Alberto Pettorossi Maurizio Proietti

First Order Predicate Calculus and Logic Programming

Third Edition

ARACNE

Table of Contents

Preface 7

Chapter 1. FIRST ORDER PREDICATE CALCULUS AND FIRST ORDER THEORIES

1. Syntax of First Order Theories 9
   - 1.1 Scope, Variables, and Formulas 10
   - 1.2 Substitutions 11
2. Axioms and Inference Rules of First Order Theories 13
   - 2.1 Classical Presentation 13
   - 2.2 Natural Deduction Presentation 19
3. Some Properties of Connectives and Quantifiers 24
   - 3.1 Basic Equivalences for Implication and Equivalence 25
   - 3.2 Distributivity of Quantifiers over Conjunctions and Disjunctions ... 26
   - 3.3 Prenex Conjunctive and Prenex Disjunctive Normal Form 27
   - 3.4 Quantifiers and Implication 28
   - 3.5 Duality 29
4. Semantics of First Order Theories and Completeness Theorem 30
5. First Order Predicate Calculus With Equality 37
   - 5.1 Definition of New Function Symbols 40
6. Peano Arithmetic and Incompleteness Theorem 40
   - 6.1 More on Incompleteness 43

Chapter 2. LOGIC PROGRAMMING

7. Towards Logic Programming: Skolem, Herbrand, and Robinson Theorems 47
8. Horn Clauses and Definite Logic Programs 55
   - 8.1 Least Herbrand Models of Definite Logic Programs 57
   - 8.2 Fixpoint Semantics of Definite Logic Programs 60
   - 8.3 Operational Semantics of Definite Logic Programs 64
9. Computing with Definite Logic Programs 69
   - 9.1 Computing Functions 70
   - 9.2 Computing Function Inverses 71
   - 9.3 Computing Relations and Nondeterministic Computing 71

9.4 Constructing Knowledge-Based Systems 76
9.5 Theorem Provers and Interpreters of Horn Clauses 79
10. Deriving Negative Information from Definite Logic Programs 85
11. Normal Programs 90
12. Programs 97
13. **Appendix A: Truth in Mathematical Structures** 102
14. **Appendix B: Remarks on Resolution** 104
14.1 Resolution for Conjunction of Clauses 104
14.2 SLD Resolution for Definite Programs 104
14.3 SLDNF Resolution for Normal Programs 106
Index 108
References 111

Preface

These lecture notes are intended to introduce the reader to the basic notions of the first order predicate calculus and logic programming. We present the axioms and the inference rules of the first order predicate calculus in two different styles: (i) the Classical style (à la Hilbert), and (ii) the Natural Deduction style (à la Gentzen). We also present the semantics of this calculus following Tarski's approach. The Skolem Theorem, the Herbrand Theorem, and the Robinson Theorem are the three steps which lead us to the study of the Definite Logic programs. For these programs we give the denotational semantics via the least Herbrand models, the fixpoint semantics via the so called T_P operator, and the operational semantics via SLD trees. Finally, we consider the issue of deriving negative information from Definite Logic programs, and we present the theory of normal programs and the theory of programs as conjunctions of statements.

Sections are devoted to the Gödel's Completeness Theorem, the first order predicate calculus with equality, and the Peano Arithmetic.

We would like to thank Andrea Bastoni, Lorenzo Clemente, Simone D'Uffizi, Alessandro Papaleo, Giorgio Ventrella, and the other students of the Theoretical Computer Science courses during the academic years 2002-03 and 2003-04 for pointing out to us a few mistakes.

Many thanks also to Lorenzo Costantini of the Aracne Publishing Company for his kind and helpful cooperation.
Roma, February 2005

In the second and third editions we have revised and improved many sections of the book.
Roma, December 2011

Alberto Pettorossi
Department of Informatics, Systems, and Production
University of Roma Tor Vergata
Via del Politecnico 1, I-00133 Roma, Italy
email: adp@iasi.rm.cnr.it
URL: http://www.iasi.cnr.it/~adp

Maurizio Proietti
Consiglio Nazionale delle Ricerche - IASI
Viale Manzoni 30, I-00185 Roma, Italy
email: proietti@iasi.rm.cnr.it
URL: http://www.iasi.cnr.it/~proietti

Chapter 1

First Order Predicate Calculus and First Order Theories

1 Syntax of First Order Theories

Let us consider a *first order language* L, that is, a language consisting of:

- a denumerable set of *variables*: $\{x, y, z, \ldots\}$ (we will feel free to use also subscripts),
- a finite or denumerable (possibly empty) set of *function symbols* (of arity $r \geq 0$): $\{f, g, \ldots\}$ (function symbols of arity 0 are also called *constant* symbols),
- a finite or denumerable non-empty set of *predicate symbols* (of arity $r \geq 0$): $\{p, q, \ldots\}$ (in particular, we assume that there are two predicate symbols of arity 0, called *true* and *false*),
- *terms* constructed from variables and function symbols of arity r applied to r terms,
- *atoms*, also called *atomic formulas*, constructed from predicate symbols of arity r applied to r terms,
- *formulas* which are either atoms or formulas constructed from formulas using the following connectives: $\neg$ (*not*: forming *negations*), $\vee$ (*or*: forming *disjunctions*), $\wedge$ (*and*: forming *conjunctions*), $\rightarrow$ (*implies*: forming *implications*), $\leftarrow$ (*reverse implies*: forming *reverse implications*), $\leftrightarrow$ (*equivalent to*: forming *equivalences*), and the quantifiers $\forall$ (*for all*: forming *universal quantifications*) and $\exists$ (*there exists*: forming *existential quantifications*). The connective $\neg$ is unary, all others are binary. A quantifier has two operands: a variable and a formula.

Different first order languages arise from different choices for the set of the function symbols and the set of the predicate symbols.

Given any two formulas A and B, we will consider: (i) $A \vee B$ as an abbreviation for $(\neg A) \rightarrow B$, (ii) $A \wedge B$ as an abbreviation for $\neg((\neg A) \vee (\neg B))$, (iii) $A \leftrightarrow B$ as an abbreviation for $(A \rightarrow B) \wedge (B \rightarrow A)$, and (iv) $\exists x\, A$ as an abbreviation for $\neg(\forall x\, (\neg A))$.

Parentheses are used for grouping formulas. In order to avoid too many parentheses, we will assume the usual priorities among connectives, so that: (i) $\forall x$, $\exists x$, and $\neg$ bind tighter than any other connective, i.e., they are applied to the smallest possible formula or parenthesized formula following them, (ii) $\wedge$ binds tighter than $\vee$, (iii) $\vee$ binds tighter than $\rightarrow$, and (iv) $\rightarrow$ binds tighter than $\leftrightarrow$. Thus, for instance, $\forall x\, \neg p(x) \rightarrow q \vee r(x)$ is the same as $(\forall x\, (\neg p(x))) \rightarrow (q \vee r(x))$.

$\forall x_1 \ldots \forall x_m\, \varphi$ is also written as $\forall x_1 \ldots x_m\, \varphi$ or $\forall x_1, \ldots, x_m\, \varphi$. Analogously for $\exists$, instead of $\forall$.

By $C[A]$ we denote an *expression*, that is, a term or a formula, C where we singled out an occurrence of a subexpression A. Then, $C[B]$ denotes C where the subexpression A has been replaced by the subexpression B. The expression $C[A]$ where A is missing, is denoted by $C[\_]$ and it is called a *context*. In particular, $C[\_]$ is also called a *term context* or a *formula context*, if $C[A]$ is term or a formula, respectively.

We will use the following auxiliary notions.

A *literal* is an atom or the negation of an atom. An atom is also called a *positive literal* and the negation of an atom is also called a *negative literal*. For instance, $p(x, a)$ is a positive literal and $\neg p(f(b, x), y)$ is a negative literal.

A *clause* is a disjunction of zero or more literals. A clause with zero literals is called the *empty clause*, denoted by $\Box$, and it is identified with the atom *false*. In the sequel, by abuse of language, we will call *clause* also the universal closure of the disjunction of zero or more literals. The context will disambiguate between these two different uses of the word 'clause'.

A formula is said to be in *clausal form* iff it is of the form:

$\forall x_1 \ldots \forall x_m\, (C_1 \wedge \ldots \wedge C_k)$

where for $1 \leq i \leq k$, C_i is a clause and the variables $x_1, \ldots, x_m$ are the distinct variables occurring in $C_1 \wedge \ldots \wedge C_k$.

1.1 Scope, Variables, and Formulas

In the formula $\forall x\, \varphi$ the subformula φ is said to be the *scope* of $\forall x$. We say that the quantifier $\forall$ in $\forall x\, \varphi$ *binds* the variable x. Analogously for $\exists$, instead of $\forall$.

An *occurrence of a variable* x is said to be *bound* in a formula φ iff *either* it is the occurrence of x in $\forall x$ or $\exists x$ in φ *or* it occurs in the scope of an occurrence of $\forall x$ or $\exists x$ in φ. An occurrence of a variable is said to be *free* iff it is not bound. A *variable* x is said to be *bound* (or *free*) in a formula φ iff there is an occurrence of x which is bound (or free, respectively) in φ.

Note that in the formula $p(x) \vee \exists x\, q(x)$ the variable x is both bound and free, but obviously, any of the three occurrences of x is either bound or free. The only free occurrence of x is the one in $p(x)$.

Given the terms (or formulas) $t_1, \ldots, t_n$, by $vars(t_1, \ldots, t_n)$ we denote the set of variables occurring in those terms (or formulas).

Given a formula φ, (i) by $freevars(\varphi)$ we denote the set of variables which are free in φ, and (ii) by $boundvars(\varphi)$ we denote the set of variables which are bound in φ.

A formula φ is said to be a *closed formula*, or a *sentence* [10, page 46], if $freevars(\varphi) = \emptyset$, where as usual $\emptyset$ denotes the emptyset. Thus, every formula in clausal form is closed.

A formula is said to be an *open formula* if it does not contain any quantifier.

Note that according to this terminology, there are formulas which are both closed and open, and formulas which are neither closed nor open. A formula is both closed and open iff it has no variables. For instance, the formula $p \vee q$ is closed and open and the formula $p(x) \vee \exists x\, q(x)$ is neither closed nor open.

A term (or a formula) is said to be a *ground term* (or a *ground formula*) if no variable occurs in that term (or formula). A term without any occurrence of a variable is also said to be a *closed term* [10, page 69].

Given a formula φ, the *universal closure* of φ is $\forall x_1 \ldots x_m\, \varphi$, where $x_1, \ldots, x_m$ are the distinct variables occurring free in φ. The universal closure is also denoted by $\forall(\varphi)$. Analogously for the *existential closure* of φ with $\exists$, instead of $\forall$.

We will feel free to write a formula φ as $\varphi(x_1,\ldots,x_n)$ to denote that $\{x_1,\ldots,x_n\} \subseteq freevars(\varphi)$. For example, if A is the formula $p(x,z) \vee \exists y\, q(y,x)$, then A can also be written as $A(x)$ or $A(z)$ or $A(x,z)$.

1.2 Substitutions

A *substitution* is a finite mapping from variables to terms. Given a substitution $\vartheta = \{x_1/t_1, \ldots, x_n/t_n\}$, the set $\{x_1, \ldots, x_n\}$, where for $i = 1, \ldots, n$, the variables x_i's are assumed to be distinct, is called the *domain* of ϑ and the set $\{t_1, \ldots, t_n\}$ is called the *range* of ϑ. For $i = 1, \ldots, n$, the pair x_i/t_i is said to be a *binding* for the variable x_i. x/x is said to be an *identity binding* for the variable x.

If we apply a substitution ϑ to a term (or formula) t we obtain a new term (or formula), denoted by $t\vartheta$, which is said to be an *instance* of t. Recall that all bindings of a substitution should be applied *in parallel and once*, so that, for instance,

$f(g(x_1, x_1), g(x, b))\ \{x/x_1,\ x_1/a\} = f(g(a, a), g(x_1, b))$.

The *composition* $\vartheta\sigma$ of two substitutions $\vartheta = \{x_1/t_1, \ldots, x_n/t_n\}$ and $\sigma = \{y_1/s_1, \ldots, y_m/s_m\}$ is the substitution obtained from the set $\{x_1/t_1\sigma, \ldots, x_n/t_n\sigma,\ y_1/s_1, \ldots, y_m/s_m\}$ by deleting the identity bindings and every binding y_i/s_i, for $1 \leq i \leq m$, such that y_i belongs to $\{x_1, \ldots, x_n\}$. This definition is motivated by the fact that we want that for any term t and substitutions ϑ and σ, $t(\vartheta\sigma) = (t\vartheta)\sigma$. Composition of substitutions is associative.

A substitution μ is *more general* than a substitution ϑ iff there exists a substitution σ such that $\vartheta = \mu\sigma$, apart from identity bindings.

A substitution $\vartheta = \{x_1/t_1, \ldots, x_n/t_n\}$ is said to be in *solved form* iff for $i = 1, \ldots, n$, the variable x_i does not occur in $\{t_1, \ldots, t_n\}$. Thus, a substitution in solved form does not have identity bindings.

A substitution ϑ is *idempotent* iff $domain(\sigma) \cap vars(range(\sigma)) = \emptyset$ where σ is the substitution derived from ϑ by deleting the identity bindings. We have that ϑ is idempotent iff $\vartheta\vartheta = \vartheta$, apart from identity bindings.

Given n (≥ 2) terms (or atoms) $u_1, \ldots, u_n$, a substitution μ is said to be a *unifier* of $u_1, \ldots, u_n$ iff $u_1\mu = \ldots = u_n\mu$. If some terms (or atoms) have a unifier, they are said to be *unifiable*. A unifier μ is said to be a *most general unifier* (or *mgu*, for short) iff μ is more general than any other unifier. An mgu can always be chosen to be an idempotent substitution. One can show that an idempotent mgu μ is *relevant*, that is, each variable in domain(μ) $\cup$ range(μ) occurs in at least one of the terms (or atoms) to be unified.

As for any function, a substitution ϑ *restricted to* a set S of variables, is a substitution with domain S which agrees with ϑ for every variable in S.

A substitution ϑ is said to be *grounding* for a term t (or a formula φ) iff $t\vartheta$ (or $\varphi\vartheta$) is ground.

Given a formula $A(x)$ and a term t, we write $A(t)$ to denote the formula obtained from $A(x)$ by replacing *every free* occurrence of the variable x in A by t. For instance, if $A(x)$ is $p(x, z) \vee \exists y\, q(y, x)$ then $A(f(a))$ is $p(f(a), z) \vee \exists y\, q(y, f(a))$.

We say that a term t *is free* (or *can be substituted*) *for the variable* x in a formula $A(x)$ iff no free occurrence of x in $A(x)$ is in the scope of a quantifier which binds a variable of t.

Thus, if a term t is free for x in $A(x)$, every variable occurrence in t generates after substitution in $A(t)$, a free occurrence of that same variable. For example,
(i) $r(y)$ is not free for x in $p(x, z) \vee (\exists y\, q(y, x))$ because the variable y of the term $r(y)$ is not free in $\exists y\, q(y, r(y))$,
(ii) the term $g(x, a)$ is free for x in $p(x, z) \vee \exists y\, q(y, x)$, and
(iii) x is free for x in any formula $A(x)$.

A *variable renaming* is a bijective mapping from the set of variables onto itself.

For other concepts not presented here we refer to [1,8,9,10].

Example 1. (i) The term $g(x, a)$ is an instance of the term $g(x, y)$.
(ii) The composition $\vartheta\sigma$ of $\vartheta = \{x/f(y),\ y/z\}$ and $\sigma = \{x/a,\ y/b,\ z/y\}$ is $\{x/f(b),\ z/y\}$, which has been obtained by deleting the identity binding y/y and the bindings x/a and y/b from the set $\{x/f(b),\ y/y,\ x/a,\ y/b,\ z/y\}$.
(iii) The composition $\vartheta\sigma$ of $\vartheta = \{x/y,\ y/b\ \}$and $\sigma = \{x/y,\ y/b\}$ is $\{x/b,\ y/b\}$.

(iv) The substitution $\{x/y\}$ is more general than the substitution $\{x/f(y), y/f(y)\}$.
(v) The most general unifier of $p(a, f(y))$ and $p(x, f(t(x)))$ is $\{x/a, y/t(a)\}$. Given the two terms x and z, the substitutions: $\{x/y, z/y\}$, $\{x/z\}$, and $\{z/x\}$ are three different unifiers. Only $\{x/z\}$ and $\{z/x\}$ are most general unifiers. Note that, in particular, $\{x/z\}$ is more general that $\{z/x\}$ and vice versa.
(vi) The substitution $\{x/y, z/g(x, a)\}$ restricted to $\{x\}$ is the substitution $\{x/y\}$.

2 Axioms and Rules of Inference of First Order Theories

We will present the axioms and the rules of inference of first order theories in two different way: the Classical presentation (à la Hilbert) [10], and the Natural Deduction presentation (à la Gentzen) [9].

2.1 Classical Presentation

The *logical axioms* of any first order theory are instances of the following axiom schemata:

A1.	$A \to (B \to A)$	
A2.	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	
A3.	$(\neg A \to \neg B) \to (B \to A)$	
A4.	$(\forall x\, A(x)) \to A(t)$	if the term t is free for x in $A(x)$
A5.	$(\forall x\, (A \to B)) \to (A \to (\forall x\, B))$	if x is not a free variable in A

Now we explain in an informal way the given restrictions on the axioms A4 and A5. Our explanation is based on the intuitive understanding of the notions of the implication and the universal quantifier. We will return to this point when presenting the semantics of a first order theory in Section 4.

The restriction on axiom A4 is explained as follows. Since y is not free for x in $\forall y\, A(x, y)$, we would have: $\forall x \neg \forall y\, A(x, y) \to \neg \forall y\, A(y, y)$. Now consider the set $\{a, b\}$ such that $A(a, a)$ and $A(b, b)$ holds, while neither $A(a, b)$ nor $A(b, a)$ holds. We have that $\forall x \neg \forall y\, A(x, y)$ holds, while $\neg \forall y\, A(y, y)$ does not hold.

The restriction on axiom A5 is explained as follows. Let us consider the formula $(\forall x\, (A(x) \to A(x))) \to (A(x) \to (\forall x\, A(x)))$, where we assume that $A(x)$ holds iff x is an even natural number. Now $\forall x\, (A(x) \to A(x))$ holds. If we take x to be an even number then $A(x)$ holds, while $\forall x\, A(x)$ does not hold because there are natural numbers which are not even numbers. Thus, $A(x) \to (\forall x\, A(x))$ does not hold.

The rules of inference of a first order theory are the following ones: for any formula A, B, and C and any variable x,

$$\frac{A \qquad A \to B}{B} \qquad \text{(MP : Modus Ponens)}$$

$$\frac{A}{\forall x\, A} \qquad \text{(Gen : Generalization)}$$

The formulas written above the horizontal lines of the rules of inference are called *premises*. The formulas written below the horizontal lines are called *conclusions.* Given a rule of inference, its conclusion is said to be the *direct consequence* of each of the premises of that rule. Axioms can be viewed as rules of inference with no premises.

If from a formula A we derive the formula $\forall x\, A$, we say that the Generalization rule has been applied to the variable x.

Note 1. Besides logical axioms, a first order theory may include also other axioms, called *proper axioms*. They are formulas which may be either closed or not closed. Recall, however, that from the Generalization rule and axiom A4, we have that for any formula $\varphi(x)$, we have that $\varphi(x) \vdash \forall x\, \varphi(x)$ and $\forall x\, \varphi(x) \vdash \varphi(x)$.

A first order theory without proper axioms is called *a first order predicate calculus*, or simply, *a predicate calculus*, when the qualification 'first order' is understood from the context.

Note 2. Many notions or properties of the first order predicate calculi do *not* depend on our choice of the first order language, and we will refer to '*the* first order predicate calculus' when we want to study the notions or properties common to all first order predicate calculi.

In what follows, for reasons of simplicity, we will feel free to say 'a theory', instead of 'a first order theory'.

Let us now define the *derivation relation*, denoted by $\vdash$, in a first order theory. Given a set Γ of formulas and a formula φ, a *derivation* (or a *proof*) of φ from Γ is a sequence of formulas ending with φ, such that each of formula in the sequence is either (i) a formula in Γ, or (ii) an axiom, or (iii) it can be obtained by the MP rule from two preceding formulas of the sequence (and in this sense the premises of the MP rule are assumed to form a conjunction), or (iv) it can be obtained by the Gen rule from a preceding formula of the sequence.

If there exists a derivation of φ from Γ we write $\Gamma \vdash \varphi$. We write $\Gamma, \varphi \vdash \psi$ to denote $\Gamma \cup \{\varphi\} \vdash \psi$.

Note that if $\Gamma \vdash \varphi$ holds, then also $\Gamma' \vdash \varphi$ holds where Γ' is the *conjunction* of the formulas in the set Γ. Thus, if $\Gamma \vdash \varphi$ holds, then we may think of it as either a set of formulas or a conjunction of formulas.

A *proof tree* of a formula φ from a set Γ of formulas, is a tree T such that:

(i) the root of T is φ,

(ii) for every node δ of T with $n\,(\geq 0)$ sons $\delta_1, \ldots, \delta_n$, there exists a rule of inference with conclusion δ and premises $\delta_1, \ldots, \delta_n$, and

(iii) every leaf is either an axiom or an element of Γ.

Each derivation of φ from Γ can be viewed as a finite proof tree of φ from Γ, and every finite proof tree of φ from Γ can be viewed as a derivation by visiting every node of that tree after its sons.

When $\Gamma = \emptyset$ we also write $\vdash \varphi$, instead of $\emptyset \vdash \varphi$, and we say that φ is a *theorem* of the first order predicate calculus.

The set of theorems of the first order predicate calculus is said to be the *theory* of the first order predicate calculus and, in general, we will identify a first order theory with the set of its theorems.

In general, given a first order theory T, possibly with proper axioms, we will write $T \vdash \varphi$, or $\vdash_T \varphi$, to denote that φ is a theorem of T. We will feel free to omit the subscript T, and we will simply write $\vdash\ \varphi$, when the theory T is understood from the context. When the theory T is a first order predicate calculus (and, thus, there are no proper axioms), we will also write φ, instead of $\vdash\ \varphi$.

The set of theorems of a first order theory does not change if together with Modus Ponens and Generalization, we allow also the following rule of inference:

$$\frac{\varphi[p]}{\varphi[\psi]} \qquad \text{(Substitution Rule)}$$

where $\varphi[p]$ is a formula φ in which we have singled out all the occurrences of a predicate symbol p (of arity 0), and $\varphi[\psi]$ is the result of substituting all those occurrences by a given *closed* formula ψ.

For instance, from $(\varphi \wedge p) \rightarrow (p \wedge r(y))$ by applying the Substitution rule we may derive the formula $(\varphi \wedge \forall x\, q(x)) \rightarrow (\forall x\, q(x) \wedge r(y))$ (which is obtained by replacing the predicate symbol p by the formula $\forall x\, q(x)$).

We have the following theorem which relates the implication connective $\rightarrow$ and the derivation relation $\vdash$. We need the following notion.

Given two formulas φ and ψ occurring in a derivation from Γ, we say that φ *depends on* ψ iff either φ is ψ or there exists a formula σ such that φ is a direct consequence of σ and σ depends on ψ.

Theorem 1. [Deduction Theorem. Version 1] Let us consider any set of formulas Γ and any two formulas φ and ψ. (i) If $\Gamma \vdash \varphi \rightarrow \psi$ then $\Gamma, \varphi \vdash \psi$.
(ii) Let us assume that in a derivation of ψ from $\Gamma \cup \{\varphi\}$, whenever we apply the Generalization rule to a formula, say A, whereby deriving $\forall x\, A$,
either (ii.1) A does not depend on φ, *or* (ii.2) x does not belong to *freevars*(φ).
If $\Gamma, \varphi \vdash \psi$ then $\Gamma \vdash \varphi \rightarrow \psi$.

In the above Theorem 1 if A depends on φ the Condition (ii.2) is needed. Indeed, by Theorem 1 without Condition (ii.2), from $A(x) \vdash \forall x A(x)$ we get $\vdash A(x) \rightarrow \forall x A(x)$. Now, $A(x) \vdash \forall x A(x)$ holds because of Gen, while $\vdash A(x) \rightarrow \forall x A(x)$ does *not* hold, as shown in the following exercise (and it can be shown by using the Completeness Theorem (see Theorem 7 on page 35).

Exercise 1. Show that in the Classical presentation of the first order predicate calculus we have that:

(1) $\varphi(x) \vdash \forall x\, \varphi(x)$ holds, and

(2) $\vdash \varphi(x) \rightarrow \forall x\, \varphi(x)$ does not hold.

Proof of (1). $\varphi(x) \vdash \forall x\, \varphi(x)$ is obtained by applying the Generalization rule.

Proof of (2). We have that: $\vdash \varphi(x) \rightarrow \forall x\, \varphi(x)$ iff $\models \varphi(x) \rightarrow \forall x\, \varphi(x)$ (by Theorem 7) iff for all I and σ, $I, \sigma \models \varphi(x)$ implies $I, \sigma \models \forall x\, \varphi(x)$ (by definition). Thus, in order to show that $\vdash \varphi(x) \rightarrow \forall x\, \varphi(x)$ does not hold, it is enough to exhibit an interpretation I and a variable assignment σ such that $I, \sigma \models \varphi(x)$ holds and $I, \sigma \models \forall x\, \varphi(x)$ does not hold. Let us consider the interpretation I such that: (i) $\mathit{domain}(I) = \{0, 1\}$, (ii) $I \models \varphi(0)$, and (iii) $I \not\models \varphi(1)$. We have that $I, \sigma \models \varphi(x)$ holds for $\sigma(x) = 0$. Since $I, \sigma \models \varphi(x)$ does not hold for $\sigma(x) = 1$ we have that $I, \sigma \models \forall x\, \varphi(x)$ does not hold.

Note 3. As a consequence of Theorem 1, if either φ is a *closed* formula or Gen is never applied to a free variable of $A(x)$, then the Deduction Theorem can be stated in the following simpler form: $\Gamma \vdash \varphi \rightarrow \psi$ iff $\Gamma, \varphi \vdash \psi$. Therefore, in order to use this simpler form, it is convenient to formally represent any given data base of facts (with its rules of inference and integrity constraints) as a set (or a conjunction) of closed formulas.

The properties stated in following theorem are useful when making proofs and they may be viewed as extra inference rules, besides Modus Ponens and Generalization. For the proof of this theorem the reader may refer to [10, pages 61–63].

Proposition 1. Let us consider any two formulas A and B, any formula context $C[\_]$, and any term t. (In particular, in Points (i) and (ii) we assume that the formula A may have the free variable x.)

(i) If t is free for x in $A(x)$, then $\forall x A(x) \vdash A(t)$.

(ii) If t is free for x in $A(x)$, then $A(t) \vdash \exists x A(x)$, where $A(t)$ is obtained from $A(x)$ by replacing all free occurrences of x by t.

(iii) $\vdash (\forall x_1 \ldots x_n\, (A \leftrightarrow B)) \;\rightarrow\; (C[A] \leftrightarrow C[B])$,
where $((\mathit{freevars}(A) \cup \mathit{freevars}(B)) \cap \mathit{boundvars}(C[A])) \;\subseteq\; \{x_1, \ldots, x_n\}$.

(iv) $\vdash A \leftrightarrow A\rho$, where ρ is a renaming of the bound variables of A such that $\mathit{vars}(A) \cap \mathit{range}(\rho) = \emptyset$.

Now we list some derived deduction rules for the first order predicate calculus. These rules can be proved by applying the Deduction Theorem on page 16 and the above Proposition 1, and can be used, besides the Modus Ponens and Generalization rules, for deriving new theorems (those below the horizontal line) from theorems which have been already derived (those above the horizontal line).

For negation:

$$\frac{\vdash \neg\neg A}{\vdash A} \qquad \frac{\vdash A}{\vdash \neg\neg A}$$

For conjunction:

$$\frac{\vdash A \wedge B}{\vdash A} \qquad \frac{\vdash A \wedge B}{\vdash B} \qquad \frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$$

For disjunction:

$$\frac{\vdash A \vee B \quad \vdash \neg A}{\vdash B} \qquad \frac{\vdash A \vee B \quad \vdash \neg B}{\vdash A} \qquad \frac{\vdash A}{\vdash A \vee B} \qquad \frac{\vdash B}{\vdash A \vee B}$$

For implication:

$$\frac{\vdash A \rightarrow B \quad \vdash \neg B}{\vdash \neg A} \qquad \frac{\vdash \neg (A \rightarrow B)}{\vdash A \wedge \neg B}$$

For equivalence (see also Property (xi) on page 25):

$$\frac{\vdash A \leftrightarrow B \quad \vdash A}{\vdash B} \qquad \frac{\vdash A \leftrightarrow B \quad \vdash B}{\vdash A}$$

The following rule is called '*proof by contradiction*':

if $\Gamma, A \vdash B \wedge \neg B$ and in this derivation from $\Gamma \cup \{A\}$ to $B \wedge \neg B$ the Generalization rule is never applied to a free variable of A (and this is the case if we assume that A is a closed formula),
then $\Gamma \vdash \neg A$

The Rule C. Suppose that during a proof we have derived $\exists x\, A(x)$. Then we can 'make a choice' and derive $A(b)$, where b is a constant which is the witness of the existential quantifier. The constant b should never occurring before in the proof. In

the derivation step from $\exists x\, A(x)$ to $A(b)$, we say that we have applied the Rule C (C for 'choice') [10, page 64] which is:

$$\frac{\exists x\, \varphi(x)}{\varphi(b)} \qquad \text{(Rule C)}$$

If at the end of the proof we get a formula without the constant b, then the proof is sound in the sense stated by Proposition 2 below. In order to state that proposition we need the following notation.

Given a set Γ of formulas and a formula φ, we write $\Gamma \vdash_{\mathrm{C}} \varphi$ to denote that there exists a derivation of φ from Γ which (i) uses Rule C, besides Modus Ponens and Generalization, (ii) uses axioms involving also the constants introduced in previous steps where Rule C was applied, and (iii) if in that derivation there is an application of Rule C with premise $\exists y\, \psi(y)$ where x occurred free in $\exists y\, \psi(y)$, then there is no subsequent application of the Generalization rule introducing the quantification $\forall x$.

Now we are ready to present Proposition 2.

Proposition 2. Given any set Γ of formulas and any formula φ, we have that if $\Gamma \vdash_{\mathrm{C}} \varphi$, then $\Gamma \vdash \varphi$.

Exercise 2. Show that the following hold: (i) $A(x) \vdash \forall x\, A(x)$, (ii) $\forall x\, A(x) \vdash A(x)$, and (iii) $\vdash \forall x\, A(x) \to A(x)$. Recall that it is not the case that $\vdash A(x) \to \forall x\, A(x)$ (see Exercise 1 on page 16).

Now let us introduce the following notions which will useful in the sequel.

A first order theory is said to be *axiomatic* if the set of its axioms is recursive [10, page 28]. (Here we stick to the standard terminology, but we think that one should refer to the axiomatic theories as *recursive axiomatic* theories.)

A first order theory is said to be *semidecidable* (or *recursively enumerable*, or *r.e.*) if the set of its theorems is recursively enumerable.

We leave it as an exercise to the reader to show the following.

Proposition 3. The set of theorems of an axiomatic first order theory is recursively enumerable (and it can be shown that, in general, is not recursive).

A first order theory is said to be *decidable* if the set of its theorems is recursive.

A first order theory T is said to be *complete* iff for any *closed* first order formula φ, either $T \vdash \varphi$ or $T \vdash \neg\varphi$ holds or both (this last case occurs only if T is inconsistent, as we will see below). Thus, given a first order theory T, if T is inconsistent then T is complete.

As a consequence of Post Theorem which states that if a set S and its complement are recursively enumerable, then S is recursive, we have the following proposition.

Proposition 4. If a first order theory is axiomatic and complete, then it is decidable.

A first order formula φ is said to be *undecidable* in a first order theory T iff $T \vdash \varphi$ does not hold and $T \vdash \neg\varphi$ does not hold. (Note that the word 'undecidable' here has a different meaning with respect to the one it has in Computability Theory [10], where a set is said to be undecidable iff it is not recursive.)

Thus, in a first order theory which is not complete there is an undecidable, closed formula.

Exercise 3. A first order theory K is complete *iff* for any two closed formulas φ and ψ in K we have that if $K \vdash \varphi \vee \psi$ then ($K \vdash \varphi$ or $K \vdash \psi$).

Solution. (*only-if part*) Assume that K is complete and $K \vdash \varphi \vee \psi$. We have to show: $K \vdash \varphi$ or $K \vdash \psi$. Indeed, assume that it is not the case that $K \vdash \varphi$. By completeness we have $K \vdash \neg\varphi$. By the tautology $(\neg\varphi) \rightarrow ((\varphi \vee \psi) \rightarrow \psi)$, we get: $K \vdash \psi$. (*if part*) Assume that K is not complete. We have to show that for some formulas φ and ψ, $K \vdash \varphi \vee \psi$ does not imply ($K \vdash \varphi$ or $K \vdash \psi$). Take ψ to be $\neg\,\varphi$ for some closed formula φ such that it is not the case that $K \vdash \varphi$ and $K \vdash \neg\varphi$. This formula φ exists because K is not complete. We have to show that: (i) $K \vdash \varphi \vee \neg\,\varphi$ holds and (ii) it is not the case that $K \vdash \varphi$ and $K \vdash \neg\varphi$. Point (i) holds because $\varphi \vee \neg\varphi$ follows from the Axioms A1–A3 (see page 13) as the reader may verify. Point (ii) holds because of our choice of the formula φ.

We say that a set Γ of formulas is *inconsistent* iff for every (closed or not closed) formula φ of the language, we have that $\Gamma \vdash \varphi$. One can show that a set Γ of formulas is inconsistent iff there is a (closed or not closed) formula φ such that $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\,\varphi$, that is, $\Gamma \vdash \varphi \wedge \neg\,\varphi$.

By using *false* as an abbreviation of $\varphi \wedge \neg\,\varphi$, for some formula φ, we have that a set Γ of formulas is inconsistent iff $\Gamma \vdash \mathit{false}$. (One can show that this use of the symbol *false* agrees with its identification with the atom *false*.)

We say that a set Γ of formulas is *consistent* iff Γ is not inconsistent.

Proposition 5. Any first order predicate calculus (thus, without proper axioms) is consistent. Thus, it does not exist a (closed or not closed) formula φ such that $\vdash \varphi$ and $\vdash \neg\,\varphi$.

Consistency is an important concept and, as we will see later, the fact that a set Γ of formulas is consistent can be characterized by a semantic property, namely, the fact that Γ has a model (see Proposition 19 on page 36).

2.2 Natural Deduction Presentation

In order to have the Deduction Theorem stated in the simpler form: $\Gamma \vdash \varphi \rightarrow \psi$ iff $\Gamma, \varphi \vdash \psi$, that is, without any extra condition on the derivation of the formula ψ (see

Theorem 1 on page 16), we now present the first order predicate calculus according to the so called Natural Deduction presentation, where the $\vdash$ relation is understood in a different way with respect to the Classical presentation.

In the Natural Deduction presentation we need the notion of a sequent.

A *sequent* is a pair consisting of: (i) a conjunction of formulas (denoted as a set without curly brackets), and (ii) a formula. If the conjunction of formulas is Γ and the formula is φ, the associated sequent is written as $\Gamma \vdash \varphi$.

This notation should not be confused with the derivation relation of the Classical presentation (see Section 2.1). As in the case of the derivation relation, we write the sequent $\Gamma \cup \{\varphi\} \vdash \psi$ as $\Gamma, \varphi \vdash \psi$, and when $\Gamma = \emptyset$ we also write $\vdash \varphi$, instead of $\emptyset \vdash \varphi$.

The formulas of the Natural Deduction presentation of the first order predicate calculus are those of the Classical presentation. Here are the axioms and the rules of inference of the Natural Deduction presentation (see [9]), where Γ denotes any set of formulas, and A, B, and C any formulas. By using these axioms and rules of inference we may deduce new sequents from zero or more old sequents.

$$\frac{}{\Gamma, A \vdash A}$$ (Assumption Axiom)

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$ (Introduction of Assumption)

$$\frac{\Gamma, A \vdash B \qquad \Gamma, \neg A \vdash B}{\Gamma \vdash B}$$ (Elimination of Assumption)

$$\frac{}{\Gamma \vdash \mathit{true}}$$ (*true* Axiom)

$$\frac{}{\Gamma \vdash \neg \mathit{false}}$$ (*false* Axiom)

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$ ($\vee$ Introduction)

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}$$ ($\vee$ Elimination)

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge}$$ ($\wedge$ Introduction)

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$ ($\wedge$ Elimination)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$ ($\to$ Introduction)
(this rule is also called Discharge Rule)

$$\frac{\Gamma \vdash A \quad \Gamma \vdash A \to B}{\Gamma \vdash B}$$ ($\to$ Elimination)
(this rule corresponds to Modus Ponens)

$$\frac{\Gamma, A \vdash B \quad \Gamma, A \vdash \neg B}{\Gamma \vdash \neg A}$$ ($\neg$ Introduction)

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash B}$$ ($\neg$ Elimination: «*ex falso sequitur quodlibet*»)

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A}$$ ($\neg\neg$ Introduction)

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$ ($\neg\neg$ Elimination)

$$\frac{\Gamma \vdash A(x)}{\Gamma \vdash \forall x\, A(x)}$$ if x is not free in Γ ($\forall$ Introduction)

$$\frac{\Gamma \vdash \forall x\, A(x)}{\Gamma \vdash A(t)}$$ if t is free for x in $A(x)$ ($\forall$ Elimination)

$$\frac{\Gamma \vdash A(t)}{\Gamma \vdash \exists x\, A(x)}$$ if t is free for x in $A(x)$ ($\exists$ Introduction)

$$\frac{\Gamma \vdash \exists x\, A(x) \quad \Gamma, A(b) \vdash C}{\Gamma \vdash C}$$ if b is a constant occurring neither in Γ nor in $\exists x\, A(x)$ nor in C ($\exists$ Elimination)

In the ($\forall$ Elimination), ($\exists$ Introduction), and ($\exists$ Elimination) rules, the formulas $A(t)$ and $A(b)$ are obtained by replacing *all* free occurrences of x in $A(x)$ by t and b, respectively.

In the ($\forall$ Elimination) rule a particular term t which is free for x in $A(x)$, is x itself, and thus, we also have that: $\dfrac{\Gamma \vdash \forall x\, A(x)}{\Gamma \vdash A(x)}$.

Note that one could consider the following simpler (but incorrect!) rule for the elimination of the existential quantifier:

$$\frac{\Gamma \vdash \exists x\, A(x)}{\Gamma \vdash A(b)}$$ if b is a constant occurring neither in Γ nor in $\exists x\, A(x)$ (E)

Using this rule we can derive our (∃ Elimination) rule above, as indicated by the following proof tree:

$$\dfrac{\dfrac{\Gamma \vdash \exists x\, A(x)}{\Gamma \vdash A(b)}\ \text{(E)} \qquad \dfrac{\Gamma, A(b) \vdash C}{\Gamma \vdash A(b) \rightarrow C}\ (\rightarrow \text{ Introduction})}{\Gamma \vdash C}\ (\rightarrow \text{ Elimination})$$

However, the condition that the constant b should not occur in the formula C, cannot be added to Rule (E) (simply, because in Rule (E) there is no reference to C) and thus, in general, the simpler Rule (E) for the elimination of the existential quantifier is *not* correct.

With reference to Rule (E), the reader may also look at what we said about Rule (C) on page 18 (and, in particular, he may look at Proposition 2 on page 18).

Given a set Γ of formulas and any two formulas $\varphi(x)$ and $\psi(x)$, we have that the following rules hold [9, pages 120 and 121]:

$$\dfrac{\Gamma, \varphi(x) \vdash \psi(x)}{\Gamma, \forall x\, \varphi(x) \vdash \forall x\, \psi(x)} \qquad \text{if } x \text{ is not free in } \Gamma \qquad (\forall\forall \text{ Introduction})$$

$$\dfrac{\Gamma, \varphi(x) \vdash \psi(x)}{\Gamma, \exists x\, \varphi(x) \vdash \exists x\, \psi(x)} \qquad \text{if } x \text{ is not free in } \Gamma \qquad (\exists\exists \text{ Introduction})$$

In the Natural Deduction presentation we say that a sequent $\Gamma \vdash \varphi$ holds iff there exists a finite tree T such that: (i) the root of T is $\Gamma \vdash \varphi$, (ii) for every node δ of T with $n\,(\geq 0)$ sons $\delta_1, \ldots, \delta_n$, there exists a rule of inference with conclusion δ and premises $\delta_1, \ldots, \delta_n$, and (iii) every leaf is an axiom.

We say that a formula φ is a *theorem* iff the sequent $\vdash \varphi$ holds in the Natural Deduction presentation.

In the Natural Deduction presentation we have the following version of the Deduction Theorem.

Theorem 2. [Deduction Theorem. Version 2] Given a set Γ of (closed or not closed) formulas and any two formulas φ and ψ, we have that $\Gamma \vdash \varphi \rightarrow \psi$ iff $\Gamma, \varphi \vdash \psi$.

Proof. The *if* part is shown by the → Introduction rule. The *only-if* part is shown as follows. By the Introduction of Assumption rule, from $\Gamma \vdash \varphi \rightarrow \psi$ we get $\Gamma, \varphi \vdash \varphi \rightarrow \psi$ (1). By the Assumption axiom from (1) we get $\Gamma, \varphi \vdash \varphi$ (2). From (2) and (1) by the → Elimination rule, we get $\Gamma, \varphi \vdash \psi$.

Note that while $A(x) \vdash \forall x\, A(x)$ is a derivation in the Classical presentation (by applying the Generalization rule), $A(x) \vdash \forall x\, A(x)$ is a sequent which does *not* hold in

the Natural Deduction presentation. Indeed, from the sequent $A(x) \vdash A(x)$ (which holds by the Assumption Axiom) we cannot deduce the sequent $A(x) \vdash \forall x\, A(x)$, because the variable x occurs free in $A(x)$ and the $\forall$ Introduction rule cannot be applied.

Let us consider the following rule, called the *Cut rule*: for all sets Γ of formulas, for all formulas φ and ψ,

$$\frac{\Gamma, \varphi \vdash \psi \qquad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \psi} \qquad \text{(Cut rule)}$$

We have the following Gentzen theorem.

Theorem 3. [Cut-Elimination. Gentzen's Hauptsatz Theorem] Any sequent which has a derivation with the Cut rule, has a derivation without the Cut rule.

We have the following theorem which relates the Classical presentation and the Natural Deduction presentation of the first order theories.

Theorem 4. [Relationship between Classical presentation and Natural Deduction presentation of the First Order Theories] For every set Γ of *closed* formulas (which is the possibly empty set of proper axioms of a first order theory) and for any (closed or not closed) formula φ, there exists a derivation of φ from Γ, that is, $\Gamma \vdash \varphi$ holds in the Classical presentation, iff $\Gamma \vdash \varphi$ is a sequent which holds in the Natural Deduction presentation.

Thus, starting from a set of closed first order formulas, the two presentations of the first order predicate calculus determine two theories which have the same theorems, that is, for any formula φ, we have that φ is a theorem in the Classical presentation iff φ is a theorem in the Natural Deduction presentation.

Note 4. Theorem 4 indicates one more reason, besides the one given in Note 3 on page 16, for the convenience of formalizing any given data base of facts (with its rules of inference and integrity constraints) as a conjunction of closed formulas. Indeed, if we use a conjunction of closed formulas, we may equivalently understand the symbol $\vdash$ as denoting either the derivation relation (as in the Classical presentation) or a sequent (as in the Natural Deduction presentation).

Exercise 4. Show the correctness of the following rule for eliminating the existential quantifier:

$$\frac{\Gamma \vdash \exists x\, A(x) \qquad \Gamma, A(x) \vdash B}{\Gamma \vdash B} \qquad \begin{array}{l}\text{if } x \text{ occurs free} \\ \text{neither in } \Gamma \text{ nor in } B\end{array} \qquad (\exists \text{ Elimination-1})$$

Solution. We have that:

$\Gamma, A(x) \vdash B$	given	
$\Gamma \vdash A(x) \to B$	$\to$ Introduction	
$\Gamma, \exists x\, A(x) \vdash A(x) \to B$	Introduction of Assumption	
$\Gamma, \exists x\, A(x) \vdash \forall x\, (A(x) \to B)$	$\forall$ Introduction (x not free in Γ)	
$\Gamma, \exists x\, A(x) \vdash A(b) \to B$	$\forall$ Elimination (x not free in B and constant b free for x in $A(x) \to B$)	
$\Gamma, \exists x\, A(x),\ A(b) \vdash A(b) \to B$	Introduction of Assumption	
$\Gamma, \exists x\, A(x),\ A(b) \vdash A(b)$	Assumption Axiom	
$\Gamma, \exists x\, A(x),\ A(b) \vdash B$	$\to$ Elimination	(1)
$\Gamma, \exists x\, A(x) \vdash \exists x\, A(x)$	Assumption Axiom	(2)
$\Gamma, \exists x\, A(x) \vdash B$	$\exists$ Elimination (1,2) (b occurring neither in Γ, nor in $\exists x\, A(x)$, nor in B)	

From the facts that: (i) $\Gamma, \exists x\, A(x) \vdash B$ can be derived from $\Gamma, A(x) \vdash B$, and (ii) by using the ($\to$ Introduction) and the ($\to$ Elimination) rules, we have that:

$$\frac{\Gamma \vdash \exists x\, A(x) \qquad \Gamma, \exists x\, A(x) \vdash B}{\Gamma \vdash B}\,,$$

we get the above Rule ($\exists$ Elimination-1).

3 Some Properties of Connectives and Quantifiers

In this section we state some important theorems of the first order predicate calculus. From now on, unless otherwise specified, the symbol $\vdash$ is used for denoting theorems in the Classical presentation (not sequents in the Natural Deduction presentation). However, the reader should bear in mind the result stated in Theorem 4 on page 23.

We have the following proposition [10, page 73] (see also Proposition 9 on page 31).

Proposition 6. For any formula φ of a first order predicate calculus without quantifiers (possibly with free variables), we have that $\vdash \varphi$ iff φ is an instance of a propositional tautology, that is, φ can be derived from the axiom schemata A1, A2, and A3 (see page 13) by using Modus Ponens.

In a first order theory, axioms and theorems may be expressed by formulas which are not closed and, thus, free variables may occur in them. Note, however, that we have the following result (see also Proposition 10 on page 32 for its semantic counterpart).

Proposition 7. [Derivation of Formulas and Their Universal Closures] For every formula φ, we have that: $\vdash \varphi(x)$ iff $\vdash \forall x\, \varphi(x)$.

Proof. The *only-if* part is a consequence of the Generalization rule and the *if* part is a consequence of the fact the variable x is free for x in $\varphi(x)$.

Thus, it is possible (and actually convenient, as indicated in Note 3 on page 16) to write every axiom of a first order theory, not as a formula, say $\varphi(x)$, with free occurrences of the variable x, but as the associated universally quantified, closed formula $\forall x\, \varphi(x)$ (see also Note 4 on page 23).

3.1 Basic Equivalences for Implication and Equivalence

We have the following facts concerning $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence).

For any formulas A, B, and C (with or without occurrences of x), we have that:

(i) $\vdash \neg\neg A \leftrightarrow A$

(ii) $\vdash (A \rightarrow B) \leftrightarrow (\neg A \vee B)$

(iii) $\vdash (A \wedge B \rightarrow C \vee D) \leftrightarrow (A \wedge B \wedge \neg C \rightarrow D)$ (moving to premises)

(iv) $\vdash (A \wedge B \rightarrow C \vee D) \leftrightarrow (A \rightarrow \neg B \vee C \vee D)$ (moving to conclusions)

(v) if $\vdash A \rightarrow B$ then $\vdash (A \wedge C) \rightarrow (B \wedge C)$ ($\wedge$ monotonicity)

(vi) if $\vdash A \rightarrow B$ then $\vdash (A \vee C) \rightarrow (B \vee C)$ ($\vee$ monotonicity)

(vii) if $\vdash A \rightarrow B$ then $\vdash (B \rightarrow C) \rightarrow (A \rightarrow C)$

(viii) $\vdash \forall x\, A(x) \leftrightarrow \neg\, \exists x \neg\, A(x)$

(ix) $\vdash A \leftrightarrow \forall x\, A$ if x does not occur free in A

(x) $\vdash A \leftrightarrow \exists x\, A$ if x does not occur free in A

(xi) if $\vdash A \leftrightarrow B$ then ($\vdash A$ iff $\vdash B$)

This Property (xi) follows from Point (iii) of Proposition 1 on page 16.

Note that the reverse implication, that is, if ($\vdash A$ iff $\vdash B$) then $\vdash A \leftrightarrow B$, does not hold. Indeed, we have that: (i) $\vdash A$ iff $\vdash \neg\, A$ by duality (see Section 3.5 on page 29), and (ii) it is not the case that $\vdash A \leftrightarrow \neg\, A$.

In many formal languages one introduces the *if-then* and *if-then-else* constructs. For those constructs one assumes that:

(i) *if A then B*	stands for	$A \rightarrow B$
(ii) *if A then B else C*	stands for	$(A \rightarrow B) \wedge ((\neg A) \rightarrow C)$
	or, equivalently,	$((\neg A) \vee B) \wedge (A \vee C)$
	or, equivalently,	$(A \wedge B) \vee ((\neg A) \wedge C)$

3.2 Distributivity of Quantifiers over Conjunctions and Disjunctions

For any formulas A and B (with or without occurrences of x),

$\forall$ distributes over $\wedge$: $\vdash \forall x\,(A \wedge B) \;\leftrightarrow\; (\forall x\,A) \wedge (\forall x\,B)$

$\exists$ distributes over $\vee$: $\vdash \exists x\,(A \vee B) \;\leftrightarrow\; (\exists x\,A) \vee (\exists x\,B)$

If an operand of $\wedge$ or $\vee$ does not have the bound variable of the quantifier $\forall$ or $\exists$, then 'it can go *in and out the scope* of the quantifier' and equivalence is preserved. By this sentence we mean that, if x does not occur free in B, then:

(i) $\vdash \forall x\,(A(x) \vee B) \;\leftrightarrow\; (\forall x\,A(x)) \vee B$, and

(ii) $\vdash \exists x\,(A(x) \wedge B) \;\leftrightarrow\; (\exists x\,A(x)) \wedge B$.

Note that the analogous equivalences, $\vdash \forall x\,(A(x) \wedge B) \;\leftrightarrow\; (\forall x\,A(x)) \wedge B$ and $\vdash \exists x\,(A(x) \vee B) \;\leftrightarrow\; (\exists x\,A(x)) \vee B$, trivially hold by distributivity.

Note also that if an operand of $\rightarrow$ does not have the bound variable of the quantifier $\forall$ or $\exists$, then 'it *cannot* go *in and out the scope* of the quantifier'. In particular, we have that $\vdash \forall x\,(A(x) \rightarrow B) \;\leftrightarrow\; ((\exists x\,A(x)) \rightarrow B)$ (note the interchange of the quantifiers $\forall$ and $\exists$) (see also Section 3.4).

If a formula has the outermost quantifier which does not distribute over the top operator of its scope, then 'that formula *is in the middle between* $\forall$ *and* $\exists$'. By this sentence we mean that the following implications hold:

Case 1. $\forall$ does not distribute over $\vee$:

$\vdash \forall x\,A(x) \vee \underline{\forall x\,B(x)} \;\rightarrow\; \underline{\forall x\,(A(x) \vee B(x))}$ and

$\vdash \underline{\forall x\,(A(x) \vee B(x))} \;\rightarrow\; \forall x\,A(x) \vee \underline{\exists x\,B(x)}$

Case 2. $\exists$ does not distribute over $\wedge$:

$\vdash \exists x\,A(x) \wedge \underline{\forall x\,B(x)} \;\rightarrow\; \underline{\exists x\,(A(x) \wedge B(x))}$ and

$\vdash \underline{\exists x\,(A(x) \wedge B(x))} \;\rightarrow\; \exists x\,A(x) \wedge \underline{\exists x\,B(x)}$

Here is the proof of Case 1. The proof of Case 2 is similar and is left to the reader.

$\vdash \forall x\,A(x) \vee \underline{\forall x\,B(x)}$		$\{x$ is not free in $\forall x\,B(x)\}$
iff	$\vdash \forall x\,(A(x) \vee \forall x\,B(x))$	$\{\forall$ elimination$\}$
implies	$\vdash \underline{\forall x\,(A(x) \vee B(x))}$	$\{\exists$ introduction$\}$
implies	$\vdash \forall x\,(A(x) \vee \exists x\,B(x))$	$\{x$ is not free in $\exists x\,B(x)\}$
iff	$\vdash \forall x\,A(x) \vee \underline{\exists x\,B(x)}$	

Figure 1 on page 27 shows some implications involving quantifiers, $\wedge$, and $\vee$.

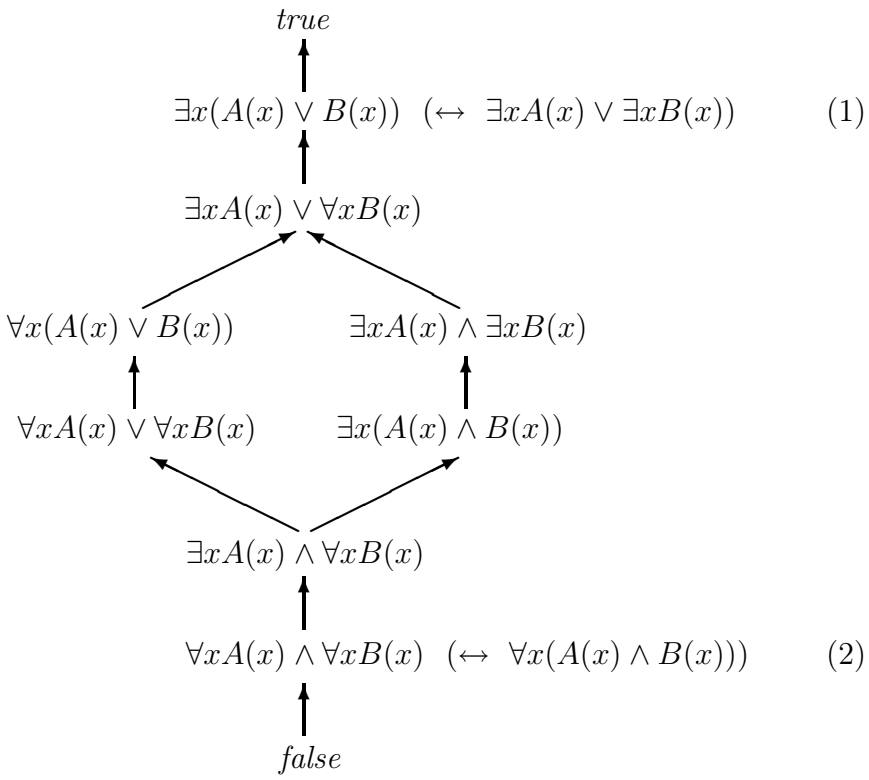

Fig. 1. Some implications involving quantifiers, $\wedge$, and $\vee$. If the formula φ_1 is below the formula φ_2, then $\vdash \varphi_1 \rightarrow \varphi_2$. Equivalence (1) is the distributivity of $\exists$ over $\vee$ and Equivalence (2) is the distributivity of $\forall$ over $\wedge$.

3.3 Prenex Conjunctive and Prenex Disjunctive Normal Form

Definition 1. A formula is said to be in *prenex conjunctive normal form* if it is of the form:

$$Q_1 x_1, \ldots, Q_n x_n \; (\; (A_{1\,1} \vee \ldots \vee A_{1,\,k_1}) \wedge (A_{2\,1} \vee \ldots \vee A_{2,\,k_2}) \wedge \vdots \wedge (A_{m\,1} \vee \ldots \vee A_{m,\,k_m}) \;) \qquad (\dagger)$$

where for $i = 1, \ldots, n$, Q_i is either a universal or an existential quantifier and each $A_{i\,j}$ is either an atomic formula or a negation of an atomic formula. Similarly, a formula is said to be in *prenex disjunctive normal form* if it is of the form (†) above, except that the operators $\vee$ and $\wedge$ have been interchanged.

Note that a formula in prenex conjunctive normal form is not necessarily closed.

Theorem 5. Every formula can be transformed into an equivalent formula in prenex conjunctive normal form and prenex disjunctive normal form [9, page 104].

Proof. We transform the given formula into an equivalent formula in prenex conjunctive normal form by applying the following six step algorithm. Each step of the algorithm preserves logical equivalence. We leave it to the reader to specify the similar algorithm for deriving an equivalent formula in prenex disjunctive normal form.

Construction of the prenex conjunctive normal form of a given formula φ

1. Eliminate all quantifiers in whose scope there is no occurrence of the corresponding bound variable.
2. Rename the bound variables, so that any two quantifiers have two distinct bound variables and no variable has an occurrence which is both bound and free.
3. Eliminate all occurrences of connectives different from $\neg$, $\wedge$, and $\vee$.
4. Push $\neg$ inward by replacing: (i) $\neg\exists x\, A$ by $\forall x\, \neg A$, (ii) $\neg\forall x\, A$ by $\exists x\, \neg A$, (iii) $\neg\neg A$ by A, and (iv) by using De Morgan's laws: $\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$ and $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$.
5. Push quantifiers outward (that is, to the left) by using the following rules (modulo commutativity of $\vee$ and $\wedge$):

   $\forall x\,(A(x) \vee B) \leftrightarrow (\forall x\, A(x)) \vee B$ $\qquad$ $\forall x\,(A(x) \wedge B) \leftrightarrow (\forall x\, A(x)) \wedge B$

   $\exists x\,(A(x) \vee B) \leftrightarrow (\exists x\, A(x)) \vee B$ $\qquad$ $\exists x\,(A(x) \wedge B) \leftrightarrow (\exists x\, A(x)) \wedge B$

   Note that the variable x does not occur free in B (this is ensured by Step 2).
6. Distribute $\vee$ over $\wedge$.

For instance, given the formula $A = \forall x\,(((\forall y\, p(y)) \vee q) \rightarrow \neg\,(\forall y \neg\, r(x,y)))$ we have that:

$$
\begin{array}{ll}
A \leftrightarrow \{\text{by renaming bound variables}\} & \forall x\,(((\forall y\, p(y)) \vee q) \rightarrow \neg\,(\forall z \neg\, r(x,z))) \\
\quad \leftrightarrow \{\text{by eliminating } \rightarrow\} & \forall x\,(\neg\,((\forall y\, p(y)) \vee q) \vee \neg\,(\forall z \neg\, r(x,z))) \\
\quad \leftrightarrow \{\text{by pushing } \neg \text{ inward}\} & \forall x\,(((\exists y \neg\, p(y)) \wedge \neg\, q) \vee (\exists z\, r(x,z))) \\
\quad \leftrightarrow \{\text{by pushing quantifiers outward}\} & \forall x\, \exists y\, \exists z\,((\neg\, p(y) \wedge \neg\, q) \vee r(x,z)) \\
\quad \leftrightarrow \{\text{by distributing } \vee \text{ over } \wedge\} & \forall x\, \exists y\, \exists z\,(\,(\neg\, p(y) \vee r(x,z)) \\
 & \qquad\qquad \wedge\ (\neg\, q\ \ \vee\ \ r(x,z))\,)
\end{array}
$$

This last formula is the prenex conjunctive normal form of the given formula A.

3.4 Quantifiers and Implication

Let us assume that x does not occur free in B. We have that:

(i) $\vdash\ \forall x\,(A(x) \rightarrow B) \ \leftrightarrow\ ((\exists x\, A(x)) \rightarrow B)$ (note that $\forall x$ becomes $\exists x$)

(ii) $\vdash\ \forall x\,(B \rightarrow A(x)) \ \leftrightarrow\ (B \rightarrow (\forall x\, A(x)))$

For Point (i) we have:

$$\vdash (\forall x\,(A(x) \to B)) \text{ iff } \vdash (\forall x\,(\neg A(x) \vee B)) \text{ iff } \vdash ((\forall x\,\neg A(x)) \vee B) \text{ iff}$$
$$\text{iff } \vdash (\neg(\exists x\,A(x)) \vee B) \text{ iff } \vdash ((\exists x\,A(x)) \to B).$$

The proof of Point (ii) is similar and is left to the reader.

Figure 2 on page 29 shows some implications involving quantifiers and $\to$.

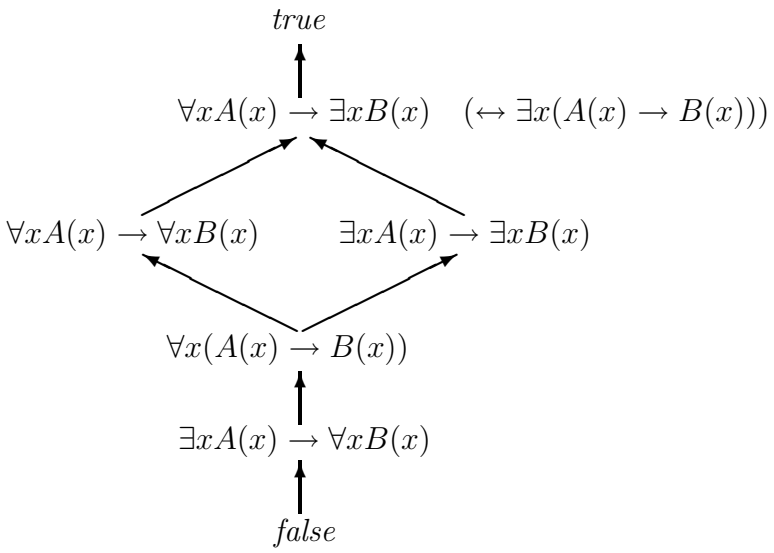

Fig. 2. Some implications involving quantifiers and $\to$. If the formula φ_1 is below the formula φ_2, then $\vdash \varphi_1 \to \varphi_2$.

3.5 Duality

Given a formula A, the dual formula, denoted A^d, is obtained by:
(i) replacing all connectives in A in favour of $\neg$, $\vee$, and $\wedge$ only, and
(ii) interchanging $\vee$ with $\wedge$, $\exists$ with $\forall$, and *true* with *false.*
Note that variables should *not* be interchanged with their negations.

We have that $\vdash A$ iff $\vdash \neg\, A^d$.

For instance, $\vdash \mathit{true} \to a$ iff $\vdash \neg(\mathit{true} \to a)^d$. It reduces to $\vdash a$ iff $\vdash \neg\, a$.
Indeed: (i) $(\mathit{true} \to a) \leftrightarrow a$, and
(ii) $\neg(\mathit{true} \to a)^d \leftrightarrow \neg(\neg \mathit{true} \vee a)^d \leftrightarrow \neg(\neg \mathit{false} \wedge a) \leftrightarrow \neg a$. Obviously, neither $\vdash a$ nor $\vdash \neg\, a$ holds.

4 Semantics of First Order Theories and Gödel's Completeness Theorem

An *interpretation* I for a first order language L is defined as follows.

(1) We take a non-empty set D, called the *domain* of the interpretation.

(2) To each function symbol of arity $r\,(\geq 0)$ we assign a function, that is, a mapping from D^r to D. The elements of D^r are r-tuples of elements in D. To each constant symbol we assign an element of D.

(3) To each predicate symbol of arity $r\,(\geq 0)$ we assign an r-ary relation, that is, a subset of D^r. To *true* we assign the subset of D^0, which is the whole D^0, which is $\{\langle\rangle\}$, that is, the set whose only element is the tuple $\langle\rangle$ with 0 components. To *false* we assign the subset of D^0 which is the empty set, denoted as usual by $\emptyset$.

A *variable assignment* σ for L is obtained by assigning an element of D to each variable of L. Thus, given I and σ, we assign an element d of D to every term t of L as follows:
(i) if t is a variable, say x, then $d = \sigma(x)$, and
(ii) if t is $f(t_1, \ldots, t_r)$ then $d = f_I(d_1, \ldots, d_r)$ where f_I is the mapping from D^r to D assigned to f according to I, and $\langle d_1, \ldots, d_r\rangle$ is the r-tuple of D^r assigned to $\langle t_1, \ldots, t_r\rangle$.

By $\sigma[x/d]$ we denote the variable assignment which is equal to σ except that $\sigma(x) = d$.

Now we define the *satisfaction relation*, which is a ternary relation denoted by $\models$ (see [10]).

Given an interpretation I, a variable assignment σ, and a formula φ, we define when $I, \sigma \models \varphi$ holds by structural induction as we now indicate.

If $I, \sigma \models \varphi$ holds, we say that I *and* σ *satisfies* φ, or equivalently, φ *is true in* the interpretation I for the variable assignment σ. To denote that $I, \sigma \models \varphi$ holds we simply write $I, \sigma \models \varphi$. To denote that $I, \sigma \models \varphi$ does not hold, we write: not $(I, \sigma \models \varphi)$, or equivalently, $I, \sigma \not\models \varphi$.

For any interpretation I, variable assignment σ, and atom $p(t_1, \ldots, t_r)$ with $r \geq 0$, $I, \sigma \models p(t_1, \ldots, t_r)$ iff the r-tuple $\langle v_1, \ldots, v_r\rangle$ belongs to the relation assigned to the predicate symbol p by I, where for $i = 1, \ldots, r$, we have that v_i is the value assigned to t_i according to the given I and σ. Thus, since *true* and *false* have arity 0, for every I and σ we have that $I, \sigma \models \mathit{true}$, because $\langle\rangle$ belongs to $\{\langle\rangle\}$, and $I, \sigma \not\models \mathit{false}$, because $\langle\rangle$ does not belong to $\emptyset$.

Note that the relations assigned to *true* and *false* do not depend on the interpretation I, and for any interpretation I and any variable assignment σ, we have that $I, \sigma \models \mathit{true}$ and $I, \sigma \not\models \mathit{false}$.

For any interpretation I, variable assignment σ, and formula φ, we stipulate that:

$I, \sigma \models \neg\varphi$ iff not $(I, \sigma \models \varphi)$
$I, \sigma \models \varphi \wedge \psi$ iff $I, \sigma \models \varphi$ and $I, \sigma \models \psi$
$I, \sigma \models \varphi \vee \psi$ iff $I, \sigma \models \varphi$ or $I, \sigma \models \psi$
$I, \sigma \models \varphi \rightarrow \psi$ iff $I, \sigma \models \varphi$ implies $I, \sigma \models \psi$, that is, (not $I, \sigma \models \varphi$) or $I, \sigma \models \psi$
$I, \sigma \models \exists x\, \varphi$ iff there exists d in D such that $I, \sigma[x/d] \models \varphi$
$I, \sigma \models \forall x\, \varphi$ iff for all d in D we have that $I, \sigma[x/d] \models \varphi$

We say that φ *is true in an interpretation* I or I *is a model of* φ or else φ *has model* I, and we write $I \models \varphi$, iff for all σ we have that $I, \sigma \models \varphi$. Note that the symbol $\models$ is overloaded in the two relations $I \models \varphi$ and $I, \sigma \models \varphi$.

We say that φ *is false in an interpretation* I iff it does not exist σ such that $I, \sigma \models \varphi$, that is, for all σ it is not the case that $I, \sigma \models \varphi$.

Note that there exist a formula φ and an interpretation I such that neither φ is true in I nor φ is false in I. However, any such formula φ cannot be closed [10, page 50].

For a closed φ, in fact, the quantification 'for all σ' is not significant, and thus, we have the following fact.

Proposition 8. Given an interpretation I and a closed formula φ, *either* (i) φ is true in I (that is, for all σ, it is the case that $I, \sigma \models \varphi$), *or* (ii) φ is false in I (that is, for all σ it is not the case that $I, \sigma \models \varphi$).

We say that a formula φ is *satisfiable* iff there exist I and σ such that $I, \sigma \models \varphi$.

Thus, a *closed* formula φ is satisfiable iff there exists an interpretation I such that $I \models \varphi$. In words, a closed φ is satisfiable iff there exists an interpretation I which is a model of φ.

A formula φ is *unsatisfiable* iff φ is not satisfiable.

Note 5. In [1] (page 513) Apt uses the following different definition of satisfiability: the formula φ is satisfiable iff there exists an interpretation I such that $I \models \varphi$, that is, iff there exists an interpretation I such that *for all* σ we have that $I, \sigma \models \varphi$. When φ is a closed formula Apt's definition of satisfiability is equivalent to ours.

We say that a formula φ is *valid* (or *logically valid*) and we write $\models \varphi$, iff for every interpretation I we have that $I \models \varphi$, that is, for all I and for all σ we have that $I, \sigma \models \varphi$.

We have the following result (see [12, page 297]).

Proposition 9. For any formula φ of a first order predicate calculus without quantifiers (possibly with free variables), we have that $\models \varphi$ holds iff φ is an instance of

a propositional tautology, that is, φ can be derived from the axiom schemata A1, A2, and A3 by using Modus Ponens (see page 13) (This proposition follows from Proposition 6 on page 24 and Gödel's Completeness Theorem on page 35).

For instance, $(p(x) \wedge \forall y\, p(y)) \rightarrow p(x)$ is an instance of the propositional tautology $(A \wedge B) \rightarrow A$.

Proposition 10. [Generalization Preserves Models] For every interpretation I and every formula φ we have that: $I \models \varphi$ (that is, I is a model of φ) iff $I \models \forall x\, \varphi$. (Note that it does not matter whether or not x occurs in φ.) In particular, for every interpretation I and formula φ we have that: $I \models \varphi$ iff $I \models \forall(\varphi)$.

Proof. See [10] (page 310). Hint: by applying the definitions we have to show that: for all σ we have that $I, \sigma \models \varphi$ iff for all σ' and for all d in the domain of I we have that $I, \sigma'[x/d] \models \varphi$.

Note that this theorem is the semantic counterpart of Proposition 7 on page 25. We leave it to the reader to show the following dual statement.

Proposition 11. For every formula φ there exist an interpretation I and a variable assignment σ such that $I, \sigma \models \varphi$ (that is, φ is satisfiable) iff there exist an interpretation I and a variable assignment σ such that $I, \sigma \models \exists x\, \varphi$ (that is, $\exists x\, \varphi$ is satisfiable).

Note that Proposition 11 does not depend on the fact that x occurs free or does not occur free in φ.

As a consequence of Propositions 10 and 11 we have the following statements:
- a formula φ has a model iff its universal closure $\forall(\varphi)$ has a model,
- a formula φ is valid iff its universal closure $\forall(\varphi)$ is valid, and
- a formula φ is satisfiable iff its existential closure $\exists(\varphi)$ is satisfiable (note that this property is not true for Apt's definition of satisfiability).

Definition 2. We say that a formula φ *logically implies* a formula ψ (or ψ is a *logical consequence* of φ), and we write $\varphi \models \psi$, iff for every interpretation I, for every variable assignment σ, we have that $I, \sigma \models \varphi$ implies $I, \sigma \models \psi$.

We have the following proposition.

Proposition 12. For all formulas φ and ψ, we have that: $\varphi \models \psi$ iff $\models \varphi \rightarrow \psi$ (that is, for all interpretations I, for all assignment σ, $I, \sigma \models \varphi \rightarrow \psi$).

In general, by considering a set Γ of formulas and a formula ψ, we write $\Gamma \models \psi$ iff for all formulas $\varphi \in \Gamma$, for all interpretations I, for all assignment σ, if $I, \sigma \models \varphi$, then $I, \sigma \models \psi$.

When Γ is the empty set of formulas, we write $\models \psi$, instead of $\emptyset \models \psi$, and if Γ is a singleton, say $\{\varphi\}$, we will feel free to write $\varphi \models \psi$, instead of $\{\varphi\} \models \psi$.

Definition 3. We say that a formula φ is *logically equivalent* to a formula ψ, and we write $\models \varphi \leftrightarrow \psi$, iff φ logically implies ψ (that is, $\models \varphi \rightarrow \psi$) and ψ logically implies φ (that is, $\models \psi \rightarrow \varphi$).

We have the following proposition.

Proposition 13. For all formulas φ and ψ, we have that: $\models \varphi \leftrightarrow \psi$ iff ($\varphi \models \psi$ and $\psi \models \varphi$).

Note 6. In the relations $\varphi \models \psi$ and $I \models \varphi$ we have overloaded the symbol $\models$. Indeed, on the left of $\varphi \models \psi$ we have a formula, while on the left of $I \models \varphi$ we have an interpretation. This overloading of $\models$, however, does not generate any ambiguity even when no formula occurs to the left of $\models$. Indeed, $\models \varphi$ stands for *true* $\models \varphi$ and this is equivalent to say that 'for every interpretation I, $I \models \varphi$'.

The following propositions follow directly from the definitions.

Proposition 14. For all formulas φ and ψ, if (i) $\varphi \models \psi$ then (ii) (for all I, $I \models \varphi$ implies $I \models \psi$). In words, if $\varphi \models \psi$ then every model of φ is a model of ψ.

Proposition 15. For all formulas φ and ψ, if φ is a closed formula then (i) $\varphi \models \psi$ iff (ii) (for all I, $I \models \varphi$ implies $I \models \psi$). In words, if φ is a closed formula, then $\varphi \models \psi$ is equivalent to say that every model of φ is a model of ψ.

Proof. By Proposition 14 we only need to show that if φ is a closed formula then (ii) implies (i). This can be shown as follows:
(ii) for all I, $I \models \varphi$ implies $I \models \psi$ {by definition}
iff for all I, ((for all σ, $I, \sigma \models \varphi$) implies (for all σ, $I, \sigma \models \psi$))
{since φ is a closed formula, σ does not modify the value of the terms in φ and, thus, σ may be moved on the left}
iff for all I, for all σ, $(I, \sigma \models \varphi)$ implies $(I, \sigma \models \psi)$.

Proposition 16. For all formulas φ and ψ, if φ is *not* a closed formula, then (ii) (for all I, $I \models \varphi$ implies $I \models \psi$) does *not* imply (i) $\varphi \models \psi$.

Proof. Let φ be $x = a$ and ψ be $\forall y\, y = a$. For these formulas we have that:
(ii) (for all I, $I \models x = a$ implies $I \models \forall y\, y = a$) holds by Proposition 10, and
(i) $x = a \models \forall y\, y = a$ does *not* hold (take the interpretation with domain $\{a, b\}$, and let $\sigma(x)$ be a, and $=$ be interpreted as the identity on $\{a, b\}$).

The notions of model, satisfiability, validity, logical implication, and logical equivalence we have defined above, can be extended to a set of formulas, instead of a formula only. Thus, for instance, we have the following definition.

Definition 4. [Models of Sets of Formulas] Given a set Γ of formulas, a *model* of Γ is an interpretation I such that for every formula φ of Γ we have that I is a model of φ, that is, $I \models \varphi$. If I is model of Γ we write $I \models \Gamma$.

When making this extension, a set of formulas represents a conjunction of formulas and we write $\Gamma \cup \{\varphi\} \models \psi$ to mean that the conjunction of every formula in Γ and φ logically implies ψ. Moreover, for reasons of simplicity, we often write $\Gamma, \varphi \models \psi$, instead of $\Gamma \cup \{\varphi\} \models \psi$.

Definition 5. [Models of First Order Theories] Given a first order theory T, a *model* of T is any interpretation I such that for every axiom φ of T, we have that $I \models \varphi$ (see [10, page 56]).

In this Definition 5, instead of saying 'every *axiom* φ of T', we can equivalently say 'every *theorem* φ of T' because the following theorem holds [10, pages 67]. Recall also that a first order theory is identified with the set of its theorems (see page 15).

Theorem 6. [Soundness of the derivation relation $\vdash$ with respect to the satisfiability relation $\models$] Given any first theory T, for all formulas φ, if $T \vdash \varphi$ (that is, φ is a theorem of T), then $T \models \varphi$ (that is, every model of T is a model of φ).

Proof. It is a consequence of: (i) Proposition 10 on page 32 which states that the Generalization rule preserves models, and (ii) the following Proposition 17 which states that Modus Ponens preserves models.

Proposition 17. [Modus Ponens Preserves Models] If $I \models \varphi$ and $I \models \varphi \rightarrow \psi$, then $I \models \psi$.

Proof. It is left as an exercise to the reader.

The following Theorem 7 on page 35 which is the well known Gödel Completeness Theorem, establishes the converse of Theorem 6 and, thus, shows the equivalence between the derivability relation ($\vdash$) and the satisfiability relation ($\models$).

Here are some lemmata which allow us to prove that theorem. Their proofs can be found in [10, pages 67–71]. First we need the following definitions.

Definition 6. [Denumerable Sets] A set D is *denumerable* if there is a bijection between D and the set of natural numbers.

Definition 7. [Recursively Enumerable Sets] A set D is said to be *recursively enumerable* (or *r.e.*, for short) of there exists a Turing Machine M such that: (i) for any natural number n, M with input n halts and returns an element $d \in D$, and (ii) for every element $d \in D$, there exists a natural number i such that M with input i halts and returns d.

If a set is recursively enumerable then it is denumerable. There are sets which are denumerable that are not recursively enumerable.

Lemma 1. The set of terms of a first order theory is denumerable and it is recursively enumerable. The same holds for the set of formulas and the set of closed formulas. (Recall the hypotheses we have made on the first order language (see page 9) and, in particular, the fact that we have denumerably many variables.)

Lemma 2. Given a first order theory K, if A is a *closed* formula such that it is *not* the case that $K \vdash \neg A$, then the theory $K \wedge A$ is consistent.

A first order theory K' is said to be an *extension* of a first order theory K iff every theorem of K is also a theorem of K'.

Lemma 3. (Lindenbaum). If a first order theory K is consistent, then there exists a consistent, complete extension of K. That extension has the same symbols as K.

The proof of Lindenbaum's Lemma uses Zorn's Lemma [10, page 213].

Lindenbaum's Lemma can be strengthened as follows: a consistent, *decidable* first order theory has an extension that is consistent, complete, and *decidable* [14, pages 15–16].

Definition 8. [Witness Theory] A first order theory is said the be *a witness theory* if for every formula $A(x)$ with one free variable only, say x, there exists a closed term t such that $K \vdash (\exists x \neg A(x)) \rightarrow \neg A(t)$.

Lemma 4. Let us consider a first order theory K which is consistent. Then K has a consistent extension which is a witness theory. That extension has a denumerable set of closed terms.

The proof of Lemma 4 can be done by introducing in the language denumerably many new constant symbols.

Lemma 5. Let us consider a first order theory J which is consistent, complete, and it is also a witness theory. Then J has a model whose domain is the set of the closed terms of J.

Lemma 6. (Henkin 1949). Every first order theory which is consistent has a model whose domain is a denumerable set [10, Proposition 2.17 on page 71].

Theorem 7. [Gödel's Completeness Theorem] (1930) For any set Γ of (closed or not closed) first order formulas and for any formula φ, we have that $\Gamma \vdash \varphi$ iff $\Gamma \models \varphi$. In particular, for $\Gamma = \emptyset$, we have $\vdash \varphi$ iff $\models \varphi$, that is, the theorems of the first order predicate calculus are precisely the logically valid formulas.

Note that Gödel's Completeness Theorem holds even if the set Γ of first order formulas is inconsistent and, thus, from Γ we may derive any formula. In that case, in fact, Γ has no models and, trivially, every model of Γ is a model of any formula.

Figure 3 gives a pictorial view of the relationship between Gödel's Completeness Theorem and the Deduction Theorem (Version 1) on page 16.

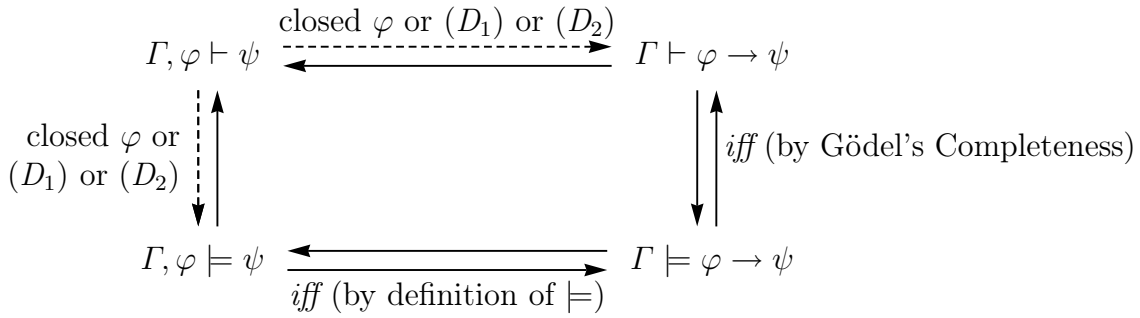

Fig. 3. A pictorial view of the relationship between Gödel's Completeness Theorem (see page 35) and the Deduction Theorem (Version 1) (see page 16). Γ is a set of (closed or not closed) formulas. The symbol $\vdash$ denotes the derivation relation in the Classical presentation of the first order predicate calculus. (D_1) and (D_2) denote Condition (ii.1) and Condition (ii.2), respectively, of the Deduction Theorem. A solid arrow $\alpha \longrightarrow \beta$ denotes the fact that if α holds, then β holds. A dashed arrow $\alpha \overset{\lambda}{\dashrightarrow} \beta$ denotes the fact that if α holds and λ holds, then β holds.

We have the following results whose proofs can be found in [10].

Proposition 18. (Skolem-Löwenheim, 1920, 1915). Any first order theory that has a model, has a denumerable model (that is, a model whose domain is a denumerable set).

Proposition 19. A first order theory K is consistent iff it has a model.

Proof. The *only-if* part is Lemma 6 above on page 35. The *if* part is a consequence of the fact that if K is not consistent then K has no models, as we now show. Indeed, if K is not consistent, then there is a formula A such that $K \vdash A$ and $K \vdash \neg A$. Then, by Gödel's Completeness Theorem, we get that $K \models A$ and $K \models \neg A$. Thus, for every model M of K we have that $M \models A$ and $M \models \neg A$. But this is impossible by definition of $\models$. Thus, K has no models.

Proposition 20. Let $\aleph_0$ denote the cardinality of the set of the natural numbers. For any cardinal number $\alpha \geq \aleph_0$, any consistent first order theory has a model with cardinality α.

Proposition 21. There is no first order theory whose models are exactly all interpretations with finite domains.

Proposition 22. [Compactness] Every finite subset of the axioms of a first order theory K has a model iff the theory K has a model [10, page 73] and [13, page 69].

5 First Order Predicate Calculus With Equality

In this section we consider the first order theory which is the first order predicate calculus with an extra binary predicate symbol $=$ and the following two extra axiom schemata:

E1. $\forall x\ x = x$

E2. $\forall x \forall y\ (x = y \rightarrow (A(x, x) \rightarrow A(x, y)))$

In the axiom schema E2 the formula $A(x, y)$ stands the formula $A(x, x)$ where *some* (maybe all) free occurrences of the variable x have been replaced by the variable y and y is free for x in $A(x, x)$. (Note that $A(x, y)$ may contain or may not contain free occurrences of the variable x.)

From E1 and E2 we get the usual properties of equality:

(i) reflexivity,

(ii) symmetry,

(iii) transitivity, and

(iv) substitutivity, that is, $\forall x \forall y\ (x = y \rightarrow (A(x, x) \leftrightarrow A(x, y)))$
(this Point (iv) follows from E2 and symmetry of $=$).

In particular, from E2 it follows that the equality predicate is symmetric, that is, $\forall x \forall y\ (x = y \rightarrow (A(x, y) \rightarrow A(x, x)))$. Indeed, from E2, by taking $A(x, y)$ to be the formula $y = x$, we get:

$\forall x \forall y\ (x = y \rightarrow (x = x \rightarrow y = x))$, that is,

$\forall x \forall y\ (x = y \rightarrow (\mathit{true} \rightarrow y = x))$, that is,

$\forall x \forall y\ (x = y \rightarrow y = x)$.

We have that $\vdash t = t$ for all terms t [10, page 75].

In what follows we will write $\exists_1 x\, \varphi(x)$ to denote the formula

$$(\exists x\, \varphi(x)) \ \wedge\ (\forall x_1, x_2\, (\varphi(x_1) \wedge \varphi(x_2)) \rightarrow x_1 = x_2)$$

which states that there is a unique x such that $\varphi(x)$.

In the Natural Deduction presentation of the first order predicate calculus with equality we have the following two extra rules for deducing new sequents from old sequents. In these rules t, t_1, and t_2 are any terms and, for $i = 1, 2$, $A(t_i)$ stands for a formula A where we have singled out an occurrence of the term t_i.

$$\frac{}{\Gamma \vdash t = t} \quad (= \text{Axiom})$$

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash A(t_1) \leftrightarrow A(t_2)} \text{ if } t_1 \text{ and } t_2 \text{ are free for } x \text{ in } A(x) \quad (= \text{Rule})$$

Exercise 5. Show that axiom schemata E1 and E2 are true in every interpretation I in which $=$ is interpreted as the identity relation of the domain D of I, that is, as the set $\{\langle x, x\rangle \mid x \in D\}$.

Let us consider the formula $B(x)$ with free occurrences of the variable x. Let the variable y be free for x in $B(x)$ and let $B(y)$ denote the result of substituting all free occurrences of x by y. We have that:

$\vdash \forall x\ (B(x) \leftrightarrow \exists y\,(x = y \wedge B(y)))$
$\vdash \forall x\ (B(x) \leftrightarrow \forall y\,(x = y \rightarrow B(y)))$
$\vdash \forall x\, \exists y\ x = y$

Thus, if t is free for y in $B(y)$ and y does not occur in t, we have that:

$\vdash B(t) \leftrightarrow \forall y\,(y = t \rightarrow B(y))$ (α)
$\vdash B(t) \leftrightarrow \exists y\,(y = t \wedge B(y))$ (β)

In these equivalences (α) and (β) we need the condition that y does not occur in t because, for instance, if t is y, we have $B(y) \leftrightarrow \forall y\, B(y)$, which does not hold (see page 16).

Now we give the proofs of (α) and (β). We leave it as an exercise to the reader construct the similar proofs in the Classical presentation.

Proof of (α).

1. $B(t),\ y{=}t \vdash y{=}t$	Assumption Axiom
2. $B(t),\ y{=}t \vdash B(y) \leftrightarrow B(t)$	$=$ Rule (t is free for y in $B(y)$ and y is free for y in $B(y)$)
3. $B(t),\ y{=}t \vdash B(t) \rightarrow B(y)$	$\leftrightarrow$ Elimination
4. $B(t),\ y{=}t \vdash B(t)$	Assumption Axiom
5. $B(t),\ y{=}t \vdash B(y)$	$\rightarrow$ Elimination (3, 4)
6. $B(t) \vdash y{=}t \rightarrow B(y)$	$\rightarrow$ Introduction
7. $B(t) \vdash \forall y\,(y{=}t \rightarrow B(y))$	$\forall$ Introduction (y is not free in $B(t)$)
8. $\vdash B(t) \rightarrow \forall y\,(y{=}t \rightarrow B(y))$	$\rightarrow$ Introduction
9. $\forall y\,(y{=}t \rightarrow B(y)) \vdash \forall y\,(y{=}t \rightarrow B(y))$	Assumption Axiom
10. $\forall y\,(y{=}t \rightarrow B(y)) \vdash t{=}t \rightarrow B(t)$	$\forall$ Elimination (t is free for y in $(y{=}t \rightarrow B(y))$ and y does not occur in t). $t{=}t \rightarrow B(t)$ is the result of substituting all free occurrences of y in $y{=}t \rightarrow B(y)$ by t.

11. $\forall y\,(y{=}t \to B(y)) \vdash t{=}t$ — $=$ Axiom
12. $\forall y\,(y{=}t \to B(y)) \vdash B(t)$ — $\to$ Elimination (10, 11)
13. $\vdash (\forall y\,(y{=}t \to B(y))) \to B(t)$ — $\to$ Introduction
14. $\vdash B(t) \leftrightarrow (\forall y\,(y{=}t \to B(y)))$ — $\leftrightarrow$ Introduction (8, 13) □

Proof of (β). The symbol b stands for a constant not occurring elsewhere.

1. $B(t) \vdash B(t)$ — Assumption Axiom
2. $B(t) \vdash t{=}t$ — $=$ Axiom
3. $B(t) \vdash t{=}t \wedge B(t)$ — $\wedge$ Introduction (2, 1). $t{=}t \to B(t)$ is the result of substituting all free occurrences of y in $y{=}t \to B(y)$ by t.
4. $B(t) \vdash \exists y\,(y{=}t \wedge B(y))$ — $\exists$ Introduction (y does not occur in t and t is free for y in $(y{=}t \wedge B(y))$)
5. $\vdash B(t) \to \exists y\,(y{=}t \wedge B(y))$ — $\to$ Introduction
6. $\exists y\,(y{=}t \wedge B(y)) \vdash \exists y\,(y{=}t \wedge B(y))$ — Assumption Axiom
7. $\exists y\,(y{=}t \wedge B(y)),\ b{=}t \wedge B(b) \vdash b{=}t \wedge B(b)$ — Assumption Axiom
8. $\exists y\,(y{=}t \wedge B(y)),\ b{=}t \wedge B(b) \vdash b{=}t$ — $\wedge$ Elimination (7)
9. $\exists y\,(y{=}t \wedge B(y)),\ b{=}t \wedge B(b) \vdash B(b)$ — $\wedge$ Elimination (7)
10. $\exists y\,(y{=}t \wedge B(y)),\ b{=}t \wedge B(b) \vdash B(b) \leftrightarrow B(t)$ — $=$ Rule (8) (b and t are free for y in $B(y)$)
11. $\exists y\,(y{=}t \wedge B(y)),\ b{=}t \wedge B(b) \vdash B(b) \to B(t)$ — $\leftrightarrow$ Elimination
12. $\exists y\,(y{=}t \wedge B(y)),\ b{=}t \wedge B(b) \vdash B(t)$ — $\to$ Elimination (9, 11).
13. $\exists y\,(y{=}t \wedge B(y)) \vdash B(t)$ — $\exists$ Elimination (6, 12). Since y does not occur in t, the formula $b{=}t \wedge B(b)$ is the result of substituting b for y in $y{=}t \wedge B(y)$.
14. $\vdash \exists y\,(y{=}t \wedge B(y)) \to B(t)$ — $\to$ Introduction
15. $\vdash B(t) \leftrightarrow (\exists y\,(y{=}t \wedge B(y)))$ — $\leftrightarrow$ Introduction (5, 14) □

Now we present the *group theory* which is an example of a first order theory with equality. The first order language of the group theory has the two function symbols 0 (with arity 0) and + (with arity 2).

The proper axioms are as follows.

G1. $\forall xyz \quad (x+y)+z = x+(y+z)$ (associativity)
G2. $\forall xyz \quad 0+x = x$ (identity)
G3. $\forall x\,\exists y \quad y+x = 0$ (inverse)

Theorem 8. (Gödel, 1930) Any consistent first order theory with equality: (i) has a *normal model*, that is, a model where the equality symbol $=$ is interpreted as the identity relation in the domain of the model, and (ii) that domain is a finite or denumerable set.

5.1 Definition of New Function Symbols

Let K be a first order theory with equality. If for all $y_1, \ldots, y_n$ there exists a *unique* object u such that $A(y_1, \ldots, y_n, u)$, with $n \geq 0$ and free variables $y_1, \ldots, y_n, u$, we may introduce a new function symbol f and stipulate that

$$\forall y_1, \ldots, y_n \, \forall u \; f(y_1, \ldots, y_n) = u \;\rightarrow\; A(y_1, \ldots, y_n, u). \qquad (\dagger)$$

More formally, let us consider a first order theory K with equality such that $K \vdash \exists_1 u \, A(y_1, \ldots, y_n, u)$. Let K' be the first order theory with equality obtained from K by adding: (i) a new function symbol f, (ii) the axiom (†), and (iii) all instances of the axiom schemata A1, A2, A3, A4, A5 (see page 13) and E1 and E2 (see page 37) with some occurrences of f.

Then we can transform, as we will indicate below, any given formula φ' of K' into a formula φ of K such that:

(i) $K' \vdash \varphi' \leftrightarrow \varphi$, and

(ii) if $K' \vdash \varphi'$, then $K \vdash \varphi$.

The transformation, call it *Elim*, from φ' to φ is by induction on the structure of the formula and, for the atomic formulas, by induction on the number of occurrences of the function symbol f.

Given an atomic formula B', the procedure *Elim* returns B' itself if there are no occurrences of f in B', otherwise it returns the formula $\exists u \, A(t_1, \ldots, t_n, u) \wedge B_u$, where B_u is obtained from B' by replacing a term of the form $f(t_1, \ldots, t_n)$, with an *innermost* occurrence of the symbol f, by the fresh new variable u.

Since in K' we have the axiom (†) and $K \vdash \exists_1 u \, A(y_1, \ldots, y_n, u)$ holds (and the same holds for K'), we have that: $K' \vdash \mathit{Elim}(B') \leftrightarrow B'$. From this equivalence we easily get Point (i) above. For Point (ii) above the reader may refer to [10, page 80].

Thus, in first order theories with equality, only predicate symbols are needed while function symbols are dispensable. Indeed, we can replace a function symbol f with arity n (≥ 0), by a new predicate symbol A of arity $n+1$ if we add the new axiom $\forall y_1, \ldots, y_n \, \exists_1 u \, A(y_1, \ldots, y_n, u)$.

6 Peano Arithmetic and Incompleteness Theorem

Let us consider the first order predicate calculus with equality and one constant 0 and three function symbols: s (unary), called *successor*, $+$ (binary), called *plus*, and $\times$ (binary), called *times*, and the following extra axioms:

PA1. $\forall xyz\ (x = y \rightarrow (x = z \rightarrow y = z))$

PA2. $\forall xy\ \ (x = y \rightarrow (s(x) = s(y)))$

PA3. $\forall x\ \ \ 0 \neq s(x)$

PA4. $\forall xy\ \ (s(x) = s(y) \rightarrow x = y)$

PA5. $\forall x\ \ \ x + 0 = x$

PA6. $\forall xy\ \ x + s(y) = s(x + y)$

PA7. $\forall x\ \ \ x \times 0 = 0$

PA8. $\forall xy\ \ x \times s(y) = (x \times y) + x$

PA9. for any formula $\varphi(x)$, $(\varphi(0) \wedge \forall x\,(\varphi(x) \rightarrow \varphi(s(x)))) \rightarrow \forall x\,\varphi(x)$

Let us refer to this first order predicate calculus as Peano Arithmetic or PA, for short.

Note that: (i) PA1 is a consequence of E1, and (ii) PA2 is a consequence of E1 and E2. PA9 is an axiom schema and it provides an axiom for each given first order formula $\varphi(x)$. PA9 is called the *principle of mathematical induction*. Note that it is not possible to present PA using a finite number of axioms only, that is, PA is not finitely axiomatizable.

Exercise 6. Show that PA is a witness theory (see Definition 8 on page 35).

The following theorem establishes the incompleteness of Peano Arithmetic [10, page 163].

Theorem 9. [Gödel-Rosser Incompleteness Theorem. Version 1] (1936) If PA is consistent, then there exists a closed formula which is undecidable in PA, that is, there exists a closed formula φ such that neither $\text{PA} \vdash \varphi$ nor $\text{PA} \vdash \neg\,\varphi$.

Now, in order to present some more results about Peano Arithmetic we introduce the notion of a Gödel number [10, page 149].

Let us consider a first order language L. Let a *symbol* σ of L be one of the following syntactic tokens (as usual, the vertical bar | separates the different alternatives):

$\sigma ::=\ (\ \ |\ \ )\ \ \ |\ ,\ \ |\ \neg\ \ |\rightarrow|\ \forall\ \ |\ \exists\ \ |\ x\ \ |\ f\ \ |\ p$

where x is any variable symbol, f is any function symbol, and p is any predicate symbol of L. An *expression* of L be either a symbol of L, or a term of L, or a formula of L.

The Gödel number of a symbol, or an expression, or a finite sequence of expressions of L, is a natural number which is the value of a Turing computable injection. Thus, in particular, any two different expressions have two different Gödel numbers.

Here we will not give the details of how to define such an injection and, actually, many definitions are possible. The interested reader may refer to [10, page 149].

One can define such an injection so that types are made explicit, in the sense that: (i) the Gödel number of any symbol σ is different from the Gödel number of the expression made out of that single symbol σ and, similarly, (ii) the Gödel number of any expression e is different from the Gödel number of the sequence of expressions made out of that single expression e.

One can define a particular injection that is a primitive recursive function (and, thus, Turing computable) whose inverse is a primitive recursive function (when restricted to its domain of definition).

In what follows we will assume a fixed primitive recursive injection (with a primitive recursive inverse) that provides the Gödel number of every symbol, or expression, or sequence of expressions we will consider. That fixed injection will be denoted by G. The number n returned by the function G, when expressed within PA, is the term $s(\ldots s(0)\ldots)$ with n occurrences of the function symbol s. That term will be denoted also by $\overline{n}$.

In order to state the following Theorem 10 we need the definition we now present [10, page 165].

Definition 9. Let us consider a first order theory K. We define the predicate $PrAx_K$ such that for any natural number n, $PrAx_K(n)$ is true iff n is the Gödel number of a proper axiom of K.

If $PrAx_K$ is recursive there is a Turing Machine, say M, which always halts and for any given natural number n, M tells us whether or not n is the Gödel number of a proper axiom of K (and that proper axiom is the value of $G^{-1}(n)$).

Theorem 10. For any theory K which has the same symbols as PA (that is, 0, s, $+$, $\times$, and $=$) and it is a consistent extension of PA such that the predicate $PrAx_K$ is recursive, we have that K has an undecidable, closed formula.

Let us consider the following interpretation $\mathcal{N}$ of Peano Arithmetic. The domain of the interpretation, also denoted $\mathcal{N}$, is the set of the natural numbers. The nullary symbol 0 is interpreted as the number 0, the unary symbol s is interpreted as the successor function $\lambda n.\, n + 1$, the binary symbol $+$ is interpreted as the sum function $\lambda m, n.\, m + n$, the binary symbol $\times$ is interpreted as the product function $\lambda m, n.\, m \times n$, and the binary symbol $=$ is interpreted as the identity relation on $\mathcal{N}$.

The interpretation $\mathcal{N}$ is a model of PA and it is said to be the *standard model* of Peano Arithmetic. Note that the standard model $\mathcal{N}$ is a normal model because of the interpretation of $=$ as the identity relation on $\mathcal{N}$.

Models of PA which are not isomorphic to the standard model are said to be *non-standard*.

The fact that $\mathcal{N}$ is a model of PA implies that PA is consistent (by Proposition 19 on page 36). Unfortunately, this model theoretic proof of consistency of PA is often considered not very satisfactory (simply because it is based on semantics) and logicians have been looking for a syntactic proof of consistency of PA.

However, the Gödel Second Theorem [10, page 166] (which we do not prove here) states that such a syntactic proof of consistency (when consistency is formulated as the absence of a formula φ for which we have a proof of φ and a proof of $\neg\,\varphi$) cannot be done within PA. There is syntactic proof of consistency of PA [4,5] which uses an induction principle, the so called ϵ_0 induction principle, that is much stronger than mathematical induction (see the axiom schema PA9 on page 41). Obviously, ϵ_0 induction cannot be reduced to mathematical induction and, thus, such a consistency proof of PA is not within PA.

Definition 10. A subset A of natural numbers is said to be *arithmetical* if in PA there is a formula $\alpha(x)$ with a single free variable x such that for all n, $n \in A$ iff $\alpha(s^n(0))$ is true in the standard model of PA, that is, $\mathcal{N} \models \alpha(s^n(0))$.

The following theorem states that in PA there is no formula (with a single free variable) which is true in the standard model of PA exactly for those numbers which are the Gödel numbers of the formulas that are true in the standard model of PA [10, page 169].

Theorem 11. [Tarski Theorem] (1936) The set of the Gödel numbers of all the formulas which are true in the standard model of PA is not arithmetical. That is, in PA no formula $\alpha(x)$ (with a single free variable x) exists such that for all formulas φ of PA, $\mathcal{N} \models \alpha(G(\varphi))$ holds iff $\mathcal{N} \models \varphi$. (Note that, since $\alpha(G(\varphi))$ should be a formula of PA, by $G(\varphi)$ we denote the Gödel number of the formula φ written in unary notation by making use of the symbols s and 0.)

Note that Tarski Theorem refers to the *standard model* $\mathcal{N}$ of PA, while Gödel-Rosser Incompleteness Theorem (see Theorem 9 on page 41) refers to the derivability relation $\vdash$ and, thus, to *all models* of PA (recall that by Gödel Completeness Theorem (see Theorem 7 on page 35) we have that $\vdash$ is equivalent to $\models$).

6.1 More on Incompleteness

In this section we would like to investigate more on the incompleteness properties of first order theories. Let us introduce the following notions.

Let N denote, as usual, the set of the natural numbers.

Definition 11. A function f from N^n to N is said to be *representable in* a first order theory K with equality with the same symbols as PA (that is, 0, s, $+$, $\times$, $=$) if there exists a formula $\alpha(x_1, \ldots, x_n, x_{n+1})$ with the free variables $x_1, \ldots, x_n, x_{n+1}$, such that for all natural numbers $k_1, \ldots, k_n, k_{n+1}$,

(i) if $f(k_1, \ldots, k_n) = k_{n+1}$ then $K \vdash \alpha(\overline{k_1}, \ldots, \overline{k_n}, \overline{k_{n+1}})$, and
(ii) $K \vdash \exists_1 x_{n+1}\, \alpha(\overline{k_1}, \ldots, \overline{k_n}, x_{n+1})$

We say that f from N^n to N is *strongly representable in* a first order theory K with equality with the same symbols as PA iff we have: (i) and
(ii)* $K \vdash \exists_1 x_{n+1}\, \alpha(x_1, \ldots, x_n, x_{n+1})$.

One can show that strong representability is equivalent to representability.

Let us consider the first order theory with equality with the same symbols as PA, called *Raphael Robinson Arithmetic*, or RR for short, whose proper axioms are the following ones [10, page 157]:

RR1. $\forall x \quad x = x$
RR2. $\forall xy \quad x = y \rightarrow y = x$
RR3. $\forall xyz \;\; x = y \rightarrow (y = z \rightarrow x = z)$
RR4. $\forall xy \quad x = y \;\leftrightarrow\; s(x) = s(y)$
RR5. $\forall xyz \;\; x = y \;\rightarrow\; (x + z = y + z \;\wedge\; z + x = z + y)$
RR6. $\forall xyz \;\; x = y \;\rightarrow\; (x \times z = y \times z \;\wedge\; z \times x = z \times y)$
RR7. $\forall x \quad 0 \neq s(x)$
RR8. $\forall xy \quad 0 \neq x \;\rightarrow\; \exists y\; x = s(y)$
RR9. $\forall x \quad x + 0 \;=\; x$
RR10. $\forall xy \quad x + s(y) \;=\; s(x + y)$
RR11. $\forall x \quad x \times 0 \;=\; 0$
RR12. $\forall xy \quad x \times s(y) \;=\; (x \times y) + x$

In [10, page 157], for reasons of simplicity, the following extra axiom (uniqueness of remainder) is also considered:

RR13. $\forall xyzr\; \forall x_1 y_1 z_1 r_1$
$(x = y \times z + r \;\wedge\; r < y \quad \wedge \quad x_1 = y_1 \times z_1 + r_1 \;\wedge\; r_1 < y_1) \;\rightarrow\; r = r_1$

where $x < y$ stands for $\exists z\; x + z = y$. However, this axiom is a consequence of the other axioms.

We have the following result for which we assume familiarity with the theory of the partial recursive functions.

Proposition 23. Every partial recursive function which is a total function, is representable (and strongly representable) in RR.

The first order theory RR is a proper subtheory of PA in the sense that every theorem of RR is a theorem of PA and not vice versa.

Note also that RR is finitely axiomatizable, while it can be shown that PA does not admit a finite axiomatization.

Let us consider any first order theory K with equality with the same symbols as PA such that:

(A1) the predicate $PrAx_K$ is recursive (see page 42),

(A2) $K \vdash 0 \neq s(0)$, and

(A3) every partial recursive function that is a total function, is representable in K.

Examples of a theory K are RR, PA, and any extension of RR which satisfies assumption (A1).

Definition 12. [ω-consistency] We say that a first order theory K with equality with the same symbols as PA is *ω-consistent* iff for every formula $\alpha(x)$, we have that if $K \vdash \neg\alpha(\overline{n})$ for every natural number n, then $K \vdash \exists x\, \alpha(x)$ does not hold.

Every theory for which $\mathcal{N}$ is a model, is an ω-consistent theory. Thus, both PA and RR are ω-consistent.

We have that ω-consistency implies consistency, but not vice versa [10, page 161].

Theorem 12. [Gödel Incompleteness Theorem] (1931) For every first order theory K with equality with the same symbols as PA satisfying assumptions (A1), (A2), and (A3), there exists a closed formula $\mathcal{G}$ such that:

if K is consistent, then it is not the case that $K \vdash \mathcal{G}$

if K is ω-consistent, then it is not the case that $K \vdash \neg\, \mathcal{G}$.

Definition 13. [ω-incompleteness] We say that a first order theory K with equality with the same symbols as PA is *ω-incomplete* iff there exists a formula $\alpha(x)$ such that $K \vdash \alpha(\overline{n})$ for every natural number n, and $K \vdash \forall x\, \alpha(x)$ does not hold.

Proposition 24. Every first order theory with equality with the same symbols as PA such that assumptions (A1), (A2), and (A3) hold, is ω-incomplete [10, Exercise 3.45].

Thus, both PA and RR are ω-incomplete.

Theorem 13. [Gödel-Rosser Incompleteness Theorem. Version 2] (1936) Consider any first order theory K with equality with the same symbols as PA satisfying assumptions (A1), (A2), and (A3), and also the following two assumptions: for every natural number n,

(A4) $K \vdash x \leq \overline{n} \;\rightarrow\; x = 0 \vee x = s(0) \vee \ldots \vee x = \overline{n}$, and

(A5) $K \vdash x \leq \overline{n} \;\vee\; \overline{n} \leq x$.

Then, there exists a closed formula $\mathcal{R}$ such that if K is consistent, then $\mathcal{R}$ is undecidable. (Note that RR and PA are particular instances of any such theory K.)

Exercise 7. If K is a first order theory with the symbols 0 and s, we have that if K is consistent and is not ω-consistent, then it is ω-incomplete [10, Exercise 3.46].
Solution. Since K is not ω-consistent we have that there exists a formula $\alpha(x)$ such that for every natural number n, $K \vdash \neg\alpha(\overline{n})$ and $K \vdash \exists x\, \alpha(x)$. Thus, from consistency of K, we get that it cannot be the case that $K \vdash \neg\, \exists x\, \alpha(x)$, that is, it cannot be the case that $K \vdash \forall x\, \neg\alpha(x)$.

We also have the following undecidability result [10, page 165] which is the reason why we have introduced the theory RR. Since RR is finitely axiomatizable, we see that undecidability due to Gödel-Rosser Incompleteness Theorem (see Theorem 9 on page 41) is *not* related to the fact that PA is not finitely axiomatizable.

Proposition 25. Consider any first order theory K with equality with the same symbols as PA. If K is a consistent extension of RR such that there exists a theory H with the same theorems of K such that $PrAx_H$ is recursive, then in K there exists a closed formula which is undecidable.

Chapter 2

Logic Programming

7 Towards Logic Programming: Skolem, Herbrand, and Robinson Theorems

In this section we return to the first order predicate calculus and we consider three theorems, namely:
(i) the Skolem Theorem (see Theorem 14 on page 47),
(ii) the Herbrand Theorem (see Theorem 16 on page 49), and
(iii) the Robinson Theorem (see Theorem 17 on page 49)
that introduce us to the theory of logic programming which we will present in the next chapter.

Theorem 14. (Skolem 1920). For any given (closed or not closed) formula φ there exists a closed formula ψ in clausal form such that φ is satisfiable iff ψ is satisfiable.

Obviously, we get an equivalent formulation of the Skolem theorem if we replace 'satisfiable' by 'unsatisfiable'. Here is the construction of ψ from φ, according to [9].

Construction of a formula ψ in clausal form which is satisfiable iff φ is satisfiable (Use of Skolemization)

1. Take the existential closure $\exists(\varphi)$ of φ (recall that φ is satisfiable iff $\exists(\varphi)$ is satisfiable), and eliminate all quantifiers in whose scope there is no occurrence of the corresponding bound variable.
2. Rename the bound variables, so that any two quantifiers have two distinct bound variables.
3. Eliminate all occurrences of connectives different from $\neg$, $\wedge$, and $\vee$.
4. Push $\neg$ inward by replacing: (i) $\neg\exists x\ A$ by $\forall x\ \neg A$, (ii) $\neg\forall x\ A$ by $\exists x\ \neg A$, (iii) $\neg\neg A$ by A, and (iv) by using De Morgan's laws: $\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$ and $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$.

5. Push quantifiers inward by using distributivity of $\exists$ over $\vee$ and distributivity of $\forall$ over $\wedge$.
6. (*Skolemization Step*) Eliminate all existential quantifiers by replacing the formula $\forall x_1 \ldots x_m \exists y\, B(y)$, where $\exists y$ is the leftmost existential quantifier occurring in the formula at hand, by $\forall x_1 \ldots x_m\, B(f(x_1, \ldots, x_m))$, where f is a new function symbol. If $m = 0$ then $\exists y\, B(y)$ is replaced by $B(c)$ where c is a new constant symbol.
7. Push universal quantifiers outward (that is, to the left).
8. Distribute $\vee$ over $\wedge$.
9. (Optional) Simplify the derived formula by: (i) using factoring (the definition of factoring and the fact that it preserves satisfiability are presented in Section 7) or any other transformation which preserves satisfiability (in particular, any transformation based on logical equivalence because logical equivalence preserves satisfiability), or (ii) eliminating tautologies.

For instance, by using factoring $\forall x(\neg p(x) \vee \neg p(a))$ is replaced by $\neg p(a)$ (because $\forall x\, (\neg p(x) \vee \neg p(a))$ is satisfiable iff $\forall x\, \neg p(a)$ is satisfiable) and the tautology $q \vee \neg q$ is replaced by *true*.

Example 2. From:

(α) $\forall x\, (p(x) \rightarrow \exists z\, (\neg \forall y\, (q(x,y) \rightarrow p(f(w))) \wedge \forall y\, (q(x,y) \rightarrow p(x))))$

we get the following sequence of formulas where we have underlined some relevant parts to highlight the changes, and we have used square brackets for improving readability:

1. $\underline{\exists w}\, \forall x\, (p(x) \rightarrow (\neg \forall y\, (q(x,y) \rightarrow p(f(w))) \wedge \forall y\, (q(x,y) \rightarrow p(x))))$
2. $\exists w\, \forall x\, (p(x) \rightarrow (\neg \forall y\, (q(x,y) \rightarrow p(f(w))) \wedge \underline{\forall z}\, (q(x,z) \rightarrow p(x))))$
3. $\exists w\, \forall x\, (\neg p(x) \vee (\underline{\neg \forall y\, (\neg q(x,y) \vee p(f(w)))} \wedge \forall z\, (\neg q(x,z) \vee p(x))))$
4. $\exists w\, \forall x\, (\neg p(x) \vee (\underline{\exists y}\, (q(x,y) \wedge \neg p(f(w))) \wedge \forall z\, (\neg q(x,z) \vee p(x))))$
5. $\underline{\exists w}\, \forall x\, (\neg p(x) \vee ((\exists y\, q(x,y) \wedge \neg p(f(\underline{w}))) \wedge (\forall z\, \neg q(x,z) \vee p(x))))$
6. $\forall x\, (\neg p(x) \vee ((q(x, g(x)) \wedge \neg p(f(c))) \wedge (\forall z\, \neg q(x,z) \vee p(x))))$
7. $\underline{\forall x \forall z}\, [\neg p(x) \vee [\underline{(q(x, g(x)) \wedge \neg p(f(c)))} \wedge \underline{(\neg q(x,z) \vee p(x))}]]$
8. $\forall x \forall z\, \underline{[[\neg p(x) \vee (q(x, g(x)) \wedge \neg p(f(c)))]} \wedge [\underline{\neg p(x)} \vee \neg q(x,z) \vee \underline{p(x)}]]$

(from 7 by distributivity)

9.1 $\forall x \forall z\, [\neg p(x) \vee \underline{[q(x, g(x)) \wedge \neg p(f(c))]}]$ (from 8 by tautology elimination)

9.2 $\forall x \forall z\, \underline{[[\neg p(x) \vee q(x, g(x))]} \wedge [\underline{\neg p(x)} \vee \underline{\neg p(f(c))}]]$ (from 9.1 by distributivity)

9.3 $\forall x\, [[\neg p(x) \vee q(x, g(x))] \wedge \neg p(f(c))]$ (from 9.2 by factoring and elimination of $\forall z$)

The two clauses of formula 9.3 are: $\neg p(x) \vee q(x, g(x))$ and $\neg p(f(c))$. We have that the formula (α) is satisfiable iff formula 9.3 is satisfiable. □

Now let us introduce the notions of Herbrand Universe, Herbrand Base, and Herbrand interpretation. We assume that the first order language L we consider, has at least one constant symbol.

The *Herbrand Universe HU* of L is the set of all ground terms which can be constructed using symbols from L. The Herbrand Universe of L is not empty because there is at least one constant symbol.

The *Herbrand Base HB* of L is the set of all ground atoms which can be constructed using symbols from L. The Herbrand Base is empty iff in L there are no predicate symbols.

An *Herbrand interpretation* on L is an interpretation I such that:

- the domain of I is HU,

- to every function symbol f of arity $r\,(\geq 0)$ I assigns the mapping from HU^r to HU which maps every r-tuple $\langle t_1, \ldots, t_r \rangle$ of ground terms $t_1, \ldots, t_r$ to the ground term $f(t_1, \ldots, t_r)$, and

- to every predicate symbol of arity $r\,(\geq 0)$ I assigns a set of r-tuples of terms, each of which belongs to HU. For any predicate symbol p and Herbrand interpretation I, we write $p(t_1, \ldots, t_r) \in I$ for denoting that $\langle t_1, \ldots, t_r \rangle$ belongs to the relation assigned to p by I. Thus, an Herbrand interpretation I of L uniquely determines a subset S_I of the Herbrand Base HB of L and vice versa. Actually, we identify an Herbrand interpretation I with the corresponding subset S_I of the Herbrand Base.

For any formula φ of the language L, an Herbrand interpretation which is a model of φ is said to be an *Herbrand model* of φ.

Theorem 15. A formula φ in clausal form is satisfiable iff φ has an Herbrand model.

Thus, we have that φ in clausal form is satisfiable iff there exists an Herbrand interpretation I such that $I \models \varphi$. Thus, a formula φ in clausal form is unsatisfiable iff φ has no Herbrand models iff φ is false in all Herbrand interpretations (recall that every formula in clausal form is a closed formula).

Theorem 16. (Herbrand 1930). A formula φ in clausal form is unsatisfiable iff there exists a finite conjunction of ground instances of its clauses which is unsatisfiable.

Theorem 17. (Robinson 1965). A formula φ in clausal form is unsatisfiable iff starting from the set of clauses of φ we can get the empty clause □ by zero or more resolution steps.

Now we define a resolution step.

Resolution Step

Given a set of clauses, consider two of them, say

C_1: $L_1 \vee \ldots \vee L_i \vee L_{i+1} \vee \ldots \vee L_k$ and C_2: $M_1 \vee \ldots \vee M_j \vee M_{j+1} \vee \ldots \vee M_h$.

Without loss of generality, we may assume that these clauses do not have variables in common.

(1) Choose nondeterministically from clause C_1 a disjunction of $i\,(\geq 1)$ atoms, say

$p(u_{11}, \ldots, u_{1r}) \vee \ldots \vee p(u_{i1}, \ldots, u_{ir})$

and from clause C_2 a disjunction of $j\,(\geq 1)$ negated atoms, say

$\neg p(v_{11}, \ldots, v_{1r}) \vee \ldots \vee \neg p(v_{j1}, \ldots, v_{jr})$

They all share the same predicate symbol p. Without loss of generality, we may assume that those disjunctions are at the leftmost positions in the corresponding clauses. Let ϑ be a most general unifier of the $i+j$ atoms $p(u_{11}, \ldots, u_{1r}), \ldots, p(u_{i1}, \ldots, u_{ir}), p(v_{11}, \ldots, v_{1r}), \ldots, p(v_{j1}, \ldots, v_{jr})$ occurring in those disjunctions, that is, ϑ is a most general substitution such that:

$p(u_{11}, \ldots, u_{1r})\vartheta = \ldots = p(u_{i1}, \ldots, u_{ir})\vartheta = p(v_{11}, \ldots, v_{1r})\vartheta = \ldots = p(v_{j1}, \ldots, v_{jr})\vartheta$.

(2) Add to the set of clauses the following clause: $(L_{i+1} \vee \ldots \vee L_k \vee M_{j+1} \vee \ldots \vee M_h)\vartheta$.

Given a set P of clauses, a resolution step generates a new set Q of clauses such that $P \subseteq Q$ and P is satisfiable iff Q is satisfiable. (We adopt the convention of Section 2.2 where sets of formulas denote conjunctions of formulas.) In other words, a resolution step preserves satisfiability (and unsatisfiability) of a set of clauses.

To compute a most general unifier of two atoms one can use the following algorithm. Let $u_1, \ldots, u_r$, $v_1, \ldots, v_r$, $s_1, \ldots, s_n, t_1, \ldots, t_n$, and t denote terms and x denote a variable.

Unification Algorithm (Martelli-Montanari) [1] (page 501).

Input: two atoms $p(u_1, \ldots, u_r)$ and $p(v_1, \ldots, v_r)$ without variables in common.

Output: an idempotent most general unifier of $p(u_1, \ldots, u_r)$ and $p(v_1, \ldots, v_r)$, if there exists one. Otherwise, the algorithm halts with failure, that is, it halts by declaring failure of unification.

Consider the set of equations $\{u_1 = v_1, \ldots, u_r = v_r\}$.

REPEAT choose nondeterministically one equation, say E, from the set of equations at hand and perform one of the following actions:

(i) *if* E is $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ *then* replace E by: $s_1 = t_1, \ldots, s_n = t_n$

(ii) *if* E is $f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)$ and f is different from g
then 'halt with failure'

(iii) *if* E is $x=x$ *then* delete E

(iv) *if* E is $t=x$ where t is not a variable *then* replace E by: $x=t$

(v) *if* E is $x=t$ where: (1) t is not x and (2) x occurs either in t or in another equation of the set of equations at hand
then *if* x occurs in t *then* 'halt with failure' *else* substitute t for x in every other equation

UNTIL 'halt with failure' or no action in (i)–(v) can be performed.

This unification algorithm can be extended to any finite set of atoms with the same predicate symbol. Indeed, it is enough to start from a set equations which, for any two atoms in the given set, entails the equalities of their corresponding arguments. For instance, if we have to unify the set $\{p(u_{11}, u_{12}), p(u_{21}, u_{22}), p(u_{31}, u_{32})\}$ of atoms, a possible set of equations to be given as input to the Martelli-Montanari's Unification Algorithm is $\{u_{11}=u_{21}, u_{21}=u_{31}, u_{12}=u_{22}, u_{22}=u_{32}\}$.

The notions of a unifier and a most general unifier of atoms can be extended to terms as follows. The unifier of the terms $t_1, \ldots, t_n$ is the unifier of the atoms $p(t_1), \ldots, p(t_n)$, where p is a new unary predicate symbol. Analogously, for the most general unifier.

Now we present a different algorithm for computing the most general unifier, if any, of $n\,(\geq 2)$ atoms, say $p(u_{11}, \ldots, u_{1r})$, ..., $p(u_{n1}, \ldots, u_{nr})$. Let σ be a substitution and $p_1, \ldots, p_n$ be n variables ranging over atoms. Let $\{\}$ denote the identity substitution.

Unification Algorithm (Robinson) [9] (page 139).
Input: n atoms $p(u_{11}, \ldots, u_{1r}), \ldots, p(u_{n1}, \ldots, u_{nr})$ without variables in common.
Output: an idempotent most general unifier of $p(u_{11}, \ldots, u_{1r}), \ldots, p(u_{n1}, \ldots, u_{nr})$, if there exists one. Otherwise, the algorithm halts with failure, that is, it halts by declaring failure of unification.

$\langle \sigma, p_1, \ldots, p_n \rangle := \langle \{\}, p(u_{11}, \ldots, u_{1r}), \ldots, p(u_{n1}, \ldots, u_{nr}) \rangle$;
WHILE (T does not 'halt with failure') and
there exist i and j in $\{1, \ldots, n\}$ such that $p_i \neq p_j$
DO $\langle \sigma, p_1, \ldots, p_n \rangle := T(\sigma, p_1, \ldots, p_n)$;
where the transformation T, given a substitution σ and the atoms p_1, ..., p_n, *either* generates the new substitution σ' and the new atoms p'_1, ..., p'_n *or* halts with failure, as we now indicate.

Let us consider two atoms, say p_i and p_j, in the set $\{p_1, \ldots, p_n\}$ such that $p_i \neq p_j$. Let s_1 and s_2 be the two terms (or subterms) in p_i and p_j, respectively, such that s_1 and s_2 are rooted at the leftmost position where p_i and p_j, when considered as strings of symbols, have different symbols.

If either $\langle s_1, s_2\rangle$ or $\langle s_2, s_1\rangle$ is equal to $\langle x, t\rangle$, where x is a variable and t is a term in which x does not occur,
then $\langle \sigma', p'_1, \ldots, p'_i, \ldots, p'_j, \ldots, p'_n\rangle :=$
$\langle \sigma\{x/t\} \cup \{x/t\},\ p_1\{x/t\}, \ldots, p_i\{x/t\}, \ldots, p_j\{x/t\}, \ldots, p_n\{x/t\}\rangle$
else T 'halts with failure'.
OD

Note that $\sigma\{x/t\} \cup \{x/t\}$ is equal to $\sigma\{x/t\}$ because we have assumed that p_i and p_j do not have variables in common.

Example 3. (See [9], page 139) Given the two atoms $p_1 = p(a, x, f(g(y)))$ and $p_2 = p(z, h(z, w), f(w))$, their idempotent most general unifier ϑ computed by the Robinson's unification algorithm, is $\{z/a,\ x/h(a, g(y)),\ w/g(y)\}$ and $p_1\vartheta = p_2\vartheta = p(a, h(a, g(y)), f(g(y)))$. This algorithm generates the following sequence of substitutions and atoms during the iterated executions of the body of the WHILE-DO statement:

	substitution σ	atom p_1	atom p_2
initialization	{}	$p(a,x,f(g(y)))$	$p(z,h(z,w),f(w))$
1st execution	$\{z/a\}$	$p(a,x,f(g(y)))$	$p(a,h(a,w),f(w))$
2nd execution	$\{z/a,\ x/h(a,w)\}$	$p(a,h(a,w),f(g(y)))$	$p(a,h(a,w),f(w))$
3rd execution	$\{z/a,\ x/h(a,w),\ w/g(y)\}$ $(=\vartheta)$	$p(a,h(a,g(y)),f(g(y)))$	$p(a,h(a,g(y)),f(g(y)))$ □

When performing a resolution step we may restrict ourselves to a version of the resolution, called 1-*to*-1 *resolution*, where each disjunction to be chosen, consists of one literal only and we need to unify two atoms only. However, there is a price to pay, as we now explain. (In the literature 1-to-1 resolution is called *binary resolution*.)

Factoring Step. (Instantiation and deletion of duplicates literals) Given a clause C, a factoring step first generates a suitable instance of C with duplicated literals and then produces as output that instance without duplicated literals.

Here are two examples of factoring.

(i) Given the clause $p(x, b)\vee p(a, y)\vee\neg q(x)$, by factoring we consider its instance for $\{x/a,\ y/b\}$ and we get the new clause $p(a, b)\vee\neg q(a)$.

(ii) Given the clause $p(x)\vee p(f(y))\vee\neg q(x)$, by factoring we consider its instance for $\{x/f(y)\}$ and we get the new clause $p(f(y))\vee\neg q(f(y))$.

A factoring step preserves satisfiability as it is easy to show by recalling that:

(i) clauses are implicitly universally quantified at the front,

(ii) for all formulas φ, for all variables x, for all terms t which are free for x in $\varphi(x)$, we have that $\forall x\ \varphi(x)$ implies $\varphi(t)$, and $(\forall x\ \varphi(x)) \vee \varphi(t))$ is satisfiable iff $\varphi(t)$ is satisfiable, and

(iii) for all formulas φ, we have that $\varphi\vee\varphi$ is equivalent to φ.

Theorem 18. (Robinson 1965). A formula φ in clausal form is unsatisfiable iff starting from the set of clauses of φ we can get the empty clause $\Box$ by zero or more steps of 1-to-1 resolution and factoring.

Factoring is required, for instance, in the case of the following two clauses: $p(x) \vee p(y)$ and $\neg p(a) \vee \neg p(b)$. Indeed, by 1-to-1 resolution we always get clauses with two literals (and never the empty clause), while by factoring we get the clauses $p(x)$ and $\neg p(a) \vee \neg p(b)$, and then, from these clauses by two steps of 1-to-1 resolution we get the empty clause.

Paramodulation. Let us consider the equality predicate $=$ which, as usual, is assumed to be reflexive, symmetric, transitive, and substitutive, that is, given any two terms r and s, and any context $C[\ldots]$, if $r = s$ then $C[r] = C[s]$ (i. e., $=$ is a congruence). In order to deal with the equality predicate, we can use the following inference rule, called *paramodulation* [3, page 168].

Let C_1 be the clause $L[t] \vee D_1$ and C_2 be either the clause $r = s \vee D_2$ or the clause $s = r \vee D_2$, where $L[t]$ is a (positive or negative) literal where we singled out an occurrence of the term t, and D_1 and D_2 are disjunctions of literals. Let us assume that the clauses C_1 and C_2 have no variables in common and let ϑ be the idempotent mgu of t and r.

Let us consider the instances $C_1\vartheta$ and $C_2\vartheta$ of the clauses C_1 and C_2, respectively. Thus, in $(L[t])\vartheta$ we have an occurrence of the term $t\vartheta$ which is identical to $r\vartheta$. By a paramodulation step from C_1 and C_2 we get the clause:

$(L[s] \vee D_1 \vee D_2)\vartheta$

where $L[s]$ is the result of replacing the selected occurrence of t in $L[t]$ by s.

Example 4. From clause C_1: $q(x, a) \vee p(x)$ and clause $C_2 : a = b$ by paramodulation we can get: $q(x, b) \vee p(x)$. In this case we have singled out the term a in C_1 and r is taken to be a. Thus, ϑ is the empty substitution, because it is the most general unifier of a and a. From clauses C_1 and C_2 by paramodulation we can also get $q(b, a) \vee p(a)$, obtained for $\vartheta = \{x/a\}$. We can also get $q(a, a) \vee p(b)$. However, we cannot get $q(a, b) \vee p(a)$, because the occurrence of the term a which has been replaced by b, does not correspond to an occurrence of x in C_1.

Example 5. From clause C_1: $p(f(g(y), z), z) \vee w(z) = m$ and clause C_2: $a = f(x, g(x))$ by paramodulation we can get: $p(a, g(g(y))) \vee w(g(g(y))) = m$. In this case, with reference to the definition of paramodulation above, we have that the term t is $f(g(y), z)$ and the term r is $f(x, g(x))$. Their idempotent mgu ϑ is the substitution $\{x/g(y),\ z/g(g(y))\}$.

Given a formula φ, let us consider the conjunction $E(\varphi)$ of the following *equality clauses* relative to φ: (i) $x = x$, and (ii) for each function symbol f of arity n (≥ 0) occurring in φ: $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$. We have the following theorem.

Theorem 19. (i) A formula φ in clausal form is unsatisfiable iff starting from the clauses of φ together with $E(\varphi)$, by zero or more steps of resolution and paramodulation we can get the empty clause $\Box$.
(ii) A formula φ in clausal form is unsatisfiable iff starting from the clauses of φ together with $E(\varphi)$, by zero or more steps of 1-to-1 resolution, factoring, and paramodulation we can get the empty clause $\Box$.

The following remarks and exercise conclude this section.

Remark 1. In the propositional calculus, that is, in the fragment of predicate calculus with neither variables nor quantifiers, we have that factoring is not significant and we can drop factoring from Theorem 18.

Remark 2. (i) Unsatisfiability, satisfiability, and validity are all *decidable* in propositional calculus (in at most exponential time on the length of the given formula).

(ii) In first order predicate calculus unsatisfiability and validity are *semidecidable.* Satisfiability is undecidable and *not semidecidable* (Recall that by Post Theorem [10, page 256], if satisfiability were semidecidable then both unsatisfiability and satisfiability would be decidable.)

(iii) In second order predicate calculus (where we may quantify also over predicate symbols) unsatisfiability, satisfiability, and validity are all undecidable and *not semidecidable.*

A theory T is said to be a *decidable theory* (or a *semidecidable theory*) if for all formulas φ there is a recursive (or semirecursive) procedure to test whether of not φ is a theorem of T, that is, $T \vdash \varphi$.

Thus, the propositional calculus is decidable (in exponential time with respect to the number of propositions in the formula) and the first order predicate calculus is semidecidable.

Remark 3. For any given *closed* formula P and a formula A, the reader should not confuse the formula $P \to A$ and the relation $P \models A$. Indeed, $P \to A$ is a formula which may be true or false in a given interpretation, while $P \models A$ is a relation stating that every model of P is also a model of A. By Gödel's Completeness (see Theorem 7 on page 35) we have that: $P \vdash A$ iff $P \models A$. We also have that $P \vdash A$ iff $\vdash P \to A$ by the Deduction Theorem 2.

Remark 4. Given a set Γ of *closed* formulas and a formula φ. We have that:
if $\Gamma \vdash \varphi$ *and* there exists an interpretation I such that for each ψ in Γ, $I \models \psi$, *then* $I \models \varphi$.
Indeed, by Theorem 7, $\Gamma \vdash \varphi$ implies $\Gamma \models \varphi$, that is, every model of Γ is a model of φ. Since I is a model of Γ we have that I is also a model of φ.

Remark 5. The resolution approach to theorem proving is based on the fact that given a set (that is, a conjunction) Γ of *closed* formulas and a *closed* formula φ, we have that $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable iff $\Gamma \vdash \varphi$. Indeed,

$\Gamma \cup \{\neg\varphi\}$ is unsatisfiable

iff (by definition) it does not exist an interpretation I and a variable assignment σ such that for every ψ in $\Gamma \cup \{\neg\varphi\}$, we have that $I, \sigma \models \psi$

iff (by pushing negation inside) for all interpretation I and variable assignment σ there exists a formula ψ in $\Gamma \cup \{\neg\varphi\}$ such that $I, \sigma \not\models \psi$

iff (by the hypothesis that Γ and φ are closed) for all interpretation I there exists a formula ψ in $\Gamma \cup \{\neg\varphi\}$ such that $I \not\models \psi$

iff for all interpretation I, $I \not\models \Gamma \wedge \neg\varphi$

iff for all I, $I \models \neg(\Gamma \wedge \neg\varphi)$

iff for all I, $I \models \Gamma \rightarrow \varphi$

iff $\models \Gamma \rightarrow \varphi$

iff $\vdash \Gamma \rightarrow \varphi$ (by Gödel's Completeness Theorem 7 on page 35)

iff $\Gamma \vdash \varphi$ (by the Deduction Theorem 1 on page 16 because Γ is a set of closed formulas).

Remark 6. When studying computations in logic programming (see Section 8.3), we will introduce a restricted form of resolution, called SLD resolution [7]. During SLD resolution steps, substitutions are recorded and composed and, as we will see, this allows us to execute programs and compute their results. Indeed, in logic programming the process of computing is reduced to the process of recording substitutions while constructing proofs. This will be explained in Section 8.3.

8 Horn Clauses and Definite Logic Programs

In this section and in the following Section 9 (starting on page 69), we will consider a restricted class of clauses, called *Horn clauses*, for which a particular form of 1-to-1 resolution, called SLD resolution, is complete, that is, given any formula φ which is a conjunction of Horn clauses, φ is unsatisfiable iff starting from φ, we can get the empty clause $\square$ by zero or more steps of SLD resolution (see the definition of an SLD resolution step on page 65 and Theorem 24 on page 68).

Note that, although Horn clauses are a particular class of clauses, they are Turing-complete in the sense that every computable function can be expressed using a conjunction of Horn clauses (see Theorem 26 on page 69).

Let us consider a first order language L. A *literal* is an atom or a negated atom in the language L. A *clause* is a disjunction of literals. The disjunction of zero literals

is called the *empty clause*, denoted by $\Box$, and it is identified with the atom *false* (see page 10).

Let $atom_0, atom_1, \ldots, atom_n$ be atoms in the language L. A *definite clause* C is a clause of the form: $atom_0 \vee \neg atom_1 \vee \ldots \vee \neg atom_n$, for $n \geq 0$, which is written as follows:

$$atom_0 \leftarrow atom_1, \ldots, atom_n \quad \text{for } n \geq 0$$

where $atom_0$ is not the atom *false*. (We will see that if we allow $atom_0$ to be *false* then a definite program may have no models.) Comma stands for $\wedge$. All variables in a definite clause are implicitly universally quantified at the front. $\leftarrow$ denotes reverse logical implication, that is, $a \leftarrow b$ stands for $b \rightarrow a$. If $n = 0$ then $a \leftarrow$ stands for $true \rightarrow a$.

The atom $atom_0$ is said to be the *head* of the definite clause C and it is denoted by $hd(C)$. The sequence $atom_1, \ldots, atom_n$ of atoms is said to be the *body* of the definite clause C and it is denoted by $bd(C)$.

A *definite goal* is a formula of the form:

$$\leftarrow atom_1, \ldots, atom_n \quad \text{for } n \geq 0$$

Comma stands for $\wedge$. All variables in a definite goal are implicitly universally quantified at the front. $\leftarrow b$ stands for $b \rightarrow false$. If $n = 0$ then a definite goal is said to be the *empty goal*, denoted by $\Box$. The empty goal is the same as the empty clause.

A *variant* of a definite clause C (or a definite goal G) is obtained by applying to C (or G) a substitution of the form $\{x_1/y_1, \ldots, x_n/y_n\}$, such that: (i) the set of variables $\{x_1, \ldots, x_n\}$ is contained in the set V of variables occurring in C (or G), (ii) the variables y_i's are distinct, and (iii) $(V - \{x_1, \ldots, x_n\}) \cap \{y_1, \ldots, y_n\} = \emptyset$.

A *Horn clause* is either a definite clause or a definite goal. Thus, a Horn clause is a clause with at most one positive literal.

A *definite program* is a finite conjunction of $n\,(\geq 1)$ definite clauses. Given a definite program, we get an equivalent definite program if we delete a clause whose head is the atom *true*. The program $C_1 \wedge \ldots \wedge C_n$ will also be written as $\{C_1, \ldots, C_n\}$, using the set notation.

Remark 7. For the rest of this Section 8, and throughout Section 9, unless otherwise specified or understood from the context, by 'clauses', 'goals', and 'programs' we mean 'definite clauses', 'definite goals', and 'definite programs', respectively. $\Box$

Given a program P, below we will define the denotational semantics of P by introducing the so called least Herbrand model of P, denoted by $M(P)$. We will also define the operational semantics of P for a given goal $\leftarrow A$ via the so called *SLD tree* for P and $\leftarrow A$. We will show that these two semantics are equivalent in the sense specified by Theorems 24 and 25 below.

8.1 Least Herbrand Models of Definite Logic Programs

Let us consider a definite program P. The first order language L associated with P has exactly the constant symbols, the function symbols, and the predicate symbols occurring in P. We assume the following:

(α) if in P there is one variable then in P there is at least one constant symbol.

This assumption is due to technical reasons and it can always be fulfilled by adding to P the clause: $newp(b) \leftarrow newp(b)$, where b is a new constant symbol and $newp$ is new predicate symbol. We will see that the addition of this extra clause modifies neither the fixpoint semantics nor the operational semantics of the given program P w.r.t. goals where $newp$ does not occur.

The notions of Herbrand Universe and the Herbrand Base generated by L can be extended to the case when we are given, together with a definite program P, also a definite goal $\leftarrow A$. In that case: (i) the language L associated with P and $\leftarrow A$ has exactly the constant symbols, the function symbols, and the predicate symbols occurring in P or in $\leftarrow A$, and (ii) if in P or in $\leftarrow A$ there is one variable then in P or in $\leftarrow A$ there is at least one constant symbol.

When referring to 'ground instances of clauses' or 'ground atoms' or 'ground literals' we mean instances of clauses or atoms or literals, respectively, whose terms belong to the Herbrand Universe generated by the given program and goal.

As already mentioned in Section 7, we will identify an Herbrand interpretation with a subset of the Herbrand Base. Let the definite program P be the conjunction $C_1 \wedge \ldots \wedge C_m$ of clauses.
For every Herbrand interpretation I, for all ground atoms $A, A_1, \ldots, A_n$, we have that:

(i) $I \models A$ iff $A \in I$,

(ii) $I \models (A_1 \wedge \ldots \wedge A_n)$ iff $\{A_1, \ldots, A_n\} \subseteq I$,

(iii) $I \models ((A_1 \wedge \ldots \wedge A_n) \rightarrow A)$ iff $(\mathit{if}\ \{A_1, \ldots, A_n\} \subseteq I\ \mathit{then}\ A \in I)$, and

(iv) $I \models P$

iff (1) $I \models \forall(C_1) \wedge \ldots \wedge \forall(C_m)$

iff (2) for each ground instance $A \leftarrow A_1, \ldots, A_n$ of a clause of P,
$I \models A_1, \ldots, A_n \rightarrow A$

iff for each ground instance $A \leftarrow A_1, \ldots, A_n$ of a clause of P,
$\mathit{if}\ \{A_1, \ldots, A_n\} \subseteq I\ \mathit{then}\ A \in I$.

Formula (2) is derived from formula (1) because by the definition of the satisfaction relation, for a universally quantified formula we have to consider all variable assignments which assign to every variable a term of the Herbrand Universe (recall that the Herbrand Universe is a set of ground terms).

We will show that every definite program P has a model which is the least subset I of the Herbrand Base such that $I \models P$. This model of the program P is called the *least Herbrand model* of P and it is denoted by $M(P)$.

As usual in the literature (see, for instance, [8]), we follow the convention that when considering the Herbrand models and the least Herbrand models, the atoms *true* and *false* do not belong to the Herbrand Base.

Example 6. For $P = \{p \leftarrow q(y),\ q(a) \leftarrow\}$ we have that $M(P) = \{p, q(a)\}$.
For $Q = \{p(0) \leftarrow,\ p(s(x)) \leftarrow p(x),\ p(s(s(a))) \leftarrow\}$ we have that:

$M(Q) = \{p(0),\ p(s(0)),\ p(s(s(0))),\ \ldots, p(s(s(a))),\ p(s(s(s(a)))),\ \ldots\}$.

$M(Q) \cup \{p(a)\}$ is an Herbrand model of Q and it properly includes the least Herbrand model.

Theorem 20. (Model Intersection Property) Given a definite program P and a set $\{M_j \mid j \in J\}$ of Herbrand models of P, their intersection $\bigcap_{j\in J} M_j$ is an Herbrand model of P.

Proof. Obviously, $\bigcap_{j\in J} M_j$ is an Herbrand interpretation. Moreover, $\bigcap_{j\in J} M_j$ is a model of P because for each ground instance $A \leftarrow A_1, \ldots, A_n$ of a clause of P, if $\{A_1, \ldots, A_n\} \subseteq \bigcap_{j\in J} M_j$ then $A \in \bigcap_{j\in J} M_j$. This can be shown as follows. $\{A_1, \ldots, A_n\} \subseteq \bigcap_{j\in J} M_j$ implies that for every $j \in J$, $\{A_1, \ldots, A_n\} \subseteq M_j$, which in turn, implies that for every $j \in J$, $A \in M_j$, because M_j is a model of P. Thus, $A \in \bigcap_{j\in J} M_j$.

Note that for set of clauses which are not definite programs, the Model Intersection Property does not hold. Indeed, let us consider the set of clauses $Q = \{p(a) \vee p(b)\}$ which consists of one clause only, namely, $p(a) \vee p(b)$. This clause is not a definite clause. Thus, Q is not a definite program. The sets $\{p(a)\}$ and $\{p(b)\}$ are two distinct Herbrand models of Q, but their intersection which is the empty set, is not a model of Q.

By Theorem 20 the intersection of all Herbrand models of a given program P is an Herbrand model. It always exists. It is the least Herbrand model. It may be the empty model, that is, the empty subset of the Herbrand Base. Note that the empty model, as every interpretation, has a non-empty domain, which is the Herbrand Universe.

For instance, the least Herbrand model of the program $\{p(0) \leftarrow q(y)\}$ is the empty model.

Theorem 21. (Kowalski-van Emden 1976). Let us consider a definite program P. $M(P)$ is the set of ground atoms which are logical consequences of P, that is, $M(P) = \{A \mid A$ is a ground atom and $P \models A\}$. Equivalently, for any ground atom A, $P \models A$ iff $A \in M(P)$ (that is, $M(P) \models A$).

Proof. Let A be a ground atom of the Herbrand Base.

$P \models A$ (that is, every model of P is a model of A, that is, for every interpretation I if P is true in I then A is true in I)

iff $P \wedge \neg A$ is false in all interpretations (a)

iff $P \wedge \neg A$ is false in all Herbrand interpretation (because a formula in clausal form is unsatisfiable iff it has no Herbrand models (see Theorem 15 on page 49)) (b)

iff $\neg A$ is false in all Herbrand models of P

iff A is true in all Herbrand models of P (because A is ground (see Note 5 on page 31)) (c)

iff A is true in the least Herbrand model of P, that is, $A \in M(P)$. (d)

Remark 8. In the above proof, we have that (b) implies (a) because P and $\neg A$ are in clausal form. Moreover, (d) implies (c) because P is a definite program and the Model Intersection Property holds.

As a consequence of Theorem 21, for any definite program P and any *ground* conjunction $A_1 \wedge \ldots \wedge A_n$ of atoms, we have that:

$$P \models (A_1 \wedge \ldots \wedge A_n) \text{ iff } M(P) \models (A_1 \wedge \ldots \wedge A_n) \qquad (\dagger)$$

Now we show that if $A_1 \wedge \ldots \wedge A_n$ is *not* ground, then (i) formula (†) holds with 'implies', instead of 'iff', and (ii) formula (†) with 'if', instead of 'iff', does not hold.

Property (i) follows from the fact that $P \models (A_1 \wedge \ldots \wedge A_n)$ holds iff $(A_1 \wedge \ldots \wedge A_n)$ holds in every model of P, and thus, it holds also in the least Herbrand model of P.

Property (ii) follows from this counterexample.

Let us consider the program $P = \{p(0) \leftarrow\}$ and the goal $\leftarrow p(x)$. Now we show that:

(ii.1) $P \models p(x)$ does not hold, and (ii.2) $M(P) \models p(x)$ holds.

Proof of (ii.1). P does not implies $p(x)$, that is, $p(x)$ is not true in *every* model of P, because we have that: (i) P is equivalent to $p(0)$, and (ii) it is not the case that $\{p(0)\} \models p(x)$ because it is not the case that $\{p(0)\} \vdash p(x)$, that is, it is not the case that $\vdash p(0) \rightarrow p(x)$ (note that $p(0)$ is a closed formula).

Proof of (ii.2). In the least Herbrand model of P which is $\{p(0)\}$, the formula $p(x)$ holds, because $\forall x\, p(x)$ holds (Recall that by Proposition 10 on page 32 we have that: $M(P) \models p(x)$ holds iff $M(P) \models \forall x\, p(x)$). Indeed, the variable x may only be assigned to the value 0 which is the only element in the Herbrand Universe of P. □

Finally, note that formula (†) with 'implies' instead of 'iff', that is,

$P \models (A_1 \wedge \ldots \wedge A_n)$ implies $M(P) \models (A_1 \wedge \ldots \wedge A_n)$

does not hold, if when constructing the domain of an interpretation we use symbols which do not occur in the given program P or goal $A_1 \wedge \ldots \wedge A_n$.

Indeed, let us consider the program $P = \{p(x) \leftarrow,\ q(0) \leftarrow\}$. We have that: $P \models p(b)$ for some constant b which does not belong to the Herbrand Universe generated by P, and it is not the case that $M(P) \models p(b)$ because $M(P) = \{p(0),\ q(0)\}$.

8.2 Fixpoint Semantics of Definite Logic Programs

Let us first recall a few basic definitions.

A *lattice* L is a set, which we denote by the same letter L, together with a partial order, denoted by $\leq$, and two binary operations, denoted by *glb* and *lub*, called the *greatest lower bound* (w.r.t. $\leq$) and the *least upper bound* (w.r.t. $\leq$), respectively.

By definition, we have that for any x, y, and z in L,
(i.1) $glb(x, y) \leq x$ and $glb(x, y) \leq y$, that is, $glb(x, y)$ is a *lower bound* of x and y, and
(i.2) if $z \leq x$ and $z \leq y$ then $z \leq glb(x, y)$, that is, $glb(x, y)$ is the greatest among the lower bounds of x and y.

Analogously, by definition, we have that for any x, y, and z in L,
(ii.1) $x \leq lub(x, y)$ and $y \leq lub(x, y)$, that is, the $lub(x, y)$ is a *upper bound* of x and y, and
(ii.2) if $x \leq z$ and $y \leq z$ then $lub(x, y) \leq z$, that is, $lub(x, y)$ is the least among the upper bounds of x and y.

Conditions (i.1), (i.2), (ii.1), and (ii.2) above implies that given any two elements x and y in L, $glb(x, y)$ and $lub(x, y)$ are unique.

A lattice L is said to be *complete* iff the *glb* and the *lub* operations are defined for every (finite or infinite) subset H of L, that is, for every set $S \subseteq L$, we have that: $glb(S) \in L$, $lub(S) \in L$, and for all $x \in S$:
(i.1) $glb(S) \leq x$, and (i.2) for all $z \in L$, if $z \leq x$ then $z \leq glb(S)$, and
(ii.1) $x \leq lub(S)$, and (ii.2) for all $z \in L$, if $x \leq z$ then $lub(S) \leq z$.

Instead of $glb(S)$ and $lub(S)$, we will also write $glb\, S$ and $lub\, S$, respectively.

The greatest lower bound of a complete lattice L is denoted by $\bot$. Thus, $\bot$ is the least element of L, that is, for all $x \in L$, $\bot \leq x$.

Let L be a complete lattice ordered by $\leq$ and T be a function from L to L. We say that x is a *prefixpoint* of T iff $T(x) \leq x$. We say that x is a *postfixpoint* of T iff $x \leq T(x)$. We also say that x is a *fixpoint* of T iff $T(x) = x$.

Let us define:

$$\begin{aligned} T^0(x) &= x \\ T^{k+1}(x) &= T(T^k(x)) \qquad \text{for any } k \geq 0 \\ T^\omega(x) &= lub\{T^k(x) \mid k \geq 0\}. \end{aligned}$$

The function $T\colon L \to L$ is said to be *monotonic on* L (or *monotonic*, for short) iff for every x and y if $x \leq y$ then $T(x) \leq T(y)$.

The function $T\colon L \to L$ is said to be *continuous on* L (or *continuous*, for short) iff it is monotonic on L and for every infinite sequence $x_0 \leq x_1 \leq \ldots \leq x_n \leq \ldots$ of elements (not necessarily distinct) of L we have that: $T(lub\{x_i \mid i \geq 0\}) = lub\{T(x_i) \mid i \geq 0\}$.

Now we recall a few known lemmata.

Lemma 7. Let $T\colon L \to L$ be a monotonic function on a complete lattice L ordered by $\leq$. $glb\{x \mid T(x) \leq x\}$, that is, the *glb* of all prefixpoints of T, is the least prefixpoint of T.

Proof. We have to show that: (1) $glb\{x \mid T(x) \leq x\}$ is a prefixpoint of T, that is, $T(glb\{x \mid T(x) \leq x\}) \leq glb\{x \mid T(x) \leq x\}$, and (2) given any other prefixpoint z of T we have that $glb\{x \mid T(x) \leq x\} \leq z$.
Proof of (1). For every y such that $T(y) \leq y$, we have that:

$$\begin{array}{ll} glb\{x \mid T(x) \leq x\} \leq y & \text{(by definition of } glb\text{, because } y \in \{x \mid T(x) \leq x\}) \\ T(glb\{x \mid T(x) \leq x\}) \leq T(y) & \text{(by monotonicity of } T) \\ T(glb\{x \mid T(x) \leq x\}) \leq y & \text{(by transitivity and } T(y) \leq y). \qquad (\dagger) \end{array}$$

Now since (†) holds for every y such that $T(y) \leq y$, $T(glb\{x \mid T(x) \leq x\})$ is a lower bound of the set $\{x \mid T(x) \leq x\}$. Since $glb\{x \mid T(x) \leq x\}$ is the greatest among the lower bounds of the set $\{x \mid T(x) \leq x\}$, we get (1).
Proof of (2). Given a complete lattice L, for every subset S of L, we have that $glb(S) \leq x$ for every element x in S. The thesis follows from: (i) $glb(S) \leq x$ by taking S to be $\{x \mid T(x) \leq x\}$, and (ii) the fact that z is a prefixpoint of T, that is, z is an element of $\{x \mid T(x) \leq x\}$. □

Lemma 8. (Knaster-Tarski 1955). Let $T\colon L \to L$ be a monotonic function on a complete lattice L ordered by $\leq$. T has a least fixpoint, denoted by $lfp(T)$. We have that: $lfp(T) = glb\{x \mid T(x) = x\} = glb\{x \mid T(x) \leq x\}$, that is, $glb\{x \mid T(x) = x\}$ is the least fixpoint of T, and it is equal to $glb\{x \mid T(x) \leq x\}$ which, by Lemma 7, is the least prefixpoint of T.

Proof. We first show that:

(1) $T(glb\{x \mid T(x) \leq x\}) = glb\{x \mid T(x) \leq x\}$,

that is, $glb\{x \mid T(x) \leq x\}$ is a fixpoint of T. This can be shown by proving that:

(1.1) $T(glb\{x \mid T(x) \leq x\}) \leq glb\{x \mid T(x) \leq x\}$, and
(1.2) $T(glb\{x \mid T(x) \leq x\}) \geq glb\{x \mid T(x) \leq x\}$.

The proof of (1.1) is Point (1) of proof of Lemma 7. The proof of (1.2) is as follows. From (1.1) by monotonicity of T we get:

$T(T(glb\{x \mid T(x) \leq x\})) \leq T(glb\{x \mid T(x) \leq x\})$.

Thus, $T(glb\{x \mid T(x) \leq x\})$ is a prefixpoint of T and hence, it belongs to the set $\{x \mid T(x) \leq x\}$. Thus, $glb\{x \mid T(x) \leq x\} \leq T(glb\{x \mid T(x) \leq x\})$, as stated in (1.2).
We also have:

(2) $glb\{x \mid T(x) = x\} \leq glb\{x \mid T(x) \leq x\}$

because $glb\{x \mid T(x) = x\} \leq x$ for every x which is a fixpoint of T, and $glb\{x \mid T(x) \leq x\}$ is a fixpoint of T, as we have shown in (1).
We also have that:

(3) $glb\{x \mid T(x) \leq x\} \leq glb\{x \mid T(x) = x\}$

because $\{x \mid T(x) = x\} \subseteq \{x \mid T(x) \leq x\}$.
From (2) and (3) we get: $glb\{x \mid T(x) \leq x\} = glb\{x \mid T(x) = x\}$. Now, since $glb\{x \mid T(x) \leq x\}$ is a fixpoint of T, as we have shown in (1) above, we have that:

(4) $glb\{x \mid T(x) = x\}$ is a fixpoint of T.

To complete the proof of this lemma we have to show that:

(5) $glb\{x \mid T(x) = x\}$ is the least fixpoint of T.

From (4) we have that $glb\{x \mid T(x) = x\}$ is a fixpoint of T. We have to show that $glb\{x \mid T(x) = x\}$ is smaller than every other fixpoint of T.

Now, since by (2) and (3), $glb\{x \mid T(x) = x\}$ is equal to $glb\{x \mid T(x) \leq x\}$, it is enough to show that $glb\{x \mid T(x) \leq x\}$ is smaller than every other fixpoint of T. By Lemma 7, $glb\{x \mid T(x) \leq x\}$ is the least prefixpoint of T. Thus, since $glb\{x \mid T(x) \leq x\}$ is smaller than every prefixpoint of T, it is also smaller than every fixpoint of T. □

Note 7. The proof of Lemma 8 shows the usefulness of introducing the notion of prefixpoint, besides the notion of fixpoint.

Lemma 9. Let $T: L \to L$ be a monotonic function on a complete lattice L ordered by $\leq$. T has a greatest fixpoint, denoted by $gfp(T)$. We have that:

$gfp(T) = lub\{x \mid T(x) = x\} = lub\{x \mid x \leq T(x)\}$,

that is, $lub\{x \mid T(x) = x\}$ is the greatest fixpoint of T, and this fixpoint is equal to $lub\{x \mid x \leq T(x)\}$ which is the greatest postfixpoint of T.

Proof. It follows from Lemma 8 because the complete lattice L ordered by the partial order $\leq$ is also a complete lattice ordered by the converse relation $\geq$, defined as follows: for all x and y in L, $x \geq y$ iff $y \leq x$. Note that also $\geq$ is a partial order. □

Lemma 10. (Kleene 1952). Let $T: L \to L$ be a continuous function on a complete lattice L ordered by $\leq$, with least element $\bot$. $T^\omega(\bot)$ is the least fixpoint of T and the least prefixpoint of T.

Proof. Note that L has the least element $\bot$ because L is a complete lattice and $glb(L)$ exists in L. We called this least element $\bot$.
(1) We first show that $T^\omega(\bot)$ is a fixpoint of T.
Indeed, $T(T^\omega(\bot)) = T(lub\{T^k(\bot) \mid k \geq 0\}) = \{$by continuity of $T\}$ =
$= lub\{T(T^k(\bot)) \mid k \geq 0\} = lub\{T^k(\bot) \mid k \geq 1\} = \{$by $\bot \leq T(\bot)\}$ =
$= lub\{T^k(\bot) \mid k \geq 0\} = T^\omega(\bot)$.
(2) We then show that $T^\omega(\bot)$ is the least fixpoint of T.
Indeed, take any other fixpoint z of T. We have that: $T(z) = z$. In order to show that $T^\omega(\bot) \leq z$ we need to show that $lub\{T^k(\bot) \mid k \geq 0\} \leq z$, or equivalently, we have to show that for any $k \geq 0$, $T^k(\bot) \leq z$.

This can be shown by induction on k.
(*Basis*) Obviously, $\bot \leq z$.
(*Step*) Assume that $T^k(\bot) \leq z$. Then $T^{k+1}(\bot) \leq T(z)$ (by monotonicity of T) and since $T(z) = z$ we get: $T^{k+1}(\bot) \leq z$, as desired.
(3) We finally have that $T^\omega(\bot)$ is the least prefixpoint of T.
Indeed, $T^\omega(\bot)$ which is the least fixpoint of T by Point (2), is also the least prefixpoint of T by Lemma 8. □

Since an Herbrand interpretation is a subset of the Herbrand Base, the set of all Herbrand interpretations is a complete lattice, that is, the lattice of all the subsets of the Herbrand Base. This complete lattice is ordered by set inclusion, denoted by $\subseteq$. The least element of this complete lattice is the empty set $\emptyset$, and the greatest element is the whole Herbrand Base. The *glb* and *lub* operations are, respectively, set intersection, denoted by $\cap$, and set union, denoted by $\cup$.

Given a definite program P, let us consider the following function T_P which takes an Herbrand interpretation and returns a new Herbrand interpretation:

$$T_P(I) = \{A \mid A \leftarrow A_1, \ldots, A_n \text{ is a ground instance of a clause of } P \text{ and } \{A_1, \ldots, A_n\} \subseteq I\}.$$

Lemma 11. An Herbrand interpretation I is a model of a definite program P, written $I \models P$, iff $T_P(I) \subseteq I$.

Proof. $I \models P$
iff for each ground instance $A \leftarrow A_1, \ldots, A_n$ of a clause of P we have that $I \models A_1, \ldots, A_n \rightarrow A$
iff for each ground instance $A \leftarrow A_1, \ldots, A_n$ of a clause of P we have that *if* $\{A_1, \ldots, A_n\} \subseteq I$ *then* $A \in I$
iff $A \in T_P(I)$ implies $A \in I$
iff $T_P(I) \subseteq I$. □

Compare this Lemma 11 with Point (iv) of Proposition 27 on page 90, where it is stated that for every Herbrand interpretation I, for every definite program P, $I \models comp(P)$ iff $T_P(I) = I$.

Theorem 22. Given a definite program P, T_P is a continuous function on the complete lattice of all Herbrand interpretations.

Proof. From the definition of T_P it follows that it is a monotonic function on the complete lattice of all Herbrand interpretations, that is, if $I_1 \subseteq I_2$ then $T_P(I_1) \subseteq T_P(I_2)$. To show that T_P is a continuous function on that lattice, let us consider an infinite sequence $I_0 \subseteq I_1 \subseteq \ldots$ of Herbrand interpretations. We have to show that: $T_P(\bigcup\{I_k \mid k \geq 0\}) = \bigcup\{T_P(I_k) \mid k \geq 0\}$. This follows from:

(1) $T_P(\bigcup\{I_k \mid k \geq 0\}) \supseteq \bigcup\{T_P(I_k) \mid k \geq 0\}$ and

(2) $T_P(\bigcup\{I_k \mid k \geq 0\}) \subseteq \bigcup\{T_P(I_k) \mid k \geq 0\}$.

Inclusion (1) easily follows from the fact that: for all $k \geq 0$, $\bigcup\{I_k \mid k \geq 0\} \supseteq I_k$ which implies that for all $k \geq 0$, $T_P(\bigcup\{I_k \mid k \geq 0\}) \supseteq T_P(I_k)$ by monotonicity of T_P. Thus, $T_P(\bigcup\{I_k \mid k \geq 0\})$ is an upper bound of the set $\{T_P(I_k) \mid k \geq 0\}$, and the least upper bound $\bigcup\{T_P(I_k) \mid k \geq 0\}$ of that set is included in the upper bound $T_P(\bigcup\{I_k \mid k \geq 0\})$.

Inclusion (2) is proved as follows. Let us consider an infinite sequence $I_0 \subseteq I_1 \subseteq \ldots$ of Herbrand interpretations and let us suppose that $A \in T_P(\bigcup\{I_k \mid k \geq 0\})$. Then, there exist some atoms $A_1, \ldots, A_n$ such that the clause $A \leftarrow A_1, \ldots, A_n$ is a ground instance of a clause of P and $\{A_1, \ldots, A_n\} \subseteq \bigcup\{I_k \mid k \geq 0\}$. Since for $i = 1, \ldots, n$, the atom A_i belongs to $\bigcup\{I_k \mid k \geq 0\}$ iff it belongs to I_{h_i} for some $h_i \geq 0$, we have that there exists $h = max\{h_1, \ldots, h_n\}$ such that $\{A_1, \ldots, A_n\} \in I_h$. Thus, $A \in T_P(I_h)$, and hence, $A \in \bigcup\{T_P(I_k) \mid k \geq 0\}$. □

Theorem 23. (van Emden-Kowalski 1976). $M(P)$ = least fixpoint of $T_P = T_P^\omega(\emptyset)$.

Proof. $M(P)$ = {by Theorem 20} =
= $\bigcap\{I \mid I$ is an Herbrand model of $P\}$ = {by set theory} =
= $glb\{I \mid I$ is an Herbrand model of $P\}$ = {by Lemma 11} =
= $glb\{I \mid T_P(I) \subseteq I\}$ = {by Lemma 7} =
= the least prefixpoint of T_P = {by Lemma 8} =
= the least fixpoint of T_P = {by continuity of T_P and Lemma 10} =
= $T_P^\omega(\emptyset)$. □

8.3 Operational Semantics of Definite Logic Programs

A *computation rule*, also called a *selection rule*, is a total function that given a non-empty goal, selects an atom in that goal. A *search rule* is a partial function that given a program and an atom, selects a clause in that program.

Given a program P, a goal G, and a computation rule R, an *SLD derivation* for P, G, and R consists in: (i) choosing a search rule, and (ii) constructing a (finite or infinite) sequence $\langle G_0, G_1, \ldots\rangle$ of goals, with $G_0 = G$, together with a

corresponding sequence $\langle C_0, C_1, \ldots \rangle$ of variants of clauses of P and a corresponding sequence $\langle \vartheta_0, \vartheta_1, \ldots \rangle$ of most general unifiers.

The construction of these three sequences is done incrementally, starting from the three sequences: $\langle G_0 \rangle$, $\langle \rangle$, and $\langle \rangle$, respectively. Then, for any $i \geq 0$, goal G_{i+1}, clause C_i, and mgu ϑ_i are constructed from goal G_i by performing an SLD resolution step, as we now indicate.

An SLD resolution step

Let G_i be the goal $\leftarrow A_1, \ldots, A_{m-1}, A_m, A_{m+1}, \ldots, A_n$, with $n > 0$.

(1) The computation rule R selects in G_i an atom, say A_m. (Recall that R is a total function.)

(2) Given P and A_m, the search rule determines a clause D_i, if any, in P such that the head of a variant of D_i unifies with A_m. We construct a variant C_i of clause D_i with no variable in common with $\langle G_0, G_1, \ldots, G_i \rangle$. Clause C_i is said to be constructed by *renaming apart* clause D_i. Then we compute an idempotent mgu, call it ϑ_i, such that $hd(C_i)\vartheta_i = A_m\vartheta_i$.

(3) G_{i+1} is the goal $\leftarrow (A_1, \ldots, A_{m-1}, bd(C_i), A_{m+1}, \ldots, A_n)\vartheta_i$. We say that goal G_{i+1} is obtained by SLD resolution from goal G_i and clause C_i.

If goal G_i is empty or no variant of clause D_i exists whose head unifies with A_m, then G_{i+1}, C_i, and ϑ_i are not constructed and the SLD derivation ends with the current sequences of goals, clauses, and mgu's.

Note that in order to perform an SLD resolution step, we have to consider *a variant* of a clause of the program P. This requirement is essential because it may be the case that a variant of a clause allows unification while a clause does not. Indeed, let us consider the goal $\leftarrow p(a, X)$ and the clause $p(X, b) \leftarrow$. Obviously, $p(a, X)$ does not unify with $p(X, b)$, but it unifies with $p(Y, b)$ which is the head of a variant of $p(X, b) \leftarrow$.

When performing an SLD resolution step from G_i, we require that clause C_i does not have any variable in common with the sequence $\langle G_0, G_1, \ldots, G_i \rangle$ of goals, for the generation of the computed answer substitutions (see below).

Note that an SLD resolution step is a 1-to-1 resolution step, and thus, the unification algorithm involves two atoms only.

Now we explain the acronym SLD.

(i) S stands for *Selection* of the atom chosen for unification (this selection is done at Point (1) of the SLD resolution step by the computation rule),

(ii) L stands for *Linear* resolution, because as in Linear resolution [3, page 131], at each step the new resolvent comes from the resolvent of the previous step and one clause of the program at hand (actually, in Linear resolution at each step the new resolvent comes from the resolvent of the previous step and either one clause of the program or a resolvent obtained in one of the previous Linear resolution steps), and

(iii) D stands for *Definite* logic programs.

An SLD derivation which ends with the empty goal $\square$, is called an *SLD refutation*.

The *computed answer substitution* (c.a.s., for short) of an SLD refutation for a program P, a goal G, and a computation rule R, is obtained by: (i) taking the composition of the most general unifiers of the sequence relative to that SLD refutation, and (ii) restricting the resulting substitution to the variables of G.

Independence of the computation rule. If there is an SLD refutation for a program P, a goal G, and a computation rule R with c.a.s. ϑ, then for any other computation rule R' there exists an SLD refutation for P, G, and R' with c.a.s. ϑ' such that $G\vartheta'$ is a variant of $G\vartheta$.

This independence may also be called *horizontal independence*, because it is related to the atoms in a goal which we usually write in a horizontal list.

Dependence on the search rule. Let P be the program $\{p(x) \leftarrow p(x),\ p(a) \leftarrow\}$. For the goal $\leftarrow p(x)$ if we always choose the clause $p(x) \leftarrow p(x)$ for performing an SLD resolution step then we always derive a variant of the goal $\leftarrow p(x)$ and we never get the empty goal. On the contrary, if we choose the clause $p(a) \leftarrow$ for performing an SLD resolution step, we derive the empty goal in one step.

This dependence may also be called *vertical dependence*, because it is related to the clauses in a program which in the programming language Prolog we often write in a vertical list, one below the other.

Let us introduce the notion of *SLD tree* for a program P, a goal G, and a computation rule R.

Construction of an SLD tree

Given a program P, a goal G, and a computation rule R, the root of the corresponding SLD tree, say T, is a node with the associated goal G. Every node of T has an associated goal and zero or more children. Every node N of T has one child-node for *each* clause D_i of P such that the head of a renamed apart variant C_i of D_i unifies with the atom selected by R in the goal, say H, associated with N. Renaming apart should ensure that no variable of C_i is in common with the goal associated with any ancestor of N. The goal H_i associated with node N_i which is a child of the node N, is obtained by performing an SLD resolution step from goal H using clause C_i.

Note that, by definition, any SLD tree is *maximal* in the sense that, if during the construction of an SLD tree, a node may have one (or one more) child-node, then we should include that child-node in the SLD tree. The various branches below a node of the SLD tree are generated by varying the search rule, and every root-to-leaf path in the SLD tree corresponds to an SLD derivation.

A finite root-to-leaf path of an SLD tree is said to be *successful*, or to have a *success leaf*, or simply to have a *success*, iff it ends with the empty goal $\Box$, that is, it is an SLD refutation. A finite root-to-leaf path of an SLD tree for a program P, a goal G, and a computation rule R, is said to be *failed*, or to have a *failure leaf*, or simply to have a *failure,* denoted by ■, iff the atom selected by R in the goal at the leaf of the path does not unify with the head of any variant of a clause in P. An SLD tree is said to be *finitely failed* iff it is finite and all its root-to-leaf paths are failed.

Note that a failed root-to-leaf path is a maximal root-to-leaf path which is not an SLD refutation. Note also that the set of successful root-to-leaf paths in an SLD tree does not depend on the computation rule, but the set of failed root-to-leaf paths *does* depend on the computation rule. For instance, given the program:

$a \leftarrow q, r$
$q \leftarrow q$

the SLD tree starting from the goal $\leftarrow a$ does *not* have any failed root-to-leaf path if we choose the leftmost selection rule, that is, the computation rule which always selects the leftmost atom. However, that SLD tree has a failed root-to-leaf path if we choose the rightmost selection rule, that is, the computation rule which always selects the rightmost atom.

In Figure 4 (adapted from [8, page 57]) we have shown an SLD tree for the program whose clauses are:

$C1.$ $p(x, z) \leftarrow q(x, y), p(y, z)$
$C2.$ $p(x, x) \leftarrow$
$C3.$ $q(a, b) \leftarrow$

and the goal $\leftarrow p(x, b)$ and the leftmost selection rule (which is the computation rule used by the programming language Prolog). The underlined atoms are the ones selected by the computation rule for SLD resolution. Every arc of the SLD tree corresponds to an SLD resolution step. We have labelled every arc by:
(i) the variant of the clause which has been used for the SLD resolution (for $1 \leq k \leq 2$, clause Ck_i is the variant of clause Ck, where every variable of Ck has been subscripted by i for $1 \leq i \leq 2$, for avoiding clashes of variable names during SLD resolution), and
(ii) the corresponding most general unifier.

We have also marked the leaves as 'success' or 'failure' according to the fact that the corresponding root-to-leaf path is or is not an SLD refutation. In the two success leaves we have written the corresponding c.a.s.

The explanation of the c.a.s. ϑ_1 is as follows. An idempotent most general unifier of clause $C2_1$ which is $p(x_1, x_1) \leftarrow$, and goal $\leftarrow p(x, b)$, is $\{x/b, x_1/b\}$. By restricting this most general unifier $\{x/b, x_1/b\}$ to $\{x\}$ we get $\{x/b\}$ which is ϑ_1. Note that

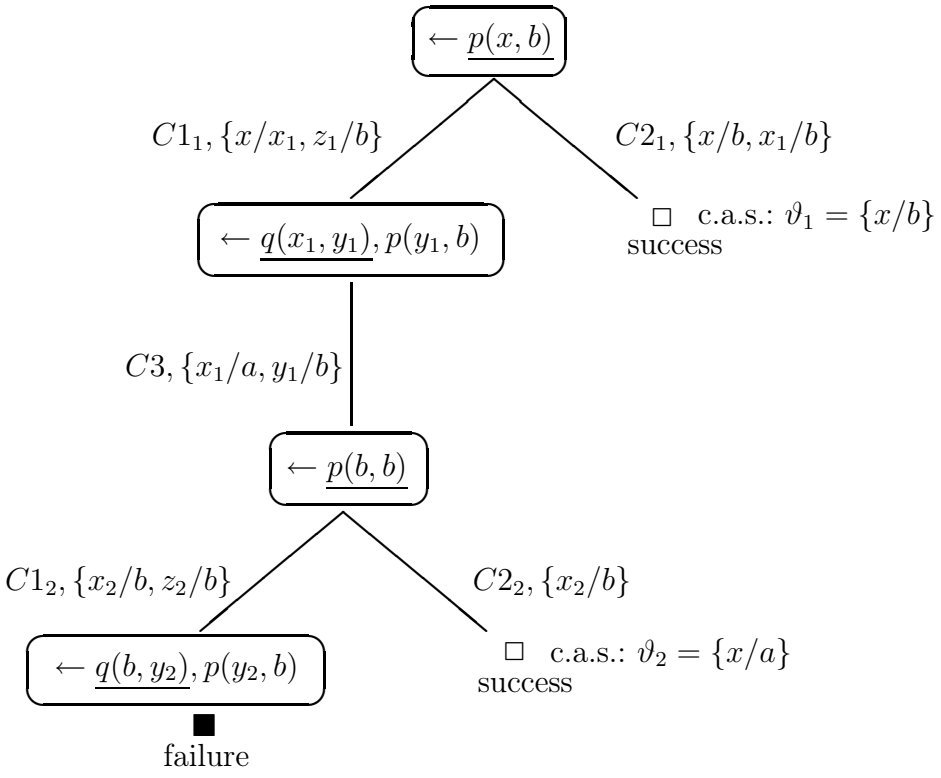

Fig. 4. A finite SLD tree for program $\{C1,\ C2,\ C3\}$, goal $\leftarrow p(x, b)$, and the leftmost selection rule.

$\{x/x_1,\ x_1/b\}$ is not a unifier of $p(x_1, x_1)$ and $p(x, b)$, because $p(x_1, x_1)\{x/x_1,\ x_1/b\}$ is $p(b, b)$ and $p(x, b)\{x/x_1,\ x_1/b\}$ is $p(x_1, b)$.

The explanation of the c.a.s. ϑ_2 is as follows. The composition of the idempotent most general unifiers along the path from the root to the corresponding success leaf is: $\{x/a,\ z_1/b,\ x_1/a,\ y_1/b,\ x_2/b\}$. The restriction of this substitution to the set $\{x\}$ which is the set of variables of the root goal $\leftarrow p(x, b)$, is $\{x/a\}$, and this is indeed ϑ_2.

Theorem 24. [Soundness and Completeness of SLD Resolution] (Clark 1979). Let us consider a program P, a (non necessarily atomic) goal $\leftarrow A$, and a computation rule R.

Soundness of SLD resolution: If a substitution ϑ is the c.a.s. of an SLD refutation for P, $\leftarrow A$, and R then $P \models \forall(A\vartheta)$.

Completeness of the SLD resolution: If a substitution ϑ for the variables of A is such that $P \models \forall(A\vartheta)$ then there exists a c.a.s. μ of an SLD refutation for P,

$\leftarrow A$, and R such that μ (whose domain is the set of variables of A, as it is the case for every c.a.s. for the program P, the goal $\leftarrow A$, and any computation rule) is more general than ϑ.

Proof. See [8, Theorem 7.1 (page 43) and Theorem 9.5 (page 52)].

In particular, if a substitution ϑ is the c.a.s. of an SLD refutation for the program P, the goal $\leftarrow A$, and the computation rule R, and we have that $A\vartheta$ is ground, then $P \models A\vartheta$, or equivalently, by Theorem 21, $M(P) \models A\vartheta$. If $A\vartheta$ is not ground, then $P \models A\vartheta$ implies $M(P) \models A\vartheta$, but it is not the case that $M(P) \models A\vartheta$ implies $P \models A\vartheta$, as shown at the end of Section 8.1.

The *success set* $SS(P)$ of a program P is $\{p(t_1, \ldots, t_r) \mid p(t_1, \ldots, t_r) \in HB$ and there exists an SLD tree with the goal $\leftarrow p(t_1, \ldots, t_r)$ at its root and at least one success leaf$\}$.

Theorem 25. (Apt-van Emden 1982). Given a program P, $SS(P) = M(P)$.

Proof. Let A be a ground atom. We have that: $A \in M(P)$
iff $P \models A$ (by Theorem 21)
iff there exists an SLD refutation for program P, goal $\leftarrow A$, and
a computation rule R (by Theorem 24)
iff $A \in SS(P)$ (by definition of $SS(P)$). □

9 Computing with Definite Logic Programs

In this section we show through some examples that definite logic programs can be used for:
(9.1) computing functions,
(9.2) computing function inverses,
(9.3) computing relations, that is, providing all computed answer substitutions for a given program and a given goal, and
(9.4) constructing knowledge-based systems.

We begin by presenting a theorem which states that definite logic programs are *Turing-complete*. In this theorem N denotes the set of natural numbers and for all natural numbers n, $\underline{n}$ is a term which denotes n. For instance, by using the constant symbol 0 and the unary successor function symbol s, we have that $s(s(0))$ is a term which denotes the natural number 2.

Theorem 26. For every partial recursive function $f\colon N \to N$, there exists a definite program P and a binary predicate symbol r such that for all natural numbers n and m we have that:

$f(n) = m$ iff $P \models r(\underline{n}, \underline{m})$.

Turing completeness of definite logic programs is proved in [8].

9.1 Computing Functions

Suppose you are given the following program *Sum*:

1. $sum(0, X, X) \leftarrow$
2. $sum(s(X), Y, s(Z)) \leftarrow sum(X, Y, Z)$

where $sum(X, Y, Z)$ holds iff Z denotes the sum of X and Y. We can use this program to compute the sum of two natural numbers as follows. Let the natural number n be denoted by the term $s(s(\ldots s(0)\ldots))$, where we have n occurrences of the symbol s.
As usual in logic programming, we use upper case letters to denote variables and lower case letters to denote constant symbols, function symbols, and predicate symbols.

Suppose we want to compute the value of 2+3, which is 5. We start from the corresponding goal:

$p.\ \leftarrow sum(s(s(0)), s(s(s(0))), Z)$

and we look for a variant of a clause of *Sum* whose head unifies with this goal. One such variant is a variant of clause 2. Thus, we rename apart clause 2, whereby getting:

$2.1\quad sum(s(X_1), Y_1, s(Z_1)) \leftarrow sum(X_1, Y_1, Z_1)$

The unification gives us the substitution $\vartheta_1 = \{X_1/s(0),\ Y_1/s(s(s(0))),\ Z/s(Z_1)\}$. We then replace the given goal p by the new goal $p1$, obtained by SLD resolution from goal p and clause 2.1:

$p1.\ \leftarrow sum(X_1, Y_1, Z_1)\vartheta_1$

that is, $\leftarrow sum(s(0), s(s(s(0))), Z_1)$. The process continues by looking again for a variant of a clause whose head unifies the new goal $p1$: $\leftarrow sum(s(0), s(s(s(0))), Z_1)$. One such variant is again a variant of clause 2. We rename apart clause 2, whereby getting:

$2.2\quad sum(s(X_2), Y_2, s(Z_2)) \leftarrow sum(X_2, Y_2, Z_2)$

The unification gives us the substitution $\vartheta_2 = \{X_2/0,\ Y_2/s(s(s(0))),\ Z_1/s(Z_2)\}$. We then replace the goal $p1$ by the new goal $p2$, obtained by SLD resolution from goal $p1$ and clause 2.2:

$p2.\ \leftarrow sum(X_2, Y_2, Z_2)\vartheta_2$

that is, $\leftarrow sum(0, s(s(s(0))), Z_2)$. The process continues by looking again for a variant of a clause whose head unifies with the new goal $p2$: $\leftarrow sum(0, s(s(s(0))), Z_2)$ and this time one such variant is a variant of clause 1. We rename apart clause 1 and we get:

$1.1\quad sum(0, W, W) \leftarrow$

The unification gives us the substitution $\vartheta_3 = \{W/s(s(s(0))), Z_2/s(s(s(0)))\}$. By SLD resolution from goal $p2$ and clause 1.1, we get the empty goal and the computation halts.

The result of the computation is given by the number which is bound by the substitution $(\vartheta_1\vartheta_2\vartheta_3)$ to the variable Z occurring in the given goal p. We have that: $Z(\vartheta_1\vartheta_2\vartheta_3) = ((Z\vartheta_1)\vartheta_2)\vartheta_3 = (s(Z_1)\vartheta_2)\vartheta_3 = (s(s(Z_2)))\vartheta_3 = s(s(s(s(0))))$, which represents 5, as expected.

9.2 Computing Function Inverses

We can use logic programs for computing function inverses. In particular, we may use the above program *Sum* (see Section 9.1) for computing both *additions* and *subtractions*. Indeed, we can use it for computing subtractions as follows. Let us assume, for instance, that we want to compute $4-1$. We start from the goal:

$r.\ \leftarrow sum(s(0), Y, s(s(s(s(0)))))$

We proceed as above. We look for a variant of a clause of *Sum* whose head unifies with this goal. One such variant is a variant of clause 2. Thus, we rename apart clause 2, whereby getting:

$2.3\quad sum(s(X_3), Y_3, s(Z_3)) \leftarrow sum(X_3, Y_3, Z_3)$

The unification gives us the substitution $\sigma_1 = \{X_3/0, Y_3/Y, Z_3/s(s(s(0)))\}$. We then replace the given goal r by the new goal $r1$, obtained by SLD resolution from goal r and clause 2.3:

$r1.\ \leftarrow sum(X_3, Y_3, Z_3)\sigma_1$

that is, $\leftarrow sum(0, Y, s(s(s(0))))$. The process continues by looking again for a variant of a clause whose head unifies the new goal $r1$: $\leftarrow sum(0, Y, s(s(s(0))))$. One such variant is a variant of clause 1. We rename apart clause 1 and we get:

$1.2\quad sum(0, X_4, X_4) \leftarrow$

The unification gives us the substitution $\sigma_2 = \{X_4/s(s(s(0))), Y/s(s(s(0)))\}$. By SLD resolution from goal $r1$ and clause 1.2, we get the empty goal and the computation halts.

The result of the computation is given by the number which is bound by the substitution $(\sigma_1\sigma_2)$ to the variable Y of the given goal r. We have that: $Y(\sigma_1\sigma_2) = (Y\sigma_1)\sigma_2 = Y\sigma_2 = s(s(s(0)))$, which represents 3, as expected.

9.3 Computing Relations and Nondeterministic Computing

We can use logic programs for computing relations, and this can be done by computing *all* computed answer substitutions for a given program and a given goal. For instance, let us consider the problem of computing all pairs of numbers whose sum is 3.

This problem is solved by computing all computed answer substitutions for the program *Sum* and the goal $g0$: $\leftarrow sum(X, Y, s(s(s(0))))$. We expect to get the following 4 computed answer substitutions: (i) $\{X/0,\ Y/s(s(s(0)))\}$, (ii) $\{X/s(0),\ Y/s(s(0))\}$, (iii) $X/s(s(0)),\ Y/s(0)\}$, and (iv) $\{X/s(s(s(0))),\ Y/0\}$, representing the 4 pairs: $\langle 0, 3\rangle$, $\langle 1, 2\rangle$, $\langle 2, 1\rangle$, and $\langle 3, 0\rangle$, respectively. Indeed, for each pair $\langle a, b\rangle$ we have that $a+b = 3$.

Now let us see how these 4 pairs are computed by executing the definite logic program starting from the initial goal:

$g0.\ \leftarrow sum(X, Y, s(s(s(0))))$

We proceed as above. We look for a variant of a clause whose head unifies with this goal. One such variant is a variant of clause 1. Thus, we rename apart clause 1, whereby getting:

1.3 $\quad sum(0, X_5, X_5) \leftarrow$

The unification gives us the substitution $\rho_1 = \{X/0,\ Y/s(s(s(0))),\ X_5/s(s(s(0)))\}$. By SLD resolution from goal $g0$ and clause 1.3, we get the empty goal and the computation halts with the c.a.s. $\{X/X\rho_1,\ Y/Y\rho_1\}$ which is $\{X/0,\ Y/s(s(s(0)))\}$, corresponding to the pair $\langle 0, 3\rangle$.

There are, however, other computed answer substitutions because also the head of a variant of clause 2 unifies with the given goal $g0$. In this sense the computation is *nondeterministic*, and together with one path of the computation which halts with the c.a.s. corresponding to $\langle 0, 3\rangle$, *there is another path of the computation* which is generated as follows. We rename apart clause 2, whereby getting:

2.4 $\quad sum(s(X_6), Y_6, s(Z_6)) \leftarrow sum(X_6, Y_6, Z_6)$

The unification of the head of clause 2.4 with goal $g0$ gives us the substitution $\rho_2 = \{X/s(X_6),\ Y/Y_6,\ Z_6/s(s(0))\}$. We then replace the given goal $g0$ by the new goal $g1$, obtained by SLD resolution from goal $g0$ and clause 2.4:

$g1.\ \leftarrow sum(X_6, Y_6, Z_6)\rho_2$

that is, $\leftarrow sum(X_6, Y_6, s(s(0)))$. We then look for a variant of a clause whose head unifies with goal $g1$. One such variant is a variant of clause 1. Thus, we rename apart clause 1, whereby getting:

1.4 $\quad sum(0, X_7, X_7) \leftarrow$

The unification gives us the substitution $\rho_3 = \{X_6/0,\ Y_6/s(s(0)),\ X_7/s(s(0))\}$. By SLD resolution from goal $g1$ and clause 1.4, we get the empty goal and the computation halts with the c.a.s. $\{X/X(\rho_2\rho_3),\ Y/Y(\rho_2\rho_3)\} = \{X/(X\rho_2)\rho_3,\ Y/(Y\rho_2)\rho_3\} = \{X/(s(X_6))\rho_3,\ Y/(Y_6)\rho_3\} = \{X/s(0),\ Y/s(s(0))\}$, corresponding to the pair $\langle 1, 2\rangle$.

There are, however, other computed answer substitutions because also the head of a variant of clause 2 unifies with goal $g1$. We again rename apart clause 2, whereby getting:

2.5 $\quad sum(s(X_8), Y_8, s(Z_8)) \leftarrow sum(X_8, Y_8, Z_8)$

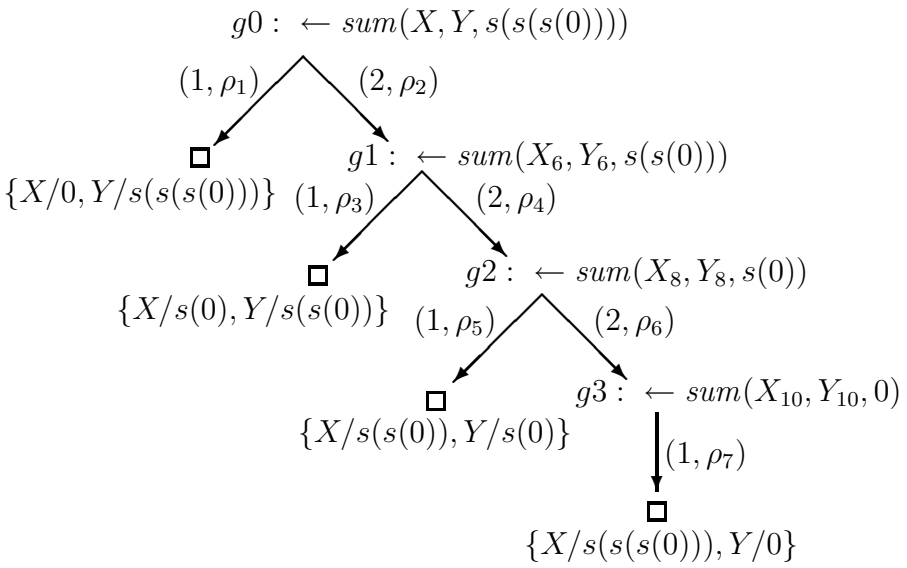

Fig. 5. The SLD tree for the program *Sum*, the goal $g0 : \leftarrow sum(X, Y, s(s(s(0))))$, and the leftmost computation rule. An arc labelled by (n, ρ_m) from node gi to node gj denotes the unification of goal gi with the head of a variant of clause n of program *Sum* via the substitution ρ_m and the SLD resolution step which uses that unification produces the goal gj.

The unification gives us the substitution $\rho_4 = \{X_6/s(X_8),\ Y_6/Y_8,\ Z_8/s(0)\}$. We then replace the given goal $g1$ by the new goal $g2$, obtained by SLD resolution from goal $g1$ and clause 2.5:

$g2.\ \leftarrow sum(X_8, Y_8, Z_8)\rho_4$

that is, $\leftarrow sum(X_8, Y_8, s(0))$. We proceed as above, by looking for a variant of a clause whose head unifies with goal $g2$. One such variant is a variant of clause 1. We rename apart clause 1, whereby getting:

$1.5\quad sum(0, X_9, X_9) \leftarrow$

The unification gives us the substitution $\rho_5 = \{X_8/0,\ Y_8/s(0),\ X_9/s(0)\}$. By SLD resolution from goal $g2$ and clause 1.5, we get the empty goal and the computation halts with the c.a.s. $\{X/X(\rho_2\rho_4\rho_5),\ Y/Y(\rho_2\rho_4\rho_5)\} = \{X/((X\rho_2)\rho_4)\rho_5,\ Y/((Y\rho_2)\rho_4)\rho_5\} = \{X/(s(X_6)\rho_4)\rho_5,\ Y/(Y_6\rho_4)\rho_5\} = \{X/s(s(X_8))\rho_5,\ Y/Y_8\rho_5\} = \{X/s(s(0)),\ Y/s(0)\}$, corresponding to the pair $\langle 2, 1\rangle$.

Besides the head of a variant of clause 1, also the head of a variant of clause 2 unifies with goal $g2$. Thus, we rename apart clause 2 and we get:

$2.6\quad sum(s(X_{10}), Y_{10}, s(Z_{10})) \leftarrow sum(X_{10}, Y_{10}, Z_{10})$

The unification gives us the substitution is $\rho_6 = \{X_8/s(X_{10}),\ Y_8/Y_{10},\ Z_{10}/0\}$. We then replace goal $g2$ by the new goal $g3$, obtained by SLD resolution from goal $g2$ and clause 2.6:

$g3.\ \leftarrow sum(X_{10}, Y_{10}, Z_{10})\rho_6$

that is, $\leftarrow sum(X_{10}, Y_{10}, 0)$. We proceed as above, by looking for a variant of a clause whose head unifies with goal $g3$. Only clause 1 has a variant whose head unifies with $g3$. We rename apart clause 1 and we get:

1.6 $\quad sum(0, X_{11}, X_{11}) \leftarrow$

The unification of goal $g3$ with the head of clause 1.6 gives us the substitution $\rho_7 = \{X_{10}/0,\ Y_{10}/0,\ X_{11}/0\}$. By SLD resolution from goal $g3$ and clause 1.6, we get the empty goal and the computation halts with the c.a.s. $\{X/X(\rho_2\rho_4\rho_6\rho_7),\ Y/Y(\rho_2\rho_4\rho_6\rho_7)\} = \{X/(((X\rho_2)\rho_4)\rho_6)\rho_7,\ Y/(((Y\rho_2)\rho_4)\rho_6)\rho_7\} = \{X/((s(X_6)\rho_4)\rho_6)\rho_7,\ Y/((Y_6\rho_4)\rho_6)\rho_7\} = \{X/(s(s(X_8))\rho_6)\rho_7,\ Y/(Y_8\rho_6)\rho_7\} = \{X/s(s(s(X_{10})))\rho_7,\ Y/(Y_{10})\rho_7\} = \{X/s(s(s(0))),\ Y/0\}$, corresponding to the pair $\langle 3, 0\rangle$.

The various computation paths starting from goal $g0$, can be arranged as the SLD tree depicted in Figure 5, where: (i) goal $g0$ is the father of the empty goal (via ρ_1) and the goal $g1$ (via ρ_2), (ii) goal $g1$ is the father of the empty goal (via ρ_3) and the goal $g2$ (via ρ_4), (iii) goal $g2$ is the father of the empty goal (via ρ_5) and the goal $g3$ (via ρ_6), and (iv) goal $g3$ is the father of the empty goal (via ρ_7). Every empty goal corresponds to a c.a.s. for the program *Sum* and the given initial goal $g0$: $\leftarrow sum(X, Y, s(s(s(0))))$.

Note that our procedure computes the four computed answers substitutions for the program *Sum* and the goal $g0$ without taking into account the commutativity of the predicate *sum*, that is, without exploiting the fact that, for all terms u and v in the set $\{0, s(0), s(s(0)), \ldots\}$ there exists a term w in the same set such that $sum(u, v, w)$ iff $sum(v, u, w)$.

In the above examples we have seen that the computation proceeds by: (i) unifying in a nondeterministic manner, that is, *in all possible ways*, a goal with the head of a variant of a clause in the program, (ii) replacing *old* goals (corresponding to heads of clauses) by *new* goals (corresponding to bodies of clauses), thereby generating a tree of goals, and (iii) recording the unifying substitutions along the arcs of that tree. In order to avoid clashes when composing substitutions, we make sure that the variables of the clauses involved in the unification process are all *new*, and this is ensured by a suitable renaming apart of the clauses.

The whole process of: (i) performing unifications, (ii) replacing old goals by new goals, (iii) recording substitutions, and (iv) renaming apart clauses, is formalized in the SLD resolution process (see Section 8.3).

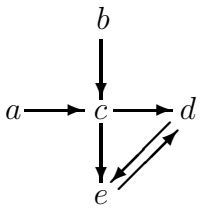

Fig. 6. A directed graph G with the set of nodes $\{a, b, c, d, e\}$.

Let us end this section by providing one more example of computing relations. From this example it will be clear that logic programming can be used for solving search problems in a very simple manner.

Example 7. We are given the directed graph G of Figure 6 and we want to compute all nodes reachable from node c.
The following clauses define the predicate $reach(X, Y)$ which holds iff from node X we can reach node Y via a sequence of directed arcs:

R.1 $reach(X, X) \leftarrow$
R.2 $reach(X, Z) \leftarrow arc(X, Y),\ reach(Y, Z)$

The encoding of the graph G is given by the conjunction of the following clauses defining the predicate $arc(X, Y)$ which holds iff in G there exists an arc from node X to node Y:

R.3 $arc(a, c) \leftarrow$
R.4 $arc(b, c) \leftarrow$
R.5 $arc(c, d) \leftarrow$
R.6 $arc(c, e) \leftarrow$
R.7 $arc(d, e) \leftarrow$
R.8 $arc(e, d) \leftarrow$

Let *Reach* be the program made out of clauses R.1–R.8. If we want to compute the set of nodes of G reachable from node c, we may use the program *Reach* with the initial goal $\leftarrow reach(c, Z)$. By using SLD resolution starting from that goal, we get the following three computed answer substitutions: (i) $\{Z/c\}$, (ii) $\{Z/d\}$, and (iii) $\{Z/e\}$, and indeed, in the graph G from node c we can reach the nodes c, d, and e.

We do not give the full account of how the computation proceeds in this case. Instead, in Figure 7 we represent the SLD tree for the program *Reach*, the initial goal $\leftarrow reach(c, Z)$, and the leftmost computation rule.

Note also that the SLD tree of Figure 7 is infinite (see the occurrences of the underlined nodes $\leftarrow reach(d, Z)$ and $\leftarrow reach(e, Z)$). Thus, it may be necessary to control the SLD resolution process to avoid the generation of infinite computations

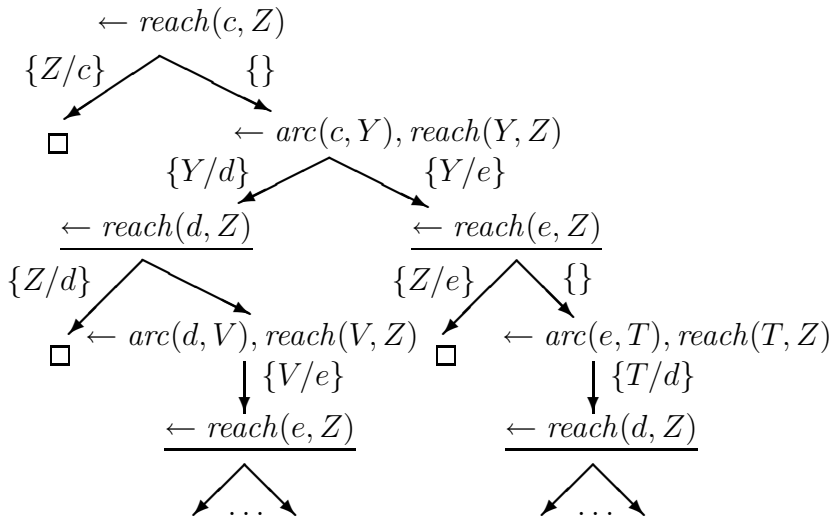

Fig. 7. The SLD tree for the program Reach, the initial goal $\leftarrow reach(c, Z)$, and the leftmost computation rule. $\{\}$ denotes the identity substitution and $\Box$ denotes the empty goal.

which are useless, that is, incapable of producing new computed answer substitutions, that is, new reachable nodes. We will not discuss this control issue here.

9.4 Constructing Knowledge-Based Systems

Now we present a simple example of a knowledge-based system realized by using definite logic programs. As we will see, we can ask questions to a knowledge-based system, and by performing SLD resolution steps, we can also get the desired answers.

A knowledge-based system is encoded as a conjunction K of closed first-order predicate calculus formulas. We stipulate that the answer to a question of the form: 'Is there any X such that the property $p(X)$ holds?' is 'yes' if $K \vdash \exists X\, p(X)$. Otherwise, if $K \nvdash \exists X\, p(X)$, the answer is 'no'.

Given a knowledge-based system K, we may compute the answer to the question: 'Is there any X such that property $p(X)$ holds?' as we now indicate.

Let us first note that $K \vdash \exists X\, p(X)$ holds iff $K \wedge \neg\exists X\, p(X)$ is unsatisfiable iff $K \wedge \forall X\, \neg p(X)$ is unsatisfiable iff, by Theorem 14, $D \wedge \forall X\, \neg p(X)$ is unsatisfiable, where D is a closed formula in clausal form which can be derived from K as described in Section 4 (page 47) (note that $\forall X\, \neg p(X)$ is already a clause). Thus, $K \vdash \exists X\, p(X)$ holds iff $D \vdash \exists X\, p(X)$ holds.

Now let us consider the case when D is a definite logic program. In this case if ϑ is the c.a.s. of an SLD refutation for the program D, the goal $\leftarrow p(X)$, and some computation rule R, then by Theorem 24, we have that $D \vdash \forall(p(X)\vartheta)$.

Thus, we have that $D \vdash \exists X\, p(X)$ holds. Hence, also $K \vdash \exists X\, p(X)$ holds and we get the answer 'yes'. Otherwise, if there is no SLD refutation for the program D, the goal $\leftarrow p(X)$, and some computation rule R, then again by Theorem 24, we have that $D \vdash \exists X\, p(X)$ does not hold and we get the answer is 'no'.

Moreover, in the particular case when the knowledge-base D is a definite logic program and K is equivalent to D, we get that $K \vdash \forall(p(X)\vartheta)$ and if $\vartheta = \{X/t\}$ then $K \vdash \forall(p(t))$, that is, any instance of $p(t)$ is a witness for the existential quantifier of the formula $\exists X\, p(X)$.

Now let us see how this method of computing answers to questions works in an example, which we take from [2].

Let us consider the following three facts which constitute our knowledge-based system:

$F1$. Some of Fiorecchio's men entered the premises unaccompanied by anyone else.

$F2$. The guard searched all who entered the premises, except those who were accompanied by members of the firm.

$F3$. The guard searched none of Fiorecchio's men.

and let us also suppose that we want to compute the answer to the following question:

Q. Were any of Fiorecchio's men members of the firm?

We proceed by encoding the above facts and question as first-order predicate calculus formulas. Let us first introduce the following predicate symbols:

predicate symbols	meaning
$e(X)$	X entered the premises
$f(X)$	X is a Fiorecchio's man
$a(X,Y)$	X entered the premises accompanied by Y (that is, X entered the premises together with Y)
$m(X)$	X is a member of the firm
$ns(X)$	the guard did not search X

(Note that is our formalization we have not specified that if $a(X,Y)$ holds for some X and Y, then X and Y are not the same term.)

Then, the following four formulas encode Facts $F1$, $F2$, $F3$, and question Q, respectively:

1. $\exists X\ [f(X) \wedge e(X) \wedge \forall Y\ (a(X,Y) \rightarrow f(Y))]$
2. $\forall X\ [(e(X) \wedge ns(X)) \rightarrow \exists Y\ (a(X,Y) \wedge m(Y))]$
3. $\forall X\ [f(X) \rightarrow ns(X)]$
4. $\exists X\ [f(X) \wedge m(X)]$

Formula 1 derives from the following understanding of Fact $F1$: there are Fiorecchio's men who entered the premises and everybody, if any, who accompanied them was Fiorecchio's man.

Formula 2 can be explained as follows. From Fact $F2$ we have that everyone who entered the premises *either* was searched by the guard (that is, it was not the case that he was not searched) *or* was accompanied by a member of the firm. Thus, we get the formula:

$$\forall X\ [e(X) \rightarrow (\neg ns(X) \vee \exists Y\ (a(X,Y) \wedge m(Y)))].$$

Then, Formula 2 derives from this formula by replacing $\neg ns(X)$ in the conclusions by $ns(X)$ in the premises.

Formulas 3 and 4 are immediate from Fact $F3$ and question Q, respectively.

As we have stipulated above, the answer to the question Q is 'yes' iff $1 \wedge 2 \wedge 3 \vdash 4$, that is, $1 \wedge 2 \wedge 3 \wedge \neg 4$ is unsatisfiable. Now we look for a conjunction D of clauses such that D is unsatisfiable iff $1 \wedge 2 \wedge 3 \wedge \neg 4$ is unsatisfiable. This can be done by applying the procedure described in Section 4 (page 47) and, in particular, by eliminating the existential quantifiers from 1 and 2 by Skolemization (the existential quantifier in $\neg 4$ can be eliminated in favour of a universal quantifier by pushing $\neg$ inside). By using the new constant 0 and the new unary function r, we get the following formulas, where the superscript s tells us that we have performed Skolemization:

1^s. $f(0) \wedge e(0) \wedge \forall Y\ (a(0,Y) \rightarrow f(Y))$
2^s. $\forall X\ [(e(X) \wedge ns(X)) \rightarrow (a(X, r(X)) \wedge m(r(X)))]$
3. $\forall X\ [f(X) \rightarrow ns(X)]$
$\neg 4$. $\forall X\ \neg(f(X) \wedge m(X))$

The conjunction $1^s \wedge 2^s \wedge 3$ is a definite logic program, call it P, made out of the following clauses (recall that, as usual, clauses are implicitly quantified at the front):

1.1 $f(0) \leftarrow$
1.2 $e(0) \leftarrow$
1.3 $f(Y) \leftarrow a(0,Y)$
2.1 $a(X, r(X)) \leftarrow e(X),\ ns(X)$
2.2 $m(r(X)) \leftarrow e(X),\ ns(X)$
3.1 $ns(X) \leftarrow f(X)$

Thus, we get that $P \vdash \exists X\, p(X)$ iff there exists an SLD refutation for the program P, the goal $\leftarrow f(X),\ m(X)$ and some computation rule R. In our case such an SLD refutation exists and it is presented in Table 1 on page 79.

goal	variant of a clause of P	mgu
G_0: $\leftarrow f(X), m(X)$	1.3: $f(Y) \leftarrow a(0,Y)$	ϑ_0: $\{X/Y\}$
G_1: $\leftarrow a(0,Y), m(Y)$	2.1: $a(X_1,r(X_1)) \leftarrow e(X_1), ns(X_1)$	ϑ_1: $\{X_1/0, Y/r(0)\}$
G_2: $\leftarrow e(0), ns(0), m(r(0))$	1.2: $e(0) \leftarrow$	ϑ_2: $\{\}$
G_3: $\leftarrow ns(0), m(r(0))$	3.1: $ns(X_3) \leftarrow f(X_3)$	ϑ_3: $\{X_3/0\}$
G_4: $\leftarrow f(0), m(r(0))$	1.1: $f(0) \leftarrow$	ϑ_4: $\{\}$
G_5: $\leftarrow m(r(0))$	2.2: $m(r(X_5)) \leftarrow e(X_5), ns(X_5)$	ϑ_5: $\{X_5/0\}$
G_6: $\leftarrow e(0), ns(0)$	1.2: $e(0) \leftarrow$	ϑ_6: $\{\}$
G_7: $\leftarrow ns(0)$	3.1: $ns(X_7) \leftarrow f(X_7)$	ϑ_7: $\{X_7/0\}$
G_8: $\leftarrow f(0)$	1.1: $f(0) \leftarrow$	ϑ_8: $\{\}$
G_9: $\square$		

Table 1. SLD refutation for the program P, the goal $\leftarrow f(X), m(X)$, and the leftmost seclection rule.

The c.a.s. ϑ of this SLD refutation is $\{X/r(0)\}$. The substitution ϑ is obtained by: (i) composing the mgu's $\vartheta_0, \vartheta_1, \ldots,$ and ϑ_8, whereby getting the substitution $\{X/r(0),\ X_1/0,\ X_3/0,\ X_5/0,\ X_7/0\}$, and then (ii) restricting this substitution to $vars(f(X),\ m(X))$. The existence of the c.a.s. ϑ tells us that 'yes' is the answer to the question Q: «Were any of Fiorecchio's men members of the firm?».

Remark 9. A different understanding of Fact $F2$ leads to its encoding as the conjunction of formula 2 and the following formula:

$$2^*. \quad \forall X, Y\ (e(X) \wedge a(X,Y) \wedge m(Y)) \rightarrow ns(X)$$

Formula 2* corresponds to the clause:

$$2.3 \quad ns(X) \leftarrow e(X),\ a(X,Y),\ m(Y)$$

Note also that: (i) 2 is not equivalent to 2*, and (ii) if we replace clauses 2.1 and 2.2 by clause 2.3 in program P the answer to the question Q does not change. □

Remark 10. As a consequence of clause 2.1, since X does not unify with $r(X)$, we have that if $a(X,Y)$ holds for some X and Y then X and Y are not the same term. □

9.5 Theorem Provers and Interpreters of Horn Clauses

Now we present an example of use of definite logic programs for proving theorems holding in an algebraic theory. The example is taken from the landmark paper by J. A. Robinson where resolution was first presented [11].

We want to show that:

"in any associative system in which all equations of the form: $X \cdot a = b$ and $a \cdot Y = b$ have left and right solutions X and Y, respectively, we have that there is a unique right identity", that is,

$$\exists Y \, \forall X \quad X \cdot Y = X. \tag{id.1}$$

By 'an associative system' we mean that the operation $\cdot$ is associative, that is,

$$\forall X, Y, Z \quad (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$$

If we introduce the predicate $p(X, Y, Z)$ which holds iff $X \cdot Y = Z$, then every associative system in which all equations $X \cdot a = b$ and $a \cdot Y = b$ have left and right solutions X and Y, respectively, is characterized by the fact that the following formulas hold, where f, ℓ, and r are new, binary function symbols:

1. $\forall X, Y \quad p(X, Y, f(X, Y))$
2. $\forall X, Y \quad p(\ell(X, Y), X, Y)$
3. $\forall X, Y \quad p(X, r(X, Y), Y)$
4. $\forall X, Y, Z, U, V, W \quad (p(X, Y, U) \wedge p(Y, Z, V)) \rightarrow (p(U, Z, W) \leftrightarrow p(X, V, W))$

Formula 1 states that the given system is closed under $\cdot$, that is, for all elements X and Y in the system, there is in the system a unique element Z which equal to $X \cdot Y$. Formula 2 (with renaming of variables) states that every equation of the form: $X \cdot Y = Z$ has a left solution, that is, X is $\ell(Y, Z)$). Similarly, formula 3 (with renaming of variables) states that every equation of the form: $X \cdot Y = Z$ has a right solution, that is, Y is $r(X, Z)$. Formula 4 states that $\cdot$ is associative, that is, if both $p(X, Y, U)$ and $p(Y, Z, V)$ hold then we have that: $p(U, Z, W)$ holds iff $p(X, V, W)$ holds.

Now in order to check Property (id.1), we construct a conjunction D of Horn clauses which is unsatisfiable iff $1 \wedge 2 \wedge 3 \wedge 4 \wedge \neg$(id.1) is unsatisfiable.

The conjunction D can be constructed by using Skolemization as follows. From formulas 1–4 above we immediately derive the following clauses (recall that the order of the atoms in the body of clauses is not significant because comma, that is, 'and', is commutative):

1^c. $\quad p(X, Y, f(X, Y)) \leftarrow$

2^c. $\quad p(\ell(X, Y), X, Y) \leftarrow$

3^c. $\quad p(X, r(X, Y), Y) \leftarrow$

4.1^c $\quad p(U, Z, W) \leftarrow p(X, Y, U),\ p(X, V, W),\ p(Y, Z, V)$

4.2^c $\quad p(X, V, W) \leftarrow p(X, Y, U),\ p(U, Z, W),\ p(Y, Z, V)$

From the formula $\neg$(id.1), that is, $\neg\exists Y \, \forall X \; p(X,Y,X)$, we get by pushing negation inside, the equivalent formula $\forall Y \, \exists X \; \neg p(X,Y,X)$. From this formula, after Skolemization, we get: $\forall Y \; \neg p(k(Y), Y, k(Y))$, where k is a new, unary function symbol.

Thus, we get the following definite goal (where the bound variable Y has been replaced by X):

$5^c.\ \leftarrow p(k(X), X, k(X))$

Now we present a SLD refutation starting from the goal 5^c: $\leftarrow p(k(X), X, k(X))$ by using the *leftmost computation rule*. This refutation shows that the conjunction $1^c \wedge 2^c \wedge 3^c \wedge 4.1^c \wedge 4.2^c \wedge 5^c$ of Horn clauses is unsatisfiable and, thus, it shows that Property (id.1) holds.
From the goal:

$\leftarrow p(k(X), X, k(X))$

by using clause 4.1^c (renamed with X', instead of X) and the idempotent mgu $\{U/k(X),\ Z/X,\ W/k(X)\}$, we get:

$\leftarrow p(X', Y, k(X)),\ p(X', V, k(X)),\ p(Y, X, V)$

By using clause 2^c (renamed with X'' and Y'', instead of X and Y, respectively) and the idempotent mgu $\{X'/\ell(X'', k(X)),\ Y/X'',\ Y''/k(X)\}$, we get:

$\leftarrow p(\ell(X'', k(X)), V, k(X)),\ p(X'', X, V)$

By using clause 2^c (renamed with X''' and Y''', instead of X and Y, respectively) and the idempotent mgu $\{X''/X''',\ Y'''/k(X),\ V/X''', \}$, we get:

$\leftarrow p(X''', X, X''')$

By using clause 3^c (renamed with X' and Y', instead of X and Y, respectively) and the idempotent mgu $\{X'/X''',\ Y'/X''',\ X/r(X''', X''')\}$, we get the empty goal $\Box$.

We leave to the reader to show that also the following property holds:

$$\forall X\, \exists Y.\ X \cdot Y = X \qquad \text{(id.2)}$$

The proof can be done along the same lines of the above proof of Property (id.1). In particular, we have that $\neg\forall X\, \exists Y\ p(X, Y, X)$ is equivalent to $\exists X\, \forall Y\ \neg p(X, Y, X)$. From this formula, after Skolemization, we get: $\forall Y\ \neg p(a, Y, a)$, where a is a new, 0-ary function symbol. Thus, we get the definite goal (where the bound variable Y has been replaced by X):

$\leftarrow p(a, X, a)$

The following Prolog program named `rightIdentity.pl`, can be used to automatically prove the above Properties (id.1) and (id.2).

Since often, by default, Prolog systems perform unification without performing occurs-check [6] (while, as indicated in Point (v) of the Unification Algorithm on page 50, occurs-check should be performed), in the program below, written in Sicstus Prolog [6], we make sure that a unifying term t is finite (i.e., acyclic) by enforcing the satisfaction of the extra atom `acyclic_term(`t`)` (`ac(`t`)`, for short). Indeed, in Sicstus Prolog (i) `acyclic_term(X)` holds iff `X` is finite (i.e., acyclic), and (ii) an unbound variable is assumed to be an acyclic (finite) term.

At the end of the program `rightIdentity.pl` we have listed some execution traces, where a variable name beginning with an underscore '_', denotes an unbound variable. In particular, `X = r(_A, _A)` denotes that the variable `X` is bound to a term whose top, binary function symbol `r` has two subterms which are both bound to the same unbound variable, different from `X` itself.

```
% ===================================================================
% Filename: rightIdentity.pl
% Sicstus Prolog
% ===================================================================
  :- use_module(library(terms)).  % needed for using acyclic_term(X)
  ac(X) :- acyclic_term(X).       % for reasons of brevity
%
% -------------------------------------------------------------------
%                 Unification with occurs_check
  q(Y,Y).
  unif0(X) :- q(X,f(X)).
% success: the unifying value of X is an infinite (i.e., cyclic)
% term, that is, X = f(f(f(f(f(f(f(f(f(f(...)))))))))).
  unif1(X) :- q(X,f(X)), ac(X).
% failure: the unifying value of X is an infinite (i.e., cyclic)
% term.
% ===================================================================
%                   Right identity property
% in any associative system which has left and right solutions X and
% Y for all equations X.a=b and a.Y=b, there is a right identity.
% -------------------------------------------------------------------
% closure under '.': p(X,Y,Z) means that X.Y=Z.
  p(X,Y,f(X,Y)) :- ac(X), ac(Y).
% left and right unique solutions of equations:
  p(l(X,Y),X,Y) :- ac(X), ac(Y).
  p(X,r(X,Y),Y) :- ac(X), ac(Y).
% associativity of '.': (X.Y=U  /\  Y.Z=V) <=> U.Z=W=X.V
% The order of the atoms in the body of the following clauses is
% relevant for termination. It is enough to enforce acyclicity
% (finiteness) of the terms to which the variables of the heads are
% bound, because this enforces acyclicity also of the terms to which
% the variables of the bodies are bound.
  p(U,Z,W) :- p(X,Y,U), p(X,V,W), p(Y,Z,V), ac(U), ac(Z), ac(W).
  p(X,V,W) :- p(X,Y,U), p(Y,Z,V), p(U,Z,W), ac(X), ac(V), ac(W).
% -------------------------------------------------------------------
% Right identity:          \exists Y  \forall X  p(X,Y,X)     (id.1)
% Negation of (id.1): \neg \exists Y  \forall X  p(X,Y,X), that is,
%                          \forall Y  \exists X  \neg p(X,Y,X)
% After Skolemization:     \forall Y \neg p(k(Y),Y,k(Y))
% By replacing Y by X and enforcing that X be bound a to finite term
% we get:
  t1(X) :- p(k(X),X,k(X)), ac(X).
```

```
% ---------------------------------------------------------------
% Weak right identity:     \forall X  \exists Y  p(X,Y,X)     (id.2)
  t2(X) :- p(a,X,a), ac(X).
/* ===============================================================
| ?- unif0(X).   (only one success with X bound to an infinite term)
X = f(f(f(f(f(f(f(f(f(f(...)))))))))) ? ;
no
| ?- unif1(X).                          (failure)
no
| ?- t1(X).
X = r(_A,_A) ? ;                       (looking for one more solution)
X = r(_A,_A) ?
yes
| ?- t2(X).
X = r(a,a) ? ;                         (looking for one more solution)
X = r(_A,_A) ? ;                       (looking for one more solution)
X = r(_A,_A) ?
yes                                                              */
% ===============================================================
```

In order to automatically prove the Properties (id.1) and (id.2), we can also use the Prolog program `rightIdentityDemo.pl`, which we list below. In this program unification is performed with occurs-check by using an interpreter (which, for historical reasons, is also called a *meta-interpreter*) of Horn clauses.

This interpreter assumes that every clause C of the form: $H \leftarrow B_1, \ldots, B_n$, is represented as an atom of the form: $cl(H, [B_1, \ldots, B_n])$, where cl is a binary predicate whose first argument is the head of C and whose second argument is the list of the atoms of the body of C (as usual, lists are represented by using the square bracket notation).

Then, our interpreter is written as the following conjunction of Horn clauses which define the predicate *demo*:

d.1 $demo([\,]) \leftarrow$

d.2 $demo([A|As]) \leftarrow demo(A), demo(As)$

d.3 $demo(A) \leftarrow cl(H, B), A = H, demo(B)$

Clause d.1 and d.2 state that a list of atoms are proved by proving each atom of the list. Clause d.3 states that an atom A is proved by finding a clause $H \leftarrow B_1, \ldots, B_n$, whose head unifies with A via the mgu ϑ, and then proving the list $[B_1\vartheta, \ldots, B_n\vartheta]$ of atoms.

For instance, let us consider the two clauses:

$cl(p(0), [\,]) \leftarrow$

$cl(p(s(X)), [p(X)]) \leftarrow$

which represent the two clauses: $p(0) \leftarrow$, and $p(s(X)) \leftarrow p(X)$, respectively. Then, for the goal $demo(p(X))$, we get, as expected, the following answers:

$X = 0$,
$X = s(0)$,
$X = s(s(0))$, and so on.

Recall that in Sicstus Prolog: `unify_with_occurs_check(X,Y)` holds iff `X` and `Y` unify to a finite (acyclic) term.

```
% ====================================================================
% Filename: rightIdentityDemo.pl
% Sicstus Prolog
% ====================================================================
%                    USING THE META-INTERPRETER demo
%
% A clause of the form:  p :- a,b,c  is denoted by:  cl(p, [a,b,c])
%
  demo([]).                                                    %  (1)
  demo([A|As]) :- demo(A), demo(As).                           %  (2)
  demo(A) :- cl(H,B), unify_with_occurs_check(A,H), demo(B).   %  (3)
%
% cl(H,B) should be 'to the left' of unify_with_occurs_check(A,H)
% because both A and H should be bound when the goal
% unify_with_occurs_check(A,H) is evaluated.
% --------------------------------------------------------------------
%                  Unification with occurs_check
  cl(q(Y,Y),[]).                    % clause: q(Y,Y).
  cl(unif0(X), [q(X,X)]).           % clause: unif0(X) :- q(X,X).
  cl(unif1(X), [q(X,f(X))]).        % clause: unif1(X) :- q(X,f(X)).

  d0(X) :- demo(unif0(X)).          % success
  d1(X) :- demo(unif1(X)).          % failure
% ====================================================================
%                    Right identity property
% in any associative system which has left and right solutions X and
% Y for all equations X.a=b and a.Y=b, there is a right identity.
% --------------------------------------------------------------------
  cl(p(X,Y,f(X,Y)),[]).             % clause: p(X,Y,f(X,Y)).
  cl(p(l(X,Y),X,Y),[]).             % clause: p(l(X,Y),X,Y).
  cl(p(X,r(X,Y),Y),[]).             % clause: p(X,r(X,Y),Y).
  cl(p(U,Z,W), [p(X,Y,U), p(X,V,W), p(Y,Z,V)]).
                 % clause: p(U,Z,W) :- p(X,Y,U), p(X,V,W), p(Y,Z,V).
  cl(p(X,V,W), [p(X,Y,U), p(Y,Z,V), p(U,Z,W)]).
                 % clause: p(X,V,W) :- p(X,Y,U), p(Y,Z,V), p(U,Z,W).
% ---------------------
  d2(X) :- demo(p(k(X),X,k(X))).    % success
  d3(X) :- demo(p(a,X,a)).          % success
/* ====================================================================
| ?- d0(X).         (only one success with X bound to a finite term)
yes
```

```
| ?- d1(X).                          (failure)
no
| ?- d2(X).
X = r(_A,_A) ? ;                     (looking for one more solution)
X = r(_A,_A) ?
yes
| ?- d3(X).
X = r(a,a) ? ;                       (looking for one more solution)
X = r(_A,_A) ? ;                     (looking for one more solution)
X = r(_A,_A) ?
yes                                                              */
% ================================================================
```

10 Deriving Negative Information from Definite Logic Programs

Given a definite program P, it is impossible to derive a negative literal as a logical consequence of P, because every clause in P has a positive literal in the head. However, given a definite program P we now define a set of atoms, which by abuse of language, we call the *negative consequences* of P. This set is also called the *finite failure set* of P and it is denoted by $FF(P)$.

Definition 14. Given a definite program P, we say that an atom A belongs to the *finite failure set* of P, written $A \in FF(P)$, iff there exists a computation rule R such that there exists a *finitely failed* SLD tree for program P, goal $\leftarrow A$, and computation rule R.

Given a node, say N, and one of its child-nodes, say M, in an SLD tree, we may relate via the *descendant relation* δ_{NM}, an occurrence of an atom of the goal in N to an occurrence of an atom of the goal in M. Instead of giving the formal definition of the descendant relation, we now give an example which, we hope, will allow the reader to construct the descendant relation δ_{NM} for any given node N and child-node M in a given SLD tree.

Let us consider the node N with goal $\leftarrow A, B, C$ and let us assume that from N we generate a child-node M via an SLD resolution step by: (i) selecting the atom B, and (ii) considering the clause $H \leftarrow B_1, B_2$ such that $H\vartheta = B\vartheta$. Thus, the node M is associated with the goal $\leftarrow A\vartheta, B_1\vartheta, B_2\vartheta, C\vartheta$. In this case the descendant relation δ_{NM} consists of the following four pairs of atom occurrences: $\langle A, A\vartheta\rangle$, $\langle B, B_1\vartheta\rangle$, $\langle B, B_2\vartheta\rangle$, and $\langle C, C\vartheta\rangle$.

From the above example one can see that given a node N and one of its child-nodes M, the relation δ_{NM} is constructed as the union of two relations:

(i) the *head-body relation* α_{NM} relative to the clause which has been used for generating the child-node M from the node N via an SLD resolution step (in our example this relation is the set made out of the two pairs $\langle B, B_1\vartheta\rangle$ and $\langle B, B_2\vartheta\rangle$), and
(ii) the *inheritance relation* β_{NM} due to the occurrences of the atoms which are *not* selected for the SLD resolution step (in our example this relation is the set made out of the two pairs $\langle A, A\vartheta\rangle$ and $\langle C, C\vartheta\rangle$).

Given a definite program P, a goal $\leftarrow A$, and a computation rule F, we say that F is *fair* iff in the (finite or infinite) SLD tree T for P, $\leftarrow A$, and F it does not exist an infinite sequence $\langle N_0, N_1, N_2, \ldots\rangle$ of nodes such that:
- in the goal at node N_0 there is the atom A_0,
- in the goal at node N_1 which is a child-node of N_0, there is the atom A_1,
- in the goal at node N_2 which is a child-node of N_1, there is the atom A_2,...
with $\langle A_0, A_1\rangle \in \beta_{N_0N_1}$, $\langle A_1, A_2\rangle \in \beta_{N_1N_2}$, ... and none of the atoms $A_0, A_1, A_2, \ldots$ is selected by F. Note that by definition of an SLD tree, for every atom B occurring in a leaf of T there is no variant of a clause of P whose head unifies with B.

Thus, any computation rule which constructs a finite SLD tree is surely fair.

Theorem 27. Given any definite program P, any definite goal $\leftarrow A$, and any fair computation rule F, $A \in FF(P)$ iff the SLD tree for program P, goal $\leftarrow A$, and computation rule F, is finitely failed.

Thus, given a definite program P, any definite goal $\leftarrow A$, by constructing the SLD tree for P and $\leftarrow A$, using *any fixed fair* computation rule, we construct a finitely failed SLD tree if there exists one which can be constructed by using *any other* (*fair or not fair*) *computation rule*.

The finite failure set $FF(P)$ of a definite program P is the set of all literals which are assumed to be negative consequences of P, but the definition of $FF(P)$ is arbitrary in the sense that, as we remarked above, there are no negative literals which are logical consequences of P. However, the finite failure set of a definite program P enjoys an important property (see Theorem 28 below) with respect to a formula related to P, called the completion of P, which we now define.

Completion of a definite program

Given a definite program P, its *completion* also called *Clark completion*, denoted by $comp(P)$, is a conjunction of first order formulas constructed as follows [1, page 535]. Assume that $=$ is a new binary predicate symbol not occurring in P.

(1) For each clause of P:

(1.1) Replace every term in the arguments of the head in favour of *new* variables and add suitable equalities in the body, so that all clauses in P with the same predicate symbol in the head, have identical heads.

For instance, the conjunction $(p(t, u) \leftarrow G_1)$, $(p(r, f(V)) \leftarrow G_2)$ of clauses is

transformed into
$(p(X,Y) \leftarrow X{=}t,\ Y{=}u,\ G_1)$, $(p(X,Y) \leftarrow X{=}r,\ Y{=}f(V),\ G_2)$.

(1.2) Existentially quantify the body w.r.t. the variables occurring in the body and not in the head.

For instance, if $vars(t,u,G_1) - \{X,Y\} = \{W,Z\}$, then $p(X,Y) \leftarrow X = t$, $Y{=}u, G_1$ is transformed into $p(X,Y) \leftarrow \exists W \exists Z\ (X{=}t,\ Y{=}u,\ G_1)$.

(2) Fuse all clauses with equal head into one implication by using $\vee$.

For instance, the conjunction $(p(X,Y) \leftarrow Body_1)$, $(p(X,Y) \leftarrow Body_2)$ of clauses is transformed into the implication $p(X,Y) \leftarrow (Body_1 \vee Body_2)$.

(3) Universally quantify every implication derived at Step (2) w.r.t. the variables occurring in its conclusion and replace $\leftarrow$ by $\leftrightarrow$.

For instance, $p(X,Y) \leftarrow (Body_1 \vee Body_2)$ is transformed into
$\forall X,Y\ (p(X,Y) \leftarrow (Body_1 \vee Body_2))$.

(4) Introduce a formula of the form: $\forall X_1 \ldots X_n \neg r(X_1,\ldots,X_n)$ for each predicate symbol r occurring in the body of a clause of P, and not in any head of P.

(5) Consider the (infinite) conjunction of the following formulas of the first order predicate calculus, which define the so called Clark Equality Theory, or CET, for short. This theory axiomatizes the usual equality predicate for the Herbrand interpretation.

(5.1) For each function symbol f,

$$\forall X_1 \ldots X_n Y_1 \ldots Y_n$$
$$[(X_1 = Y_1 \wedge \ldots \wedge X_n = Y_n) \leftrightarrow f(X_1,\ldots,X_n) = f(Y_1,\ldots,Y_n)].$$

(In any Herbrand interpretation function symbols are interpreted as themselves)

(5.2) For each pair of distinct function symbols f and g,

$$\forall X_1 \ldots X_n Y_1 \ldots Y_m \neg(f(X_1,\ldots,X_n) = g(Y_1,\ldots,Y_m)).$$

(5.3) For each predicate symbol p,

$$\forall X_1 \ldots X_n Y_1 \ldots Y_n$$
$$[(X_1 = Y_1 \wedge \ldots \wedge X_n = Y_n) \rightarrow (p(X_1,\ldots,X_n) \leftrightarrow p(Y_1,\ldots,Y_n))].$$

(5.4) $\forall X\ X{=}X$.

(5.5) For each term t such that $vars(t) = \{X, X_1, \ldots, X_n\}$ and t is not X,

$$\forall X\ X_1 \ldots X_n \neg(X = t).$$

The completion $comp(P)$ of P is the conjunction of:

- the formulas derived at Step (3), denoted by $iff(P)$ (these formulas have, in general, the symbols $\exists, \vee, \forall$, and $\leftrightarrow$),
- the formulas introduced at Step (4) (these formulas have also the symbol $\neg$), and
- the formulas introduced at Step (5), that is, CET (these formulas have also the symbol $=$).

Note that in Point (5.3) $\rightarrow$ cannot be replaced by $\leftrightarrow$ because, for instance, it may be the case that both $p(0)$ and $p(1)$ hold and yet $0 \neq 1$.

For any program P, we have that CET is a *consistent* and *complete* theory for the set of closed first order formulas whose predicate symbols are *true*, *false*, and $=$ only, and thus, for any closed first order formula φ in that set, either CET $\vdash \varphi$ or CET $\vdash \neg\varphi$ holds, but not both.

Remark 11. From the formulas (5.3) and (5.4) we have that $=$ denotes an equivalence relation [8, page 79], and CET forces the predicate symbol $=$ to be interpreted as the identity relation on the domain of I, for every Herbrand interpretation I. We also have that:
CET $\vdash \exists X_1 \ldots X_n \; u = v$ iff
the terms u and v are unifiable, where $\{X_1, \ldots, X_n\} = vars(u, v)$.
Thus,
CET $\vdash \forall X_1 \ldots X_n \; \neg(u = v)$ iff
the terms u and v are *not* unifiable, where $\{X_1, \ldots, X_n\} = vars(u, v)$.

Example 8. (i) Given the definite program $P_1 = \{p(0) \leftarrow,\ p(s(Y)) \leftarrow p(Y)\}$, $comp(P_1)$ is $\forall X\,(p(X) \leftrightarrow (X = 0 \vee \exists Y\,(X = s(Y) \wedge p(Y)))) \wedge \text{CET}$.

(ii) Given the definite program $P_2 = \{p(0) \leftarrow,\ p(s(Y)) \leftarrow r(a, Z)\}$, $comp(P_2)$ is $\forall X\,(p(X) \leftrightarrow (X = 0 \vee \exists Y \exists Z\,(X = s(Y) \wedge r(a, Z)))) \wedge \forall X \forall Y\, \neg r(X, Y) \wedge \text{CET}$.

The following theorem establishes for any given definite program P, the relationship between its least Herbrand model $M(P)$, the finite failure set $FF(P)$, and the completion $comp(P)$.

Theorem 28. Let us consider a definite program P and a ground atom A. By $M(P)$ we denote, as usual, the least Herbrand model of P.
(i) $A \in M(P)$ iff $comp(P) \models A$, and (ii) $A \in FF(P)$ iff $comp(P) \models \neg A$.

As a consequence of Theorem 7 on page 35, Theorem 28 also holds if we write $\vdash$, instead of $\models$.

Proposition 26. For every definite program P and for every ground atom A,
(i) if $comp(P) \models \neg A$ then $A \notin M(P)$, and thus, $P \not\models A$, and
(ii) $M(P) \models comp(P)$. Thus, for every formula φ, if $comp(P) \models \varphi$ then $M(P) \models \varphi$.

Proof. Point (i) follows from Theorem 28, because $comp(P) \models \neg A$ iff $comp(P) \not\models A$, and by Theorem 21, $M(P) \models A$ iff $P \models A$. Point (ii) follows from the fact that, as stated in Proposition 27 on page 90, for every definite program P, every fixpoint of the T_P operator is an Herbrand model of $comp(P)$, and $M(P)$ is a fixpoint of T_P (see Theorem 23 on page 64). □

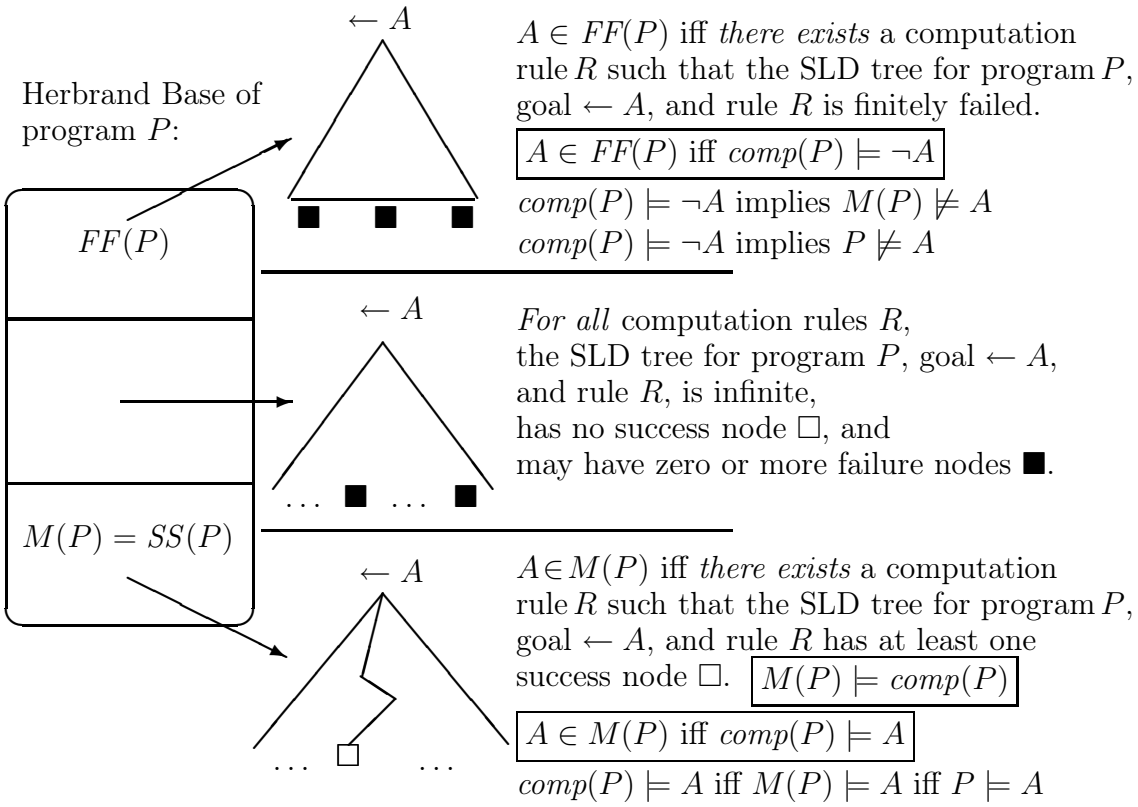

Fig. 8. The Herbrand Base of a definite program P with the least Herbrand model $M(P)$, the success set $SS(P)$, and the finite failure set $FF(P)$. By definition, we have that $M(P)$ and $FF(P)$ are disjoint sets of atoms.

As illustrated in Figure 8 we have the following theorem.

Theorem 29. For every definite program P and definite goal $\leftarrow (A_1 \wedge \ldots \wedge A_n)$, where $A_1, \ldots, A_n$ are (possibly non-ground) atoms, we have that:

(i) $P \models \forall((A_1 \wedge \ldots \wedge A_n)\vartheta)$ iff $comp(P) \models \forall((A_1 \wedge \ldots \wedge A_n)\vartheta)$, where ϑ is any substitution such that $domain(\vartheta) = vars(A_1 \wedge \ldots \wedge A_n)$ [8, Theorem 14.6 on page 82], and

(ii) $comp(P) \models \neg(A_1 \wedge \ldots \wedge A_n)$ iff *there exists* a computation rule R and a finitely failed SLD tree for program P, goal $\leftarrow (A_1 \wedge \ldots \wedge A_n)$, and rule R [8, Theorem 16.5 on page 102] iff
for every fair computation rule F we get a finitely failed SLD tree for program P, goal $\leftarrow (A_1 \wedge \ldots \wedge A_n)$, and rule F [8, Theorem 16.1 on page 95, and Theorem 16.5 on page 102].

Recall that since $comp(P)$ is a closed formula, then $comp(P) \models \neg(A_1 \wedge \ldots \wedge A_n)$ iff $comp(P) \models \forall(\neg(A_1 \wedge \ldots \wedge A_n))$. More properties of the completion of definite programs are stated in the following section with reference to the class of normal programs which includes the class of definite programs. In particular, we have the following proposition (where the first point has number (iii), because Points (i) and (ii) are assumed to be those of Theorem 29).

Proposition 27. For every definite program P,
(iii) $comp(P) \models P$, and
(iv) for every Herbrand interpretation I, $I \models comp(P)$ iff $T_P(I) = I$.
(v) for a ground atom A in the Herbrand Base, $A \in gfp(T_P)$ iff $comp(P) \cup \{A\}$ has a Herbrand model.

Compare Point (iv) of this Proposition with Lemma 11 on page 63, where it is stated that for every Herbrand interpretation I, for every definite program P, $I \models P$ iff $T_P(I) \subseteq I$.

Contrary to the case of definite programs, we have that: (i) a normal program may not have a least Herbrand model (see Proposition 30 on page 96), and (ii) there exists a normal program P with least Herbrand model $M(P)$ and $M(P) \not\models comp(P)$ (see Proposition 31 on page 96).

11 Normal Programs

We may extend the class of definite programs by allowing negated atoms to occur in the body of the clauses. By doing so, we get the class of *normal programs*, also called *general programs*. Similarly, we may extend the class of definite goals by allowing negated atoms to occur in them and we get the class of *normal goals*, also called *general goals*.

As for the case of definite programs, given a normal program P, it is impossible to derive a negative literal as a logical consequence of P, because every clause in P has a positive literal in its head.

We can extend the definition of the completion of a program P also to the case of normal programs by making the same construction we have described in Section 10 for definite programs.

For any given definite program, definite goal, and computation rule, we have introduced in the previous section the concepts of SLD derivation, SLD refutation, and SLD tree. These concepts are all based on the concept of the SLD resolution step. Analogously, for any given normal program, normal goal, and computation rule, now we introduce the concepts of SLDNF derivation, SLDNF refutation, and SLDNF tree, which are all based on the concept of SLDNF resolution step. This step is identical to an SLD resolution step, except for the case when negated atoms are selected. In this case special actions must be taken as we specify below in the definition of the SLDNF derivation.

The definitions of SLDNF resolution, SLDNF derivation, SLDNF refutation, and SLDNF tree will be given in a mutually dependent way (see also [8, Chapter 3]).

For constructing an SLDNF derivation we use a *safe* computation rule, that is, a *partial* function from the set of non-empty goals to the set of literals such that given a goal, (i) it selects either a *positive* literal or a *ground negative* literal in that goal, and (ii) it does not select any literal iff the given goal has non-ground negative literals only. Safe computation rules are needed to ensure the soundness of the SLDNF resolution (see Theorem 31 below).

An SLDNF derivation

The definition of an *SLDNF derivation* for a normal program P, a normal goal G, and a *safe* computation rule R, is like the one of an SLD derivation in the case of definite programs (see page 64), except that in any goal of the sequence $\langle G_0, G_1, \ldots \rangle$ of goals and in any clause of the sequence $\langle C_0, C_1, \ldots \rangle$ of clauses we allow also negative literals to occur, and

(1) if goal G_i is $\leftarrow A_1, \ldots, A_{m-1}, \neg A, A_{m+1}, \ldots, A_n$, and the literal $\neg A$ is the selected one, and there exists a finitely failed SLDNF tree (defined below) for P, $\leftarrow A$, and R, then: (1.i) goal G_{i+1} is $\leftarrow A_1, \ldots, A_{m-1}, A_{m+1}, \ldots, A_n$, (1.ii) ϑ_i is the identity substitution, and (1.iii) the element C_i in the sequence $\langle C_0, C_1, \ldots \rangle$ of clauses is the negative literal $\neg A$,

(2) if goal G_i is $\leftarrow A_1, \ldots, A_{m-1}, \neg A, A_{m+1}, \ldots, A_n$, and the literal $\neg A$ is the selected one, and there exists an SLDNF refutation (defined below) for P, $\leftarrow A$, and R, then the SLDNF derivation ends with that goal G_i, and

(3) if in G_i there are non-ground negative literals only, then the SLDNF derivation ends with that goal G_i. In this case we also say that the SLDNF derivation *flounders*.

An SLDNF derivation which ends with the empty goal $\square$, is called an *SLDNF refutation*.

The *computed answer substitution* (*c.a.s.*, for short) of an SLDNF refutation for a normal program P, a normal goal G, and a computation rule R, is obtained by: (i) taking the composition of the most general unifiers of the sequence relative to that SLDNF refutation, and (ii) restricting the resulting substitution to the variables occurring in G.

Let us now specify how to construct an SLDNF tree. We need first the following definition.

Definition 15. (i) We say that an SLDNF tree T has *an* SLDNF *refutation* (or has a path with a *success leaf*) iff T has a finite root-to-leaf path which ends with the empty goal. (ii) An SLDNF tree T is said to be *finitely failed* iff T is finite and every root-to-leaf path in T ends with a failure leaf (see page 66 and Point (α) in the following algorithm for the construction of an SLDNF tree).

Construction of an SLDNF tree

The construction of an SLDNF tree for a normal program P, a normal goal G, and a *safe* computation rule R, proceeds as the one of an SLD tree in the case of definite programs, except when a ground negative literal, say $\neg A$, is selected by the computation rule R at a node, say N. In that case we proceed as follows.

(α) If the SLDNF tree for P, $\leftarrow A$, and R has an SLDNF refutation then: (α.1) we do not generate any child-node of N and N is made a failure leaf of T, and (α.2) we resume the construction of the SLDNF tree T as specified at Point (γ).

(β) If the SLDNF tree for P, $\leftarrow A$, and R is finitely failed then: (β.1) we generate a child-node, say $N1$, of the node N (which will have exactly one child-node) and we attach to $N1$ the same goal of N without the literal $\neg A$, and (β.2) we resume the construction of T as specified at Point (γ).

(γ) The construction of the tree T is resumed by considering a node which is not a failure leaf and whose goal is non-empty and it has either a positive literal or a ground negative literal.

Let us now make the following notes.

(1) It is not necessary to completely construct the SLDNF tree of Point (α). It is enough to construct an SLDNF refutation, that is, a suitable path of that tree.

(2) Step (α.1) is based on the fact that if there exists an SLDNF refutation for P, $\leftarrow A$, and R then $comp(P) \vdash A$.

(3) The deletion of $\neg A$ at Step (β.1) is said to be an application of the *negation-as-failure* rule. Indeed, by applying that rule, if there exists a finitely failed SLDNF tree for P, $\leftarrow A$, and R, we replace the literal $\neg A$ by *true*. The negation-as-failure rule is justified by Theorem 31 below, whereby if there exists a finitely failed SLDNF tree for P, $\leftarrow A$, and R, then $comp(P) \vdash \neg A$.

(4) As it is the case for SLD trees, also SLDNF trees are *maximal* in the sense that, if during the construction of an SLDNF tree, a node can have one (or one more) child-node, then we should include that child-node in the SLDNF tree.

As a consequence of this fact, we have that for every non-empty goal H at a leaf of a finitely failed SLDNF tree for a program P, a goal G, and a safe computation rule R, either: (i) no clause head in P unifies with the positive literal in H selected by R, or (ii) there is in H a negative literal, say $\neg A$, such that there is an SLDNF refutation for P, $\leftarrow A$, and R.

(5) There are cases in which an SLDNF tree cannot be constructed. For instance, let us consider the program $P = \{p \leftarrow p\}$ and the goal $G : \leftarrow \neg p$. No SLDNF tree can be constructed for program P, goal G, and any computation rule R because for the goal $\leftarrow p$ there exists neither an SLDNF refutation nor a finitely failed SLDNF tree.

It is undecidable whether or not an SLDNF derivation for a normal program, a normal goal, and a safe computation rule, does not flounders [1, Theorem 6.12]. The same holds if we do not require that the computation rule be safe.

As we will see below, it is important to know whether or not the SLDNF derivation for a given normal program, a given normal goal, and a given safe computation rule, does flounder. We now give a sufficient condition which ensures that an SLDNF derivation does not flounder [8, Proposition 15.1 on page 89]. This condition is based on the following concepts.

A clause is *admissible* iff every variable in that clause has at least one occurrence either in the head or in a positive literal in the body.

A clause is *allowed* iff every variable in that clause has at least one occurrence in a positive literal in the body.

A goal $\leftarrow L_1, \ldots, L_n$ is *allowed* iff every variable in that goal has at least one occurrence in a positive literal of that goal.

Thus, if a clause C is allowed and it has an empty body then the head of C must be ground.

Given a normal program P and a normal goal G, the pair $[P, G]$ is said to be *allowed* iff
(i) every clause of P is admissible,
(ii) G is allowed, and
(iii) every clause of P whose head predicate occurs either in a positive literal of the body of a clause of P or in a positive literal of G, is allowed.

For the following theorem recall that a substitution ϑ is said to be *grounding* for a formula φ iff $\varphi\vartheta$ has not free variables.

Theorem 30. Let us consider a normal program P, a normal goal G, and a safe computation rule R. Assume that $[P, G]$ is allowed. Then every SLDNF derivation

for P, G, and R does not flounder and the corresponding c.a.s. is a grounding substitution for G [8, Proposition 15.1].

A program P is said to be *hierarchical* iff we can assign to every predicate p of P a non-negative integer, called the *level* of p, such that in every clause C of P the level of the predicate in the head of C is *strictly greater* than the level of every predicate in the body of C.

Theorem 31. [Soundness and Relative Completeness of SLDNF Resolution for Normal Programs] Let us consider a normal program P, a normal goal G: $\leftarrow L$ where L is a literal, and a safe computation rule R.

Soundness of SLDNF resolution: If ϑ is the c.a.s. of an SLDNF refutation for P, G, and R then $comp(P) \models \forall(L\vartheta)$ [8, Theorem 15.6]. If there exists a finitely failed SLDNF tree for P, G, and R then $comp(P) \models \forall(\neg L)$ [8, Theorem 15.4].

Completeness of the SLDNF resolution for hierarchical normal programs: If P is hierarchical, $[P, G]$ is allowed, $comp(P) \models L\vartheta$, and ϑ is grounding substitution for L, then ϑ is a c.a.s. of an SLDNF refutation for P, G, and R. (Actually, for every safe computation rule R the SLDNF tree for P, G, and R is finite and ϑ is the c.a.s. of one of its SLDNF refutations [8, Theorem 16.3].)

Theorem 31 can be stated for the goal $\leftarrow (L_1, \ldots, L_n)$, with $n \geq 1$, instead of $\leftarrow L$, by replacing the literal L by the conjunction $L_1 \wedge \ldots \wedge L_n$ of literals.

Note also that for the soundness of SLDNF resolution the hypothesis that R is a safe computation rule is indeed necessary as the following example shows.

Let us consider the normal program $P = \{p \leftarrow \neg q(x) \;\; q(a) \leftarrow\}$. We have that: (i) there exists a finitely failed SLDNF tree for P, $\leftarrow p$, and a computation rule which is *not* safe and selects the negative literal $\leftarrow \neg q(x)$ (because there exists a SLDNF refutation for P, $\leftarrow q(x)$, and the computation rule which selects the atom $q(x)$), and (ii) it is not the case that $comp(P) \models \forall(\neg p)$. Point (ii) holds because: (ii.1) $comp(P)$ is $p \leftrightarrow \exists X\, \neg q(X) \wedge \forall X\, (q(X) \leftrightarrow X = a) \wedge \mathrm{CET}$, (ii.2) $\forall(\neg p) \leftrightarrow \neg p$, and (ii.3) it is not the case $comp(P) \models \neg p$. In order to show Point (ii.3) let us consider the interpretation I with domain $\{a, b\}$ such that both $q(a)$ and p are true in I and $q(b)$ is false in I. Thus, $comp(P) \not\models \forall X\, q(X)$ and we have that $(\forall X\, q(X)) \leftrightarrow \neg p$.

In the completeness of SLDNF resolution for hierarchical normal programs, we have that ϑ should be grounding for L because of Theorem 30. Note that SLDNF resolution is *not* complete for all normal programs (in particular, in Theorem 31 the normal programs are required to be hierarchical), while SLD resolution is complete for all definite programs as stated in Theorem 24.

The incompleteness of SLDNF resolution is due to the following two limitations of the negation-as-failure rule (see page 92):
(1) the negation-as-failure rule does not generate bindings for variables, because a safe computation rule selects negative literals only if they are ground, and
(2) the negation-as-failure rule does not take into account infinite failures, in the sense that it does not allow to infer $\neg A$ in the case when the SLDNF tree for the given program, the goal $\leftarrow A$, and the given computation rule, has no success leaf and it is infinite.

These two limitations are shown through the following two examples.

Example 9. [Limitation 1] Let us consider the program $P = \{p(x) \leftarrow,\ q(a) \leftarrow,\ r(b) \leftarrow\}$ and the goal $\leftarrow p(x), \neg q(x)$. We have that $comp(P) \models p(b) \wedge \neg q(b)$, and yet $\{x/b\}$ is not a computed answer substitution for program P, goal $\leftarrow p(x), \neg q(x)$, and any safe computation rule, because any safe computation rule does not select the non-ground literal $\neg q(x)$. □

Example 10. [Limitation 2] Let us consider the program $P = \{p \leftarrow q,\ p \leftarrow \neg q,\ q \leftarrow q\}$ and the goal $\leftarrow p$. We have that $comp(P) \models p$, and yet the identity substitution $\{\}$ is not a computed answer substitution for program P, goal $\leftarrow p$, and any safe computation rule, because the SLDNF tree for P, $\leftarrow q$, and any safe computation rule has no success leaf and it is infinite. □

Now given a normal program P, we define a class of operators, called T_P^J, where J is a so called pre-interpretation defined as follows.

Given a domain D, a *pre-interpretation* J assigns a function in $D^r \rightarrow D$ to every function symbol of arity r (≥ 0). An *interpretation* I *based on* the pre-interpretation J, assigns an r-ary relation, that is, a subset of D^r, to every predicate symbol of arity r (≥ 0).

Given an interpretation I based on a pre-interpretation J, a variable assignment σ which assigns an element of D to every variable, and given an atom $p(t_1, \ldots, t_r)$, we stipulate that: $I, \sigma \models p(t_1, \ldots, t_r)$ holds iff the r-tuple $\langle v_1, \ldots, v_r \rangle$ belongs to the relation assigned to the predicate symbol p by I, where $\langle v_1, \ldots, v_r \rangle$ is the r-tuple of elements of D assigned to $\langle t_1, \ldots, t_r \rangle$, according to J and σ.

Analogously to the case of the Herbrand interpretations for definite programs, we identify an interpretation I based on a pre-interpretation J, with the set of expressions of the form $p(v_1, \ldots, v_r)$, where: (i) p is a predicate symbol in the program P, and (ii) $\langle v_1, \ldots, v_r \rangle$ is an r-tuple of elements of D which belongs to the relation assigned to p by I.

Definition 16. Given a normal program P and a pre-interpretation J, we define T_P^J to be a function which maps an interpretation I based on the pre-interpretation J into a new interpretation $T_P^J(I)$ also based on J, as follows:

$$T_P^J(I) = \{A\sigma \mid A \leftarrow L_1, \ldots, L_n \text{ in } P \text{ and } I, \sigma \models L_1 \wedge \ldots \wedge L_n \text{ for some variable assignment } \sigma\}$$

To understand this definition the reader should recall that:
(i) if A is the atom $p(t_1, \ldots, t_r)$ then $A\sigma$ is the atom $p(v_1, \ldots, v_r)$, where $\langle v_1, \ldots, v_r \rangle$ is the r-tuple of elements of D assigned to $\langle t_1, \ldots, t_r \rangle$, according to J and σ, and
(ii) given an atom A, we have that $I, \sigma \models \neg A$ iff it is not the case that $I, \sigma \models A$.

The pre-interpretation J is said to be an *Herbrand pre-interpretation* iff
(i) we take the domain D to be the Herbrand Universe of a language (which we assume to have at least one constant) which includes the symbols occurring in the program P, and
(ii) we assign to every function symbol f of arity r (≥ 0), the function which maps the terms $t_1, \ldots, t_r$ into the term $f(t_1, \ldots, t_r)$.

An interpretation based on an Herbrand pre-interpretation is said to be an *Herbrand interpretation*. We will write T_P, instead of T_P^J, iff J is an Herbrand pre-interpretation.

Let us now state some facts about the T_P operator and the completion in the case of normal programs.

Proposition 28. For a normal program P which is not a definite program, the T_P operator may be not monotonic.

Proof. Take, for instance, the program $\{p \leftarrow \neg p\}$. We have: $T_P(\emptyset) = \{p\}$, and $T_p(\{p\}) = \emptyset$. Recall that if P is a definite program then T_P is monotonic.

Proposition 29. For any normal program P, $comp(P) \models P$ [8, Proposition 14.1].

Proposition 30. Every normal program P has an Herbrand model. This model is constructed by taking the ground atoms which are the instances of the heads of the clauses of P. A normal program may *not* have the least Herbrand model.

Proof. Indeed, for instance, the program $\{p \leftarrow \neg q\}$ has two Herbrand models: $\{p\}$ and $\{q\}$, and none of them is included in the other.

Proposition 31. Let us consider a normal program P and let us assume that P has the least Herbrand model $M(P)$. Then it may be the case that $M(P) \models comp(P)$ does not hold.

Proof. Indeed, for instance, the program $P = \{p \leftarrow \neg p\}$ has $\{p\}$ as its least Herbrand model. However, $comp(P)$ implies that $p \leftrightarrow \neg p$. Thus, $comp(P)$ is inconsistent and it has no models.

Proposition 32. Let us consider a normal program P and an interpretation I based on a pre-interpretation J. (i) Then I is a model of P iff $T_P^J(I) \subseteq I$.
(ii) Let us suppose that in the interpretation I the predicate symbol $=$ is interpreted as the identity relation on the domain of I. If $I \models \mathrm{CET}$ then ($I \models comp(P)$ iff $T_P^J(I) = I$) [8, Proposition 14.3].

Let us consider a normal program P. We have that if I is an Herbrand interpretation where the predicate symbol $=$ is interpreted as the identity relation on the Herbrand Universe, then $I \models \mathrm{CET}$. Thus, by Proposition 32 above, we have the following proposition.

Proposition 33. Let P be a normal program. For any Herbrand interpretation I, $I \models comp(P)$ iff $T_P(I) = I$ [1, Theorem 5.18].

Note 8. The notion of pre-interpretation is required only for the above Proposition 32.

Since the completion of normal programs may be inconsistent, many properties of the completion of definite programs presented in Section 10, no longer hold in the case of normal programs. In particular, for every normal program P and atom A, it is not the case that: $comp(P) \models \neg A$ iff there exists a computation rule R and a finitely failed SLDNF tree for program P, goal $\leftarrow A$, and R (see Theorem 29, page 89). Indeed, take P to be $\{p \leftarrow \neg p,\ \ a \leftarrow\}$ and A to be a. We have that $comp(P)$ is inconsistent, and thus, $comp(P) \models \neg A$, while there is no finitely failed SLDNF tree for program P, goal $\leftarrow a$, and any computation rule.

12 Programs

Now we make a further extension to the class of programs and goals we consider. We allow arbitrary first order formulas (possibly not closed) to occur in bodies of clauses and in goals. By this extension we get the so called *programs* and *goals* (without further qualifications).

Programs are conjunctions of so called *statements* [8]. Statements and goals are formulas of the form: $A \leftarrow W$ and $\leftarrow W$, respectively, where A is an atom and W is any first order logic formula. Given the statement $A \leftarrow W$, we say that A is its head and W is its body. As for clauses, statements and goals are assumed to be universally quantified at the front.

We can also introduce the notion of the completion of a program as we did for definite logic programs. Indeed, the form of the body of a clause does not play any role in the definition of the completion we have given in Section 10.

For programs and goals we can introduce the notions of SLDNF derivation, SLDNF refutation, SLDNF tree, computed answer substitution, and finitely failed SLDNF tree as follows. Given a program P, a goal G, and a safe computation rule R, those notions are assumed to be the ones for a normal program, say P', a normal goal, say G', and the safe computation rule R, where P' and G' are constructed from P and G by using the so called Lloyd-Topor transformation, which we now present.

The Lloyd-Topor transformation takes a conjunction of statements and produces a conjunction of normal clauses. What is preserved by that transformation is indicated in the Facts 1 and 2 below.

As usual, we will feel free to denote conjunctions as sets.

In order to present the Lloyd-Topor transformation we introduce the following notation.

We write $\underline{\mathrm{C}}[\gamma]$ to denote a first order formula where the subformula γ occurs as an *outermost conjunct*, that is, $\underline{\mathrm{C}}[\gamma] = \rho_1 \wedge \ldots \wedge \rho_r \wedge \gamma \wedge \sigma_1 \wedge \ldots \wedge \sigma_s$ for some first order formulas $\rho_1, \ldots, \rho_r, \sigma_1, \ldots, \sigma_s$, with $r \geq 0$ and $s \geq 0$. We will say that the formula $\underline{\mathrm{C}}[\gamma]$ is transformed into the formula $\underline{\mathrm{C}}[\delta]$ if $\underline{\mathrm{C}}[\delta]$ is obtained from $\underline{\mathrm{C}}[\gamma]$ by replacing the conjunct γ by the new conjunct δ.

Lloyd-Topor Transformation

Input: a conjunction P of statements.

Output: a conjunction P' of normal clauses such that Facts 1 and 2 below hold.

(Rule A) Eliminate from the body of every statement all occurrences of logical constants, connectives, and quantifiers other than *true*, $\neg$, $\wedge$, and $\exists$. For every statement st rename the bound variables occurring in $\forall(st)$ so that all of them are distinct. For instance, $p(x,y) \leftarrow ((\exists w\, q(w,y)) \rightarrow \forall w\, r(w))$ is transformed into

$p(x,y) \leftarrow \neg((\exists w\, q(w,y)) \wedge (\exists z\, \neg r(z)))$.

(Rule B) Apply *as long as possible* the following rules:

(Rule B.1) $A \leftarrow \underline{\mathrm{C}}[\neg \mathit{true}]$ is deleted

(Rule B.2) $A \leftarrow \underline{\mathrm{C}}[\neg\neg\varphi]$ is transformed into $A \leftarrow \underline{\mathrm{C}}[\varphi]$

(Rule B.3) $A \leftarrow \underline{\mathrm{C}}[\neg(\varphi \wedge \psi)]$ is transformed into $A \leftarrow \underline{\mathrm{C}}[\neg \mathit{newp}(y_1, \ldots, y_k)] \wedge \mathit{newp}(y_1, \ldots, y_k) \leftarrow \varphi \wedge \psi$

where *newp* is a new predicate symbol and $\{y_1, \ldots y_k\} = \mathit{freevars}(\varphi \wedge \psi)$. We assume that $\varphi \neq \mathit{true}$ and $\psi \neq \mathit{true}$.

(Rule B.4) $A \leftarrow \underline{\mathrm{C}}[\neg\exists x_1 \ldots x_n\, \varphi]$ is transformed into $A \leftarrow \underline{\mathrm{C}}[\neg \mathit{newp}(y_1, \ldots, y_k)] \wedge \mathit{newp}(y_1, \ldots, y_k) \leftarrow \varphi$

where *newp* is a new predicate symbol and $\{y_1, \ldots y_k\} = \mathit{freevars}(\exists x_1 \ldots x_n\, \varphi)$.

(Rule B.5) $A \leftarrow \underline{\mathrm{C}}[\exists x_1 \ldots x_n\, \varphi]$ is transformed into $A \leftarrow \underline{\mathrm{C}}[\varphi]$.

During the Lloyd-Topor transformation, in order to simplify programs, we may use tautologies. Thus, for instance, we may replace a formula of the form $\varphi \wedge \varphi$ by φ and vice versa.

Example 11. Let us apply the Lloyd-Topor transformation starting from the statement:

$r \leftarrow \forall x\, (q(x, v, w) \rightarrow (\exists v\, p(x, v)))$

By renaming the bound variables and eliminating $\rightarrow$ (Rule A), we get:

$r \leftarrow \forall x\, (\neg q(x, v, w) \vee \exists y\, p(x, y))$

By eliminating $\forall x$ (Rule A), we get:

$r \leftarrow \neg\exists x\, \neg(\neg q(x, v, w) \vee \exists y\, p(x, y))$

By eliminating $\neg\exists x$ (Rule B.4), we get:

$\{r \leftarrow \neg new1(v, w),\ \ new1(v, w) \leftarrow \neg(\neg q(x, v, w) \vee \exists y\, p(x, y))\}$

By pushing $\neg$ inward (Rule B.2), we get:

$\{r \leftarrow \neg new1(y, w),\ \ new1(y, w) \leftarrow q(x, v, w) \wedge \neg\exists y\, p(x, y)\}$

By eliminating $\neg\exists y$ (Rule B.4), we get:

$\{r \leftarrow \neg new1(y, w),\ \ new1(v, w) \leftarrow q(x, v, w) \wedge \neg new2(x),\ \ new2(x) \leftarrow p(x, y)\}$

which is a set of three normal clauses. □

Now given a program P and a goal G: $\leftarrow W$, where W is a first order formula, we may construct the new program P^G which is $P \wedge (ans(x_1, \ldots, x_n) \leftarrow W)$, where *ans* is a new predicate symbol and $x_1, \ldots, x_n$ are the free variables occurring in W. Then we may apply the Lloyd-Topor transformation to the program P^G, thereby deriving a normal program P'. Let G' be the normal goal $\leftarrow ans(x_1, \ldots, x_n)$ (actually, G' is a definite goal).

The normal program P' and the normal goal G' are said to be *a normal form* of program P and goal G.

Note that we said '*a normal form*' and not '*the normal form*', because the outcome of the Lloyd-Topor transformation is not uniquely determined if we do not specify the tautologies which we apply. However, the results we will state below do not depend on the particular normal form of P and G we consider.

Let us consider a program P and let P' be a normal form of P. We have the following facts.

Fact 1. If the formula φ contains only predicate symbols occurring in P and $comp(P') \models \varphi$ then $comp(P) \models \varphi$ [8, Lemma 18.4 on page 115]. (This fact is needed for the soundness result of Theorem 32 below.)

Fact 2. We have that $comp(P') \models comp(P)$ [8, Lemma 18.8 on page 118]. (This fact is needed for the completeness result of Theorem 32 below.)

Note that every rule of the Lloyd-Topor transformation is based on a logical equivalence, except for Rules B.3 and B.4. However, for Rule B.3 we have that:

(B.3.i)	$A \leftarrow \underline{C}[\neg newp(y_1, \ldots, y_k)] \wedge$ $newp(y_1, \ldots, y_k) \leftrightarrow (\varphi \wedge \psi)$	implies	$A \leftarrow \underline{C}[\neg(\varphi \wedge \psi)]$
(B.3.ii)	$A \leftarrow \underline{C}[\neg newp(y_1, \ldots, y_k)] \wedge$ $newp(y_1, \ldots, y_k) \leftarrow (\varphi \wedge \psi)$	does not imply	$A \leftarrow \underline{C}[\neg(\varphi \wedge \psi)]$

(For (B.3.ii) take, for instance, $\varphi \wedge \psi$ to be *false* and $\underline{C}[\ldots]$ to be the empty context.) Analogously for Rule B.4, we have that:

(B.4.i)	$A \leftarrow \underline{C}[\neg newp(y_1, \ldots, y_k)] \wedge$ $newp(y_1, \ldots, y_k) \leftrightarrow \varphi$	implies	$A \leftarrow \underline{C}[\neg \exists x_1 \ldots x_n \varphi]$
(B.4.ii)	$A \leftarrow \underline{C}[\neg newp(y_1, \ldots, y_k)] \wedge$ $newp(y_1, \ldots, y_k) \leftarrow \varphi$	does not imply	$A \leftarrow \underline{C}[\neg \exists x_1 \ldots x_n \varphi]$

These facts easily follow from the hypotheses that: (i) every statement is implicitly universally quantified at the front, (ii) variables have been renamed as specified in Rule A, (iii) $\{y_1, \ldots y_m\} = freevars(\exists x_1 \ldots x_n \varphi)$, and (iv) *newp* is a new predicate symbol.

Let us now introduce the following concepts. Let us consider a program P, a goal G: $\leftarrow W$, and a computation rule R. Let P' and G' be a normal form of P and G.

(i) We define a c.a.s. of an SLDNF refutation for P, G and R to be a c.a.s. of an SLDNF refutation for P', G', and R.

(ii) We define a finitely failed SLDNF tree for P, G, and R is a finitely failed SLDNF tree for P', G', and R.

(iii) As for normal programs, a program P is said to be *hierarchical* iff we can assign to every predicate p of P a non-negative integer, called the *level* of p, such that in every statement C of P the level of the predicate in the head of C is *strictly greater* than the level of every predicate in the body of C. Note that if P is hierarchical then any normal form P' of P is hierarchical.

(iv) We say that $[P, G]$ is allowed iff $[P', G']$ is allowed.

Theorem 32. [Soundness and Relative Completeness of SLDNF Resolution for Programs] Let us consider a program P, a goal G: $\leftarrow W$, and a safe computation rule R.

Soundness of SLDNF resolution for programs: If ϑ (not necessarily a grounding substitution for W) is the c.a.s. of an SLDNF refutation for P, G and R, then $comp(P) \models \forall(W\vartheta)$. If there exists a finitely failed SLDNF tree for P, G and R then $comp(P) \models \forall(\neg W)$ [8, Theorem 18.6 and 18.7 on page 117].

Completeness of the SLDNF resolution for hierarchical programs: If P is hierarchical, $[P, G]$ is allowed, $comp(P) \models W\vartheta$, and ϑ is grounding substitution for W, then we have that ϑ is the c.a.s. of an SLDNF refutation for P, G, and R. (Actually, for every safe computation rule R, the SLDNF tree for P, G, and R is finite, and ϑ is a c.a.s. of *one* of its SLDNF refutations [8, Theorem 18.9 on page 119].) □

13 Appendix A: Truth in Mathematical Structures

Here we discuss a few basic issues about the notion of truth in logic. We begin by asking ourselves the following question: Is 1+1 equal to 2 (as in ordinary arithmetic) or equal to 10 (as in binary arithmetic)?

To distinguish between these two answers, and in general, to establish what is a true statement in a mathematical structure, say M, we may characterize that structure by a set of axioms and inference rules, collectively called $AR(M)$. Then we say that φ is true in a structure M iff φ can be deduced using the axioms and the inference rules in $AR(M)$.

In order to illustrate these ideas let us now consider the following example referring to boolean algebras.

A boolean algebra is a non-empty set B together with two binary operations $\vee$ and $\wedge$, one unary operation $\neg$, and two nullary operations 0 and 1. Below we will present the set *Bool* of axioms for boolean algebras. In these axioms it is assumed that all free variables are implicitly universally quantified at the front.

1.a	$x \wedge y = y \wedge x$	1.b	$x \vee y = y \vee x$
2.a	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	2.b	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
3.a	$x \wedge (\neg x) = 0$	3.b	$x \vee (\neg x) = 1$
4.a	$x \vee 0 = x$	4.b	$x \wedge 1 = x$
5.	$0 \neq 1$		

We say that a property φ is true in any boolean algebra iff φ can be deduced from *Bool* by using the replacement of *equals by equals* as the only rule of inference. This rule allows us to deduce $E[a] = E[b]$ from $a = b$, for any given expression context $E[\_]$ and expressions a and b. (As usual, $E[a]$ denotes the expression context $E[\_]$ with the subexpression a, instead of the missing expression.)

Here is the proof that in every boolean algebra $\langle B, \vee, \wedge, \neg, 0, 1\rangle$ it is true that $\forall x \in B,\ 1 = x \vee 1$. For brevity we omit to write '$\forall x \in B$' and we write within curly brackets the axioms used for the deduction.

$$1 = \{3.\text{b}\} = x \vee \neg x = \{4.\text{b}\} = x \vee (\neg x \wedge 1) = \{2.\text{b}\} = (x \vee \neg x) \wedge (x \vee 1) =$$
$$= \{3.\text{b}\} = 1 \wedge (x \vee 1) = \{1.\text{a}\} = (x \vee 1) \wedge 1 = \{4.\text{b}\} = x \vee 1.$$

Note, however, that it is not always the case that given a structure M, we can indeed construct $AR(M)$ so that every statement φ which is true in M can be derived from $AR(M)$. Here we cannot provide more evidence for this fact (see the Gödel-Rosser Incompleteness Theorem 9 on page 41). Let us simply recall that the mathematical structures are, indeed, very rich structures and not all their properties can be captured by deductive systems based on axioms and inference rules. Obviously, to formalize this claim, we should give a formal definition of what we mean by axioms and inference rules. But we do not do that here.

We may follow the approach of defining in an independent way the notions of: (1) what we mean for a statement φ *to be true* in a structure M, denoted $M \models \varphi$, and (2) what we mean for a statement *to be derivable* via a deductive system associated with M. Then, we may study the relationship between these two notions.

Indeed, when presenting the first order predicate calculus we have followed this approach and, in particular, we have provided the notion of truth in that calculus, that is, its semantics (see Section 4), by using a technique due to Tarski. Note also that, since every statement of the first order predicate calculus is a formula in a given formal language, we began by presenting that formal language (see Section 1).

14 Appendix B: Remarks on Resolution

Here we present some results concerning the application of resolution steps to: (i) conjunctions of clauses, (ii) definite programs, and (iii) normal programs.

14.1 Resolution for Conjunction of Clauses

Let us consider any conjunction A of clauses (not necessarily Horn clauses) and let us assume that from A we get the new conjunction B of clauses by performing a resolution step. We have that: $\forall(A)$ is unsatisfiable iff $\forall(B)$ is unsatisfiable. (Recall that $\forall(\varphi)$ denotes the universal closure of the formula φ.)

Since $\forall(A)$ is a closed formula, we have that $\forall(A)$ is unsatisfiable iff $\forall(A)$ is false in all interpretations. Thus, for all interpretations I, it is not the case that $I \models \forall(A)$, that is, for all I, $I \models \neg\forall(A)$. As a consequence, $\models \neg\forall(A)$ and, by Theorem 7 on page 35, we have that $\vdash \neg\forall(A)$, that is, $\forall(A) \vdash \varphi \wedge \neg\varphi$, for some formula φ. Thus, we have that:

(α) $\forall(A)$ is inconsistent iff $\forall(B)$ is inconsistent.

14.2 SLD Resolution for Definite Programs

Let us consider: (i) a definite program P, (ii) a definite goal $\leftarrow Conj_0$, where $Conj_0$ is the conjunction $A_1 \wedge \ldots \wedge A_n$ of possibly non-ground atoms, and (iii) a computation rule R. Let us construct a finite upper portion of the SLD tree starting from the goal $\leftarrow Conj_0$ by performing some SLD resolution steps according to the computation rule R (see Figure 9).

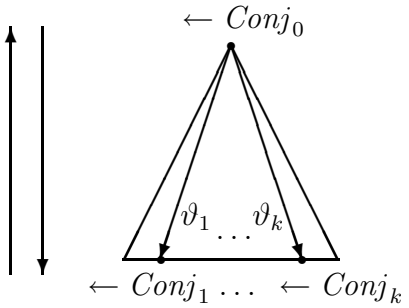

Fig. 9. An upper portion of an SLD tree for program P, goal $\leftarrow Conj_0$, and a selection rule. The up-arrow and the down-arrow depict the *iff* relation of the formula (β^*) (see page 105).

Suppose that the k leaves (≥ 0) of this upper portion of the SLD tree have the following goals: $\leftarrow Conj_1$, ..., $\leftarrow Conj_k$. For $i = 1, \ldots, k$, the substitution ϑ_i denotes the composition of the mgu's from the root to the leaf with goal $\leftarrow Conj_i$.

As usual, we assume that the clauses of P are implicitly universally quantified at the front. Thus, P is the same as $\forall(P)$. We have that:

$$
\begin{array}{lll}
(\beta) & P \wedge \forall(\neg Conj_0) \text{ is inconsistent} & (\beta.1) \\
\text{iff} & P \wedge \forall(\neg Conj_1\vartheta_1) \wedge \ldots \wedge \forall(\neg Conj_k\vartheta_k) \text{ is inconsistent} & (\beta.2) \\
\text{iff} & P \wedge \forall(\neg Conj_1\vartheta_1) \text{ is inconsistent or } \ldots & \\
 & \qquad \ldots \text{ or } P \wedge \forall(\neg Conj_k\vartheta_k) \text{ is inconsistent.} & (\beta.3)
\end{array}
$$

We will not provide a proof of (β), but for the reader's convenience we make the following remarks.

(i) The goal $\neg Conj_0$ and the goals $\neg Conj_i\vartheta_i$'s, for $i = 1, \ldots, k$, are universally quantified at the front because they are Horn clauses and, as usual, Horn clauses are universally quantified at the front.

(ii) $(\beta.1)$ iff $(\beta.2)$ holds because of Theorems 17 on page 49 and 24 on page 68 (note also that in order to show inconsistency, we need to explore all possible resolution steps).

(iii) $(\beta.2)$ iff $(\beta.3)$ holds because P is a definite program, the goals $\neg Conj_i\vartheta_i$'s for $i = 1, \ldots, k$, are definite goals.

Thus, inconsistency cannot arise from the conjunctions of some definite goals, because they have negative literals only (once written in terms of $\neg$ and $\vee$ only). Instead, inconsistency can arise from the conjunction of a goal with clauses of P. We will see in Section 14.3 that this property does not hold for normal programs and normal goals.

Formula (β) can also be written as follows:

$$
\begin{array}{lll}
(\beta^*) & P \vdash \exists(Conj_0) & \\
\text{iff} & P \vdash \exists(Conj_1\vartheta_1) \vee \ldots \vee \exists(Conj_k\vartheta_k) & \\
\text{iff} & P \vdash \exists(Conj_1\vartheta_1) \text{ or } \ldots \text{ or } P \vdash \exists(Conj_k\vartheta_k) &
\end{array}
$$

because P is a closed formula and thus, $P \wedge \forall(\neg\varphi)$ is inconsistent iff $P \vdash \exists(\varphi)$ holds. Since $\forall$ distributes over $\wedge$ and $\exists$ distributes over $\vee$, we have that in (β) the formula $\forall(\neg Conj_1\vartheta_1) \wedge \ldots \wedge \forall(\neg Conj_k\vartheta_k)$ can also be written as $\forall(\neg Conj_1\vartheta_1 \wedge \ldots \wedge \neg Conj_k\vartheta_k)$, and in (β^*) the formula $\exists(Conj_1\vartheta_1) \vee \ldots \vee \exists(Conj_k\vartheta_k)$ can also be written as $\exists(Conj_1\vartheta_1 \vee \ldots \vee Conj_k\vartheta_k)$.

The above two formulas (β) and (β^*) also hold if we replace P by $comp(P)$. Indeed, recall that, for every definite program P and every definite goal $\leftarrow Conj$, where $Conj$ is the conjunction $A_1 \wedge \ldots \wedge A_n$ of possibly non-ground atoms, we have that:

$P \vdash \forall(Conj\,\vartheta)$ iff $comp(P) \vdash \forall(Conj\,\vartheta)$

where ϑ is any substitution for the variables in *Conj* (see Theorem 14.6 in [8, page 82] and replace $\models$ with $\vdash$ because of Theorem 7 on page 35).

14.3 SLDNF Resolution for Normal Programs

Let us consider a normal program P, a normal goal $\leftarrow Conj_0$, and a safe computation rule R.

Let us construct a finite upper portion of the SLDNF tree starting from the goal $\leftarrow Conj_0$ by performing some SLDNF resolution steps according to the safe computation rule R. Suppose that the k (≥ 0) leaves of this upper portion of the SLDNF tree have goals: $\leftarrow Conj_1$,..., $\leftarrow Conj_k$. For $i = 1, \ldots, k$, the substitution ϑ_i denotes the composition of the mgu's from the root to the leaf with goal $\leftarrow Conj_i$.

Analogously to the case of definite programs, we have that:

$$
\begin{array}{lll}
(\gamma) & & comp(P) \wedge \forall(\neg Conj_0) \text{ is inconsistent} \\
& \text{iff} & comp(P) \wedge \forall(\neg Conj_1\vartheta_1) \wedge \ldots \wedge \forall(\neg Conj_k\vartheta_k) \text{ is inconsistent} \\
& \text{is implied by} & comp(P) \wedge \forall(\neg Conj_1\vartheta_1) \text{ is inconsistent or } \ldots \\
& & \quad \ldots \text{ or } comp(P) \wedge \forall(\neg Conj_k\vartheta_k) \text{ is inconsistent}
\end{array}
$$

which can also be written as follows (see Figure 10):

$$
\begin{array}{lll}
(\gamma^*) & & comp(P) \vdash \exists(Conj_0) \\
& \text{iff} & comp(P) \vdash \exists(Conj_1\vartheta_1) \vee \ldots \vee \exists(Conj_k\vartheta_k) \\
& \text{is implied by} & comp(P) \vdash \exists(Conj_1\vartheta_1) \text{ or } \ldots \text{ or } comp(P) \vdash \exists(Conj_k\vartheta_k).
\end{array}
$$

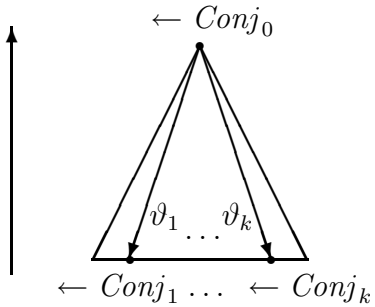

Fig. 10. An upper portion of an SLD tree for program P, goal $\leftarrow Conj_0$, and a selection rule. The up-arrow depicts the *if* relation of the formula (γ^*) (see page 106).

Note that in (γ) and (γ^*) above, we cannot replace 'is implied by' by 'iff '.

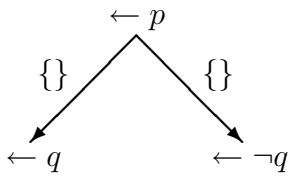

Fig. 11. An upper portion of the SLDNF tree for program $P = \{p \leftarrow q,\ p \leftarrow \neg q,\ q \leftarrow q\}$, goal $\leftarrow p$, and the leftmost selection rule. $\{\}$ denotes the identity substitution.

Indeed, let us consider the normal program $P = \{p \leftarrow q,\ p \leftarrow \neg q,\ q \leftarrow q\}$ and the goal $\leftarrow p$. By performing two SLDNF steps we get the upper portion of the SLDNF tree shown in Figure 11 on page 107.

Now, we have that: $comp(P) = p \wedge \mathrm{CET}$, $Conj_1 = q$, $Conj_2 = \neg q$, and $\vartheta_1 = \vartheta_2 = \{\}$. Note that, since $q \leftrightarrow q$ is true, we have discarded $q \leftrightarrow q$ from the expression of $comp(P)$.

We also have that:

$$comp(P) \vdash \exists(Conj_1) \vee \exists(Conj_2)$$

does not imply $comp(P) \vdash \exists(Conj_1)$ or $comp(P) \vdash \exists(Conj_2)$

because

$$p \wedge \mathrm{CET} \vdash \mathit{true}$$

does not imply $p \wedge \mathrm{CET} \vdash q$ or $p \wedge \mathrm{CET} \vdash \neg q$.

□

Index

ϵ_0 induction principle, 43
$\exists_1$ quantifier, 37
$\mathcal{N}$: standard model, 42
ω-consistency, 45
ω-incompleteness, 45

admissible clause, 93
allowed [program, goal] pair, 93
allowed clause, 93
allowed goal, 93
atom, atomic formula, 9
axiom, axiom schema, 13
axiomatic first order theory, 18

binding, 11
body of a clause, 56
boolean algebra, 102
boolean algebra: axioms, 102
boundvars, 11

c.a.s., computed answer substitution, 92
Clark completion, 86
Clark equality theory, CET, 87
clausal form, 10
clause, 10, 55
closed term, 11
complete lattice, 60
complete theory, 18
completeness of SLD resolution for definite programs, 68
completeness of SLDNF resolution for hierachical programs, 101
completeness of SLDNF resolution for hierarchical normal programs, 94
completion of a definite program, 86
computation rule, 64
computed answer substitution, c.a.s., 66, 92, 100
conclusion, 14
conjunction, 9
connective, 9
consistency, 19
context, 10
continuous function, 61
Cut rule, 23

decidable first order theory, 18
decidable theory, 54
deduction theorem (for Natural Deduction), 22, 23
deduction theorem (for the Classical Presentation), 16
definite clause, 56
definite goal, 56
definite program, 56
denumerable set, 34
dependence relation between formulas, 15
derivation, 14
descendant relation, 85
direct consequence, 14
disjunction, 9
distributivity, 26
domain of an interpretation, 30
duality, dual formula, 29

empty clause, 10, 56
empty goal, 56
enumerable set, 34
equivalence, 9
existential closure, 11
existential quantification, 9
expression, 10
expression of a first order language, 41
extension of a first order theory, 35

factoring step, 52
failed path in an SLD tree, 67
failed path in an SLDNF tree, 92
failure leaf in an SLD tree (failure), 67
failure leaf in an SLDNF tree (failure), 92
fair computation rule, 86
false, 9, 56
finite failure set, $FF(P)$, 85
finitely failed SLD tree, 67
finitely failed SLDNF tree, 92, 100
first order language, 9
first order predicate calculus, 14
first order predicate calculus with equality, 37
first order predicate calculus: classical presentation, 13
first order predicate calculus: natural deduction presentation, 19
first order predicate calculus: theory, 15
fixpoint, 60
floundering, 91
formula, 9
formula context, 10
formula: (logically) valid, 31
formula: closed, 11
formula: false in an interpretation, 31
formula: logical consequence, 32

formula: logical equivalence, 33
formula: open, 11
formula: satisfiable, 31
formula: true in a interpretation, 30
formula: unsatisfiable, 31
freevars, 11
function T_P, 63
function symbol, 9, 40

general goals, 90
general programs, 90
Generalization rule, 14
goals, 97
greatest lower bound, glb, 60
ground formula, 11
ground term, 11
group theory, 39
Gödel completeness theorem, 35
Gödel incompleteness theorem, 45
Gödel-Rosser incompleteness theorem, 41, 45

head of a clause, 56
head-body relation, 86
Henkin lemma, 35
Herbrand base, 49
Herbrand interpretation, 49, 96
Herbrand model, 49
Herbrand pre-interpretation, 96
Herbrand theorem, 49
Herbrand universe, 49
hierarchical program, 94, 100
Horn clause, 56

identity binding, 11
if-then, 25
if-then-else, 25
iff-formula of a definite program, 88
implication, 9
inconsistency, 19
inheritance relation, 86
instance, 11
interpretation, 30
interpretation based on a pre-interpretation, 95

Kleene lemma, 62
Knaster-Tarski lemma, 61

lattice, 60
least Herbrand model, 58
least upper bound, lub, 60
Lindenbaum lemma, 35
literal, 10, 55
Lloyd-Topor transformation, 98
logical axioms, 13
lower bound, 60

mathematical induction, 41
model, 31
model intersection property, 58
model of a first order theory, 34
model of a set of formulas, 34
Modus Ponens rule, 14
monotonic function, 60
most general unifier, mgu, 12
most general unifier: relevant, 12

negation, 9
negation-as-failure rule, 92
negative consequences of a definite logic program, 85
non-standard model of PA, 42
normal form of a program, 99
normal goals, 90
normal model of a first order predicate calculus with equality, 40
normal programs, 90

outermost conjunct, 98

PA theory, 40
paramodulation, 53
partial recursive functions, 44
Peano arithmetic, 40
postfixpoint, 60
pre-interpretation, 95
predicate symbol, 9
prefixpoint, 60
premise, 14
prenex conjunctive normal form, 27
prenex disjunctive normal form, 27
programs, 97
proof, 14
proof tree, 14
proper axiom, 14
propositional tautology, 32

r.e. first order theory, 18
r.e. set, 34
Raphael Robinson Arithmetic, 44
recursively enumerable first order theory, 18
recursively enumerable set, 34
renaming apart, 65
representable function, 43
resolution for conjunctions of clauses: properties, 104
resolution step, 50
resolution: 1-to-1, 52
resolution: binary, 52

reverse implication, 9
Robinson theorem, 49
RR theory, 44
rule C, 18
rule of inference, 13
rule: Generalization, 14
rule: Modus Ponens, 14

safe computation rule, 91
satisfaction relation, 30
satisfiability, preservation using Skolemization, 47
scope, 10
search rule, 64
selection rule, 64
semidecidable first order theory, 18
semidecidable theory, 54
sentence, 11
sequent, 20
Skolem theorem, 47
SLD derivation, 64
SLD refutation, 66
SLD resolution for definite programs: properties, 104
SLD resolution step for definite programs, 65
SLD tree, 66
SLDNF derivation, 91
SLDNF refutation, 91
SLDNF resolution for normal programs: properties, 106
SLDNF resolution step for normal programs, 91
SLDNF tree, 92
soundness of first order predicate calculus, 34
soundness of SLD resolution for definite programs, 68
soundness of SLDNF resolution for normal programs, 94
soundness of SLDNF resolution for programs, 101
standard model $\mathcal{N}$, 42
standard model of PA, 42
statements, 97
strongly representable function, 44
substitution, 11
Substitution rule, 15
substitution: composition, 11
substitution: domain, 11
substitution: grounding, 12
substitution: idempotent, 12
substitution: more general substitution, 12
substitution: range, 11
substitution: restricted to, 12
substitution: solved form, 12
success leaf in an SLD tree (success), 67
success leaf in an SLDNF tree (success), 92
success set, $SS(P)$, 69
successful path in an SLD tree, 67
successful path in an SLDNF tree, 92

Tarski theorem, 43
term, 9
term context, 10
term: free for a variable, 12
theorem, 15, 22
theory: first order theory, 15
true, 9
Turing completeness of definite logic programs, 69

undecidable formula, 19
unifiable terms (or atoms), 12
unification algorithm (Martelli-Montanari), 50
unification algorithm (Robinson), 51
unifier, 12
universal closure, 11
universal quantification, 9
upper bound, 60

variable, 9
variable assignment, 30
variable renaming, 12
variable: bound and free, 10
variable: bound and free occurrence, 10
variant, 56
vars, 11

witness theory, 35

References

1. K. R. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–576. Elsevier, 1990.
2. F. Black. A deductive question-answering system. In Marvin Minsky, editor, *Semantic Information Processing*, pages 354–402. The MIT Press, Cambridge, Massachusetts, USA, 1968.
3. C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press Inc., 1973.
4. G. Gentzen. Die Widerspruchfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936.
5. G. Gentzen. Neue Fassung des Widerspruchsfreiheitsbeweises fuer die reine Zahlentheorie. *Forschungen zur Logik und zur Grundlagen der exakten Wissenschaften. New series*, 4:19–44, 1938.
6. Intelligent Systems Laboratory. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 1995.
7. R. A. Kowalski. *Logic for Problem Solving*. North Holland, 1979.
8. J. W. Lloyd. *Foundations of Logic Programming*. Second Edition. Springer-Verlag, Berlin, 1987.
9. Z. Manna. *Mathematical Theory of Computation*. MacGraw-Hill, 1974.
10. E. Mendelson. *Introduction to Mathematical Logic*. Third Edition. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, California, Usa, Monterey, California, Usa, 1987.
11. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
12. H. Rasiowa and R. Sikorski. *The Mathematics of Metamathematics*. Third Edition. Tom 41. PWN, Polish Scientific Publishers, Warszawa, Poland, 1970.
13. J. R. Shoenfield. *Mathematical Logic*. Addison-Wesley Publishing Company, 1967.
14. A. Tarski, A. Mostowski, and R. Robinson. *Undecidable Theories*. North-Holland Publishing Company, 1953.