A Unification-Free Introduction to Logic Programming

Roberto Di Meglio Vladimiro Sassone
Dipartimento di Informatica
Università di Pisa

Abstract

In this paper, we give a new presentation of the fundamental results of the theory of Logic Programming, which differs from classical introductions in at least two ways: the use of predicate algebras to deal with model theoretical aspects and the parameterization of the resolution algorithm with respect to the specific unification algorithm implemented.

In our opinion, the search for an elegant and concise meta-language to introduce the theory of Logic Programming (LP) has two major goals: to make the basic concepts of the theory easier to understand and to smooth the differences between the various semantics of LP.

In the usual introductions to LP, unification and substitution are among the key notions of the theory; at the same time, they are among the most difficult ones, since many of the difficulties in the proofs of LP theorems are due to them. However, unification is not a methodology; we would rather say that it is just a tool (although a fundamental one) and it could perhaps be considered just an "implementation issue".

In the present approach, unification is not directly introduced. It becomes just one of the algorithms —or, if you prefer, of the proof systems— that may be chosen to solve (or prove) equations and used as modules to use in resolution for solving constraints. On the other hand, since substitutions are algebraically nothing but homomorphisms, they are elegantly included in our meta-language of predicate algebras. For this reason, substitutions do not appear in derivations anymore, so they have not to be included in the operational semantics of LP itself. Moreover, by using (idempotent) substitutions as solutions to equation systems, we get a natural equivalence relation between them, which allows to avoid many cumbersome details in the proofs, so that in our approach many of the intricacies of classical introductions to LP are somewhat made easier.

This also achieves our second goal. In fact, using as a starting point the usual introductions to LP, it is not easy to extend LP to Constraint Logic Programming (CLP). Many new concepts must be introduced, e.g. multi-sorted pre-interpreted algebras or constraint solvers. For this reason, even if from a conceptual point of view CLP is a generalization of LP, it seems to lie in a completely different universe. With our approach, CLP may be explained without introducing any new tool; we can just generalize the tools we defined and used for LP. In particular, we can extend our introduction to encompass CLP just admitting more general constraints and replacing the "unification module" with a new module to solve this type of constraints.

The contents of the sections are as follows.

Section 1, introduces signatures, terms, predicates, atoms and clauses and defines the notion of logic program. Section 2 introduces the class of the algebras over a given signature as the possible interpretations of a program on that signature. Furthermore, we introduce the notions of validity, satisfiability and model of a program.

Declarative and procedural semantics of logic programs are introduced in Section 3 and 4, respectively.

Section 5 includes the definition of SLD_{EQ}–Resolution and the proof of its soundness and completeness with respect to declarative and procedural semantics.

We conclude by summarizing our results in Section 6.

1 Definite Logic Programs

In this section we introduce some basic concepts in a purely syntactical way, using an algebraic notation.

Definition 1.1 *(Signature with predicates)*
A signature *(one-sorted) with predicates is a pair* $\langle \Sigma, \Pi \rangle$ *where* Σ *and* Π *are disjoint families of disjoint sets of, respectively, operator symbols* $\{\Sigma_n | n \in \omega\}$ *and predicate symbols* $\{\Pi_n | n \in \omega\}$. Σ_n *is the set of symbols for* n*-ary operators and* Π_n *is the set of symbols for* n*-ary predicates.*

In the following, we shall always use one-sorted signatures and we shall suppose that the set of operators with zero arity (i.e. the constants), Σ_0, is *not* empty.

Definition 1.2 *(Term and Atom)*
The set $T_{\Sigma,\Pi}(X)$*, of* terms *with variables* X *over a signature* $\langle \Sigma, \Pi \rangle$ *is the smallest set s.t.:*
(i) $\Sigma_0 \cup X \subseteq T_{\Sigma,\Pi}(X)$*;*
(ii) $\forall t_1, \ldots, t_n \in T_{\Sigma,\Pi}(X)$ *and* $\sigma \in \Sigma_n$, $\sigma(t_1, \ldots, t_n) \in T_{\Sigma,\Pi}(X)$
The set of atoms *with variables* X *over* $\langle \Sigma, \Pi \rangle$ *is the set of formulas:*

$$B_{\Sigma,\Pi}(X) = \Big\{ \rho(t_1, \ldots, t_n) \;\Big|\; \rho \in \Pi_n \text{ and } t_1, \ldots, t_n \in T_{\Sigma,\Pi}(X), n \in \mathbb{N} \Big\}.$$

When the variable set is empty, the previous definitions give us the set of *closed (or ground) terms* $T_{\Sigma,\Pi}(\emptyset)$, denoted by $T_{\Sigma,\Pi}$ and called *Herbrand universe* for $\langle \Sigma, \Pi \rangle$, and the set of *closed (or ground) atoms* $B_{\Sigma,\Pi}(\emptyset)$, denoted by $B_{\Sigma,\Pi}$ and called *Herbrand base* for $\langle \Sigma, \Pi \rangle$.
Given $B \subseteq B_{\Sigma,\Pi}(X)$, we shall denote by $[\![B]\!]_\rho$ the set $\Big\{ (t_1, \ldots, t_n) \;\Big|\; \rho(t_1, \ldots, t_n) \in B \Big\}$.

Definition 1.3 *(Definite clause, Goal, Definite program)*
A clause *is a formula of the type* $A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$*, where* $A_1, \ldots A_n, B_1, \ldots, B_m$ *are atoms in* $B_{\Sigma,\Pi}(X)$ *and* $n, m \in \mathbb{N}$ *and* $n \geq 1$*. When* $n = 1$*, the clause is called* definite.
A goal (or query) *is a formula of the type* $\leftarrow B_1, \ldots, B_m$*, where* $B_1, \ldots, B_m \in B_{\Sigma,\Pi}(X)$ *and* $m \in \mathbb{N}$, $m \geq 1$.
A definite program P *is a (finite) set of definite clauses.*

2 Interpretations

In order to start talking about semantics, we need to define the notion of *interpretation* of the symbols in a program. We shall do this by identifying the interpretations of a program with the class of the algebras over the signature defined by the program itself.

Definition 2.1 *($\mathbf{Alg}_{\Sigma,\Pi}$)*
*A $\langle\Sigma,\Pi\rangle$-*algebra $\mathcal{A}$ *consists of*

(i) *a set A, called* carrier *of the algebra and denoted by $|\mathcal{A}|$;*
(ii) *for any $n \in \mathbb{N}$ and $\sigma \in \Sigma_n$ an operation $\sigma_{\mathcal{A}}\colon A^n \to A$;*
(iii) *for any $n \in \mathbb{N}$ and $\rho \in \Pi_n$ a predicate $\rho_{\mathcal{A}} \subseteq A^n$.*

*A $\langle\Sigma,\Pi\rangle$-*homomorphism *between two $\langle\Sigma,\Pi\rangle$-algebras $\mathcal{A}$ and $\mathcal{B}$ is a function $\phi\colon|\mathcal{A}| \to |\mathcal{B}|$ which preserves operations and predicates, i.e. such that:*

(i) *for any $n \in \mathbb{N}$ and $\sigma \in \Sigma$, $\phi(\sigma_{\mathcal{A}}(a_1,\ldots,a_n)) = \sigma_{\mathcal{B}}(\phi(a_1),\ldots,\phi(a_n))$*
(ii) *for any $n \in \mathbb{N}$ and $\rho \in \Pi$ $\phi^n(\rho_{\mathcal{A}}) \subseteq \rho_{\mathcal{B}}$,*

where ϕ^n is the cartesian product of n copies of ϕ.
We shall call $\mathbf{Alg}_{\Sigma,\Pi}$ the class of $\langle\Sigma,\Pi\rangle$-algebras defined over the signature $\langle\Sigma,\Pi\rangle$.

To make notations simpler, we shall denote ϕ^n simply by ϕ. We shall also use $\pi\mathcal{A}$ to denote the set $\{\rho(e_1,\ldots,e_n) \mid (e_1,\ldots,e_n) \in \rho_{\mathcal{A}},\ \rho \in \Pi\}$, and will extend the use of the notation $[\![\_]\!]_\rho$ to subsets of $\pi\mathcal{A}$, $\mathcal{A} \in \mathbf{Alg}_{\Sigma,\Pi}$.

We can give the structure of a $\langle\Sigma,\Pi\rangle$-algebra (called *closed term algebra*) to the set of terms with variables $T_{\Sigma,\Pi}(X)$, defined in the previous section. We do this by defining operations and predicates as follows: the operation $\sigma_{T_{\Sigma,\Pi}(X)}$ with $\sigma \in \Sigma_n$ is defined by

$$\sigma_{T_{\Sigma,\Pi}(X)}(t_1,\ldots,t_n) = \sigma(t_1,\ldots,t_n)$$

and, for any $\rho \in \Pi$, $\rho_{T_{\Sigma,\Pi}(X)} = \emptyset$.

In a similar way, we can also give the structure of a $\langle\Sigma,\Pi\rangle$-algebra to $T_{\Sigma,\Pi}$. The variables in $T_{\Sigma,\Pi}(X)$ are not considered operators with an arity of 0, but just elements of the algebra. This allows $\langle\Sigma,\Pi\rangle$-homomorphisms to map a variable to any element of the algebra corresponding to their codomain. In this way, we capture —as will be made clear in the following— the standard notions of *substitution* and *evaluation* using homomorphisms.

When we give semantics to syntactical objects we are not interested in their actual representation, only to their structure. This is why we usually work up to isomorphisms: an interpretation for a given program is as good as any other isomorphic interpretation. As usual, we will work using this assumption. First, we need to introduce the notion of *initiality*.

Definition 2.2 *(Initiality)*
An algebra $\mathcal{A}$ is initial *in a class $\mathcal{C}$ if and only if $\mathcal{A} \in \mathcal{C}$ and, for any algebra $\mathcal{B} \in \mathcal{C}$, there exists only one morphism from $\mathcal{A}$ to $\mathcal{B}$.*

An important property of an initial algebra, which we shall often use, is that, if it exists, it is uniquely determined up to isomorphisms. This result may obviously be applied to classes of $\langle\Sigma,\Pi\rangle$-algebras as well: if two algebras $\mathcal{A}$ and $\mathcal{B}$ are both initial in $\mathbf{Alg}_{\Sigma,\Pi}$, they are isomorphic. We can easily show that the algebra $T_{\Sigma,\Pi}$ is initial in $\mathbf{Alg}_{\Sigma,\Pi}$.

The algebra $T_{\Sigma,\Pi}(X)$, on the other hand, has a property similar to initiality: it is the *free $\langle\Sigma,\Pi\rangle$-algebra* over the set of generators X. Its main characteristic is shown by the following theorem (look at Figure 2, too).

Theorem 2.3 *(Free algebra)*
Let $\langle\Sigma,\Pi\rangle$ be a signature with predicates, X a set of variables and $\mathcal{A}$ a $\langle\Sigma,\Pi\rangle$-algebra. Given an assignment *of values in $\mathcal{A}$ to the variables, i.e. a function $\alpha\colon X \to \mathcal{A}$, exists only one $\langle\Sigma,\Pi\rangle$-homomorphism $\alpha^\circ\colon T_{\Sigma,\Pi}(X) \to \mathcal{A}$ extending α, i.e. such that $(\alpha^\circ)_{|X} = \alpha$.*

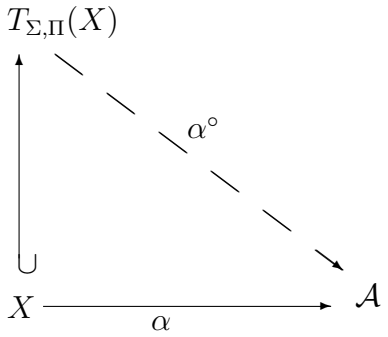


Figure 1: Free algebra over a set of generators X

Proof. We just note that the $\langle\Sigma,\Pi\rangle$-algebra $\mathcal{A}$, together with an assignment $\alpha\colon X \to \mathcal{A}$, gives a structure of $\langle\Sigma(X),\Pi\rangle$-algebra to $\mathcal{A}$, where $\alpha(x)$, for $x \in X$ is the constant assigned to x in $\mathcal{A}$. So, there is only one $\langle\Sigma(X),\Pi\rangle$-homomorphism $\phi\colon T_{\Sigma,\Pi}(X) \to \mathcal{A}$, which must extend α because $\phi(x) = \alpha(x)$. □

An assignment α and its corresponding extension α° are also called an *evaluation*. An assignment $\theta\colon X \to T_{\Sigma,\Pi}(X)$ and its corresponding θ° are called a *substitution*. Evaluations may be composed by composing their extensions as homomorphisms.

To have a more readable notation, we shall usually write αt to indicate $\alpha^\circ(t)$, i.e. the application of the evaluation α to the term t. In a similar way, $\alpha\rho(t_1,\ldots,t_n)$ stands for $\rho(\alpha^\circ(t_1),\ldots,\alpha^\circ(t_n))$. So, for example,

$$\alpha\theta\rho(t_1,\ldots,t_n) = \rho\Big(\alpha^\circ\circ\theta^\circ(t_1),\ldots,\alpha^\circ\circ\theta^\circ(t_n)\Big) = \rho\Big((\alpha^\circ\circ\theta)^\circ(t_1),\ldots,(\alpha^\circ\circ\theta)^\circ(t_n)\Big).$$

We defined interpretations and evaluations in order to understand the meaning of a program. In short, an interpretation gives a meaning to the symbols in a signature, i.e. to the non-variable symbols appearing in a program, while an evaluation gives an interpretation of the variables appearing in the atoms. We also need a formal definition to discuss the *truth (validity)* of formulas under a given interpretation. This definition may be given in terms of a relation ($\models$), which associates to each $\langle\Sigma,\Pi\rangle$-algebra the atoms, the clauses and the programs which are valid in it. In the following definition, we denote by $\mathcal{A}$ a $\langle\Sigma,\Pi\rangle$-algebra; α is a generic assignment $X \to \mathcal{A}$; $A, B_1, \ldots, B_n$ are atoms; C is a generic clause and P is a program.

Definition 2.4 *(Validity and satisfiability)*
The relation $\models$ is the smallest relation such that:

$\mathcal{A} \models_\alpha \rho(t_1,\ldots,t_n)$	*iff* $(\alpha t_1,\ldots,\alpha t_n) \in \rho_\mathcal{A}$;
$\mathcal{A} \models_\alpha A_1,\ldots,A_n$	*iff* $\mathcal{A} \models_\alpha A_1,\ldots,\mathcal{A} \models_\alpha A_n$;
$\mathcal{A} \models A_1,\ldots,A_n$	*iff* $\mathcal{A} \models_\alpha A_1,\ldots,A_n$ *for any* α;
$\mathcal{A} \models_\alpha A_1,\ldots,A_n \leftarrow B_1,\ldots,B_m$	*iff* $\mathcal{A} \models_\alpha B_1,\ldots,B_m \Rightarrow \mathcal{A} \models_\alpha A_1,\ldots,A_n$;
$\mathcal{A} \models A_1,\ldots,A_n \leftarrow B_1,\ldots,B_m$	*iff* $\mathcal{A} \models_\alpha A_1,\ldots,A_n \leftarrow B_1,\ldots,B_m$ *for any* α;
$\mathcal{A} \models P$	*iff* $\mathcal{A} \models C$ *for any* $C \in P$.

We say that a goal $\leftarrow B_1,\ldots,B_m$ is satisfied *in $\mathcal{A}$ (with some notational abuse, we shall denote this by $\mathcal{A} \models \leftarrow B_1,\ldots,B_m$) if and only if there exists an evaluation $\alpha\colon X \to \mathcal{A}$ such that $B_1,\ldots,B_m$ is* true *in $\mathcal{A}$ using the evaluation α (α is called a* witness*). Formally,*

$$\mathcal{A} \models \leftarrow B_1,\ldots,B_m \text{ iff } \mathcal{A} \models_\alpha B_1,\ldots,B_m \text{ for some } \alpha\colon X \to \mathcal{A}.$$

If a formula is true under a certain evaluation, so will be all its instances, i.e. all the formulas which may be obtained from it by substitution. This may be formally shown for conjunctions of atoms in the following lemma, of which we omit the proof. It is quite simple to extend a similar result to clauses.

Lemma 2.5 *(Validity and substitutions)*
If $\mathcal{A} \models A_1, \ldots, A_n$ *then, for any* $\theta: X \to T_{\Sigma,\Pi}(X)$, $\mathcal{A} \models \theta A_1, \ldots, \theta A_n$.

Using the formal definition of truth just given, we are able to formalize an important notion, that of model of a program.

Definition 2.6 *(Models of a program)*
The class of the models of a program P*, denoted by* $\mathbf{Mod}(P)$*, is the sub-class of the* $\langle \Sigma, \Pi \rangle$*-algebras where* P *is valid, i.e. of the algebras* $\mathcal{M}$ *such that* $\mathcal{M} \models P$.

3 Declarative Semantics

The models of a program are the interpretations where all the logical implications specified by program clauses are correct. Some of the facts which are true in a model of P will be logical consequences of the program, while others will depend on the nature of the model itself. For this reason, the consequences of a program, i.e. its semantics, are defined by the set of facts which are valid in *every* model of P.

Definition 3.1 *(Logical consequence and Theory)*
An atom A *is a* logical consequence *of a program* P *(denoted by* $P \models A$*) if and only if* $\mathcal{M} \models A$ *for any* $\mathcal{M} \in \mathbf{Mod}(P)$.
The set of logical consequences of a program P *is called* theory *of* P *and is denoted by* $\mathbf{TH}(P)$.

In the following we shall give several characterization of $\mathbf{TH}(P)$, i.e. we shall see how to define the semantics of a program P in several ways. In this section we introduce declarative-style semantics. First, we define *modus ponens* as a relation on the algebra $T_{\Sigma,\Pi}(X)$. We call *renaming* a substitution $\eta = \{x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n\}$ (i.e. a substitution such that $\eta x_i = y_i$) if $\forall x \in var(\eta),\ x \in cod(\eta)$ implies $x \in dom(\eta)$, where x_i, y_i are variables, $var(\eta)$ is the set of the variables appearing in η and *dom* and *cod* are, respectively, the domain and codomain of functions.

Definition 3.2 *(Modus Ponens)*
The relation $\vdash_{\text{mp}}$ *between programs and atoms in* $T_{\Sigma,\Pi}(X)$ *is the smallest closed relation w.r.t. the following inference rule:*

$$\frac{P \vdash_{\text{mp}} \theta B_1, \ldots, P \vdash_{\text{mp}} \theta B_n}{P \vdash_{\text{mp}} \theta A} \qquad (MP)$$

where $A \leftarrow B_1, \ldots, B_n$ *is a renaming of a clause in* P *and* $\theta: X \to T_{\Sigma,\Pi}(X)$ *is a substitution. We shall often write* $P \vdash_{\text{mp}} A_1, \ldots, A_n$ *instead of* $P \vdash_{\text{mp}} A_1, \ldots, P \vdash_{\text{mp}} A_n$.

The following lemma provides an important observation on the behaviour of modus ponens.

Lemma 3.3 *(Observation on $\vdash_{\text{mp}}$)*
If $P \vdash_{\text{mp}} A$, then there exists a derivation of A from P where, for every rule used, the renaming of variables is such that no rule has a variable in common with another or with the atoms used in the proof.

So, we can assume that, each time we have to execute a derivation step, we can apply the desired renaming to the variables included in a program clause (in general, to make them different from the variables used in the rest of the derivation). We are guaranteed that this operation has no influence on the outcome of the derivation itself.

Definition 3.4 *(Least Herbrand model)*
The least model *of P is the algebra $T_{\Sigma,\Pi,P}$ obtained from $T_{\Sigma,\Pi}$ by assuming,*

$$\forall \alpha \colon X \to T_{\Sigma,\Pi},\ (\alpha t_1, \ldots, \alpha t_n) \in \rho_{T_{\Sigma,\Pi,P}} \Leftrightarrow P \vdash_{\text{mp}} \rho(t_1, \ldots, t_n)$$

.

Proposition 3.5 *($T_{\Sigma,\Pi,P}$ is equivalent to $\vdash_{\text{mp}}$)*
$T_{\Sigma,\Pi,P} \models A_1, \ldots, A_n$ if and only if $P \vdash_{\text{mp}} A_1, \ldots, A_n$.

Proof. Let's consider the problem when $n = 1$.The proof is easily extended to a generic n.

$$\begin{aligned} T_{\Sigma,\Pi,P} \models \rho(t_1, \ldots, t_n) &\Leftrightarrow \forall \alpha \colon X \to T_{\Sigma,\Pi,P},\ (\alpha t_1, \ldots, \alpha t_n) \in \rho_{T_{\Sigma,\Pi,P}} \\ &\Leftrightarrow \forall \alpha \colon X \to T_{\Sigma,\Pi},\ (\alpha t_1, \ldots, \alpha t_n) \in \rho_{T_{\Sigma,\Pi,P}} \\ &\Leftrightarrow P \vdash_{\text{mp}} \rho(t_1, \ldots, t_n). \end{aligned}$$

□

It is also easy to show that the least Herbrand model is actually a model for P. The least Herbrand model is *universal*: a conjunction of atoms is valid in $T_{\Sigma,\Pi,P}$ if and only if it is valid in every other model of the program P. This is stated by the following theorem, whose proof may be done by induction on the depth of the derivation $P \vdash_{\text{mp}} A$.

Theorem 3.6 *(Universality of $T_{\Sigma,\Pi,P}$)*
$T_{\Sigma,\Pi,P} \models A_1, \ldots, A_n \Leftrightarrow \forall \mathcal{M} \in \mathbf{Mod}(P),\ \mathcal{M} \models A_1, \ldots, A_n$.

The theorem, using Definition 3.1 and Proposition 3.5, may be rewritten as follows:

$$T_{\Sigma,\Pi,P} \models A_1, \ldots, A_n \Leftrightarrow A_1, \ldots, A_n \in \mathbf{TH}(P),\ \text{i.e.}$$

$$T_{\Sigma,\Pi,P} \models A_1, \ldots, A_n \Leftrightarrow P \vdash_{\text{mp}} A_1, \ldots, A_n.$$

In this way, we obtain operational-style semantics for our programs: the theory of P is just what we can prove using modus ponens and starting from P. However, one of the most interesting things in Logic Programming is to work with existential quantification, i.e. to be able to solve queries such as $\exists x.p(x)$ (i.e. $\leftarrow p(x)$). It is worth noticing that it is this possibility that provides the duality between logic and computation which is fundamental in Logic Programming: evaluating the truth of a query is equivalent to finding a witness. Modus ponens does not enable us to do this: if we want to show that a formula is valid in $T_{\Sigma,\Pi,P}$ we have no strategy to guide us towards our goal. In general, we have to enumerate all the possible derivations (and to consider all the possible witnesses, too). In other words, modus ponens is not "*goal-oriented*". We shall consider effective operational semantics for logic programs in Section 5.

We have also seen that in the least Herbrand model all the consequences of a program, and only these, are valid. A full characterization of this model may be provided by showing that it satisfies all and only the queries which are satisfied in any model of the program. It is easy to see that the least Herbrand model $T_{\Sigma,\Pi,P}$ is initial in $\mathbf{Alg}_{\Sigma,\Pi}$. This is useful in the proof of the following theorem.

Theorem 3.7 *(Herbrand's theorem)*
$T_{\Sigma,\Pi,P} \models \leftarrow B_1, \ldots, B_m$ *if and only if* $\forall \mathcal{M} \in \mathbf{Mod}(P)$, $\mathcal{M} \models \leftarrow B_1, \ldots, B_m$.

Proof. ($\Leftarrow$) Trivial.

($\Rightarrow$) Let $\alpha: X \to T_{\Sigma,\Pi,P}$ be an evaluation and let's assume that

$$T_{\Sigma,\Pi,P} \models_\alpha B_1, \ldots, T_{\Sigma,\Pi,P} \models_\alpha B_m,$$

i.e. that $\alpha B_1, \ldots, \alpha B_m \in \pi T_{\Sigma,\Pi,P}$. Then, if ϕ is the unique homomorphism from $T_{\Sigma,\Pi,P}$ to $\mathcal{M}$, by definition of homomorphism we have that $\phi\alpha B_1, \ldots, \phi\alpha B_m \in \pi\mathcal{M}$, i.e. $\mathcal{M} \models_{\phi\alpha} B_1, \ldots, \mathcal{M} \models_{\phi\alpha} B_m$, so that $\phi\alpha$ is a witness. □

Until now we considered evaluations α which substituted terms with elements in the algebra; the previous result, however, may be generalized by allowing terms with variables as witnesses.

Theorem 3.8 *(Generalized Herbrand's theorem)*
There exists a substitution $\theta: X \to T_{\Sigma,\Pi}(X)$ *such that* $T_{\Sigma,\Pi,P} \models \theta B_1, \ldots, \theta B_m$ *if and only if* $\forall \mathcal{M} \in \mathbf{Mod}(P)$, $\mathcal{M} \models \leftarrow B_1, \ldots, B_m$.

In general, it is worth noticing that if we have a witness $\alpha: X \to T_{\Sigma,\Pi,P}$, we can always translate it into a witness $\phi\alpha: X \to \mathcal{M}$ by means of the unique homomorphism $\phi: T_{\Sigma,\Pi,P} \to \mathcal{M}$. Furthermore, if there is a substitution $\theta: X \to T_{\Sigma,\Pi}(X)$, which makes a conjunction of atoms valid in $T_{\Sigma,\Pi,P}$, we can consider it as a witness that the conjunction may be satisfied in $\mathcal{M}$ by any evaluation $\alpha: T_{\Sigma,\Pi}(X) \to \mathcal{M}$.

It is interesting to note that the assumption of non-emptiness of the set of constant symbols of a signature $\langle \Sigma, \Pi \rangle$ plays a central role in this translation. Without such an assumption, a substitution $\theta: X \to T_{\Sigma,\Pi}(X)$ is not always a witness, both in the mathematical and the intuitive meaning of the word. As a matter of fact, it can be shown that, starting from θ, we can find a witness if and only if $T_{\Sigma,\Pi}$ is not empty, and this happens if and only if Σ_0 is not empty. To understand this better, we notice that

$$T_{\Sigma,\Pi,P} \models \theta A_1, \ldots, \theta A_n \Rightarrow T_{\Sigma,\Pi,P} \models \leftarrow A_1, \ldots, A_n,$$

but the converse is true if and only if $T_{\Sigma,\Pi}$ is not empty.

4 Procedural Semantics

One of the classical ways of providing semantics to logic programs are the so called fixpoint semantics, where the set of logical consequences is obtained as the least fixpoint of an endofunction on the set of the subsets of the ground atoms. In this section we recall this approach and study its relationship with the declarative semantics given in the previous section.

4.1 The Least fixpoint

Given a lattice L and a function $T: L \to L$, a *fixpoint* for T is an element $l \in L$ such that $T(l) = l$. We call *lfp*(T) the *least fixpoint* for T, if it exists.

The most important consequence of the continuity of an endofunction on a lattice is that the set of its fixpoints is a complete lattice. Furthermore, we can represent the minimum of these fixpoints in a very simple way. In the following, we write $T^{(\omega)}(e)$ to denote $\bigsqcup_n T^{(n)}(e)$. Obviously, this notation makes sense only if the least upper bound of $T^{(n)}(e)$ on $\mathbb{N}$ exists.

Theorem 4.1 *(Continuity ⇒ Least fixpoint)*
Let $T: L \to L$ *be a continuous function. Then,* T *has a least fixpoint,* $T^{(\omega)}(\bot)$*, where* $\bot$ *is the minimum of* L.

4.2 Construction of $T_{\Sigma,\Pi,P}$ as a fixpoint

This approach to the semantics of logic programs is centered on defining an endofunction on the Herbrand base $B_{\Sigma,\Pi}$ of a program P such that it has a least fixpoint which is the set of consequences of P.

Definition 4.2 *(Immediate consequence operator)*
Let P be a program on the signature $\langle \Sigma, \Pi \rangle$ and $T_P \colon 2^{B_{\Sigma,\Pi}} \to 2^{B_{\Sigma,\Pi}}$ the function which maps $B \subseteq B_{\Sigma,\Pi}$ to the set $T_P(B)$ defined as follows:

$$A \in T_P(B) \;\Leftrightarrow\; \begin{array}{l} \exists \theta \colon X \to T_{\Sigma,\Pi} \text{ and } A' \leftarrow B_1, \ldots, B_n \in P \\ \text{such that } \theta B_1, \ldots, \theta B_n \in B \text{ and } \theta A' = A \end{array}$$

We call the function $T_P(B)$ immediate consequence operator*.*

It is possible to show that T_P is continuous, and this implies, as we said earlier, that T_P has a least fixpoint, $T_P^{(\omega)}(\emptyset)$, which we shall call M_P. M_P includes all and only the consequences of P, as we shall now see. For this reason, it is in a sense equivalent to $\models_{\text{mp}}$ and to $T_{\Sigma,\Pi,P}$: if we define the algebra $T_{\Sigma,\Pi}/P$ as the $\langle \Sigma, \Pi \rangle$-algebra obtained from $T_{\Sigma,\Pi}$ by assuming

$$(t_1, \ldots, t_n) \in \rho_{T_{\Sigma,\Pi}/P} \Leftrightarrow \rho(t_1, \ldots, t_n) \in M_P$$

we can show that the following theorem holds.

Theorem 4.3 *($T_{\Sigma,\Pi}/P$ is equal to $T_{\Sigma,\Pi,P}$)*
$T_{\Sigma,\Pi}/P = T_{\Sigma,\Pi,P}$.

5 Operational Semantics

In this section we give operational semantics to logic programs by means of the SLD$_{EQ}$–Resolution. SLD$_{EQ}$–Resolution is a *goal oriented* proof system. The situation is exactly reversed from what we had with *modus ponens*: to answer a query, we now start from the query itself, and the substitution which satisfies the query —the witness— is calculated through the resolution itself.

To find this witness, at each step of the calculation we need to solve equations to *unify* formulas with variables (i.e. to make them syntactically identical). It is not important which algorithm we use to solve these equations: we need only to make sure that it satisfies a few simple properties. in this way, we are able to give a definition of SLD$_{EQ}$–Resolution which is parametrical with respect to the chosen equation solver.

In order to have a clear distinction between the properties of the search method used to find a witness and the properties of the equation solver, we shall introduce first some basic concepts of equations, solutions and solvers.

5.1 Equations

Definition 5.1 *(Equations and Equation systems)*
Given a signature $\langle \Sigma, \Pi \rangle$ and a set of variables X, an equation *over $\langle \Sigma(X), \Pi \rangle$ is a pair of terms $\langle t_1, t_2 \rangle$ (usually written as $t_1 = t_2$) where $t_i \in T_\Sigma(X)$.*
A finite set of equations $E = \{E_1, \ldots, E_n\}$ is called equation system*.*

The solution of an equation is, as usual, an assignment of values to variables which makes the terms involved equal in the equation domain. In our case, the domain is $T_{\Sigma,\Pi}(X)$, so that solving $t_1 = t_2$ is equivalent to finding a substitution $\theta\colon X \to T_\Sigma(X)$ that, when applied to t_1 and t_2, makes them syntactically the same. We say that $t_1 = t_2$ is *consistent* if it has at least a solution.

We also say that θ is a solution of E if it is a solution of each equation E_i in E. We denote by $Sol[E]$ the set of the solutions of E. This set is a set of substitutions; for this reason, using the ordering on sets defined, as usual, by inclusion, we can define a *preorder* on sets of equations.

Definition 5.2 *(Preorder on sets of equations)*
There exists a natural preorder *on sets of equations, defined by:*

(i) $E_1 \leq E_2 \Leftrightarrow Sol[E_1] \subseteq Sol[E_2]$
(ii) $E_1 \approx E_2 \Leftrightarrow E_1 \leq E_2$ *and* $E_2 \leq E_1$

In other words, $E_1 \leq E_2$ means that any solution of E_1 is also a solution of E_2, whereas $E_1 \approx E_2$ means that they have exactly the same solutions. Obviously, these solution may well be syntactically very different from each other: for example, all systems which are not consistent, i.e. without solutions, are in the same relation $\approx$. It is easy to show that $\approx$ is an equivalence relation.

We are interested in a particular type of equation systems, which are special because their solution is explicitly included in the system itself: these are the systems in *solved form.*

Definition 5.3 *(Systems in solved form)*
A system of equations is in solved form *if it has the form* $R = \{x_1 = t_1, \ldots, x_n = t_n\}$*, where* $x_i \in X$ *are all distinct and* $t_i \in T_{\Sigma,\Pi}(X \setminus \{x_1 \ldots x_n\})$.
Given a system in solved form, $R = \{x_1 = t_1, \ldots, x_n = t_n\}$*, we denote by* $\hat{R}$ *the substitution given by* $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$*, i.e. the only substitution such that* $\hat{R}x_i = t_i$*, for* $i = 1, \ldots, n$.

It is easy to show that the substitution $\hat{R}$ is idempotent.

Definition 5.4 *(Preorder on substitutions)*
Let θ *and* η *be two substitutions; we say that* θ is more general than η *or that* η is a specialization of θ *(denoted by* $\theta \prec \eta$*) if there exists a substitution* ε *such that* $\eta = \varepsilon\theta$.

It is easy to see that the relation $\prec$ is a *preorder.*

An important set which we can associate to a given substitution θ is the set of its specializations, i.e. the set of the substitutions which "factor" via θ.

Definition 5.5 *(Specialization set)*
Given a substitution $\theta\colon X \to T_{\Sigma,\Pi}(X)$*, we denote by* $Sp[\theta]$ *the set of the specializations of* θ*, i.e.* $\{\eta\colon X \to T_{\Sigma,\Pi}(X) \mid \theta \prec \eta\}$.

Given a system E and an equivalent system in solved form, the relationship between the solutions of E and the specializations of the substitution provided by the system in solved form is shown in the following theorem.

Theorem 5.6 *(Most general solution of a system)*
Let E *be an equation system and* R *a system in solved form such that* $E \approx R$*; then* $Sol[E] = Sp[\hat{R}]$*. The solution* $\hat{R}$ *is called* most general solution *of the system.*

The binary relation $\vdash_{\text{mm}}$ between equation sets E and sets of equations extended by adding $\bot$, is the smallest closed relation w.r.t. the following transition rules:

$$(1)\quad \frac{E \cup \{f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)\}}{E \cup \{s_1 = t_1,\ldots,s_n = t_n\}};$$

$$(2)\quad \frac{E \cup \{f(s_1,\ldots,s_n) = g(t_1,\ldots,t_n)\}}{\bot};$$

$$(3)\quad \frac{E \cup \{x = x\}}{E};$$

$$(4)\quad \frac{E \cup \{t = x\}}{E \cup \{x = t\}}\ t \text{ not a variable};$$

$$(5)\quad \frac{E \cup \{x = t\}}{\bot}\ x \in var(E) \text{ s.t. } x \neq t \text{ and } x \in var(t);$$

$$(6)\quad \frac{E \cup \{x = t\}}{\theta E \cup \{x = t\}}\ x \in var(E) \text{ s.t. } x \neq t \text{ and } x \notin var(t).$$

where $\theta\colon X \to T_{\Sigma,\Pi}(X)$ is such that $\theta x = t,\ \theta E = \{\theta t' = \theta t'' \mid (t' = t'') \in E\}$.

Figure 2: Unification

Proof. By definition of $\approx$, $Sol[E] = Sol[R]$. So, we need just to show that

$$Sol[R] = Sp[\hat{R}].$$

Let $R = \{x_1 = t_1,\ldots,x_n = t_n\}$; obviously, $\hat{R}$ is a solution:

$$\hat{R}x_i = t_i = \hat{R}(t_i),\ \text{for } i = 1,\ldots,n,$$

because, as R is in solved form, t_i does not include any occurrence of x_i. Then, $\forall \mu \in Sp[\hat{R}],\ \mu = \varepsilon\hat{R} \Rightarrow \varepsilon\hat{R}x_i = \varepsilon t_i = \varepsilon\hat{R}t_i,\ \text{for } i = 1,\ldots,n \Rightarrow \mu \in Sol[R]$. If we now look at $\theta \in Sol[R]$ we notice that, by definition of solution, $\theta x_i = \theta t_i$, for $i = 1,\ldots,n$. For this reason, we have that $\theta\hat{R}x_i = \theta t_i = \theta x_i$. At the same time, for any $x \notin \{x_1,\ldots,x_n\}$, we have $\hat{R}x = x$ that implies $\theta\hat{R}x = \theta x$. Using these two equivalences, we can easily show that $\theta = \theta\hat{R}$, i.e. $\theta \in Sp[\hat{R}]$. □

The intuitive meaning of this theorem is that R includes all the information we need about the solutions of E; any solution can be generated as a specialization of $\hat{R}$. We also notice that the choice of the solved system equivalent to E does not really matter: any $R' \approx R$ will have the same properties, which is the only concern we have, as

$$Sol[E] = Sol[R'] = Sol[R] = Sp[\hat{R}] = Sp[\hat{R}'].$$

As a matter of fact, it is possible to show that any $R' \approx R$ is *isomorphic* to it, as it is obtained by R through a simple change of variables.

For this reason, in the rest of this work we shall not make distinctions between the systems in solved form equivalent to E, as well as between the substitutions corresponding to them; as usual, we shall define our objects *up to isomorphisms*.

Definition 5.7 *(Equation solver)*
An equation solver *is a formal proof system $\vdash_{\text{eq}}$ which associates to each equation system E an equivalent equation system in solved form R, if and only if E is consistent.*
We shall denote this by $E \vdash_{\text{eq}} R$ and sometimes we shall denote R by $\vdash_{\text{eq}} E$.

Let's look at some properties of these proof systems which will be useful later on.

Lemma 5.8 *(Properties of equation solvers)*
The following propositions are equivalent:

(i) $E \cup E' \models_{\overline{eq}} R$;
(ii) $(E \models_{\overline{eq}} R_1)$ *and* $(E' \cup R_1 \models_{\overline{eq}} R)$;
(iii) $(E' \models_{\overline{eq}} R_2)$ *and* $(E \cup R_2 \models_{\overline{eq}} R)$;
(iv) $(E \models_{\overline{eq}} R_1)$ *and* $(E' \models_{\overline{eq}} R_2)$ *and* $(R_1 \cup R_2 \models_{\overline{eq}} R)$.

Proof. As $E \approx R_1$ and $E' \approx R_2$ we have that:

$$E \cup E' \approx E \cup R_2 \approx R_1 \cup E' \approx R_1 \approx R_2.$$

So, the solutions calculated via $\models_{\overline{eq}}$ are the same (are isomorphic) in all these cases. □

Lemma 5.9 *(Union of equation systems)*
Let $E_1 = \{t_1 = t'_1, \ldots, t_n = t'_n\} \models_{\overline{eq}} R_1$ *and* $E_2 = \{s_1 = s'_1, \ldots, s_n = s'_n\} \models_{\overline{eq}} R_2$ *and let's assume that* $\exists\theta\colon X \to T_{\Sigma,\Pi}(X)$ *such that* $\hat{R}_1 \prec \theta$ *and* $\hat{R}_2 \prec \theta$. *Then,* $E_1 \cup E_2 \models_{\overline{eq}} R$, *and* $\hat{R} \prec \theta$.

Proof. As $\hat{R}_1$ is a solution of E_1, we have that: $\hat{R}_1 t_1 = \hat{R}_1 t'_1, \ldots, \hat{R}_1 t_n = \hat{R}_1 t'_n$ and $\theta = \varepsilon\hat{R}_1 \Rightarrow \theta t_1 = \theta t'_1, \ldots, \theta t_1 = \theta t'_1$. So, by definition of solution, $\theta \in Sol[E_1]$. In a similar way, using $\hat{R}_2$, we find that $\theta \in Sol[E_2]$. So, θ is a solution of $E_1 \cup E_2$, i.e. $\theta \in Sol[E_1 \cup E_2]$. Then $E_1 \cup E_2$ is consistent and so $\exists R$, $(E_1 \bigcup E_2) \models_{\overline{eq}} R$. By definition, it follows that $\hat{R} \prec \theta$. □

In Figure 2 we show an example of an equation solver, $\models_{\overline{eq}}$, in order to better show what, in practice, a formal system of this kind entails. The example is based on the Martelli – Montanari algorithm for unification. It is possible to show that such a system is correct and complete.

5.2 SLD$_{EQ}$–Resolution

In this section we present the SLD$_{EQ}$–Resolution. We shall see that the semantics obtained in this way are equivalent to the semantics defined earlier by means of $T_{\Sigma,\Pi,P}$. First, we must give some definitions.

Definition 5.10 *(Resolvent)*
A resolvent *is a formula of one of the following types:*

(i) $[\![B_1, \ldots, B_n$,
(ii) $R[\![B_1, \ldots, B_n$,
(iii) $R[\![$,

where $B_1 \ldots B_n$ *are atoms and* R *is a system of equations in solved form.*

Definition 5.11 *(Resolution step)*
Let $N = R[\![\rho(t_1, \ldots, t_n), A_1, \ldots, A_n$ *be a resolvent and* $C = \rho(t'_1, \ldots, t'_n) \leftarrow B_1, \ldots, B_m$ *a defined clause. If* $R \cup \{t_1 = t'_1, \ldots, t_n = t'_n\} \models_{\overline{eq}} R'$, *then the resolvent*

$$N' = R'[\![B_1, \ldots, B_m, A_1, \ldots, A_n$$

is called resolvent of N and C with selected atom $\rho(t_1 \ldots t_n)$.

Definition 5.12 *(SLD$_{EQ}$–Derivation)*
An SLD$_{EQ}$–Derivation *of a resolvent N, using a program P, is a maximal sequence of resolvents $N_0, N_1, N_2, \ldots$ where $N = N_0$, and each N_{i+1} is obtained as resolvent of N_i and of a clause C_i; C_i is obtained from a clause $C \in P$ renaming its variables in such a way that it has not any variable in common with $N_0, C_0, \ldots, C_{i-1}$.*

We are obviously interested in *finite* SLD$_{EQ}$–Derivations, which we shall denote by $P \vdash_{\text{sld}} N_0 \rightsquigarrow N_k$. Furthermore, these derivations must have the following property.

Definition 5.13 *(SLD$_{EQ}$–Refutation)*
An SLD$_{EQ}$–Derivation where one of the resolvents is of the type $R[\![$, so that it is the last resolvent in the derivation, is called SLD$_{EQ}$–Refutation*.*

We can now show that SLD$_{EQ}$–Resolution is *sound*.

Lemma 5.14 *(Soundness)*
If $P \vdash_{\text{sld}} R_1[\![A_1, \ldots, A_n \rightsquigarrow R_2[\![B_1, \ldots, B_k$, then $T_{\Sigma,\Pi,P} \models \hat{R}_2 A_1 \ldots \hat{R}_2 A_n \leftarrow \hat{R}_2 B_1 \ldots \hat{R}_2 B_k$.

Proof. By induction on the length of the derivation. The initial step is trivial: if the derivation has zero length we do not have anything to prove, as in this case, $R_1 = R_2$, $A_1, \ldots, A_n = B_1, \ldots, B_k$ and the thesis is obviously true.

Let's now consider a derivation with length n:

$$\underline{R_1[\![A_1, \ldots, A_n}$$

$$\vdots$$

$$\frac{R_2[\![\rho(t_1, \ldots, t_n), C_1, \ldots, C_m}{R_3[\![B_1, \ldots, B_k, C_1, \ldots, C_m}$$

By induction on the derivation of length $(n-1)$ we have that:

$$T_{\Sigma,\Pi,P} \models \hat{R}_2 A_1, \ldots, \hat{R}_2 A_n \leftarrow \hat{R}_2 \rho(t_1 \ldots t_n), \hat{R}_2 C_1, \ldots, \hat{R}_2 C_m.$$

The n-th resolvent is obtained from the clause $\rho(t'_1 \ldots t'_n) \leftarrow B_1, \ldots, B_k$ which is valid in $T_{\Sigma,\Pi,P}$ as it is the renaming of a clause in P, and with

$$R_2 \cup \{t_1 = t'_1, \ldots, t_n = t'_n\} \vdash_{\text{eq}} R_3.$$

Obviously, as $\hat{R}_3 \rho(t_1, \ldots, t_n) = \hat{R}_3 \rho(t'_1, \ldots, t'_n)$ and the clause is valid in $T_{\Sigma,\Pi,P}$, we have that $T_{\Sigma,\Pi,P} \models \hat{R}_3 \rho(t_1, \ldots, t_n) \leftarrow \hat{R}_3 B_1, \ldots, \hat{R}_3 B_k$, which implies

$$T_{\Sigma,\Pi,P} \models \hat{R}_3 \rho(t_1, \ldots, t_n), \hat{R}_3 C_1, \ldots, \hat{R}_3 C_m \leftarrow \hat{R}_3 B_1, \ldots, \hat{R}_3 B_k, \hat{R}_3 C_1, \ldots, \hat{R}_3 C_m.$$

Let $\theta\colon X \to T_{\Sigma,\Pi,P}(X)$. Let's assume that $T_{\Sigma,\Pi,P} \models_\theta \hat{R}_3 B_1, \ldots, \hat{R}_3 B_k, \hat{R}_3 C_1, \ldots, \hat{R}_3 C_m$. Then, $T_{\Sigma,\Pi,P} \models_\theta \hat{R}_3 \rho(t_1, \ldots, t_n), \hat{R}_3 C_1, \ldots, \hat{R}_3 C_m$ i.e. $\theta\hat{R}_3(t_1, \ldots, t_n) \in \rho_{T_{\Sigma,\Pi,P}}$, $\theta\hat{R}_3 C_1, \ldots, \theta\hat{R}_3 C_1 \in \pi T_{\Sigma,\Pi,P}$. As $\hat{R}_3$ is also a solution of R_2 we have that, given $\theta\hat{R}_3$, $\exists\eta\colon X \to T_{\Sigma,\Pi,P}(X)$ such that $\theta\hat{R}_3 = \eta\hat{R}_2$. Then, by induction hypothesis, we have that $\eta\hat{R}_2 A_1, \ldots, \eta\hat{R}_2 A_n \in \pi T_{\Sigma,\Pi,P}$, which is equivalent to saying that $T_{\Sigma,\Pi,P} \models_\theta \hat{R}_3 A_1, \ldots, \hat{R}_3 A_n$.
So,

$$T_{\Sigma,\Pi,P} \models_\theta \hat{R}_3 A_1, \ldots, \hat{R}_3 A_n \leftarrow \hat{R}_3 B_1, \ldots, \hat{R}_3 B_k, \hat{R}_3 C_1, \ldots, \hat{R}_3 C_m.$$

This concludes our proof. □

As the empty conjunction is valid in any interpretation, we are able to prove the following theorem and corollary by applying the previous lemma.

Theorem 5.15 *(Soundness, part 1)*
If $P \vdash_{\text{sld}} [\![A_1, \ldots, A_n \rightsquigarrow R]\!]$, *then* $T_{\Sigma,\Pi,P} \models \hat{R}A_1, \ldots, \hat{R}A_n$.

Corollary 5.16 *(Soundness, part 2)*
If $P \vdash_{\text{sld}} A_1, \ldots, A_n \rightsquigarrow R[\![$ *then* $\mathcal{M} \models \leftarrow \hat{R}B_1, \ldots, \hat{R}B_n,\ \forall \mathcal{M} \in Mod(P)$.
Furthermore, $\mathcal{M} \models \leftarrow B_1, \ldots, B_n\ \forall \mathcal{M} \in Mod(P)$, *with witness* $\phi\hat{R},\ \forall \phi \colon X \to \mathcal{M}$.

We can now prove the completeness of SLD_{EQ}–Resolution. First, we need to introduce some lemmas.

Lemma 5.17 *(Completeness, part 1)*
If $P \vdash_{\text{sld}} R_B[\![B_1, \ldots, B_n \rightsquigarrow R_1[\![C_1, \ldots, C_k$ *and* $P \vdash_{\text{sld}} [\![A \rightsquigarrow R_2[\![D_1, \ldots, D_m$ *with* $\hat{R}_1 \prec \theta$ *and* $\hat{R}_2 \prec \theta$, *then* $P \vdash_{\text{sld}} R_B[\![B_1, \ldots, B_n, A \rightsquigarrow R[\![C_1, \ldots, C_k, D_1, \ldots, D_m$ *with* $\hat{R} \prec \theta$.

Proof. By induction on the maximum length of the two derivations.

If both derivations have a length $n = 0$ we have nothing to show, as in this case the first and last resolvent in a derivation are identical.

We consider now two derivations with a length of n:

$$\begin{array}{cc} \underline{R_B[\![B_1, \ldots, B_n} & \underline{[\![A} \\ \vdots & \vdots \\ \dfrac{R_1'[\![\rho(t_1, \ldots, t_n), B_1', \ldots, B_k'}{R_1[\![B_1, \ldots, B_l, B_1', \ldots, B_k'} & \dfrac{R_2'[\![\rho'(s_1, \ldots, s_n), A_1', \ldots, A_n'}{R_2[\![A_1, \ldots, A_p, A_1', \ldots, A_n'} \end{array},$$

where the last resolvents are obtained by applying the rules:

$$\rho(t_1', \ldots, t_n') \leftarrow B_1, \ldots, B_l \text{ and } \rho'(s_1', \ldots, s_n') \leftarrow A_1, \ldots, A_p.$$

By induction hypothesis, as $R_1' \prec R_1 \prec \theta$ and $R_2' \prec R_2 \prec \theta$, we have that

$$P \vdash_{\text{sld}} [\![B_1, \ldots, B_n, A \rightsquigarrow$$

$$\rightsquigarrow R'[\![B_1, \ldots, B_l, B_1', \ldots B_k', A_1, \ldots, A_p, A_1', \ldots, A_n', \rho(t_1, \ldots, t_n), q(s_1, \ldots, s_n),$$

with $\hat{R}' \prec \theta$, i.e.

$$\begin{array}{c} \underline{R_B[\![B_1, \ldots, B_n, A} \\ \vdots \\ \overline{R'[\![B_1', \ldots, B_k', A_1', \ldots, A_n', \rho(t_1, \ldots, t_n), \rho'(s_1, \ldots, s_n)} \end{array}.$$

As $\{t_1 = t_1', \ldots, t_n = t_n'\} \prec R_1 \prec \theta$ and $\hat{R}' \prec \theta$, from Lemma 5.9 we have that $\{t_1 = t_1', \ldots, t_n = t_n'\} \cup R \vdash_{\text{eq}} R''$, with $\hat{R}'' \prec \theta$. So, we can go a step further in the derivation:

$$\begin{array}{c} \vdots \\ \overline{R''[\![B_1, \ldots, B_l, B_1', \ldots, B_k', A_1', \ldots, A_n', \rho'(s_1, \ldots, s_n)} \end{array}.$$

In a similar way, $\{s_1 = s_1', \ldots, s_n = s_n'\} \prec R_2 \prec \theta$ and $\hat{R}'' \prec \theta$; so $\{s_1 = s_1' \ldots s_n = s_n'\} \cup R'' \vdash_{\text{eq}} R$ with $\hat{R}'' \prec \theta$. With a final derivation step we then find:

$$\begin{array}{c} \vdots \\ \overline{R[\![B_1, \ldots, B_l, B_1', \ldots, B_k', A_1', \ldots, A_n', A_1, \ldots, A_n} \end{array},$$

which is just what was to be proved. □

Lemma 5.18 *(Completeness, part 2)*
If $P \vDash_{\text{sld}} [\![B_1 \rightsquigarrow R_1]\!], \ldots, P \vDash_{\text{sld}} [\![B_n \rightsquigarrow R_n]\!]$ *with* $\hat{R}_1 \ldots, \hat{R}_n \prec \theta$*, then* $P \vDash_{\text{sld}} [\![B_1 \ldots B_n \rightsquigarrow R]\!]$ *with* $\hat{R} \prec \theta$*.*

Proof. By repeated application of the previous lemma. □

Lemma 5.19 *(Completeness, part 3)*
If $P \vDash_{\text{sld}} [\![B_1, \ldots, B_n \rightsquigarrow R_1]\!]$ *and* $\hat{R}_1 \prec \theta$ *and there exists* $\hat{R}_2 \prec \theta$*, then* $P \vDash_{\text{sld}} R_2 [\![B_1, \ldots, B_n \rightsquigarrow R]\!]$ *with* $\hat{R} \prec \theta$*.*

Definition 5.20 *(Correct answer substitution and Computed answer substitution)*
Let $\leftarrow A_1, \ldots, A_n$ *be a goal and let* $V \subseteq X$ *be the set of its variables. We call* $\theta \colon V \to T_{\Sigma,\Pi}(X)$ *a* correct answer substitution *if and only if* $T_{\Sigma,\Pi,P} \models \theta A_1, \ldots, \theta A_n$*.*
We call $\hat{R}_{|V}$ *a* computed answer substitution *if and only if* $P \vDash_{\text{sld}} [\![A_1, \ldots, A_n \rightsquigarrow R]\!]$*.*

We can now show the completeness theorem.

Theorem 5.21 *(Completeness, part 1)*
If $T_{\Sigma,\Pi,P} \models \theta A_1, \ldots, \theta A_n$ *where* $\theta \colon V \to T_{\Sigma,\Pi}(X)$ *is a correct answer substitution, then* $\exists P \vDash_{\text{sld}} [\![A_1, \ldots, A_n \rightsquigarrow R]\!]$ *with* $\hat{R}_{|V} \prec \theta$*.*

Proof. From Proposition 3.5 we know that:

$$T_{\Sigma,\Pi,P} \models \theta A_1, \ldots, \theta A_n \Leftrightarrow P \vDash_{\text{mp}} \theta A_1, \ldots, \theta A_n.$$

We can prove the theorem by induction on the depth of the derivation trees. Using Lemma 3.3, we can assume that the clauses used and any atom appearing in the deduction have not any variable in common.

Let's consider the base case, $n = 1$. We have that $P \vDash_{\text{mp}} \theta A_1, \ldots, P \vDash_{\text{mp}} \theta A_n$, with $A'_1 \leftarrow , \ldots, A'_n \leftarrow \in P$ and $\eta_i A'_i = \theta A_i$, $A'_i = \rho_i(t_1^{i'}, \ldots, t_{k_i}^{i'})$, $A_i = \rho_i(t_1^i, \ldots, t_{k_i}^i)$. From our hypothesis we can assume that any $\eta_i = \eta$, for some $\eta \colon V \to T_{\Sigma,\Pi}(X)$. We also have that $\eta\theta A_i = \eta\theta A'_i$.

So, $\exists E = \{t_1^i = t_1^{i'}, \ldots, t_{k_i}^i = t_{k_i}^{i'}\} \vDash_{\text{eq}} R_i$ and $\dfrac{[\![\rho_i(t_1^i, \ldots, t_{k_i}^i)}{R_i [\![}$ with $\hat{R}_i \prec \eta\theta$ per $i = 1 \ldots n$. Then, from Lemma 5.19:

$$\frac{\rho_1(t_1^1, \ldots, t_{k_1}^1), \ldots, \rho_n(t_1^n, \ldots, t_{k_n}^n)}{R [\![}$$

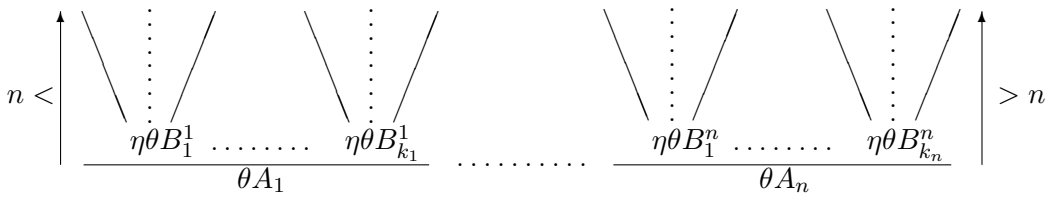
with $\hat{R} \prec \eta\theta$. From our hypothesis, $\eta\theta_{|V} = \theta$, so we can easily prove that (we do not show here a detailed proof) $\hat{R}_{|V} \prec \theta$.

Let's now consider the induction step. We have that

$$\frac{\begin{array}{cccc} \vdots & \vdots & & \vdots \\ P \vDash_{\text{mp}} \eta_1 B_1^1 & P \vDash_{\text{mp}} \eta_1 B_{k_1}^1 & \cdots & P \vDash_{\text{mp}} \eta_n B_1^n \end{array} \quad \cdots \quad \begin{array}{c} \vdots \\ P \vDash_{\text{mp}} \eta_n B_{k_n}^n \end{array}}{P \vDash_{\text{mp}} \theta A_1 \qquad\qquad\qquad P \vDash_{\text{mp}} \theta A_n}$$

where $\eta_i A'_i = \theta A_i$ and in the last step of each tree we used a renaming $A'_i \leftarrow B_1^i, \ldots, B_{k_i}^i$ of a rule in P. From our initial hypothesis we can now assume that all η_i are equal to η. Then, we have that $\eta\theta A'_i = \eta\theta A_i$; furthermore, $\eta B_k^i = \eta\theta B_k^i$.

We then get a forest of the following form:



And, by induction hypothesis,

$$P \vdash_{\text{sld}} [\![B_1^1, \ldots, B_{k_1}^1, \ldots, B_1^n, \ldots, B_{k_n}^n \rightsquigarrow R_1]\!]$$

with $\hat{R}_1 \prec \varepsilon\eta\theta$ and $\varepsilon\eta\theta_{|B} = \eta\theta$, where B is the set of variables appearing in $\eta\theta$ and ε is introduced because, using the induction hypothesis, we only know that $\hat{R}_{|B} \prec \eta\theta$.

As far as the last step is concerned, assuming that $A_i = p_i(t_1^i \ldots t_{k_i}^i)$ and $A_i' = p_i(t_1^{i'} \ldots t_{k_i}^{i'})$, we have that

$$E = \{t_1^1 = t_1^{1'}, \ldots, t_{k_1}^1 = t_{k_1}^{1'}, \ldots, t_1^n = t_1^{n'}, \ldots, t_{k_n}^n = t_{k_n}^{n'}\} \vdash_{\text{eq}} R_2$$

with $\hat{R}_2 \prec \eta\theta$. So, we have that $P \vdash_{\text{sld}} [\![A_1, \ldots, A_n \rightsquigarrow R_2 [\![B_1^1, \ldots, B_{k_1}^1, \ldots, B_1^n, \ldots, B_{k_n}^n$. As $\hat{R}_2 \prec \eta\theta \prec \varepsilon\eta\theta$, $\hat{R}_1 \prec \varepsilon\eta\theta$, from Lemma 5.19 we know that $P \vdash_{\text{sld}} [\![A_1, \ldots, A_n \rightsquigarrow R]\!]$ with $\hat{R} \prec \varepsilon\eta\theta$. As $V \subseteq B$ we have

$$\varepsilon\eta\theta_{|V} = (\varepsilon\eta\theta_{|B})_{|V} = \eta\theta_{|V} = \theta$$

and from this follows $\hat{R_{|V}} \prec \theta$, which ends the proof. □

The completeness theorem may also be rewritten as follows.

Theorem 5.22 *(Completeness, part 2)*
Given a goal $\leftarrow A_1, \ldots, A_n$ *and a* correct answer substitution θ *there exists a* computed answer substitution $\hat{R}$ *such that* $\hat{R}_{|V} \prec \theta$*, where* V *is the set of the variables in* $\{A_1, \ldots, A_n\}$*.*

To conclude our work, we want to show that the soundness of SLD_{EQ}–Resolution is independent the way we select the atoms for the resolution. Formally, we say that a *selection rule* is a function $\mathcal{R}$ from resolvents to atoms, and that an SLD_{EQ}–Derivation $N_0, \ldots, N_i, \ldots$ is an SLD_{EQ}–Derivation *via* $\mathcal{R}$ if, at each step, the atom to expand in the resolvent N_i is $\mathcal{R}(N_i)$.

First, we show that if we switch the order of two resolvents in a derivation, we get an equivalent derivation.

Lemma 5.23 *(Switching)*
Given a finite SLD_{EQ}*–Derivation* $N_0, \ldots, N_i, N_{i+1}, \ldots, N_M$*, we can find an* SLD_{EQ}*–Derivation* $N_0, \ldots, N_i', N_{i+1}', \ldots, N_M$ *where the selected atom in* N_i' *is the same atom that was selected in* N_{i+1}*, and the selected atom in* N_{i+1}' *is the same atom that was selected in* N_i*.*

Proof. Let's assume

$$\begin{array}{rc} & N_0 \\ & \vdots \\ N_i = & \dfrac{}{R_1 [\![A, B, C_1, \ldots, C_m} \\ N_{i+1} = & \dfrac{}{R_2 [\![A_1, \ldots, A_n, B, C_1, \ldots, C_m} \\ & \dfrac{}{R_3 [\![A_1, \ldots, A_n, B_1, \ldots, B_k, C_1, \ldots, C_m} \\ & \vdots \\ & N_M \end{array}$$

where we have $A = \rho(t_1, \ldots, t_n)$, $B = \rho'(s_1, \ldots, s_n)$ and the clauses $\rho(t'_1, \ldots, t'_n) \leftarrow A_1, \ldots, A_n$ and $\rho'(s'_1, \ldots, s'_n) \leftarrow B_1, \ldots, B_k$.

Then, we have that $R_1 \cup \{t_1 = t'_1, \ldots, t_n = t'_n\} \models_{\text{eq}} R_2$ and $R_2 \cup \{s_1 = s'_1, \ldots, s_n = s'_n\} \models_{\text{eq}} R_3$, which implies that $R_1 \cup \{t_1 = t'_1, \ldots, t_n = t'_n\} \cup \{s_1 = s'_1, \ldots, s_n = s'_n\} \models_{\text{eq}} R_3$ and $R_1 \cup \{s_1 = s'_1, \ldots, s_n = s'_n\} \models_{\text{eq}} R'$, $R' \cup \{t_1 = t'_1, \ldots, t_n = t'_n\} \models_{\text{eq}} R_3$ from Lemma 5.8.

So, there exists an SLD_{EQ}–Derivation:

$$
\begin{array}{rc}
 & N_0 \\
 & \vdots \\
N'_i = & \dfrac{R_1 \| A, B, C_1, \ldots, C_m}{} \\
N'_{i+1} = & \dfrac{R_2 \| A, B_1, \ldots, B_k, C_1, \ldots, C_m}{R_3 \| A_1, \ldots, A_n, B_1, \ldots B_k, C_1, \ldots, C_m} \\
 & \vdots \\
 & N_M.
\end{array}
$$

□

Theorem 5.24 *(SLD_{EQ}–Refutations are independent from $\mathcal{R}$)*
The SLD_{EQ}–Refutations we get starting from a given program are the same under any selection rule, i.e.:

$$P \models_{\text{sld}} \| A_1, \ldots, A_n \rightsquigarrow R \| \text{via } \mathcal{R} \Leftrightarrow P \models_{\text{sld}} \| A_1, \ldots, A_n \rightsquigarrow R \| \text{via } \mathcal{R}'.$$

Proof. By induction on the length of the refutation. We do not have to show anything for the base case, $n = 1$.

Let's consider instead the induction step:

$$P \models_{\text{sld}} \| A_1, \ldots, A_n \rightsquigarrow R' \| B_1, \ldots, B_n \rightsquigarrow \ldots \rightsquigarrow R \| \text{via } \mathcal{R}.$$

Let A_i and A_j be the atoms selected by $\mathcal{R}$ and $\mathcal{R}'$. By repeated application of Lemma 5.23 to the derivation via $\mathcal{R}$, we get a refutation $P \models_{\text{sld}} \| A_1, \ldots, A_n \rightsquigarrow R \|$ where A_j is selected at the first step. By induction hypothesis, the rest of the refutation has an $(n-1)$ length, so there exists a refutation via $\mathcal{R}'$ with final resolvent $R\|$. Joining these two derivations, we get a refutation of $\leftarrow A_1, \ldots, A_n$ in $\leftarrow R \|$via $\mathcal{R}'$. □

6 Conclusions

In this paper, we characterized in a number of different ways the least Herbrand model and its main properties. To conclude our work, we would like to give a summary of these characterizations and their relationship.

Corollary 6.1 *(Summary)*
All the following facts are equivalent.

- (i) $T_{\Sigma,\Pi,P} \models A_1, \ldots, A_n$;
- (ii) $P \models_{\text{mp}} A_1, \ldots, A_n$;
- (iii) $A_1, \ldots, A_n \in \mathbf{TH}(P)$;
- (iv) $\forall \mathcal{M} \in \mathbf{Mod}(P),\ \mathcal{M} \models A_1, \ldots, A_n$;
- (v) $T_{\Sigma,\Pi}/P \models A_1, \ldots, A_n$;
- (vi) $\text{lfp}(\mathcal{T}_P) \models A_1, \ldots, A_m$;
- (vii) $\mathcal{T}_P^{(\omega)}(T_{\Sigma,\Pi}) \models A_1, \ldots, A_m$;
- $(viii)$ $P \models_{\text{sld}} \| A'_1, \ldots, A'_n \rightsquigarrow R \|$, *with* $\hat{R} A'_i = A_i$;
- (ix) $P \models_{\text{sld}} \| A'_1, \ldots, A'_n \rightsquigarrow R \| \text{via } \mathcal{R}, \forall \mathcal{R}$, *with* $\hat{R} A'_i = A_i$.

All the following facts are equivalent.

(i) $T_{\Sigma,\Pi,P} \models \leftarrow B_1, \ldots, B_m$;
(ii) $\forall \mathcal{M} \in \mathbf{Mod}(P),\ \mathcal{M} \models \leftarrow B_1, \ldots, B_m$;
(iii) $\exists \theta\colon X \to T_{\Sigma,\Pi}(X),\ T_{\Sigma,\Pi,P} \models \theta B_1, \ldots, \theta B_m$;
(iv) $T_{\Sigma,\Pi}/P \models \leftarrow B_1, \ldots, B_m$;
(v) $\mathrm{lfp}(\mathcal{T}_P) \models \leftarrow B_1, \ldots, B_m$;
(vi) $\mathcal{T}_P^{(\omega)}(T_{\Sigma,\Pi}) \models \leftarrow B_1, \ldots, B_m$;
(vii) $P \vdash_{\mathrm{sld}} [\![B_1, \ldots, B_m \rightsquigarrow R]\!]$ *with a witness* $\phi\hat{R},\ \forall \phi\colon X \to \mathcal{M}$;
$(viii)$ $P \vdash_{\mathrm{sld}} [\![B_1, \ldots, B_m \rightsquigarrow R]\!]$via $\mathcal{R}, \forall \mathcal{R}$, *with a witness* $\phi\hat{R},\ \forall \phi\colon X \to \mathcal{M}$.

ACKNOWLEDGEMENTS. The authors wish to thank Giorgio Levi for his valuable suggestions and encouragement in writing this work.

References

[1] K.R. Apt. *Introduction to Logic Programming*, TR-87-35, Department of Computer Science, University of Texas, 1988.

[2] G. Birkhoff. "On the Structure of Abstract Algebras", Proceedings of the Cambridge Philosophical Society, n. 31, 1935.

[3] G. Birkhoff and J. Lipson. "Heterogeneous Algebras", Journal of Combinatorial Theory, n. 8, 1970.

[4] J. Goguen and J. Meseguer. "Initiality, Induction, and Computability", in *Algebraic Methods in Semantics*, M. Nivat and J. Reynolds, Eds., Cambridge University Press, 1985.

[5] J. Goguen and J. Meseguer. "Models and Equality for Logical Programming", in *Proceedings of TAPSOFT '87*, LNCS 250, Springer-Verlag, 1987.

[6] R.A. Kowalski. "Predicate Logic as a Programming Language", in *Proceedings of IFIP '74*, North-Holland, 1974.

[7] J. Jaffar and J.L. Lassez. "Constraint Logic Programming", in *Proceedings of POPL '87*, ACM, 1987.

[8] J.W. Lloyd. *Foundations of Logic Programming*, Second Edition, Springer-Verlag, 1987.

[9] J.L. Lassez, M.J. Maher and K. Marriot. "Unification Revisited", in *Foundations of Deductive Databases and Logic Programming*, Ed. J. Minker, M. Kaufmann, Los Altos, 1988.

[10] A. Martelli and U. Montanari. "An Efficient Unification Algorithm", ACM Transactions on Programming Languages and Systems, vol.4, n.2, 1982.

[11] J. Meseguer. General Logics, in *Logical Colloquium '87*, North-Holland, 1987.