *a vector of variables,* $\varphi$ *a constraint store and* $n$ *a natural number.*

$$
\begin{array}{rcl}
\langle \psi, \boldsymbol{p} \,|\, \varphi \rangle_n & \xrightarrow{constraint}_p & \langle \boldsymbol{p} \,|\, \varphi \wedge \psi \rangle_n \\
\langle \psi, \boldsymbol{p} \,|\, \varphi \rangle_n & \xrightarrow{fail}_p & \langle fail \rangle \\
& & \boxed{\text{if } \varphi \wedge \psi \text{ is not satisfiable}} \\
\langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n & \xrightarrow{call}_p & \left\langle^{\pi} \blacktriangleright (\langle \boldsymbol{q}_1 \,|\, \top \rangle_h \,\|\, \ldots \,\|\, \langle \boldsymbol{q}_k \,|\, \top \rangle_h), \boldsymbol{p} \,|\, \varphi \right\rangle_n \\
& & \boxed{\text{if } p(\boldsymbol{x}_h) \leftarrow \exists^{h\uparrow}.(\boldsymbol{q}_1 \vee \ldots \vee \boldsymbol{q}_k) \in P',\ \pi(\boldsymbol{x}) = \boldsymbol{x}_h} \\
\left\langle^{\pi} \blacktriangleright (\langle \boldsymbol{q} \,|\, \psi \rangle_h \,\|\, PS), \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{select}_p & \left( \left\langle^{\pi} \langle \boldsymbol{q} \,|\, \psi \wedge \Delta_h^{\pi}(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \,\|\, \left\langle^{\pi} \blacktriangleright PS, \boldsymbol{p} \,|\, \varphi \right\rangle_n \right) \\
\left\langle^{\pi} \langle \square \,|\, \psi \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{return}_p & \langle \boldsymbol{p} \,|\, \nabla_h^{\pi}(\psi, \varphi) \rangle_n \\
\left\langle^{\pi} \langle fail \rangle, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{return}_p & \langle fail \rangle \\
\left\langle^{\pi} PS, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{sub}_p & \left\langle^{\pi} PS', \boldsymbol{p} \,|\, \varphi \right\rangle_n \\
& & \boxed{\text{if } PS \neq \langle \square \,|\, \psi \rangle_n,\ PS \neq \langle fail \rangle, \text{ and } PS \to_p PS'} \\
(\langle fail \rangle \,\|\, PS) & \xrightarrow{backtrack}_p & PS \\
(PS_1 \,\|\, PS_2) & \xrightarrow{seq}_p & (PS_1' \,\|\, PS_2) \\
& & \boxed{\text{if } PS \neq \langle fail \rangle, \text{ and } PS_1 \to_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 ($k = 1$)).

**Fig. 6.** Resolution Transition System.

- $\left\langle^{\pi} PS, \boldsymbol{p} \,|\, \varphi \right\rangle_n$, *where* $PS$ *is a resolution state, and* $\pi$ *a permutation.*
- $\left\langle^{\pi} \blacktriangleright PS, \boldsymbol{p} \,|\, \varphi \right\rangle_n$, *the "select state". It represents the state just before selecting a clause to proceed with proof search.*
- $(PS_1 \,\|\, PS_2)$. *The bar is parallel composition, capturing choice in the proof search tree.*

The *resolution transition system* $\to_P \subseteq (\mathcal{PS} \times \mathcal{PS})$ is shown in Fig. 6. 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 $(n, \pi)$; 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.

Then, we have the following lemma:

**Lemma 9.** *For all queries* $\langle \boldsymbol{p} \,|\, \varphi \rangle_n$, *the first successful* $\to_l$ *derivation using a SLD strategy uniquely corresponds to a* $\to_p$ *derivation:*

$$\langle \boldsymbol{p} \,|\, \varphi \rangle_n \to_l \ldots \to_l \langle \square \,|\, \varphi' \rangle_n \quad \Longleftrightarrow \quad \langle \boldsymbol{p} \,|\, \varphi \rangle_n \to_p \ldots \to_p (\langle \square \,|\, \varphi' \rangle_n \,\|\, PS)$$

*for some resolution state PS.*

**Corollary 1.** *The transition systems of Def. 10 and Fig. 6 are answer-equivalent: for any query they return the same answer constraint.*

With this lemma in place, the proof of Thm. 3 is completed by showing a simulation between the resolution system and a transition system induced by relation rewriting.

## 6 Related and Future Work

*Previous Work:* The paper is the continuation of previous work in [4,15,11] considerably extended to include constraint logic programming, which requires a different translation procedure and a different 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. The operational semantics is also new.

*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 [19], 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 [23,5] in logic programming, another, the formalization of logic programming in categorical logic [2,20,13,1,8] which provides a mathematical framework for the treatment of variables, as well as for derivations [14]. None of the cited work gives a solution that simultaneously includes logic variables, constraints and proof search strategies however.

Bellia and Occhiuto [3] 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 relations is more applicable to extensions of logic programming. Furthermore the authors do not consider constraints.

*Future Work:* A complementary approach to this work is the use of category theory, in particular the Freyd's theory of *tabular allegories* [9] which extends the 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 [10]. It would be interesting to extend the translation to hereditarily Harrop or higher order logic [18] 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.

## 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. Our framework has several significant advantages.

Programs can be analyzed, transformed and optimized entirely within this framework. Execution is carried out by rewriting over relational terms. In these two ways, specification and implementation are brought much closer together than in the traditional logic programming formalism.

## References


1. Amato, G., Lipton, J., McGrail, R.: On the algebraic structure of declarative programming languages. Theoretical Computer Science 410(46), 4626 – 4671 (2009), `http://www.sciencedirect.com/science/article/B6V1G-4WV15VS-7/2/5475111b9a9642244a208e9bd1fcd46a`, abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi
2. Asperti, A., Martini, S.: Projections instead of variables: A category theoretic interpretation of logic programs. In: ICLP. pp. 337–352 (1989)
3. Bellia, M., Occhiuto, M.E.: C-expressions: A variable-free calculus for equational logic programming. Theor. Comput. Sci. 107(2), 209–252 (1993)
4. Broome, P., Lipton, J.: Combinatory logic programming: computing in relation calculi. In: ILPS '94: Proceedings of the 1994 International Symposium on Logic programming. pp. 269–285. MIT Press, Cambridge, MA, USA (1994)
5. Cheney, J., Urban, C.: Alpha-prolog: A logic programming language with names, binding, and alpha-equivalence (2004)
6. Clark, K.L.: Negation as failure. In: Gallaire, Minker (eds.) Logic and Data Bases. pp. 293–322. Plenum Press (1977)
7. Comini, M., Levi, G., Meo, M.C.: A theory of observables for logic programs. Inf. Comput. 169(1), 23–80 (2001)
8. Finkelstein, S.E., Freyd, P.J., Lipton, J.: A new framework for declarative programming. Theor. Comput. Sci. 300(1-3), 91–160 (2003)
9. Freyd, P., Scedrov, A.: Categories, Allegories. North Holland Publishing Company (1991)
10. Gallego Arias, E.J., Lipton, J.: Logic programming in tabular allegories. In: Dovier, A., Costa, V.S. (eds.) Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary. LIPIcs, vol. 17, pp. 334–347. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)

11. Gallego Arias, E.J., Lipton, J., Mariño, J., Nogueira, P.: First-order unification using variable-free relational algebra. Logic Journal of IGPL 19(6), 790–820 (2011), http://jigpal.oxfordjournals.org/content/19/6/790.abstract
12. Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. Journal of Logic Programming 19/20, 503–581 (1994), http://citeseer.ist.psu.edu/jaffar94constraint.html
13. Kinoshita, Y., Power, A.J.: A fibrational semantics for logic programs. In: Dyckhoff, R., Herre, H., Schroeder-Heister, P. (eds.) ELP. Lecture Notes in Computer Science, vol. 1050, pp. 177–191. Springer (1996)
14. Komendantskaya, E., Power, J.: Coalgebraic derivations in logic programming. In: Bezem, M. (ed.) CSL. LIPIcs, vol. 12, pp. 352–366. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
15. Lipton, J., Chapman, E.: Some notes on logic programming with a relational machine. In: Jaoua, A., Kempf, P., Schmidt, G. (eds.) Using Relational Methods in Computer Science. pp. 1–34. Technical Report Nr. 1998-03, Fakultät für Informatik, Universität der Bundeswehr München (Jul 1998)
16. Lloyd, J.W.: Foundations of logic programming. Springer-Verlag New York, Inc., New York, NY, USA (1984)
17. Mal'tsev, A.I.: On the elementary theories of locally free universal algebras. Soviet Math pp. 768–771 (1961)
18. Miller, D., Nadathur, G., Pfenning, F., Scedrov, A.: Uniform proofs as a foundation for logic programming. Annals of Pure and Applied Logic 51(1-2), 125–157 (1991)
19. Pfenning, F., Elliot, C.: Higher-order abstract syntax. In: PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation. pp. 199–208. ACM, New York, NY, USA (1988)
20. Rydeheard, D.E., Burstall, R.M.: A categorical unification algorithm. In: Proceedings of a tutorial and workshop on Category theory and computer programming. pp. 493–505. Springer-Verlag New York, Inc., New York, NY, USA (1986)
21. Sterling, L., Shapiro, E.: The Art of Prolog. The MIT Press (1986)
22. Tarski, A., Givant, S.: A Formalization of Set Theory Without Variables, Colloquium Publications, vol. 41. American Mathematical Society, Providence, Rhode Island (1987)
23. Urban, C., Pitts, A.M., Gabbay, M.J.: Nominal unification. Theoretical Computer Science 323(1–3), 473–497 (2004)

## A Proofs

*Proof (Thm: 1).* The proof is straightforward. The rules of equational reasoning 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 $(w, u) \in ([\![S]\!]^{\mathcal{D}^\dagger})^o$ hence $(w, v)$ is in both $([\![S]\!]^{\mathcal{D}^\dagger})^o [\![R]\!]^{\mathcal{D}^\dagger}$ and $[\![T]\!]^{\mathcal{D}^\dagger}$, so $(u, v) \in [\![S(S^o R \cap T)]\!]^{\mathcal{D}^\dagger}$ as was to be shown.

The equational theory of $\mathbf{QRA}_{\mathbb{H}}$ is sound for $[\![\cdot]\!]^{\mathcal{D}^\dagger}$ if $\mathcal{D}$ is a term algebra (or a locally free algebra in the terminology of [17]).

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

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

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

*Proof (Lem. 2).* By induction on the structure of the formulas:

- We consider the case of a unary constraint predicate $p$, where our atomic formula is just $p(\boldsymbol{t}[\boldsymbol{x}])$ (the argument extends easily to higher arities). Observe that $(\boldsymbol{au}, \boldsymbol{au'}) \in [\![\dot{K}(p(t))]\!]^{\mathcal{D}^\dagger}$, i.e. $(\boldsymbol{au}, \boldsymbol{au'}) \in [\![K(t)^\circ P_1^\circ; \mathsf{p}; P_1; K(t)]\!]^{\mathcal{D}^\dagger}$ is equivalent to the assertion that for some $b \in \mathcal{D}, \boldsymbol{b}, \boldsymbol{v}, \boldsymbol{v'} \in \mathcal{D}^\dagger$

  $$(\boldsymbol{au}, b) \in [\![K(t)^\circ]\!]^{\mathcal{D}^\dagger} \text{ and } (b, \boldsymbol{b}) \in [\![P_1^\circ]\!]^{\mathcal{D}^\dagger} \text{ and } (b_1\boldsymbol{v}, b_1\boldsymbol{v'}) \in [\![\mathsf{p}]\!]^{\mathcal{D}^\dagger}.$$

  Equivalently, we have

  $$b = b_1, \quad (b\boldsymbol{v}, b\boldsymbol{v'}) \in [\![\mathsf{p}]\!]^{\mathcal{D}^\dagger} \text{ and } b = t^{\mathcal{D}}[\boldsymbol{a}/\boldsymbol{x}]$$

  the latter equation by Lem. 1. By definition of $[\![\mathsf{p}]\!]^{\mathcal{D}^\dagger}$ this implies that $p^{\mathcal{D}}(t^{\mathcal{D}}[\boldsymbol{a}/\boldsymbol{x}])$, that is to say, that $\mathcal{D} \models p(t)[\boldsymbol{a}/\boldsymbol{x}]$.
  Conversely, if $p^{\mathcal{D}}(t^{\mathcal{D}}[\boldsymbol{a}/\boldsymbol{x}])$ then for some $\boldsymbol{v}, \boldsymbol{v'} \in \mathcal{D}^\dagger$ we have

  $$(t^{\mathcal{D}}[\boldsymbol{a}/\boldsymbol{x}]\boldsymbol{v}, t^{\mathcal{D}}[\boldsymbol{a}/\boldsymbol{x}]\boldsymbol{v'}) \in [\![\mathsf{p}]\!]^{\mathcal{D}^\dagger}.$$

By the equivalences stated above, we obtain $(\boldsymbol{au}, \boldsymbol{au'}) \in [\![K(t)^\circ P_1^\circ; \mathsf{p}; P_1; K(t)]\!]^{\mathcal{D}^\dagger}$ for any $\boldsymbol{u}, \boldsymbol{u'} \in \mathcal{D}^\dagger$.

– For the case $\varphi[\boldsymbol{x}] \wedge \theta[\boldsymbol{x}]$,

$$(\boldsymbol{au}, \boldsymbol{au'}) \in [\![\dot{K}(\varphi[\boldsymbol{x}]) \cap \dot{K}(\theta[\boldsymbol{x}])]\!] \iff (\boldsymbol{au}, \boldsymbol{au'}) \in [\![\dot{K}(\varphi[\boldsymbol{x}])]\!] \text{ and } (\boldsymbol{au}, \boldsymbol{au'}) \in [\![\dot{K}(\theta[\boldsymbol{x}])]\!]$$

By the induction hypothesis this is equivalent to $\mathcal{D} \models \varphi[\boldsymbol{a}/\boldsymbol{x}]$ and $\mathcal{D} \models \theta[\boldsymbol{a}/\boldsymbol{x}]$, i.e. $\mathcal{D} \models \varphi[\boldsymbol{a}/\boldsymbol{x}] \wedge \theta[\boldsymbol{a}/\boldsymbol{x}]$.

– For the case $\exists x_i.\varphi[\boldsymbol{x}]$, let $\boldsymbol{a}$ be an arbitrary sequence in $\mathcal{D}^\dagger$ of the same length $m$ as $\boldsymbol{x}$, $\boldsymbol{a_{i-1}} \equiv a_1, \ldots, a_{i-1}$ and $\boldsymbol{a_{i+1}} \equiv a_{i+1}, \ldots, a_m$. For arbitrary sequences $\boldsymbol{u}, \boldsymbol{v}$ we have that

$$(\boldsymbol{au}, \boldsymbol{av}) \in Q_i[\![\varphi[\boldsymbol{x}]]\!]Q_i \iff \exists \boldsymbol{bqr}\ (\boldsymbol{au}, \boldsymbol{b}) \in Q_i \wedge (\boldsymbol{q}, \boldsymbol{r}) \in [\![\varphi[\boldsymbol{x}]]\!] \wedge (\boldsymbol{r}, \boldsymbol{av}) \in Q_i.$$

Equivalently, by the definition of $Q_i$

$$(\boldsymbol{au}, \boldsymbol{av}) \in Q_i[\![\varphi[\boldsymbol{x}]]\!]Q_i \iff \exists b_i\ (\boldsymbol{a_{i-1}} b_i \boldsymbol{a_{i+1}} \boldsymbol{u}, \boldsymbol{a_{i-1}} b_i \boldsymbol{a_{i+1}} \boldsymbol{v}) \in [\![\varphi[\boldsymbol{x}]]\!]^{\mathcal{D}^\dagger}$$

and by the induction hypothesis for $\varphi$

$$(\boldsymbol{au}, \boldsymbol{av}) \in Q_i[\![\varphi[\boldsymbol{x}]]\!]Q_i \iff \exists b_i\ \mathcal{D} \models \varphi[\boldsymbol{a_{i-1}} b_i \boldsymbol{a_{i+1}}/\boldsymbol{x}]$$

Equivalently, we have

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

as we wanted to show.

**Lemma 10.**

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

*Proof (Lem. 10).* Immediate: just observe that for each $i$ $[\![P_i(P_i)^\circ]\!]^{\mathcal{D}^\dagger} = \{(\boldsymbol{u}a\boldsymbol{v}, \boldsymbol{u'}a\boldsymbol{v'}) || \boldsymbol{u}| = |\boldsymbol{u'}| = i-1 \text{ and } \boldsymbol{u}, \boldsymbol{u'}, \boldsymbol{v}, \boldsymbol{v'}, a \in \mathcal{D}^\dagger\}$ that is to say, that $P_i(P_i)^\circ$ relates arbitrary sequences except for the position $i$, where it is the identity.

*Proof (Lem. 4).* Straightforward, This is just a restatement of the claim:

$$(\boldsymbol{a'u}, \boldsymbol{a'u}) \in [\![R]\!] \iff (\boldsymbol{au}, \boldsymbol{au}) \in [\![WRW^\circ]\!]$$

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

*Proof (Lem. 3).* $(\boldsymbol{au}, \boldsymbol{av}) \in [\![I_n; \dot{K}(\varphi[\boldsymbol{xy}]); I_n]\!]^{\mathcal{D}^\dagger} \iff$ for some $b_{n+1}, \ldots, b_m, \boldsymbol{u'}, \boldsymbol{v'} \in \mathcal{D}^\dagger$
$(\boldsymbol{abu'}, \boldsymbol{abv'}) \in [\![\dot{K}(\varphi[\boldsymbol{xy}])]\!]^{\mathcal{D}^\dagger}$.

By Lem. 2, we know that $(\boldsymbol{abu'}, \boldsymbol{abv'}) \in [\![\dot{K}(\varphi[\boldsymbol{xy}])]\!]^{\mathcal{D}^\dagger} \iff \mathcal{D} \models \varphi[\boldsymbol{ab}/\boldsymbol{xy}]$.

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

*Proof (Lem. 6).* This is a straightforward consequence of the constraint translation lemma, Lem. 2. 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 $\psi$ are among $\boldsymbol{x}$, where $\boldsymbol{x}$ is chosen to be of length greater then $m$.

Given $\boldsymbol{a}, \boldsymbol{u}, \boldsymbol{u}' \in \mathcal{D}^\dagger$ with $\mid \boldsymbol{a} \mid$ equal to $m$, we have $(\boldsymbol{au}, \boldsymbol{au}') \in [\![I_m\dot{K}(\psi))I_m]\!]^{\mathcal{D}^\dagger}$ if there are $\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{w}' \in \mathcal{D}^\dagger$ with $\mid \boldsymbol{av} \mid = \mid \boldsymbol{x} \mid$ such that $(\boldsymbol{avw}, \boldsymbol{avw}') \in [\![\dot{K}(\psi)]\!]^{\mathcal{D}^\dagger}$. By Lem. 2, this is the case if and only if there is a $\boldsymbol{v}$ such that $\mathcal{D} \models \psi[\boldsymbol{av}/\boldsymbol{x}]$, i.e. iff $\mathcal{D} \models \exists^{m\uparrow}\ \psi[a_1, \ldots, a_m/x_1, \ldots, x_m]$, which in turn, implies $(\boldsymbol{au}, \boldsymbol{au}') \in [\![\dot{K}(\exists^{m\uparrow}\ \psi[\boldsymbol{x}])]\!]^{\mathcal{D}^\dagger}$. The argument for the converse is symmetric.

The soundness of rule $m_2$ follows immediately from Lem. 4.

The soundness of rule $m_3$ follows from the fact that by definition $\dot{K}(\varphi \wedge \psi) = \dot{K}(\varphi) \cap \dot{K}(\psi)$ and the fact that $[\![\cdot]\!]^{\mathcal{D}^\dagger}$ commutes with $\cap$. By Lem. 2 $[\![\dot{K}(\varphi \wedge \psi)]\!]^{\mathcal{D}^\dagger}$ is empty if and only if $\varphi \wedge \psi$ is not satisfiable in $\mathcal{D}$.

**Lemma 11.** *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 (Lem. 11).* 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.

*Proof (Lem. 7).* All of the rules are easy consequences of relation algebra, except for $p_9$. For $p_9$, we apply the equational version of 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.

*Proof (Lem.8).* We study critical pairs and prove that all the existing ones join. Our systems have three critical pairs:

- $m_1$ 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 $\dot{K}(\psi_1 \wedge \psi_2) \cap W(\overline{q})$.

## B Operational Equivalence

We prove that rewriting relational terms simulates "traditional" SLD proof search specified as a transition-based operational semantics (i.e. [12,7]).

We proceed in two steps: we first define two intermediate transition systems — internalizing renaming apart and the search tree — proving them equivalent. Second, we show a simulation relation between the fully internalized transition system and a transition system between relations, implemented by the rewriting system of Sec. 4.

### B.1 Operational Semantics in Logic Style for SLD-resolution

Before defining the *Call-Return* and *Resolution* transition systems, we define the standard SLD semantics and extend the notion of General Purified Form to program states. A *program state* is an ordered pair $\langle A_1, \dots, A_n |\ \varphi\rangle$ where $A_1, \dots, A_n$ is a sequence of atomic formulas or constraints known as the resolvent, and $\varphi$ is a constraint formula known as the constraint store. We write $\Box$ for the null resolvent, i.e. the empty sequence of formulas. We assume free variables in the constraint store to be existentially quantified.

**Definition 12.** *The* standard transition system *capturing SLD resolution is:*

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

We write GPF for general purified form. 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 13.** *The* GPF form *of state* $\langle \boldsymbol{p}(\boldsymbol{u}[\boldsymbol{x}]) \,|\, \varphi[\boldsymbol{x}]\rangle$ *is* $\langle \boldsymbol{p}(\boldsymbol{x}') \,|\, \varphi[\boldsymbol{x}] \wedge \boldsymbol{x}' = \boldsymbol{u}[\boldsymbol{x}]\rangle$, *with* $\boldsymbol{x} = x_1, \dots, x_m$, $k = |\boldsymbol{u}|$, *and* $\boldsymbol{x}' = x_{m+1}, \dots, x_{m+k}$.

**Lemma 12.** *Let $\varphi$ be the constraint store of a state $Q$, and $\varphi'$ the constraint store of $Q'$. Then, $\mathcal{D} \models \varphi$ iff $\mathcal{D} \models \varphi'$.*

*Proof (Lem. 12).* A consequence of soundness. Take a formula $\exists \boldsymbol{x}.\varphi$, then, for $\boldsymbol{x}'$ fresh, and any sequence of terms $\boldsymbol{t}$ from $\mathcal{T}_\Sigma(\mathcal{X})$, $\exists \boldsymbol{x}.\varphi \iff \exists \boldsymbol{x}.\varphi \wedge \boldsymbol{t} = \boldsymbol{t} \iff (\exists \boldsymbol{x}.\varphi \wedge \boldsymbol{x}' = \boldsymbol{t})[\boldsymbol{x}'/\boldsymbol{t}] \iff \exists \boldsymbol{x}\boldsymbol{x}'.\varphi \wedge \boldsymbol{x}' = \boldsymbol{t}$.

**Definition 14.** *We define an equivalence relation $\approx_\mathcal{D}$ on states:*

$$\langle \boldsymbol{p}(\boldsymbol{t}[\boldsymbol{x}_1]) \,|\, \psi_1[\boldsymbol{x}_1]\rangle \approx_\mathcal{D} \langle \boldsymbol{p}(\boldsymbol{t}[\boldsymbol{x}_2]) \,|\, \psi_2[\boldsymbol{x}_2]\rangle$$

*iff* $\mathcal{D} \models \exists \boldsymbol{x}_1\ \psi_1[\boldsymbol{x}_1] \iff \mathcal{D} \models \exists \boldsymbol{x}_2\ \psi_2[\boldsymbol{x}_2]$.

**Lemma 13.** *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 $Q_2 \approx_\mathcal{D} R_2$.*

*Proof (Lem. 13).* 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 $\varphi$ arising from either a resolution or a constraint step, it is the case that $\psi_1 \land \varphi \iff \psi_2 \land \varphi$.

**Lemma 14 (GPF Equivalence).** *For a state $Q_1$ and its GPF form $Q_1'$:*

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

*Proof (Lem. 14).* We annotate the number of variables in use in each constraint store in order to help the reader to follow the proof. Let $|\boldsymbol{x}| = m$, $|\boldsymbol{u}| = |\boldsymbol{x}'| = k$. Recall that $\boldsymbol{x}' = x_{m+1}, \ldots, x_{m+k}$ then

$$\begin{aligned} Q_1 &= \langle \boldsymbol{p}(\boldsymbol{u}[\boldsymbol{x}]) \,|\, \varphi[\boldsymbol{x}] \rangle & \text{[m]} \\ Q_1' &= \langle \boldsymbol{p}(\boldsymbol{x}') \,|\, \varphi[\boldsymbol{x}] \land \boldsymbol{x}' = \boldsymbol{u}[\boldsymbol{x}] \rangle & \text{[m+k]} \end{aligned}$$

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 $Q_1 \xrightarrow{cs}_l Q_2$ and $Q_1' \xrightarrow{cs}_l C_2$. The new states are:

$$\begin{aligned} Q_2 &= \langle \boldsymbol{p}_{|2}(\boldsymbol{u}_{|2}[\boldsymbol{x}]) \,|\, \varphi[\boldsymbol{x}] \land \psi \rangle & \text{[m]} \\ Q_2' &= \langle \boldsymbol{p}_{|2}(\boldsymbol{x}') \,|\, \varphi[\boldsymbol{x}] \land \psi \land \boldsymbol{x}' = \boldsymbol{u}_{|2}[\boldsymbol{x}] \rangle & \text{[m+k]} \\ C_2 &= \langle \boldsymbol{p}_{|2}(\boldsymbol{x}') \,|\, \varphi[\boldsymbol{x}] \land \psi \land \boldsymbol{x}' = \boldsymbol{u}_{|2}[\boldsymbol{x}] \rangle & \text{[m+k]} \end{aligned}$$

  They are the same identical state given that we don't purify constraints.
- If $p_1$ is a defined predicate with clause:

$$\begin{aligned} cl &: \; p_1(\boldsymbol{t}[\boldsymbol{y}]) \leftarrow \boldsymbol{q}(\boldsymbol{v}[\boldsymbol{y}]) \\ cl' &: p_1(\boldsymbol{x}_h) \;\; \leftarrow \exists^{m'}_{h+1}.((\boldsymbol{x}_h = \boldsymbol{t}[\boldsymbol{x}_y] \land \boldsymbol{x}_v = \boldsymbol{v}[\boldsymbol{x}_y]), \boldsymbol{q}(\boldsymbol{x}_v)) \end{aligned}$$

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

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

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

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

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.

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

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 4.** *$Q \to_l \ldots \to_l \langle \Box \,|\, \varphi\rangle$ iff $Q' \to_l \ldots \to_l \langle \Box \,|\, \varphi'\rangle$ and $\langle \Box \,|\, \varphi\rangle \approx_{\mathcal{D}} \langle \Box \,|\, \varphi'\rangle$.*

**Call-Return Transition System** Prior to using a predicate definition in proof search, renaming apart must be performed in order to avoid clashes. In resolution we often have a constraint $\exists \boldsymbol{x}.\varphi[\boldsymbol{zx}]$ that should be combined with $\exists \boldsymbol{y}.\psi[\boldsymbol{zy}]$ to obtain a new constraint $\exists \boldsymbol{x}\boldsymbol{y}_\sigma.(\varphi[\boldsymbol{zx}] \wedge \psi[\boldsymbol{z}\boldsymbol{y}_\sigma])$ with $\boldsymbol{y}_\sigma = \sigma_{\boldsymbol{x}}(\boldsymbol{y})$ a renaming apart of $\boldsymbol{y}$ for $\boldsymbol{x}$. Note however that we can use the logically equivalent formula $(\exists \boldsymbol{x}.\varphi[\boldsymbol{x}]) \wedge (\exists \boldsymbol{y}.\psi[\boldsymbol{y}])$. We will use the last form to capture renaming apart. Doing so requires a carefully chosen canonical naming scheme and the use of the variables $\boldsymbol{z}$ to propagate constraints outside the scope of the quantifiers.

We will keep track of "local" versus "global" variables using a cut-off index $n$. Then, we will existentially quantify variables with index greater than $n$ to preserve local scope. To this end we define an extended notion of state that reflects the index and is closed under sub-states.

**Definition 15.** *The set $\mathcal{CS}$ of* call-return states *is defined inductively as:*
- *$\langle \boldsymbol{p} \,|\, \varphi[\boldsymbol{x}]\rangle_n$, where $p_i \equiv P_i(\boldsymbol{x}_i)$ is an atom or $p_i \equiv \psi$ a constraint, $\boldsymbol{x}_i$ a vector of variables, $n$ a natural number, and $\varphi[\boldsymbol{x}]$ a constraint store.*
- *$\left\langle^{\pi} CS, \boldsymbol{p} \,|\, \varphi[\boldsymbol{x}]\right\rangle_n$, where $CS$ is a call-return state, $\pi$ is a permutation, $\boldsymbol{p}$ a vector of atoms similar to the previous case, $n$ a natural number and $\varphi[\boldsymbol{x}]$ a constraint store.*

$n$ captures the number of arguments involved in a predicate call and $\pi$ captures a permutation of variables local to the state that will 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. We define constraint store operators $\Delta$ and $\nabla$ for the call and return manipulations.

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

$\Delta^\pi_n(\varphi)$ may be read as "modify $\varphi$ to be placed in a context with $n$ open variables and permutation $\pi$". $\nabla^\pi_n(\varphi,\psi)$ may be read as "merge constraint $\varphi$ with scope $(\pi,n)$ with $\psi$". With these in place, we get a new transition system logically formalizing predicate call, capturing in a logical way the notion of call-frame of most Prolog implementations:

**Definition 16.** *The* Call-Return transition system *is:*

$$\begin{array}{lll}
\langle \psi, \boldsymbol{p} \,|\, \varphi\rangle_n & \xrightarrow{constraint}_{cr} & \langle \boldsymbol{p} \,|\, \varphi \wedge \psi\rangle_n \\
 & & \boxed{\textit{if } p_1 \equiv \psi \textit{ and } \varphi \wedge \psi \textit{ satisfiable}} \\
\langle p(\boldsymbol{x}_1), \boldsymbol{p} \,|\, \varphi\rangle_n & \xrightarrow{call/cl_i}_{cr} & \left\langle^\pi \langle \boldsymbol{q} \,|\, \Delta^\pi_h(\varphi)\rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \\
 & & \boxed{\textit{if } cl_i : p(\boldsymbol{x}_h) \leftarrow \exists^{h\uparrow}.\boldsymbol{q} \in P' \textit{ and } \pi(\boldsymbol{x}_1) = \boldsymbol{x}_h} \\
\left\langle^\pi \langle \Box \,|\, \psi\rangle_m, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{return}_{cr} & \langle \boldsymbol{p} \,|\, \nabla^\pi_m(\psi,\varphi)\rangle_n \\
\left\langle^\pi PS, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{sub}_{cr} & \left\langle^\pi PS', \boldsymbol{p} \,|\, \varphi \right\rangle_n \\
 & & \boxed{\textit{if } PS \neq \langle \Box \,|\, \varphi'\rangle_h, \textit{ and } PS \rightarrow_p PS'}
\end{array}$$

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

**Lemma 15.** *Write $\boldsymbol{p}_{|2}$ for subsequence of $\boldsymbol{p}$ starting at the second element. Then, given a GPF state $Q_1 = \langle p_1(\boldsymbol{x}_1), \boldsymbol{p}_{|2}(\boldsymbol{x}) \,|\, \varphi\rangle_n$ and program $P$:*

$$\begin{array}{ll}
 & Q_1 \rightarrow_l \ldots \rightarrow_l \langle \boldsymbol{p}_{|2}(\boldsymbol{x}) \,|\, \varphi'\rangle_n \iff Q_1 \rightarrow_{cr} \ldots \rightarrow_{cr} \langle \boldsymbol{p}_{|2}(\boldsymbol{x}) \,|\, \varphi''\rangle_n \\
\textit{with} & \langle \boldsymbol{p}_{|2}(\boldsymbol{x}) \,|\, \varphi'\rangle_n \approx_{\mathcal{D}} \langle \boldsymbol{p}_{|2}(\boldsymbol{x}) \,|\, \varphi''\rangle_n
\end{array}$$

*It is easily seen that this implies:*

$$Q_1 \rightarrow_l \ldots \rightarrow_l \langle \Box \,|\, \varphi'\rangle_n \iff Q_1 \rightarrow_{cr} \ldots \rightarrow_{cr} \langle \Box \,|\, \varphi''\rangle_n \quad \varphi' \iff \varphi''$$

*Proof (Lem. 15).* By induction over the length of the first derivation. They key point of the proof is to research the behavior the new sub-state notion induces.

*Base Case:* The base case is a derivation of length 1, corresponding either to a constraint step or a *empty* clause $p_1(\boldsymbol{x}_h) \leftarrow$ .

- 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, then the proof is also immediate as the transition for the first system is:

$$\begin{array}{l}
\langle p(\boldsymbol{x}_p), \boldsymbol{p}(\boldsymbol{x}) \,|\, \varphi\rangle_n \quad \xrightarrow{res}_l \\
\langle \boldsymbol{p}(\boldsymbol{x}) \,|\, \varphi \wedge \boldsymbol{x}_{h\sigma} = \boldsymbol{x}_p\rangle_n
\end{array}$$

  and for the call-return one is:

$$\begin{array}{l}
\langle p(\boldsymbol{x}_p), \boldsymbol{p}(\boldsymbol{x}) \,|\, \varphi\rangle_n \quad \xrightarrow{call}_{cr} \\
\left\langle^\pi \langle \Box \,|\, \exists^{h\uparrow}.\pi(\varphi)\rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \quad \xrightarrow{return}_{cr} \\
\langle \boldsymbol{p} \,|\, \varphi \wedge \pi^{-1}(\exists^{h\uparrow}.\exists^{h\uparrow}.\pi(\varphi))\rangle_n
\end{array}$$

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

*Inductive Case:* The inductive case is when $p_1$ is a defined predicate with a non-empty clause:

$$p_1(\boldsymbol{x}_h) \leftarrow \exists_m^n.\boldsymbol{q}(\boldsymbol{x}').$$

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

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

$$\begin{array}{l} \langle \boldsymbol{p}(\boldsymbol{x}) \,|\, \varphi[\boldsymbol{x}]\rangle_m \xrightarrow{call}_{cr} \left\langle^{\pi} \langle \boldsymbol{q}(\boldsymbol{x}') \,|\, \exists^{h\uparrow}.\pi(\varphi[\boldsymbol{x}])\rangle_h, \boldsymbol{p}_{|2} \,|\, \varphi[\boldsymbol{x}]\right\rangle_m \overbrace{\rightarrow \cdots \rightarrow}^{i'} \\ \left\langle^{\pi} \langle \Box \,|\, \exists^{h\uparrow}.\pi(\varphi[\boldsymbol{x}]) \wedge \varphi'[\boldsymbol{x}']\rangle_h, \boldsymbol{p}_{|2} \,|\, \varphi[\boldsymbol{x}]\right\rangle_m \xrightarrow{return}_{cr} \\ \langle \boldsymbol{p}_{|2}(\boldsymbol{x}) \,|\, \varphi[\boldsymbol{x}] \wedge \pi^{-1}(\exists^{h\uparrow}.(\exists^{h\uparrow}.\pi(\varphi[\boldsymbol{x}]) \wedge \varphi'(\boldsymbol{x}')))\rangle_m \end{array}$$

with $\pi(\boldsymbol{x}_1)$. 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.

We use the fact that derivations for the first atom or constraint of a resolvent don't depend on the rest of it:

$$\begin{array}{l} \langle \boldsymbol{p}(\boldsymbol{x}) \,|\, \varphi[\boldsymbol{x}]\rangle \quad \rightarrow_l \ldots \rightarrow_l \langle \Box, \boldsymbol{p}_{|2}(\boldsymbol{x}) \,|\, \varphi[\boldsymbol{x}] \wedge \varphi'[\boldsymbol{x}_1]\rangle \text{ iff} \\ \langle p_1(\boldsymbol{x}_1) \,|\, \varphi[\boldsymbol{x}]\rangle \rightarrow_l \ldots \rightarrow_l \langle \Box \,|\, \varphi[\boldsymbol{x}] \wedge \varphi'[\boldsymbol{x}_1]\rangle \end{array}$$

Then, we check the equivalence of the two states:

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

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

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

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

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

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

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

We apply the induction hypothesis. Actually, we are applying induction as many times as elements or constraints $\boldsymbol{q}$ has. We could recast this lemma to make this fact more explicit:

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

but we think the current presentation is clearer.

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

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

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

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

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

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

which is clearly equivalent to:

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

This concludes the proof.

**Folding of SLD derivations** We extend call-return states to internalize the proof-search tree. The set of possible derivations from a $CS$ state is folded into a single one between *resolution states*, an extension of our previous states with a parallel constructor $(PS_1 \,\|\, PS_2)$. Making failure explicit is necessary, so we introduce a new $\langle \mathit{fail} \rangle$ state. A *resolution state* captures all the meta-theory of constraint logic programming except recursion, which operates meta-logically by grafting predicate symbols onto their definitions.

**Definition 17.** *The set $\mathcal{PS}$ of* resolution states *is inductively defined as:*

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

The key point of the resolution transition system is to split a resolution step into two tasks: clause selection and parameter passing.

**Definition 18.** *The* resolution transition system $\rightarrow_P \subseteq (\mathcal{PS} \times \mathcal{PS})$ *is shown in Fig. 7.*

The main property of this system is the internalization of the SLD search strategy.

$$
\begin{array}{rcl}
\langle \psi, \boldsymbol{p} \,|\, \varphi \rangle_n & \xrightarrow{constraint}_p & \langle \boldsymbol{p} \,|\, \varphi \wedge \psi \rangle_n \\
\langle \psi, \boldsymbol{p} \,|\, \varphi \rangle_n & \xrightarrow{fail}_p & \langle fail \rangle \\
 & & \boxed{\text{if } \varphi \wedge \psi \text{ is not satisfiable}} \\
\langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n & \xrightarrow{call}_p & \left\langle^\pi \blacktriangleright (\langle \boldsymbol{q}_1 \,|\, \top \rangle_h \,\rule{2pt}{8pt}\, \ldots \,\rule{2pt}{8pt}\, \langle \boldsymbol{q}_k \,|\, \top \rangle_h), \boldsymbol{p} \,|\, \varphi \right\rangle_n \\
 & & \boxed{\text{if } p(\boldsymbol{x}_h) \leftarrow \exists^{h\uparrow}.(\boldsymbol{q}_1 \vee \ldots \vee \boldsymbol{q}_k) \in P',\ \pi(\boldsymbol{x}) = \boldsymbol{x}_h} \\
\left\langle^\pi \blacktriangleright (\langle \boldsymbol{q} \,|\, \psi \rangle_h \,\rule{2pt}{8pt}\, PS), \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{select}_p & \left( \left\langle^\pi \langle \boldsymbol{q} \,|\, \psi \wedge \Delta^\pi_h(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \,\rule{2pt}{8pt}\, \left\langle^\pi \blacktriangleright PS, \boldsymbol{p} \,|\, \varphi \right\rangle_n \right) \\
\left\langle^\pi \langle \Box \,|\, \psi \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{return}_p & \langle \boldsymbol{p} \,|\, \nabla^\pi_h(\psi, \varphi) \rangle_n \\
\left\langle^\pi \langle fail \rangle, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{return}_p & \langle fail \rangle \\
\left\langle^\pi PS, \boldsymbol{p} \,|\, \varphi \right\rangle_n & \xrightarrow{sub}_p & \left\langle^\pi PS', \boldsymbol{p} \,|\, \varphi \right\rangle_n \\
 & & \boxed{\text{if } PS \neq \langle \Box \,|\, \psi \rangle_n,\ PS \neq \langle fail \rangle, \text{ and } PS \to_p PS'} \\
(\langle fail \rangle \,\rule{2pt}{8pt}\, PS) & \xrightarrow{backtrack}_p & PS \\
(PS_1 \,\rule{2pt}{8pt}\, PS_2) & \xrightarrow{seq}_p & (PS_1' \,\rule{2pt}{8pt}\, PS_2) \\
 & & \boxed{\text{if } PS \neq \langle fail \rangle, \text{ and } PS_1 \to_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 ($k = 1$))

**Fig. 7.** Resolution Transition System

**Lemma 16 (Clause Selection).** *Suppose we are given a set of clauses:*

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

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

$$\left\{ \begin{array}{l} \langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n \xrightarrow{call/cl_1}_{cr} \left\langle^\pi \langle \boldsymbol{q} \,|\, \Delta^\pi_h(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \not\to_{cr} \\ \langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n \xrightarrow{call/cl_2}_{cr} \left\langle^\pi \langle \boldsymbol{r} \,|\, \Delta^\pi_h(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \xrightarrow{constraint}_{cr} \left\langle^\pi \langle \boldsymbol{r}_{|2} \,|\, r_1 \wedge \Delta^\pi_h(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \end{array} \right\}$$

*iff the derivation in the resolution system is:*

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

*Proof (Lem. 16).* The derivation set is only possible if $q_1 \wedge \Delta^\pi_h(\varphi)$ is not satisfiable. We check the transitions using $\to_p$ (we label *sub* and *seq* transitions with the actual atomic ones):

$$
\begin{array}{l}
\langle p(\boldsymbol{x}) \,|\, \varphi \rangle_n \xrightarrow{call}_p \left\langle^\pi \blacktriangleright (\langle \boldsymbol{q} \,|\, \top \rangle_h \,\rule{2pt}{8pt}\, \langle \boldsymbol{r} \,|\, \top \rangle_h), \boldsymbol{p} \,|\, \varphi \right\rangle_n \xrightarrow{select}_p \\
\left( \left\langle^\pi \langle \boldsymbol{q} \,|\, \Delta^\pi_h(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \,\rule{2pt}{8pt}\, \left\langle^\pi \blacktriangleright \langle \boldsymbol{r} \,|\, \top \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \right) \xrightarrow{fail}_p \\
\left( \left\langle^\pi \langle fail \rangle, \boldsymbol{p} \,|\, \varphi \right\rangle \,\rule{2pt}{8pt}\, \left\langle^\pi \blacktriangleright \langle \boldsymbol{r} \,|\, \top \rangle, \boldsymbol{p} \,|\, \varphi \right\rangle \right) \xrightarrow{return}_p \left( \langle fail \rangle \,\rule{2pt}{8pt}\, \left\langle^\pi \blacktriangleright \langle \boldsymbol{r} \,|\, \top \rangle, \boldsymbol{p} \,|\, \varphi \right\rangle \right) \xrightarrow{backtrack}_p \\
\left\langle^\pi \blacktriangleright \langle \boldsymbol{r} \,|\, \top \rangle, \boldsymbol{p} \,|\, \varphi \right\rangle \xrightarrow{select}_p \left\langle^\pi \langle \boldsymbol{r} \,|\, \Delta^\pi_h(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n \xrightarrow{constraint}_{cr} \left\langle^\pi \langle \boldsymbol{r}_{|2} \,|\, r_1 \wedge \Delta^\pi_h(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \right\rangle_n
\end{array}
$$

Clearly, if $\to_p$ could carry out any other transition the derivation set would be different.

**Lemma 17 (Backtracking).** *For a set of clauses:*

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

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

$$
\left\{
\begin{array}{ll}
\langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n \xrightarrow{call/cl_1}_{cr} & \big\langle^{\pi} \langle q(\boldsymbol{x}_1), \boldsymbol{q} \,|\, \Delta_h^{\pi}(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \big\rangle_n \xrightarrow{call/cl_3}_{cr} \\
& \big\langle^{\pi} \big\langle^{\pi_1} \langle \boldsymbol{s} \,|\, \Delta_i^{\pi_1}(\Delta_h^{\pi}(\varphi)) \rangle_h, \boldsymbol{q} \,|\, \Delta_h^{\pi}(\varphi) \big\rangle, \boldsymbol{p} \,|\, \varphi \big\rangle_n \not\rightarrow_{cr} \\
\langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n \xrightarrow{call/cl_2}_{cr} & \big\langle^{\pi} \langle r(\boldsymbol{x}_2), \boldsymbol{r} \,|\, \Delta_h^{\pi}(\varphi) \rangle_h, \boldsymbol{p} \,|\, \varphi \big\rangle_n \xrightarrow{call/cl_4}_{cr} \\
& \big\langle^{\pi} \big\langle^{\pi_2} \langle \boldsymbol{t} \,|\, \Delta_j^{\pi_2}(\Delta_h^{\pi}(\varphi)) \rangle_h, \boldsymbol{r} \,|\, \Delta_h^{\pi}(\varphi) \big\rangle, \boldsymbol{p} \,|\, \varphi \big\rangle_n \xrightarrow{constraint}_{cr} \\
& \big\langle^{\pi} \big\langle^{\pi_2} \langle \boldsymbol{t}_{|2} \,|\, t_1 \wedge \Delta_j^{\pi_2}(\Delta_h^{\pi}(\varphi)) \rangle_h, \boldsymbol{r} \,|\, \Delta_h^{\pi}(\varphi) \big\rangle, \boldsymbol{p} \,|\, \varphi \big\rangle_n
\end{array}
\right\}
$$

*iff the derivation using the resolution system is:*

$$
\langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n \rightarrow_p \ldots \rightarrow_p \big\langle^{\pi} \big\langle^{\pi_2} \langle \boldsymbol{t}_{|2} \,|\, t_1 \wedge \Delta_j^{\pi_2}(\Delta_h^{\pi}(\varphi)) \rangle_h, \boldsymbol{r} \,|\, \Delta_h^{\pi}(\varphi) \big\rangle, \boldsymbol{p} \,|\, \varphi \big\rangle_n
$$

*Proof (Lem. 17).* We check the transitions as in the previous lemma.

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

**Lemma 18.** *For all queries* $\langle \boldsymbol{p} \,|\, \varphi \rangle_n$*, the first* $\rightarrow_{cr}$ *successful derivation using a SLD strategy uniquely corresponds to a* $\rightarrow_p$ *derivation:*

$$
\langle \boldsymbol{p} \,|\, \varphi \rangle_n \rightarrow_{cr} \ldots \rightarrow_{cr} \langle \Box \,|\, \varphi' \rangle_n \quad \iff \quad \langle \boldsymbol{p} \,|\, \varphi \rangle_n \rightarrow_p \ldots \rightarrow_p \big( \langle \Box \,|\, \varphi' \rangle_n \,\|\, PS \big)
$$

*Proof.* By induction over the length of the successful derivation, repeatedly applying Lem. 16 and Lem. 17.

**Theorem 5.** *The transition systems of Def. 12 and Fig. 7 are answer-equivalent: for any query they return the same answer constraint.*

*Proof (Thm. 5).* The standard transition system is equivalent to the call-return system by Lem. 14 and Lem. 15. The call-return system is equivalent to the resolution transition system by Cor. 18.

### B.2 Relational Operational Semantics for SLD-resolution

The rewriting system of Sec. 4.2 is too fine-grained to be directly related to the resolution operational semantics. We use a set $\mathcal{RS}$ of relational states and a transition system over them that simulates the resolution 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 19.** *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,* $n$, $\varphi$, $\overline{p}$ *parameters.*
- $I_n(W^\circ_\pi(\dot{K}(\varphi) \cap RS) \cap \overrightarrow{W(\overline{p})_\cap})$, *selection,* $n$, $\pi$, $\varphi$, $\overline{p}$, $RS$ *parameters.*
- $I_n(W^\circ_\pi(RS \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap})$, *subquery,* $n$, $\pi$, $\varphi$, $\overline{p}$, $RS$ *parameters.*
- $(RS_1 \cup RS_2)$, *parallel,* $RS_1$, $RS_2$ *parameters.*

Recall that a predicate $p$ is translated to a relational term such that we have the equation $\overline{p} \doteq \Theta_1 \cup \cdots \cup \Theta_k$, and $\Theta_i = I_{\alpha(p)}(\dot{K}(\varphi_i) \cap \overrightarrow{W(\overline{q})_\cap})$.

**Definition 20.** *The* transition system for relational states *is defined by the rules of Fig. 8.*

$$I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{constraint}_r I_n(\dot{K}(\varphi \wedge \psi) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \boxed{\text{if } \varphi \wedge \psi \text{ satisfiable}}$$

$$I_n((\dot{K}(\varphi) \cap \dot{K}(\psi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{fail}_r \mathbf{0} \quad \boxed{\text{if } \varphi \wedge \psi \text{ not satisfiable}}$$

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

$$I_n(W^\circ_\pi(\dot{K}(\varphi) \cap (\Theta_1 \cup \Theta)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{select}_r I_n(W^\circ_\pi(\Theta'_1 \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \cup I_n(W^\circ_\pi(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \boxed{\Theta_1 \equiv I_m(\overrightarrow{W(\overline{q})_\cap})} \quad \boxed{\Theta'_1 \equiv I_m(\dot{K}(\exists^{m\uparrow}.\varphi) \cap \overrightarrow{W(\overline{q})_\cap})}$$

$$I_n(W^\circ_\pi(I_m(\dot{K}(\varphi_1)) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{return}_r I_n(\dot{K}(\pi^{-1}(\varphi \wedge (\exists^{m\uparrow}.\ \varphi_1))) \cap \overrightarrow{W(\overline{p})_\cap})$$

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

$$I_n(W^\circ_\pi(RS \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{sub}_r I_n(W^\circ_\pi(RS' \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \quad \boxed{\text{if } RS \rightarrow_r RS'}$$

$$(\mathbf{0} \cup RS) \xrightarrow{backtrack}_r RS$$

$$(RS_1 \cup RS_2) \xrightarrow{seq}_r (RS'_1 \cup RS_2) \quad \boxed{\text{if } RS_1 \rightarrow_r RS'_1}$$

**Fig. 8.** Relational Transition Rules

### B.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. The last step is to check that the rewriting system of Sec. 4.2 implements the relational transition system, which gives a proof of completeness.

**Definition 21.** *We define functions* $\mathsf{R} : \mathcal{PS} \to \mathcal{RS}$ *and* $\mathsf{R}^{-1} : \mathcal{RS} \to \mathcal{PS}$ *by induction over the structure of the states:*

$$\begin{aligned}
\mathsf{R}(\langle \mathit{fail} \rangle) &= \mathbf{0} \\
\mathsf{R}(\langle \boldsymbol{p} \,|\, \varphi \rangle_n) &= I_n(\dot{K}(\varphi) \cap \overrightarrow{W(\bar{p})_\cap}) \\
\mathsf{R}(\langle^{\pi}\!\blacktriangleright PS, \boldsymbol{p} \,|\, \varphi \rangle_n) &= I_n(W^{\circ}_{\pi}(\dot{K}(\pi(\varphi)) \cap \mathsf{R}(PS)) \cap \overrightarrow{W(\bar{p})_\cap}) \\
\mathsf{R}(\langle^{\pi} PS, \boldsymbol{p} \,|\, \varphi \rangle_n) &= I_n(W^{\circ}_{\pi}(\mathsf{R}(PS) \cap \dot{K}(\pi(\varphi))) \cap \overrightarrow{W(\bar{p})_\cap}) \\
\mathsf{R}((PS_1 \,\|\, PS_2)) &= (\mathsf{R}(PS_1) \cup \mathsf{R}(PS_2))
\end{aligned}$$

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

$$\begin{aligned}
\mathsf{R}^{-1}(\mathbf{0}) &= \langle \mathit{fail} \rangle \\
\mathsf{R}^{-1}(I_n(\dot{K}(\varphi) \cap \overrightarrow{W(\bar{p})_\cap})) &= \langle \boldsymbol{p} \,|\, \varphi \rangle_n \\
\mathsf{R}^{-1}(I_n(W^{\circ}_{\pi}(\dot{K}(\varphi) \cap RS) \cap \overrightarrow{W(\bar{p})_\cap})) &= \langle^{\pi}\!\blacktriangleright \mathsf{R}^{-1}(RS), \boldsymbol{p} \,|\, \pi^{-1}(\varphi) \rangle_n \\
\mathsf{R}^{-1}(I_n(W^{\circ}_{\pi}(RS \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\bar{p})_\cap})) &= \langle^{\pi} \mathsf{R}^{-1}(RS), \boldsymbol{p} \,|\, \pi^{-1}(\varphi) \rangle_n \\
\mathsf{R}^{-1}((RS_1 \cup RS_2)) &= (\mathsf{R}^{-1}(RS_1) \,\|\, \mathsf{R}^{-1}(RS_2))
\end{aligned}$$

Note that $\mathsf{R}$ is an extension of the translation function of Sec. 4.2, and an isomorphism.

**Lemma 19.** $\mathsf{R}$ *is an isomorphism.*

*Proof (Lem. 19).* By induction over the structure of the states. For the failure, base and parallel states the proof is immediate. We check the subquery case:

$$\mathsf{R}^{-1}(\mathsf{R}(\langle^{\pi} PS, \boldsymbol{p} \,|\, \varphi \rangle_n)) = \mathsf{R}^{-1}(I_n(W^{\circ}_{\pi}(\mathsf{R}(PS) \cap \dot{K}(\pi(\varphi))) \cap \overrightarrow{W(\bar{p})_\cap})) = \\ \langle^{\pi} \mathsf{R}^{-1}(\mathsf{R}(PS)), \boldsymbol{p} \,|\, \pi^{-1}(\pi(\varphi)) \rangle_n =_{\{\mathrm{IH}\}} \langle^{\pi} PS, \boldsymbol{p} \,|\, \varphi \rangle_n$$

and the select case:

$$\mathsf{R}^{-1}(\mathsf{R}(\langle^{\pi}\!\blacktriangleright PS, \boldsymbol{p} \,|\, \varphi \rangle_n)) = \mathsf{R}^{-1}(I_n(W^{\circ}_{\pi}(\dot{K}(\pi(\varphi)) \cap \mathsf{R}(PS)) \cap \overrightarrow{W(\bar{p})_\cap})) = \\ \langle^{\pi}\!\blacktriangleright \mathsf{R}^{-1}(\mathsf{R}(PS)), \boldsymbol{p} \,|\, \pi^{-1}(\pi(\varphi)) \rangle_n =_{\{\mathrm{IH}\}} \langle^{\pi}\!\blacktriangleright PS, \boldsymbol{p} \,|\, \varphi \rangle_n$$

**Lemma 20.** *$R$ is a simulation between the resolution transition system of Fig. 7 and the relational transition system of Fig. 8.*

*Proof (Lem. 20).* We check that the relation $R \subseteq (\mathcal{PS} \times \mathcal{RS})$ induced by the isomorphism $\mathsf{R}$ is a simulation relation, that is to say:

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

Given the one to one nature of our relation and the fact that the transition systems are deterministic, the truth of the above statement can be reduced to checking that any of the following properties:

$$PS \rightarrow_p PS' \Rightarrow \mathsf{R}(PS) \rightarrow_r \mathsf{R}(PS')$$
$$RS \rightarrow_r RS' \Rightarrow \mathsf{R}^{-1}(PS) \rightarrow_p \mathsf{R}^{-1}(PS')$$

holds for every transition of the system. Note that one case implies the other. We check the non-obvious transitions constraint, fail, call, select, and return. In order to help the reader, we show both transitions, then perform the check outlined above.

• *constraint*:

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

and the corresponding check:

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

• *fail*:

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

the simulation check is:

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

• *call*:

$$\begin{array}{ll} \langle p(\boldsymbol{x}), \boldsymbol{p} \,|\, \varphi \rangle_n & \xrightarrow{call}_p \left\langle {}^{\pi}\blacktriangleright (\langle \boldsymbol{q}_1 \,|\, \top \rangle_h \,\rule{2pt}{8pt}\, \ldots \,\rule{2pt}{8pt}\, \langle \boldsymbol{q}_k \,|\, \top \rangle_h), \boldsymbol{p} \,|\, \varphi \right\rangle_n \\ & \boxed{\text{if } p(\boldsymbol{x}_h) \leftarrow \exists^{h\uparrow}.(\boldsymbol{q}_1 \vee \ldots \vee \boldsymbol{q}_k) \in P',\ \pi(\boldsymbol{x}) = \boldsymbol{x}_h} \\ I_n((\dot{K}(\varphi) \cap W_\pi(\overline{p})) \cap \overrightarrow{W(\overline{p})_\cap}) & \xrightarrow{call}_r I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) \\ & \boxed{\text{with } \overline{p} = \Theta} \end{array}$$

$\mathsf{R}(\langle \boldsymbol{q}_i \,|\, \top \rangle_h) = I_h(\overrightarrow{W(\overline{q_i})_\cap}) \equiv \Theta_i$, thus $\mathsf{R}((\langle \boldsymbol{q}_1 \,|\, \top \rangle_h \,\rule{2pt}{8pt}\, \ldots \,\rule{2pt}{8pt}\, \langle \boldsymbol{q}_k \,|\, \top \rangle_h)) = \Theta_1 \cup \cdots \cup \Theta_k \equiv \Theta$. The check is:

$$\mathsf{R}(\langle \boldsymbol{p} \,|\, \varphi \rangle_n) = I_n(\dot{K}(\varphi) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{call}_r$$
$$I_n(W^\circ_\pi(\dot{K}(\pi(\varphi)) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap}) = \mathsf{R}(\left\langle {}^{\pi}\blacktriangleright (\langle \boldsymbol{q}_1 \,|\, \top \rangle_h \,\rule{2pt}{8pt}\, \ldots \,\rule{2pt}{8pt}\, \langle \boldsymbol{q}_k \,|\, \top \rangle_h), \boldsymbol{p} \,|\, \varphi \right\rangle_n)$$

• *select*:

$$\left\langle{}^{\pi}\blacktriangleright(\langle \boldsymbol{q}\,|\,\top\rangle_h \,\|\, PS), \boldsymbol{p}\,|\,\varphi\right\rangle_n \xrightarrow{select}_p \left(\left\langle{}^{\pi}\langle \boldsymbol{q}\,|\,\Delta^{\pi}_h(\varphi)\rangle_h, \boldsymbol{p}\,|\,\varphi\right\rangle_n \,\|\, \left\langle{}^{\pi}\blacktriangleright PS, \boldsymbol{p}\,|\,\varphi\right\rangle_n\right)$$

$$I_n(W^{\circ}_{\pi}(\dot K(\varphi)\cap(\Theta_1\cup\Theta))\cap\overrightarrow{W(\bar p)_\cap}) \xrightarrow{select}_r I_n(W^{\circ}_{\pi}(\Theta'_1\cap\dot K(\varphi))\cap\overrightarrow{W(\bar p)_\cap})\cup I_n(W^{\circ}_{\pi}(\dot K(\varphi)\cap\Theta)\cap\overrightarrow{W(\bar p)_\cap})$$

$$\boxed{\Theta_1 \equiv I_m(\overrightarrow{W(\bar q)_\cap})}$$

$$\boxed{\Theta'_1 \equiv I_m(\dot K(\exists^{m\uparrow}.\varphi)\cap\overrightarrow{W(\bar q)_\cap})}$$

The check is:

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

• *return*:

$$\left\langle{}^{\pi}\langle\Box\,|\,\psi\rangle_h, \boldsymbol{p}\,|\,\varphi\right\rangle_n \xrightarrow{return}_p \langle\boldsymbol{p}\,|\,\nabla^{\pi}_h(\psi,\varphi)\rangle_n$$

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

the check:

$$\mathsf{R}(\left\langle{}^{\pi}\langle\Box\,|\,\psi\rangle_h, \boldsymbol{p}\,|\,\varphi\right\rangle_n) = I_n(W^{\circ}_{\pi}(I_m(\dot K(\psi))\cap\dot K(\pi(\varphi)))\cap\overrightarrow{W(\bar p)_\cap}) \xrightarrow{return}_p$$
$$I_n(\dot K(\pi^{-1}(\pi(\varphi)\wedge(\exists^{m\uparrow}.\ \psi)))\cap\overrightarrow{W(\bar p)_\cap}) = I_n(\dot K(\varphi\wedge\pi^{-1}(\exists^{m\uparrow}.\ \psi))\cap\overrightarrow{W(\bar p)_\cap}) =$$
$$\mathsf{R}(\langle\boldsymbol{p}\,|\,\nabla^{\pi}_n(\varphi,\psi)\rangle)$$

• *return* second case:

$$\left\langle{}^{\pi}\langle \mathit{fail}\rangle, \boldsymbol{p}\,|\,\varphi\right\rangle_n \xrightarrow{return}_p \langle \mathit{fail}\rangle$$
$$I_n(W^{\circ}_{\pi}(\mathbf{0}\cap\dot K(\varphi))\cap\overrightarrow{W(\bar p)_\cap}) \xrightarrow{return}_r \mathbf{0}$$

is immediate.

The last step of the proof is checking that the transition relation is properly embedded into the rewriting relation.

**Lemma 21.** *The relational transition system of Fig. 8 is implemented by the rewriting system of Fig. 5 That is to say, for every transition* $(r_1, r_2) \in (\rightarrow_r)$,

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

*Proof (Lem. 21).* 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 check the most relevant transitions constraint, fail, call, select and return.
• *constraint*:

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

This transition is implemented by the rewriting rule $m_3$.

• *fail*:

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

This transition is implemented by the rewriting chain:

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

• *call*:

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

the transition is implemented by the rewriting chain:

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

• *select*:

$$I_n(W_\pi^\circ(\dot{K}(\varphi) \cap (\Theta_1 \cup \Theta)) \cap \overrightarrow{W(\overline{p})_\cap}) \xrightarrow{select}_r I_n(W_\pi^\circ(\Theta_1' \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\overline{p})_\cap}) \cup I_n(W_\pi^\circ(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(\overline{p})_\cap})$$
$$\boxed{\Theta_1 \equiv I_m(\overrightarrow{W(\overline{q})_\cap})}$$
$$\boxed{\Theta_1' \equiv I_m(\dot{K}(\exists^{m\uparrow}.\varphi) \cap \overrightarrow{W(\overline{q})_\cap})}$$

This transition is implemented by the rewriting chain:

$$\begin{array}{ll} I_n(W_\pi^\circ(\dot{K}(\varphi) \cap (\Theta_1 \cup \Theta)) \cap \overrightarrow{W(p)_\cap}) & \overset{P}{\longmapsto} (p_6) \\ I_n(W_\pi^\circ((\dot{K}(\varphi) \cap \Theta_1) \cup (\dot{K}(\varphi) \cap \Theta)) \cap \overrightarrow{W(p)_\cap}) & \overset{P}{\longmapsto} (p_3) \\ I_n((W_\pi^\circ(\dot{K}(\varphi) \cap \Theta_1) \cup W_\pi^\circ(\dot{K}(\varphi) \cap \Theta)) \cap \overrightarrow{W(p)_\cap}) & \overset{P}{\longmapsto} (p_5) \\ I_n((W_\pi^\circ(\dot{K}(\varphi) \cap \Theta_1) \cap \overrightarrow{W(p)_\cap}) \cup (W_\pi^\circ(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(p)_\cap})) & \overset{P}{\longmapsto} (p_4) \\ I_n(W_\pi^\circ(\dot{K}(\varphi) \cap \Theta_1) \cap \overrightarrow{W(p)_\cap}) \cup I_n(W_\pi^\circ(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(p)_\cap}) & \overset{P}{\longmapsto} (p_9) \\ I_n(W_\pi^\circ(I_m(I_m(\dot{K}(\varphi)) \cap \overrightarrow{W(q)_\cap}) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(p)_\cap}) \cup I_n(W_\pi^\circ(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(p)_\cap}) & \overset{P}{\longmapsto} (m_1) \\ I_n(W_\pi^\circ(I_m(\dot{K}(\exists^{m\uparrow}.\ \varphi) \cap \overrightarrow{W(q)_\cap}) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(p)_\cap}) \cup I_n(W_\pi^\circ(\dot{K}(\varphi) \cap \Theta) \cap \overrightarrow{W(p)_\cap}) & \end{array}$$

• *return*:

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

the transition is implemented by the rewriting chain:

$$\begin{array}{ll}
I_n(W_\pi^\circ(I_m(\dot{K}(\psi)) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\bar{p})_\cap}) & \xrightarrow{P} (m_1) \\
I_n(W_\pi^\circ(\dot{K}(\exists^{m\uparrow}.\ \psi) \cap \dot{K}(\varphi)) \cap \overrightarrow{W(p)_\cap} & \xrightarrow{P} (m_3) \\
I_n(W_\pi^\circ(\dot{K}(\exists^{m\uparrow}.\ \psi \wedge \varphi)) \cap \overrightarrow{W(p)_\cap} & \xrightarrow{P} (m_2) \\
I_n(\dot{K}(\pi^{-1}(\exists^{m\uparrow}.\ \psi \wedge \varphi)) \cap \overrightarrow{W(p)_\cap} &
\end{array}$$

• *return* second case:

$$I_n(W_\pi^\circ(\mathbf{0} \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\bar{p})_\cap}) \xrightarrow{return}_r \mathbf{0}$$

the transition is implemented by the rewriting chain:

$$\begin{array}{ll}
I_n(W_\pi^\circ(\mathbf{0} \cap \dot{K}(\varphi)) \cap \overrightarrow{W(\bar{p})_\cap}) & \xrightarrow{P} (p_2) \\
I_n(W_\pi^\circ(\mathbf{0}) \cap \overrightarrow{W(p)_\cap}) & \xrightarrow{P} (m_2*) \\
I_n(\mathbf{0} \cap \overrightarrow{W(p)_\cap}) & \xrightarrow{P} (p_2) \\
I_n(\mathbf{0}) & \xrightarrow{P} (m_1*) \\
\mathbf{0} &
\end{array}$$

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 6.** *The rewriting system simulates SLD-resolution.*

*Proof (Thm. 3).* By Lem. 21 and Theorem 5. Indeed, when SLD-resolution diverges, the relational rewriting system does so in the same way.