1 Preliminaries

In this chapter we first give a summary of the basic notations, terminology and results which will be used in this thesis. The treatment here is reduced to a list of definitions. For the terminology not explicitly shown and for a more motivated introduction, the reader can consult [**?**] for logic, [**?**, **?**, **?**] for the fixpoint theory and for the algebraic notions, and [**?**, **?**] for a specific introduction to logic programming. Here we essentially follows [**?**]. Some more specific notions will be introduced in the chapters were they are needed. The reader acquainted with logic programming can skip sections 2 and 3 (the non-standard terminology used in this thesis is shown at the end of section 3). Section 4 contains some preliminary results on unification and a lemma on SLD derivations that we will need in the following. In section **??** finally we show a categorical approach to the semantics of logic programs.

2 Basic logical and mathematical definitions

Logic programs are sets of first order formulas of a particular class. We then start with some basic logic notions necessary to define formally the syntax of logic programs.

2.0.1 First order logic

A *first order language* consists of an *alphabet* and of the (well formed) *formulas* defined on it. An alphabet contains two kind of symbols: the logical and the non-logical symbols. All the sets we consider below are denumerable.

The *logical symbols* are common to all the FOL languages and consist of the following sets

- logical connectives: $\wedge$ (conjunction), $\vee$ disjunction), $\neg$ (negation), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence),
- an infinite set V of *variables*,
- propositional constants: *true* and *false*,
- quantifiers: $\exists$ (there exists) and $\forall$ (for all),
- punctuation symbols: the parentheses (,) and the comma , .

The *non-logical* symbols determine a specific FOL language and consists of the following sets

- a one-sorted *signature* Σ,
- a set Π of *predicate symbols* of fixed arity.

An one-sorted signature is defined as a family of disjoint sets

$$\Sigma = \{\Sigma_n\}_{n<\omega}$$

where Σ_n is a set which contains the function symbols of arity n (i.e. the symbols which has n arguments). In the following, when no ambiguity arise, we denote by Σ both the family $\{\Sigma_n\}_{n<\omega}$ and the set $\bigcup_{n<\omega}\{\Sigma_n\}$. Function whose arity is 0 are called also constants. We assume that V, Σ and Π are pairwise disjoint and do not share any symbol with the other sets listed above.

We have chosen this slightly different definition of the non-logical symbols because it allows a natural algebraic treatment and it can easily be generalized to the case of S-sorted signatures, and hence to many-sorted first order languages. We will need many-sorted languages in chapter **??** when considering Constraint Logic Languages. In the following by signature we mean a one-sorted signature. Sometime it

will be used also the terminology *signature with predicate* to denote a pair $\langle \Sigma, \Pi \rangle$ where Σ is a signature and Π is a disjoint set of predicate symbols.

The set $\tau(\Sigma \cup V)$ of *terms* of the language is the set of terms on the signature Σ and on the (disjoint) set V of variables, and is defined inductively as the least $\tau(\Sigma \cup V)$ such that

1. $V \subseteq \tau(\Sigma \cup V)$,
2. $\Sigma_0 \subseteq \tau(\Sigma \cup V)$,
3. $\forall t_1, \ldots, t_n \in \tau(\Sigma \cup V)$ if $f \in \Sigma_n$ then $f(t_1, \ldots, t_n) \in \tau(\Sigma \cup V)$.

The set $\tau(\Sigma \cup V)$ of *ground terms* is obtained by setting $V = \emptyset$ in previous definition. Finally we can define the class of *formulas* of the language as follows

1. if $t_1, \ldots, t_n$ are terms and $p \in \Pi$ is a predicate symbol of arity n then $p(t_1, \ldots, t_n)$ is a formula (called *atom*),
2. *true* and *false* are formulas,
3. if F and G are formulas then $\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$ and $(F \leftrightarrow G)$ are formulas,
4. if F is a formula and x is a variable the $\forall x.F$ and $\exists x.F$ are formulas,
5. (no other string on the alphabet is a formula).

We use the standard convention that $\neg, \forall, \exists$ bind stronger than $\vee$ which in turn binds stronger than $\wedge$ which bind stronger than $\rightarrow$ and $\leftrightarrow$. Moreover we denote by $\forall x_1, \ldots x_n F$ the formula $\forall x_1 \ldots \forall x_n F$ and, if $x_1, \ldots, x_n$ are all the variables in the quantifier free formula F, we denote by $\forall F$ the formula $\forall x_1, \ldots, x_n F$.

2.0.2 Unification theory

We introduce now some general definitions and some results concerning unification that we will need in the following. The interested reader can see [**?**, **?**, **?**] for more details on substitutions, equations and unification.

An *equation* is an atom $s = t$, where s, t are terms and $=$ is a predicate symbol which is interpreted as the syntactic equality on the Herbrand universe ($=$ is written in infix form to improve the readability). A *substitution* ϑ is a finite mapping from variables to terms and it is written as

$$\vartheta = \{x_1/t_1, \ldots, x_n/t_n\}$$

Note that the notation implies that the variables $x_1, \ldots, x_n$ are different. Moreover we assume that $t_i \not\equiv x_i$ for $i = 1, \ldots, n$. A pair x_i/t_i is called a binding and if $t_1, \ldots, t_n$ are ground then ϑ is called ground. ε denotes the empty substitution. For ϑ as before we define

$$\begin{array}{ll} Dom(\vartheta) & = \{x_1, \ldots, x_n\}, \\ Range(\vartheta) & = \{y \mid y \text{ occurs in } t_i \text{ for some } t_i\ \} \\ Var(\vartheta) & = Dom(\vartheta) \cup Range(\vartheta) \end{array}$$

An *expression* is either a term, a literal or a conjunction or a disjunction of literals. A *simple expression* is either a term or an atom. The result of an application of a substitution ϑ to an expression E, denoted by $E\vartheta$, is obtained by simultaneously replacing all the free occurrences of the x_i's in E by the corresponding t_i's. $E\vartheta$ is called an *instance* of E. If $W \subseteq V$, we denote by $\vartheta_{|W}$ the *restriction* of ϑ to the variables in W defined as follows

$$\vartheta_{|W}(Y) = \begin{cases} Y & \text{for } Y \notin W \\ Y\vartheta & \text{for } Y \in W \end{cases}$$

The *composition* $\vartheta\sigma$ of the substitutions $\vartheta = \{x_1/t_1, \ldots, x_n/t_n\}$ and $\sigma = \{y_1/s_1, \ldots, y_m/s_m\}$ is defined as the substitution obtained by removing from the set

$$\{x_1/t_1\sigma, \ldots, x_n/t_n\sigma, y_1/s_1, \ldots, y_m/s_m\}$$

those pairs $x_i/t_i\sigma$ such that $x_i \equiv t_i\sigma$ and those pairs y_i/s_i such that $y_i \in \{x_1, \ldots, x_n\}$. The composition is associative, ϵ is the neutral element, and for any term t, $t(\gamma\sigma) = (t\gamma)\sigma$ holds. A *renaming* is a substitution ρ for which there exists the inverse ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \varepsilon$.

We can define a pre-order $\leq$ (more general than) on substitutions by defining $\vartheta \leq \sigma$ iff there exists a substitution γ such that $\vartheta\gamma = \sigma$. Similarly, given two terms t and t', we define $t \leq t'$ (t is more general than t') iff there exists ϑ such that $t\vartheta = t'$. The relation $\leq$ is a preorder and by $\equiv_v$ we denote the associated equivalence relation which is called *variance*. Therefore t and t' are variants iff t is an instance of t' and vice versa. It easy to see that this definition is equivalent to say that t and t' are variants iff there exists a renaming ρ such that $t \equiv t'\rho$. This definitions can be extended to expressions in the obvious way.

Given two atoms A and B, a substitution ϑ is an *unifier* of A and B if $A\vartheta \equiv B\vartheta$. Given a set of equations $E = \{s_1 = t_1, \ldots, s_n = t_n\}$, the substitution ϑ is an unifier for E iff it is an unifier of $p(s_1, \ldots, s_n)$ and $p(t_1, \ldots, t_n)$. A unifier of A and B is called the *most general unifier* (mgu) of A and B if it is more general than any other unifier of A and B. The following theorem, due to Robinson [**?**], shows that if two atoms are unifiable, than they have a most general unifier.

Theorem 2.1 (Unification theorem) *There exists an algorithm, called* unification algorithm*, which for any two atoms produce their most general unifier if they are unifiable, and reports failure otherwise.*

A substitution ϑ is *idempotent* iff $\vartheta\vartheta = \vartheta$. Note that, by definition, if ϑ is idempotent than $Dom(\vartheta) \cap Range(\vartheta) = \emptyset$. The following corollary can be obtained by observing how the unification algorithm of previous theorem works (see, for example, [**?**]).

Corollary 2.2 *If two atoms are unifiable then they have an idempotent mgu.*

Moreover, it is possible to prove that the idempotent mgu of two atoms (or, equivalently, of a set of equations) is unique up to renaming (see [**?**]).

2.0.3 Fixpoint theory

Most of the semantics defined in this thesis are least fixpoints of suitable operators. We then need the following basic notions.

A *partially ordered set* (poset) $(S, \leq)$ is a set S which has a binary relation $\leq$ such that $\forall x, y, z \in S$

$$\begin{array}{ll} x \leq x & (\leq \text{ is reflexive}) \\ \text{if } x \leq y \text{ and } y \leq x, \text{ then } x = y & (\leq \text{ is anti-symmetric}) \\ \text{if } x \leq y \text{ and } y \leq z, \text{ then } x \leq z & (\leq \text{ is transitive}) \end{array}$$

S is *totally ordered* if it is partially ordered and also the following condition holds

$$\forall x, y \in S.\ x \leq y \text{ or } y \leq x$$

A (possibly empty) totally ordered subset of S is called a *chain*.

An element $z \in S$ is an *upper bound* of a subset X of S if

$$\forall x \in X.\ x \leq z$$

$z \in S$ is the *least upper bound* (called also join) of $X \subseteq S$, and is denoted by $lub_S\ X$ or by $\sqcup_S X$, if z is an upper bound of X and moreover

$$\text{if } \forall x \in X.\ x \leq y \text{ for } y \in S, \text{ then } z \leq y$$

Dually, an element $z \in S$ is a *lower bound* of $X \subseteq S$ if

$$\forall x \in X.\ z \leq x$$

$z \in S$ is the *greatest lower bound* (called also meet) of $X \subseteq S$, and is denoted by $glb_S\ X$ or by $\sqcap_S X$, if z is a lower bound of X and moreover

$$\text{if } \forall x \in X.\ y \leq x \text{ for } y \in S, \text{ then } y \leq z$$

It is easy to see that if the *glb* and the *lub* exist then they are unique. When it is clear from the context, the subscript $_S$ will be omitted.

A subset D of a poset S is *directed* if every finite subset of D has an upper bound in D. A *complete partial order* (cpo) is a poset $(S, \leq)$ such that for each directed set $D \subseteq S$ there exists *lub* D. A poset $(S, \leq)$ such that for every subset X of S there exists *lub* X and *glb* X is called a *complete lattice*. In particular, $\bot = lub\ \emptyset = glb\ S$ is the least element of S, while $\top = glb\ \emptyset = lub\ S$ is the greatest element. It is easy to see that the set of the subsets of S (called also the powerset of S) $\wp(S)$, equipped with the subset ordering is a complete lattice where *lub* is given by the union, and *glb* is given by the intersection.

Let $(S, \leq_S), (T, \leq_T)$ be posets. A function $f : S \to T$ is *monotonic* iff

$$\forall x, y \in S.\ x \leq_S y \text{ implies } f(x) \leq_T f(y)$$

f is *continuous* iff

$$\forall D \subseteq S \text{ directed}.\ f(lub_S\ D) = lub_T\ \{f(d) \mid d \in D\}$$

Every continuous function is also monotonic. Often in the definitions of *cpo* and of continuity are used *chains* instead than directed sets. It is possible to show that if the set S is denumerable then the definitions are equivalent.

Given a poset $(S, \leq)$ and a function $f : S \to S$, $x \in S$ is a *fixpoint* of f iff $x = f(x)$. x is a *prefixpoint* of f iff $x \leq f(x)$ and, dually, x is a *postfixpoint* of f iff $f(x) \leq x$.

$f : S \to S$ is an (increasing) *closure operator* iff

1)	$\forall x \in S.\ x \leq f(x)$	(f is extensive)
2)	$\forall x, y \in S.\ x \leq y \Rightarrow f(x) \leq f(y)$	(f is monotonic)
3)	$\forall x \in S.\ f(f(x)) = f(x)$	(f is idempotent)

A closure operator on a complete lattice $(L, \leq)$ can be represented by the set of its fixpoints. In fact, if f is a closure operator then

$$f(x) = glb\ (\check{x} \cap F) = min(\check{x} \cap F)$$

where $F = \{x \in L \mid f(x) = x\}$ and $\check{x} = \{y \in L \mid x \leq y\}$.

It can be proved that the composition of monotonic functions is monotonic, the composition of continuous functions is continuous and the composition of increasing functions is increasing. The composition of idempotent functions instead is not idempotent and therefore the composition of closure operators is not a closure operator.

The ordinal powers of a monotonic function $f : C \to C$ on a complete partial order $(C, \leq)$ are defined as

$$f \uparrow \alpha = \begin{cases} f(f \uparrow \alpha - 1) & \text{for } \alpha \text{ successor ordinal} \\ lub\ \{f \uparrow \beta \mid \beta < \alpha\} & \text{otherwise} \end{cases}$$

Note that, by definition, $f \uparrow 0 = \bot_C$. Moreover we use the notation

$$\begin{array}{ll} f^0(x) = x, & \\ f^{\alpha+1}(x) = f(f^\alpha(x)) & \text{for any ordinal } \alpha \\ f^\gamma(x) = lub\{f^\alpha(x) \mid \alpha < \gamma\} & \text{for } \gamma \text{ limit ordinal} \end{array}$$

We show now two important results which characterize the fixpoints of a function under different hypotheses. A great part of this thesis will deal with continuous functions on complete lattices. Therefore the second theorem will be particularly relevant to us.

Theorem 2.3 (Fixpoint theorem) (Knaster and Tarski [?]). *A monotonic function f on a complete lattice $(L, \leq)$ has a least fixpoint $lfp(f)$ and a greatest fixpoint $gfp(f)$. Moreover*

$$\begin{array}{l} lfp(f) = glb\ \{x \mid f(x) \leq x\} = glb\ \{x \mid x = f(x)\} \\ gfp(f) = lub\ \{x \mid x \leq f(x)\} = lub\ \{x \mid x = f(x)\} \end{array}$$

The next result is usually attributed to Kleene.

Theorem 2.4 *If f is a continuous function on a complete partial order $(C, \leq)$ and $x_0 \in C$ if a prefixpoint of f, then $lub\ \{f^n(X_0) \mid n \leq \omega\}$ is the least fixpoint of f greater than X_0. In particular, $f \uparrow \omega$ is the least prefixpoint and the least fixpoint of f.*

In the following we will use the following notation. If f and g are functions on a poset S,

$$(f + g)(X) = f(X) \cup g(X)$$

Moreover we denote by $f = g$ the extensional equality, i.e.

$$f = g \text{ iff } \forall x \in S.\ f(x) = g(x).$$

3 Logic programs

We are ready now to introduce the syntax of logic programs and to show the basic classical results which characterize their semantics.

3.0.4 Syntax

Given a first order language we need to identify a particular class of formulas to define what a logic program is. A *clause* is a formula of the form

$$\forall(L_1, \ldots, L_n)$$

where $L_1, \ldots, L_n$ are *literals*, i.e. either atoms or negated atoms. Usually clauses are written in a special, so called clausal, form. Namely the above clause is written as

$$A_1, \ldots A_m : -B_1, \ldots, B_k \tag{1}$$

where $A_1, \ldots, A_m$ are the atoms which occur non negated in $L_1, \ldots, L_n$, while $B_1, \ldots, B_k$ are the remaining atoms in $L_1, \ldots, L_n$. $: -$ is written also as the implication $\leftarrow$ and this clarifies the relation between the disjunctive and the clausal form of a clause.

A *program clause* (or a *definite clause*) is obtained by setting $m = 1$ in the general definition 1. A definite clause is then a formula of the form

$$A_1 : -B_1, \ldots, B_k$$

where A_1 is called the *head* and $B_1, \ldots, B_k$ is called the *body* of the clause. If the body is empty the clause is called *unit* clause or *fact*.

If we put $m = 0$ in definition 1 we obtain a *negative clause* called also a *goal*

$$: -B_1, \ldots, B_k$$

When no ambiguity arise, in the following we will denote the above goal simply by $B_1, \ldots, B_k$.

A *logic program* (or simply a program) is then a finite non-empty set of program clauses. Sometime we will consider also infinite or empty programs as infinite or empty sets of program clauses. In the following we will call a program clause simply clause. Moreover by "programs and goals defined on a signature Σ" we mean that Σ is the signature of the first order language on which program clauses and goals are defined.

Throughout this thesis we will assume programs and goals being defined on a fixed FOL given by a non-empty signature Σ and a finite set of predicates Π. Moreover, several results assume that Σ contains an infinite number of constant symbols. We will explicitly mention such an assumption when needed.

3.0.5 Declarative semantics

A logic program can be considered as a set of first order (non-logical) axioms which define a first order theory. Therefore to understand the meaning of a program we can surely use the classical model theory for first order logic, based on the notion of interpretation a la Tarski. An *interpretation* I (sometimes called a structure) for a (one-sorted) first order language L then consists of

1. a non empty set D called the *domain* of I,
2. an assignment to each 0-ary function symbol $c \in L$ of an element $c_I \in D$,
3. an assignment to each n-ary function symbol $f \in L$, $n \geq 1$, of a function $f : D^n \to D$,
4. an assignment to each n-ary predicate symbol $p \in L$ of a subset p_I of D^n.

where D^n is n the product $D \times D \times \ldots \times D$ (n times). To give a truth value to formulas containing variables we need the notion of *variable assignment* which is a mapping $\sigma : V \to D$ assigning to each variable an element of the domain D. A *variable assignment* σ can be lifted homomorphically to a function, still denoted by σ and called *valuation*, which maps terms to D. Namely we can define inductively $\sigma(c) = c_I$ for a constant c and $\sigma(f(t_1, \ldots, t_n)) = f_I(\sigma(t_1), \ldots, \sigma(t_n))$. It is possible to show that such a lifting is unique (see section **??**).

We can now define the truth of a formula F in the interpretation I for the valuation σ, written $I \models_\sigma F$, as follows

1. if $p(t_1, \ldots, t_n)$ is an atom then $I \models_\sigma p(t_1, \ldots, t_n)$ iff $(\sigma(t_1), \ldots, \sigma(t_n)) \in p_I$,
2. $I \models_\sigma true$ and not $I \models_\sigma false$
3. if F and G are formulas then
   - $I \models_\sigma \neg F$ iff not $I \models_\sigma F$,
   - $I \models_\sigma F \wedge G$ iff $I \models_\sigma F$ and $I \models_\sigma G$
   - $I \models_\sigma \forall F$ iff $I \models_{\sigma[x/d]} F$ for all $d \in D$

where $\sigma[x/d]$ is the valuation which differs from σ only in the assignment of d to the variable x.

The truth of the other formulas can be reconduced to the above cases, by recalling that we can express $F \vee G$ as $\neg(\neg F \wedge \neg G)$, $F \rightarrow G$ as $\neg F \vee G$, $F \leftrightarrow G$ as $(F \rightarrow G) \wedge (G \rightarrow F)$, and $\exists x F$ as $\neg\forall\neg F$.

We then say that F *is true* in the interpretation I, written $I \models F$, iff for any valuation σ $I \models_\sigma F$ holds. An interpretation M is a *model* of a set of formulas T if each formula f is true in M and, given a different set of formulas T', we say that T' is a *logical consequence* of T if each model of T is also a model of T'. A set of formulas is *satisfiable* (or consistent) iff it has a model and it is *unsatisfiable* (or inconsistent) otherwise.

When considering logic programs there exists a particular class of interpretations which are relevant, namely *Herbrand interpretations*. Let assume that the first order language L is defined on a signature Σ which contains at least one 0-ary function symbol. The set $\tau(\Sigma)$ of the ground terms is called the *Herbrand universe* for the language L, while the *Herbrand base* B_L of L is the set of all the ground atoms of L.

An *Herbrand interpretation* is then defined, according to the previous general definition, as an interpretation where the domain is the Herbrand universe, each constant in L is assigned to itself, each n-ary function symbol $f \in L$ is assigned to the mapping $f_H : \tau(\Sigma)^n \rightarrow \tau(\Sigma)$ defined by $f_H(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for $t_1, \ldots, t_n$ ground terms, each n-ary predicate symbol $p \in L$ is assigned to a subset p_H of $\tau(\Sigma)^n$. Note that each Herbrand interpretations can be uniquely identified by a subset H of the Herbrand base, where $p(t_1, \ldots, t_n) \in H$ iff $(t_1, \ldots, t_n) \in p_H$. In the following we will then call Herbrand interpretation a subset of B_L.

The importance of Herbrand interpretations relies on the following result.

Theorem 3.1 (Herbrand theorem) *A set of clauses S has a model S iff it has an Herbrand model.*

As a consequence, when considering the model theoretic semantics of logic programs we need to look only at Herbrand models. It is easy to see that a logic program P has always an Herbrand model and that the intersection of two Herbrand models of P is still a model of P. Therefore the *least Herbrand model*, obtained by the intersections of all the Herbrand models of a program, was proposed by Van Emdem and Kowalski [**?**] as the model-theoretic semantics for logic programs.

A classical result in [**?**] shows an alternative fixpoint characterization of the least Herbrand model. The key idea was the introduction in [**?**] of the *immediate consequences operator* T_P which maps Herbrand interpretations to Herbrand interpretations (the name of the operator is due to Clark [**?**]). Given a program P and and Herbrand interpretation I, the T_P operator associated to P can be defined by

$$T_P(I) = \{A \quad \begin{array}{l} \mid \exists \text{ a clause } H :- B_1, \ldots, B_n \in P, \\ I \models (B_1, \ldots, B_n)\vartheta \text{ and} \\ A \equiv B\vartheta \text{ is ground} \end{array} \quad \}$$

As first observed in [**?**], the postfixpoint of T_P are exactly the Herbrand models of P, i.e. given an Herbrand interpretation I, $T_P(I) \subseteq I$ iff I is a model of P. Moreover T_P is continuous on the lattice of Herbrand interpretations (with $\subseteq$ ordering). This observations allow to obtain the mentioned result.

Theorem 3.2 (Characterization theorem) (Van Emdem and Kovalski [?**])** *Let P be a program. Then P has an Herbrand model $M(P)$ which satisfies the following properties*

- *$M(P)$ is the least Herbrand model of P,*
- *$M(P)$ is the least pre-fixpoint and fixpoint of T_P,*
- *$M(P) = T_P \uparrow \omega$*

A third alternative characterization of the semantics of logic languages will be shown in next section.

3.0.6 Operational semantics

Definite clauses allow a natural computational reading based on the resolution procedure. Resolution was introduced by Robinson [**?**] as an inference rule for clauses. The version that we consider here was first described in [**?**] and was called SLD-resolution in [**?**] (Linear resolution with Selection rule for Definite clauses). Here we give only the basic terminology and some results. For more details see [**?**] and [**?**] which contain also an accurate bibliography on the subject.

Let $G = A_1, \ldots, A_k$ be a goal and $C = H : -B_1, \ldots, B_n$ be a (definite) clause. Then G' is *derived* from G and C using ϑ or, equivalently, G is a *resolvent* of G and C iff the following conditions hold

1. A_m, $1 \leq m \leq k$, is an atom *selected* among those in G,
2. $\vartheta = mgu(A_m, H)$,
3. $G' = (A_1, \ldots, A_{m-1}, B_1, \ldots, B_n, A_{m+1}, \ldots, A_k)\vartheta$.

Given a goal G and a program P, an *SLD-derivation* (or simply a derivation) of $P \cup G$ consists of a (possibly infinite) sequence of goals $G_0, G_1, G_2, \ldots$, a sequence $C_1, C_2, \ldots$ of *renamed apart* clauses in P and a sequence $\vartheta_1, \vartheta_2, \ldots$ of mgu's such that $G_0 = G$ and, for $i \geq 1$, each G_i is derived from G_{i-1} and C_i using ϑ_i. Each clause C_i used in a derivation is renamed (or standardized) apart, i.e. it is a variant of a clause in the program which do not share any variable with $G_0, C_1, C_2, \ldots C_{i_1}$. This condition is clearly necessary to avoid name clashes.

An *SLD-refutation* of $P \cup G$ is a finite SLD derivation of $P \cup G$ which has the empty clause $\Box$ as the last goal in the derivation. If $\vartheta_1, \vartheta_2, \ldots, \vartheta_n$ are the mgu's used in the refutation of $P \cup G$, we say that $\vartheta = (\vartheta_1 \vartheta_2 \ldots \vartheta_n)_{|Var(G)}$ is the *computed answer* for $P \cup G$ or, equivalently, for the goal G in the program P. The main classical results on SLD-resolution concern its soundness and completeness. The first one is due to Clark.

Theorem 3.3 (Soundness of SLD-resolution) (Clark [?]) *Let P be a program and $G = A_1, \ldots, A_n$ be a goal. Let $\vartheta_1, \vartheta_2, \ldots, \vartheta_n$ be the sequence of $mgu's$ computed in an SLD-refutation of $P \cup G$. Then $(A_1 \wedge \ldots \wedge A_n)\vartheta_1\vartheta_2 \ldots \vartheta_n$ is a logical consequence of P.*

Corollary 3.4 *Let P be a program and G be a goal. If there exists an SLD-refutation of $P \cup G$ then $P \cup G$ is inconsistent.*

The converse of previous corollary gives a first completeness result. It is due to Hill [**?**] even if the proof is due to Apt and Ven Emden [**?**].

Theorem 3.5 (Completeness of SLD-resolution) *Let P be a program and G be a goal. If $P \cup G$ is inconsistent then there exists an SLD-refutation of $P \cup G$.*

Previous theorem can be generalized by taking into account the computed answer and can be considered a kind of converse of theorem 3.3. A substitution ϑ is called a *correct answer* for $P \cup G$ if $Dom(\vartheta) \subseteq Var(G)$ and, for $G = A_1, \ldots, A_n$, $P \models (A_1 \wedge \ldots \wedge A_n)\vartheta$.

Theorem 3.6 (Clark [?]) *Let P be a program and $G = A_1, \ldots, A_n$ be a goal. If ϑ is a correct answer for $P \cup G$, then there exists a computed answer γ of $P \cup G$ such that $G\gamma$ is more general than $G\vartheta$.*

Another possible generalization of the completeness theorem concerns the *selection* rule. According to the definition of SLD-derivation, to obtain a new resolvent from the goal G_i we have to choice

1. a selected atom A_m from G_i,

2. a clause whose head unifies with A_m.

The first choice depends on the "history" of G_i, i.e. on the sequence $G_0, G_1, \ldots G_i$ of resolvents, the sequence $C_1, \ldots C_{i-1}$ of input clauses and the sequence $\vartheta_1, \ldots, \vartheta_{i-1}$ of mgu's as previously specified. Let HIS the set of all the histories. Following [**?**] we can define a *selection rule* R as a function which when applied to an element of HIS with sequence of resolvents $G_0, G_1, \ldots G_i$ returns an atom in G_i. Such an atom is the selected atom in G_i.

Given a selection rule R, an SLD-derivation is called via R (or with selection rule R) if all the selections of atoms in the resolvents are performed according to R. Moreover an R-computed answer of $P \cup G$ is a computed answer of a SLD-refutation via R. We can then give the following results.

Theorem 3.7 (Strong completeness of SLD-resolution)(Hill [?]) *Let P be a program, G be a goal and R be a selection rule. If $P \cup G$ is inconsistent then there exists an SLD-refutation via R of $P \cup G$.*

Analogously, we can generalize theorem 3.6.

Theorem 3.8 (Clark [?]) *Let P be a program, $G = A_1, \ldots, A_n$ be a goal and R a selection rule. If ϑ is a correct answer for $P \cup G$, then there exists a R-computed answer γ of $P \cup G$ such that $G\gamma$ is more general than $G\vartheta$.*

Theorem 3.9 *[**?**] Let P be a program, G be a goal and R a selection rule. If ϑ is a computed answer for $P \cup G$, then there exists a R-computed answer ϑ' of $P \cup G$ such that $G\vartheta$ and $G\vartheta'$ are variants.*

We have shown how SLD-derivation furnishes a proof-theory for logic programs. The *operational semantics* of logic programs can then be defined naturally in terms of such an inference rule. Namely, given a program P on the language L, we can define the *success set* of a program P as the set of all the (ground) atoms A in the Herbrand base B_L such that $P \cup A$ has a refutation. The following result shows the mentioned equivalence between this operational definition and the declarative one.

Theorem 3.10 (Succes set theorem) (Apt and Van Emdem [?]) *The success set of a program P is equal to its least Herbrand model.*