Order Number 9108165

Fully declarative programming with logic mathematical foundations

Plaza, Jan Andrzej, Ph.D.

City University of New York, 1990

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A

Fully Declarative Programming with Logic

Mathematical Foundations

by

Jan A. Plaza

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1990

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

31 July 1990
Date

Melvin Fitting
Chair of Examining Committee
Prof. Melvin Fitting

31 July, 1990
Date

Executive Officer
Prof. Thomas Wesselkamper

Prof. Michael Anshel

Prof. Howard Blair

Prof. Kenneth McAloon

Prof. Rohit Parikh

Supervisory Committee

The City University of New York

Abstract

FULLY DECLARATIVE PROGRAMMING WITH LOGIC

MATHEMATICAL FOUNDATIONS

by

Jan A. Plaza

Adviser: Professor Melvin Fitting

Great hope for programming correctness has been centered in logic programming because of its declarativeness, but the declarativeness depends on the technical condition of completeness:

> solutions computed by resolution should be exactly the logical consequences of an easily understandable completion of the program.

For resolutions allowing negation $\neg$, such as SLDNF-resolution or Prolog's resolution, this goal has not been yet achieved, even for propositional programs.

In this paper we consider programs involving $\neg, \forall, \exists$ and $=$. At the propositional level we prove completeness of SLDNF-resolution for the full class of programs with $\neg$

(including normal programs). This is done in several ways in classical, intuitionistic, intermediate and in modal logics. We use intuitively natural notions of constructive completions of programs. These results can be of importance for expert systems. At the first-order level, we formulate resolution systems extending SLD-resolution, but alternative to SLDNF. We prove their completeness for the full class of positive programs (involving $\forall, \exists, =$), and also for some classes of programs involving negative information. This guarantees declarativeness of logic programming for a wide variety of programs. Examples suggest that extending completeness (declarativeness) further, is not possible because of certain inherent properties of these wider classes of programs.

Several techniques of this paper, connected with an alterative notion of a Herbrand model, least fix-points of operators, and decidability of an equality theory that extends Clark's theory, may be applicable in other problems from the theory of logic programming.

KEYWORDS: logic programming, declarative programming, SLD-resolution, negation as failure, constructive negation, guards, soundness and completeness, non-classical logics, theories of equality, decidability.

Acknowledgemets

My warmest thanks to Professor Melvin Fitting, my mentor. I deeply appreciate, and will always value, everything I learned from him. Without him this work would not have come into being - thanks for inviting me to the United States to study with him at the City University of New York - thanks for the very special friendship.

Most heartfelt thanks also to Professor Helena Rasiowa at the Mathematical Institute of the Polish Academy of Sciences in Warsaw who first showed me the beauty and strength of algebraic methods in logic.

Many thanks to Professor Rohit Parikh and Professor Kenneth McAloon for invaluable seminars and lectures which deeply influenced my understanding of mathematical logic and theoretical computer science.

Finaly, very special thanks to my mother, Janina for unwavering support, always.

Contents

1 INTRODUCTION

1.1 The problem and its context

A major concern in programming is the *correctness* of program P with respect to intended specification $S; \alpha$. This can be formulated as

> $S \vdash \alpha(I, O)$ iff the computational mechanism, given program P and input I, computes output O.

(In the specification we distinguish two parts: a defining part S and a goal part α. For instance, for a program that finds the last element of a list, S contains definitions explaining what is meant by the "last element", and α is the formula saying that O is the last element in I.) However, proving correctness is a difficult task and it is never undertaken for big programs. As the result, all big programs contain mistakes and their actual behavior differs from what was intended. In this situation R. A. Kowalski and A. Colmerauer proposed creating languages of a very high level, for which proofs of correctness of programs become very simple or even obvious. Their revolutionary ideas underlying *logic programming* consist of the following:

1. Write specifications, not programs, using a fragment of the language of first-order classical logic, representing possible inputs, outputs and all the intermediate data by means of terms.
2. Take the specification itself as a program!
3. Use a resolution system as a computational mechanism. (Linear resolution may be efficient.) Substitutions calculated by the resolution will be considered as

output.

4. The (desired) correctness of a program with respect to the specification is automatic; programming turns into *declarative programming*:

   Procedural meaning of program P $\equiv$ declarative meaning of program P.

   More specifically:

   Given program P, for any formula $\alpha(I, x)$, resolution computes $x{=}O$ iff the specification P proves $\alpha(I, O)$.

To make this idea work one has to choose the resolution system and the fragment of first-order logic in such a way that the equivalence from (4.) becomes true. This is partly done in Prolog — the first logic programming language. Further theoretical work suggested that one should not insist the program and its specification be identical. K. L. Clark proposed that, as a specification for a program P, we may admit a certain completion of P, which can be obtained from P in an automatic way.

Example 1.1.1 The empty list can be represented by a constant *nil*. The list consisting of an element e followed by another list *rest* can be represented as a term $[e \mid rest]$. (We abbreviate *nil* as [], and $[e_1 \mid [e_2 \mid \ldots [e_n \mid nil] \ldots]]$ as $[e_1, e_2, \ldots, e_n]$.) One can write the following specification S for a program that finds last element of the input list.

$last([x], x)$

$last([x \mid rest], y) \equiv last(rest, y)$

This can be automatically turned into the following Prolog program P (we do not observe Prolog's exact syntax):

$last([x], x)$

$last([x \mid rest], y) \leftarrow last(rest, y)$

Given goal $last([1,5,3], z)$ Prolog computes the substitution $z = 3$. We also have $S \vdash last([1,5,3], z)(3/z)$. S is equivalent to Clark's completion $Comp(P)$.

Using the notion of completion the (desired) declarativeness equivalence from (4.) can be represented as:

Soundness and completeness:

Resolution with program P and a query A computes substitution θ iff *Completion*$(P) \vdash A\theta$.

So soundness and completeness are technical conditions guaranteeing the declarativeness of logic programming.

The first results of this sort, for SLD-resolution, definite programs P and definite queries A, were proved by K. L. Clark [1]:

Soundness:

If SLD-resolution with program P and a query A computes substitution θ then $Comp(P) \vdash A\theta$.

Completeness:

If $Comp(P) \vdash A\theta$ then there exists a substitution θ', more general than θ, such that SLD-resolution with program P and a query A computes θ'.

[1] Originally Clark used the *Comp* only in theorems for SLDNF-resolution, but not for SLD-resolution

Definite programs, for which these theorems hold, do not admit negations in bodies of clauses. Therefore a question arises:

> Can one incorporate negation $\neg$ into the syntax of logic programs while still keeping declarativeness (i.e. soundness and completeness) ?

A partial answer to this problem was provided by K. L. Clark who, with the same notion of completion, proved soundness for SLDNF-resolution, normal programs and normal goals:

> If SLDNF-resolution with program P and a query L computes θ then $Comp(P) \vdash L\theta$

Unfortunately several counter-examples are known, (both at the propositional and the first-order level), showing that the converse theorem (completeness) doesn't hold.

This paper is written to address the following:

Problem

Can one extend the syntax of logic programs (allowing $\neg, \forall, \exists, =, \neq$, etc.), and also extend SLD-resolution to handle such programs, still keeping declarativeness (i.e. soundness and completeness) ?

Our answer consists of several parts and in each part we intend to specify four components:

1. The fragment of the language of first-order logic used to write programs.

2. Resolution and additional computational mechanisms (handling $\neg$, $=$, etc.)

3. The completion of the program (possibly different from Clark's).

4. The logic used to make inferences in the intended completeness theorem.

1.2 Results and their relation to other work

In Chapter 2 we consider propositional programs. This class of programs is often used while writing expert systems. We define the notion of constructive completion of a program, and prove completeness of SLDNF-resolution in classical logic. Then we show other completeness results which use intuitionistic or intermediate logics, and modal logics. Results of this chapter hold for the full class of propositional programs with negation (including normal programs), and show that SLDNF-resolution is fully satisfactory at the propositional level — to achieve completeness it was enough to change Clark's completion to a weaker completion.

In Chapter 3 we define ω-Herbrand models — an alternative to conventional Herbrand models. We prove that the theory of equality determined by these models is decidable. Then we consider positive programs, i.e. ones consisting of statements $A \leftarrow B$, where B is a positive formula (involving $=, \forall, \exists, \wedge, \vee, \top, \bot$) and prove that operators on lattices of ω-Herbrand interpretations, associated with such programs, despite being not continuous, reach their least fix-points in ω steps. So ω-Herbrand models differ greatly from conventional Herbrand models. This and other results indicate that ω-Herbrand models overcome deficiencies of the conventional setting, and are strong and convenient technical tools in the theory of logic programming.

In Chapter 4 we consider programs of the first order. At this level SLDNF-resolution is not satisfactory, and if completeness is attempted another resolution system is needed. Approaching this problem, equipped with techniques mentioned earlier, we formulate SLPG-resolution, capable of handling so called positive programs with guards. Guards are built of inqualities between terms. We show that SLPG is an extension of SLD-resolution. We prove that SLPG is complete (in classical logic), for the full class of positive programs and also for another class in which some negative information (guards) is allowed in programs. Then by augmenting SLPG with mechanisms based on the idea of constructive negation, we define SLPGCN-resolution. This yields completeness for some classes of programs with negation. However the full class of normal programs is not covered, and we discuss an example suggesting that in this case, incompleteness is not the fault of resolution, but an inherent property of the class of normal programs. Independently of that, fully declarative logic programming can be achieved for classes of programs mentioned earlier.

Resolution systems specified in the paper can be efficiently implemented, and this issue is (very) briefly discussed in corresponding sections and in the Conclusion.

The problem of soundness and completeness for programs with extended syntax, has been studied from the early days of logic programming. In references we give an extensive selection of research papers related to this topic. A big part of that work is devoted to obtaining completeness, in classical logic and with Clark's notion of completion, for some restricted classes of programs and goals (allowed, admissible,

hierarchical, stratified, locally stratified, etc.) Other attempts, in which many-valued logics were used, resulted in soundness theorems. In yet another approach Gabbay recently proved completeness of propositional SLDNF-resolution using a complex notion of completion in the modal logic of arithmetical provability. In our considerations we obtain completeness theorems for unrestricted classes of programs, using simple and intuitive notions of completions.

1.3 Organization of the paper

In terminology and notation we follow [44]: *Foundations of Logic Programming* (Second extended edition) by J. W. Lloyd. Chapters 1-3 of this book can be also used as a handy reference for basic results in the domain.

In the paper, Chapter 2 is devoted to propositional programs and Chapters 3, 4 to first-order programs. Chapter 2 and the set consisting of 3, 4 are independent of each other (although some ideas of 4.4 originate in 2.2). Chapter 3 develops methodology used in Chapter 4; a reader interested in the theory underlying completeness results should read it, a reader interested only in descriptions of resolution systems may omit it and pass to Chapter 4. Proofs are given separately in Chapters 6-8.

2 PROPOSITIONAL SLDNF-RESOLUTION

In this chapter we consider propositional programs consisting of statements $A \leftarrow B$ where B is an arbitrary formula (with $\neg, \wedge, \vee$, etc.) and prove completeness of SLDNF-resolution for the full class of such programs. Completeness in classical logic could have not been achieved without a new notion of completion of a program, which uses the idea of constructive negation, and differs from Clark's completion. Other completeness results employ intuitionistic or intermediate logics and modal logics. We also propose a Gentzen-style system and conjecture its completeness. Results of this chapter can be of importance for expert systems, which are often written in propositional Prolog.

2.1 Basic definitions

In this section we set the terminology and notation which is used in the rest of this chapter. In general we assume the terminology, definitions and results from chapters 1-3 of [44]: *Foundations of Logic Programming* (second extended edition) by J.W. Lloyd, but some additions are needed.

First, we generalize the notion of programs admitting arbitrary formulas in bodies of statements. With this generalization, every propositional program may be transformed to Clark's form in which every propositional letter occurs as the head of exactly one statement. All the basic definitions and completeness results for definite/normal programs extend automatically to positive/arbitrary generalized programs.

Second, we define a set of notions related to Clark's completion.

Third, Lloyd defined the notion of finite failure set only for definite programs. According to our needs, we extend this notion to programs containing negation in clause bodies, defining *SLDNF-finite failure set.*

Fourth, we consider SLDNF-computations, i.e. sets of derivations computed by the backtracking mechanism, and define: $SLDNF(P, L) = YES, SLDNF(P, L) = NO$ and $SLDNF(P, L) = \uparrow$, which intuitively correspond to situations when an idealized Prolog answers YES, NO or loops. These definitions are used later in this chapter to establish several non-procedural characterizations of SLDNF-computations for propositional programs.

At the end of this section, we state some procedural properties which show internal symmetries of the notion of computation.

Now the technicalities.

Recall that by a *definite clause* we understand any formula $A \leftarrow B$ where A is an atom, other than $\top$ or $\bot$, and B is a conjunction of atoms. Recall that by a *normal clause* we understand any formula $A \leftarrow B$ where A is an atom, other than $\top$ or $\bot$, and B is a conjunction of literals. The next definition generalizes these notions.

Definition 2.1.1

By a *propositional* $\vee$*-definite formula* we understand any disjunction of conjunctions of propositional atoms. By a *propositional* $\vee$*-definite statement* we understand any formula $A \leftarrow B$ where A is a propositional atom, other than $\top$ or $\bot$, and B

is a propositional $\vee$-definite formula. By a *propositional $\vee$-definite program* we understand any finite set of propositional $\vee$-definite statements. By a *propositional $\vee$-definite goal* we understand any formula $\leftarrow B$ where B is a propositional $\vee$-definite formula.

By a *propositional positive formula* we understand any formula built from atoms, including $\top, \bot$, by means of $\wedge$ and $\vee$. By a *propositional positive statement* we understand any formula $A \leftarrow B$ where A is a propositional atom, other than $\top$ or $\bot$, and B is a propositional positive formula. By a *propositional positive program* we understand any finite set of propositional positive statements. By a *propositional positive goal* we understand any formula $\leftarrow B$ where B is a propositional positive formula.

By a *propositional $\vee$-normal formula* we understand any disjunction of conjunctions of propositional literals. By a *propositional $\vee$-normal statement* we understand any formula $A \leftarrow B$ where A is a propositional atom, other than $\top$ or $\bot$, and B is a propositional $\vee$-normal formula. By a *propositional $\vee$-normal program* we understand any finite set of propositional $\vee$-normal statements. By a *propositional $\vee$-normal goal* we understand any formula $\leftarrow B$ where B is a propositional $\vee$-normal formula.

By a *propositional statement* we understand any formula $A \leftarrow B$ where A is a propositional atom, other than $\top$ or $\bot$, and B is arbitrary propositional formula. By a *propositional program* we understand any finite set of propositional statements. By a *propositional goal* we understand any propositional formula $\leftarrow B$.

SLDNF-resolution was defined originally for normal programs and normal goals.

The generalization to (arbitrary) propositional programs and goals is immediate. Given a program P, transform body B of any statement $A \leftarrow B$ into disjunctive normal form $\bigvee_{i \in I} B_i$ and replace $A \leftarrow B$ by the collection of clauses $A \leftarrow B_i : i \in I$. Denote the normal program obtained in this way by P'. Given a goal $\leftarrow B'$, transform B' into disjunctive normal form $\bigvee_{j \in J} B'_j$ and consider the collection of normal goals $\leftarrow B'_j : j \in J$. We say that there exists an SLDNF-refutation (in the new sense) for $P \cup \{\leftarrow B\}$ if there exists $j_0 \in J$ and an SLDNF-refutation (in the old sense) for $P' \cup \{\leftarrow B'_{j_0}\}$. We say that there exists an SLDNF-finite failure tree (in the new sense) if for every $j \in J$ there exists an SLDNF-finite failure tree (in the old sense) for $P' \cup \{\leftarrow B'_j\}$. So starting from this point we assume that SLDNF-resolution deals with (arbitrary) propositional programs and goals. Similarly we assume that SLD-resolution can handle (arbitrary) propositional positive programs and goals. (It is possible to redefine SLD or SLDNF so that they handle positive programs or arbitrary programs, without turning them into definite or normal programs. This will be done in definitions of SLPG- and SLPGCN-resolutions in Chapter 4.)

Definition 2.1.2

1. Propositional program P is said to be in *Clark's form* if for every propositional letter p in its language, P contains exactly one statement defining p.

2. Propositional program P is said to be in *Clark's disjunctive form* if it is in Clark's form and every statement body is in disjunctive normal form.

3. We define *Clark's transformation* $(\cdot)^C$ which transforms a propositional program

P into propositional program P^C. Perform the following steps. For every propositional letter p in P list all the statements defining p:

$$p \leftarrow B_1, \ldots, p \leftarrow B_n$$

and form $p \leftarrow \bigvee_{i \leq n} B_i$. (If there are no statements defining p, this construction gives $p \leftarrow \bot$.) The program P^C consists of all statements obtained in this way.

4. We define *Clark's disjunctive transformation* $(\cdot)^{Cd}$ which transforms a propositional program P into propositional program P^{Cd}. The program P^{Cd} is obtained from P^C by turning bodies of the statements into disjunctive normal form.

5. *Clark's completion* $Comp(P)$ of the propositional program P is defined as

$$Comp(P) = \{(p \equiv B) \mid (p \leftarrow B) \in P^C\}.$$

Basic properties of these notions are given (without a proof) in the following proposition.

Proposition 2.1.3 Let P be a propositional program. Then:

1. P^C is in Clark's form; P^{Cd} is in Clark's disjunctive form.
2. P, P^C and P^{Cd} are equivalent in classical logic.
3. $Comp(P) = Comp(P^C)$.
4. $Comp(P)$ and $Comp(P^{Cd})$ are equivalent in classical logic.

Lloyd defines finite failure sets only for definite programs (p. 75). In the next definition we extend this notion to normal programs.

Definition 2.1.4 Let P be a propositional program. By the *SLDNF-finite failure*

set for P we understand the set of all propositional letters p in the language of the program, such that there exists a finitely failed SLDNF-tree for $P; \leftarrow p$.

For definite programs P, Lloyd's notion of SLD-finite failure set and notion of SLDNF-finite failure set above coincide.

One of the main problems in the theory of logic programming is that of giving a non-procedural characterization of existence/non-existence of the successful/failed SLDNF-derivations. The next definition introduces notation convenient for propositional resolution.

Definition 2.1.5 Let P be a propositional program and let $\leftarrow B$ be a propositional goal.

1. $SLDNF(P, B) = YES$ is read: "SLDNF-computation for $P \cup \{\leftarrow B\}$ returns answer YES" and means: there is an SLDNF-refutation for $P \cup \{\leftarrow B\}$.
2. $SLDNF(P, B) = NO$ is read: "SLDNF-computation for $P \cup \{\leftarrow B\}$ returns answer NO" and means: there is a finitely failed SLDNF-tree for $P \cup \{\leftarrow B\}$.
3. $SLDNF(P, B) = \uparrow$ is read: "SLDNF-computation for $P \cup \{\leftarrow B\}$ loops" and means: there is neither an SLDNF-refutation nor a finitely failed tree for $P \cup \{\leftarrow B\}$.

Notation $SLD(P, B) = YES$, $SLD(P, B) = NO$, $SLD(P, B) = \uparrow$ is defined in an analogous way.

Proposition 2.1.6 Let P be a propositional program and let p be a propositional letter in P. Then:

1. p belongs to the SLDNF-success set of P iff $SLDNF(P,p) = YES$
2. p belongs to the SLDNF-finite failure set of P iff $SLDNF(P,p) = NO$

The next two propositions show internal symmetries of the notion of SLDNF-computation.

Proposition 2.1.7 Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then exactly one of the following conditions holds:

$$SLDNF(P,B) = YES,\ \ SLDNF(P,B) = NO,\ \ SLDNF(P,B) = \uparrow.$$

Proposition 2.1.8 Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

$$SLDNF(P,B) = YES \iff SLDNF(P,\neg B) = NO$$

$$SLDNF(P,B) = NO \iff SLDNF(P,\neg B) = YES$$

$$SLDNF(P,B) = \uparrow \iff SLDNF(P,\neg B) = \uparrow$$

Notice also that $SLDNF(P,B) = SLDNF(P^C,B)$.

2.2 Completeness in classical logic

> "I learned to recognise the thorough and primitive duality of man; I saw that, of the two natures that contended in the field of my consciousnesness, even if I could rightly be said to be either, it was only because I was radically both; and from an early date, even before the course of my scientific discoveries had begun to suggest the most naked possibility of such a miraccle, I had learned to dwell with pleasure, as a beloved day-dream, on the thought of the separation of these elements. If each, I told myself, could but be housed in separate identities, life would be relieved of all that was unbearable ... It was the curse of mankind that these incongruous faggots were thus bound together – that in the agonised womb of consciousness, these polar twins should be continuously struggling, How, then, were they dissociated?"
>
> Robert Louis Stevenson
>
> *Strange case of Dr. Jekyll and Mr. Hyde*, 1885.

In this section we define the constructive completion $Comp(P^+)$ for a propositional program P and prove completeness of propositional SLDNF-resolution. $Comp(P^+)$ is different from Clark's completion $Comp(P)$ – it uses an extended language in which every propositional letter p has a counterpart $\overline{p}$ representing the negation of p in a positive

way. This idea originates from the Prolog programming method of constructive negation: instead of using a negative literal $\neg A$, define in a positive way a new predicate with the same properties as $\neg A$.

Definition 2.2.1 (Constructive transformation and completion)

Let P be a propositional program in language $\mathcal{L}$. Let $\overline{\mathcal{L}}$ be an extension of $\mathcal{L}$ obtained by adjoining for every propositional letter p, a new propositional letter $\overline{p}$.

1. *Constructive transformation* P^+ of program P, is defined as the propositional program in $\overline{\mathcal{L}}$ obtained in the following way. For every statement $A \leftarrow B$ in P^C construct $\neg A \leftarrow B'$, where B' is a disjunctive normal form of $\neg B$; denote the set of all formulas obtained in this way by P'. Program P^+ is obtained from $P^{Cd} \cup P'$ by replacing each negative literal $\neg p$ (in heads as well as in bodies of clauses) by $\overline{p}$.

2. *Constructive transformation* $\leftarrow B^+$ of a propositional goal $\leftarrow B$ is obtained by turning B into disjunctive normal form and replacing each negative literal $\neg p$ by $\overline{p}$.

3. *Constructive completion* of program P is defined as $Comp(P^+)$.

Example 2.2.2 Let P be the following program:

$p \leftarrow p$

$q \leftarrow \neg q$

$q \leftarrow p$

We produce the constructive completion $Comp(P^+)$. First we turn P into Clark's

disjunctive form P^{Cd}:

$p \leftarrow p$

$q \leftarrow \neg q \vee p$

The set P' used to form P^+ is:

$\neg p \leftarrow \neg p$

$\neg q \leftarrow q \wedge \neg p$

We obtain P^+ from $P^{Cd} \cup P'$ by replacing $\neg p$ by $\overline{p}$ and $\neg q$ by $\overline{q}$:

$p \leftarrow p$ $\qquad$ $\overline{p} \leftarrow \overline{p}$

$q \leftarrow \overline{q} \vee p$ $\qquad$ $\overline{q} \leftarrow q \wedge \overline{p}$

Now $Comp(P^+)$ is obtained by changing implications to equivalences:

$p \equiv p$ $\qquad$ $\overline{p} \equiv \overline{p}$

$q \equiv \overline{q} \vee p$ $\qquad$ $\overline{q} \equiv q \wedge \overline{p}$

Notice that the constructive transformation P^+ is a positive program, and that the constructive completion $Comp(P^+)$ is always consistent in classical logic. This feature distinguishes it from Clark's completion $Comp(P)$ which may be inconsistent. As the following theorem shows, the constructive completion fully characterizes the behavior of propositional SLDNF-resolution.

Theorem 2.2.3 (Soundness and completeness)

Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLDNF(P, B) = YES$ iff $Comp(P^+) \vdash B^+$
2. $SLDNF(P, B) = NO$ iff $Comp(P^+) \vdash (\neg B)^+$
3. $SLDNF(P, B) = \uparrow$ iff $Comp(P^+) \nvdash B^+$ and $Comp(P^+) \nvdash (\neg B)^+$

We could use just P^+ instead of $Comp(P^+)$ in this theorem, but using $Comp(P^+)$ seems to be closer to the way programmers think.

Example 2.2.4 Continuing Example 2.2.2 notice that for that program, $SLDNF(P,p) = \uparrow$. At the same time p follows from Clark's completion: $Comp(P) = \{p \equiv p, q \equiv (\neg q \vee p)\} \vdash p$ so this example shows that SLDNF-resolution is not complete with respect to $Comp(P)$. Let us see how the constructive completion $Comp(P^+)$ behaves in this situation. As stated before $Comp(P^+)$ is:

$$p \equiv p \qquad \overline{p} \equiv \overline{p}$$

$$q \equiv \overline{q} \vee p \qquad \overline{q} \equiv q \wedge \overline{p}$$

One can see that $Comp(P^+) \nvdash p$ and $Comp(P^+) \nvdash \overline{p}$ so according to the theorem above $SLDNF(P,p) = \uparrow$.

From Theorem 2.2.3 , by the completeness of SLD-resolution, we immediately obtain the following important corollary.

Corollary 2.2.5 Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLDNF(P,B) = YES$ iff $SLD(P^+, B^+) = YES$
2. $SLDNF(P,B) = NO$ iff $SLD(P^+, B^+) = NO$
3. $SLDNF(P,B) = \uparrow$ iff $SLD(P^+, B^+) = \uparrow$

This corollary suggests the following strategy: instead of performing SLDNF resolution on $P \cup \{\leftarrow B\}$, construct the positive program and goal $P^+ \cup \{\leftarrow B^+\}$ and

perform SLD-resolution. In Chapter 4 we will generalize this idea to the case of first-order programs.

Example 2.2.4 shows that Clark's completion is in some sense too strong: it implies things that are not computed by resolution. The constructive completion $Comp(P^+)$ is weaker; roughly speaking: $Comp(P) = Comp(P^+) + \{\neg p \equiv \overline{p} : p \in Prop\}$. The precise relation between the two notions of completion is given in the following proposition.

Proposition 2.2.6 Let P be a propositional program in a language with the set *Prop* of propositional letters. Then (in classical logic):

1. $Comp(P) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$ is equivalent to

   $Comp(P^+) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$.

2. $Comp(P) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$ is a conservative extension of $Comp(P)$.

The constructive transformation P^+ is also in some sense symmetrical. A precise description of those internal symmetries is given in the next theorem.

Proposition 2.2.7 Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLD(P^+, B^+) = YES$ iff $SLD(P^+, (\neg B)^+) = NO$
2. $SLD(P^+, B^+) = NO$ iff $SLD(P^+, (\neg B)^+) = YES$
3. $SLD(P^+, B^+) = \uparrow$ iff $SLD(P^+, (\neg B)^+) = \uparrow$

2.3 Completeness in intuitionistic and intermediate logics

In this section we generalize the results of the previous section, obtaining completeness theorems for propositional SLDNF-resolution with respect to intuitionistic or intermediate logics. Every intuitionistic tautology is a classical tautology, but not vice versa. Intuitionistic logic is considered to be a logic of constructive reasoning. Much common sense reasoning has constructive flavor, and when formalized, can be placed between intuitionistic and classical logic. As in programming practice reasoning about SLDNF-derivations for bigger programs is not expected to be formal, it is important to know that the entire machinery of classical logic is not necessary. This assures us that there is a good chance to deduce all the needed properties of propositional SLDNF-derivations while reasoning in an informal way.

Theorem 2.3.1 (Soundness and completeness)

Let $\mathbf{L}$ be any intermediate logic between intuitionistic logic and classical logic (including the two). Let P be a propositional program and let $\leftarrow B$ ba a propositional goal. Then:

1. $SLDNF(P, B) = YES$ iff $Comp(P^+) \vdash B^+$
2. $SLDNF(P, B) = NO$ iff $Comp(P^+) \vdash (\neg B)^+$
3. $SLDNF(P, B) = \uparrow$ iff $Comp(P^+) \nvdash_{\mathbf{L}} B^+$ and $Comp(P^+) \nvdash_{\mathbf{L}} (\neg B)^+$

2.4 Completeness in modal logics

In this section we define two notions of modal completions which yield completeness the-

orems for propositional SLDNF-resolution in wide families of modal logics. Modal logics are extensions of classical logic involving additional logical operators $\Box$ and $\Diamond$. Several modal logics were constructed in an attempt to capture the notion of knowledge. In those logics $\Box\alpha$ can be read "I know that α is true"; and $\Diamond\alpha$ as "I can assume α" or "α is consistent with my knowledge". It is natural to interpret a statement, say:

$$p \leftarrow (q \wedge \neg r) \vee s$$

as:

"I know p if/iff (I know q and I know $\neg r$) or I know s",

in symbols:

$$\Box p \leftarrow (\Box q \wedge \Box \neg r) \vee \Box s.$$

It seems also intuitive to add the following characterization of negation:

"I know $\neg r$ if/iff I know that I don't know r",

in symbols:

$$\Box r \leftarrow \Box \neg \Box r.$$

After all, this is what negation as failure does: it derives $\neg r$ if it can see that it never will be able to derive r. Unfortunately formulas like $\Box r \leftarrow \Box \neg \Box r$ turned out to be technically problematic, and in the approach that is presented in this section, we dropped them.

In this section we consider only propositional modal logics that extend classical propositional logic, and contain the schema $\Diamond\alpha \equiv \neg\Box\neg\alpha$.

The following definition recalls an important distinction, often overlooked in pre-

sentations of the theory of modal logics.

Definition 2.4.1

1. Propositional modal logic is called *standard* [2] if it contains the rule $(\alpha \equiv \beta)/(\Box\alpha \equiv \Box\beta)$, (which can be applied to logical and non-logical axioms as well).

2. If **L** is a modal propositional calculus, **dL** stands for the logic in which theorems of **L** are treated as schemata, and whose only inference rule is the rule of detachment: $\alpha, \alpha \rightarrow \beta/\beta$.

3. If **L** is a modal propositional calculus, **sL** stands for the logic in which theorems of **L** are treated as schemata, and whose only inference rules are: $\alpha, \alpha \rightarrow \beta/\beta$ and $(\alpha \equiv \beta)/(\Box\alpha \equiv \Box\beta)$.

4. By a *standard Kripke model* or *s-model* we understand any tuple $M = \langle W, R, v\rangle$, where:

   $W \neq \emptyset$,

   $R \subseteq W \times W$,

   $v_P : W \times Prop \longrightarrow \{true,\ false\}$.

   The notion of *satisfaction of a formula in a world w* is defined as the smallest relation meeting the following conditions:

[2] A particular case of a *standard propositional calculus* in the sense of Rasiowa

$M, w \models \top$

$M, w \models p$ iff $v(w, p) = true$ (for any propositional letter $p \in Prop$)

$M, w \models \neg\alpha$ iff not $M, w \models \alpha$

$M, w \models \alpha \wedge \beta$ iff $M, w \models \alpha$ and $M, w \models \beta$

$M, w \models \alpha \vee \beta$ iff $M, w \models \alpha$ or $M, w \models \beta$

$M, w \models \alpha \rightarrow \beta$ iff $M, w \models \alpha$ implies $M, w \models \beta$

$M, w \models \Box\alpha$ iff for any w', wRw' implies $M, w' \models \alpha$

$M, w \models \Diamond\alpha$ iff there exists w' such that wRw' and $M, w' \models \alpha$

A formula is said to be *true in the model* M iff for every $w \in W$, $M, w \models \alpha$; this is denoted by $M \models \alpha$.

5. By a *Kripke model with origin*, or *d-model*, we understand any tuple $M = \langle W, w_0, R, v\rangle$, such that $w_0 \in W$ and $M = \langle W, R, v\rangle$ satisfies the conditions of item 4. The notion of satisfaction of a formula in a world is defined as in 4. A *formula is true in the model* M if $M, w_0 \models \alpha$; this is denoted by $M \models \alpha$.

Example 2.4.2 Propositional calculus **S5** consists of all formulas that can be derived using the following schemata:

CL: theorems of classical propositional calculus

K: $\Box\alpha \wedge \Box\beta \rightarrow \Box(\alpha \wedge \beta)$

T: $\Box\alpha \rightarrow \alpha$

4: $\Box\alpha \rightarrow \Box\Box\alpha$

5: $\Diamond\alpha \rightarrow \Box\Diamond\alpha$

$\alpha, \alpha \rightarrow \beta / \beta$

$\alpha/\Box\alpha$

One can show that **S5** is closed under the rule [3] $(\alpha \equiv \beta)/(\Box\alpha \equiv \Box\beta)$:

if $\vdash_{\mathbf{S5}} \alpha \equiv \beta$ then $\vdash_{\mathbf{S5}} \Box\alpha \equiv \Box\beta$. **S5** defined as above is a modal propositional calculus, but not a logic: it makes sense to write $\vdash_{\mathbf{S5}} \alpha$, but $\Gamma \vdash_{\mathbf{S5}} \alpha$ is meaningless.

There is a variety of logics that have **S5** as their propositional calculus. Two of them are of special interest: **dS5** and **sS5**. We have:

$p \equiv q \not\vdash_{\mathbf{dS5}} \Box p \equiv \Box q$

$p \equiv q \vdash_{\mathbf{sS5}} \Box p \equiv \Box q$

One can show that **dS5** can be axiomatized by CL, K, T, 4, 5 and

$(\vdash_{\mathbf{dS5}} \alpha)/\Box\alpha$

$\alpha, \alpha \rightarrow \beta/\beta$

The assertion mark $\vdash_{\mathbf{dS5}}$ in the first of these rules indicates that the rule can be applied only to logical theorems α.

sS5 can be axiomatized by CL, K, T, 4, 5 and

$\alpha/\Box\alpha$

$\alpha, \alpha \rightarrow \beta/\beta$

Now both rules can be applied to logical and non-logical axioms as well. Thus we have:

$\alpha \not\vdash_{\mathbf{dS5}} \Box\alpha$

$\alpha \vdash_{\mathbf{sS5}} \Box\alpha$

It is important to notice that s-logics are not sound with respect to Kripke models

[3] This means that modal propositional calculus **S5** is *classical* in the sense of Segerberg.

with origins. **sS5** is sound and complete with respect to the class of s-models in which R is an equivalence relation, while **dS5** is sound and complete with respect to the class of d-models in which R is an equivalence relation.

Any d-logic satisfies the following version of the deduction lemma:

$\Gamma, \alpha \vdash \beta$ iff $\Gamma \vdash \alpha \rightarrow \beta$

The implication to the right is usually not true for s-logics. s-logics containing **sS4**, (among them **sS5**) satisfy the following version:

$\Gamma, \alpha \vdash \beta$ iff $\Gamma \vdash \Box\alpha \rightarrow \beta$

Definition 2.4.3 Let C be a propositional $\vee$-normal statement. We associate with it two modal formulas: C_1 and C_2. C_1 is obtained from C by prefixing every literal (in the head and in the body) by $\Box$; C_2 is obtained from C by prefixing every literal by $\Diamond$ and by changing $\leftarrow$ to $\rightarrow$. If P is a propositional program we define its *modal completion* $C^{\Box}(P)$ as $\{C_1 \mid C \in (P^{Cd})\} \cup \{C_2 \mid C \in (P^{Cd})\}$. If $\leftarrow B$ is a propositional goal, we define its *modal version* $\leftarrow B^{\Box}$. $B^{\Box}$ is obtained by turning B into disjunctive normal form and prefixing each literal by $\Box$.

Example 2.4.4 Consider the program P:

$q \leftarrow \neg q$

$q \leftarrow p$

$p \leftarrow p$

We will construct its modal completion $C^{\Box}(P)$. First we transform P into Clark's disjunctive form P^{Cd}:

$q \leftarrow \neg q \vee p$

$p \leftarrow p$

So the modal completion $C^{\Box}(P)$ consists of the following formulas:

$\Box q \leftarrow \Box\neg q \vee \Box p$ $\qquad$ $\Diamond q \rightarrow \Diamond\neg q \vee \Diamond p$

$\Box p \leftarrow \Box p$ $\qquad$ $\Diamond p \rightarrow \Diamond p$

Theorem 2.4.5 (Soundness and completeness) [4] Let **L** be any modal logic extending $\mathbf{CL} + \Box\top$ and contained in **sS5**. Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLDNF(P, B) = YES \iff C^{\Box}(P) \vdash_L B^{\Box}$
2. $SLDNF(P, B) = NO \iff C^{\Box}(P) \vdash_L (\neg B)^{\Box}$
3. $SLDNF(P, B) = \uparrow \iff C^{\Box}(P) \nvdash_L B^{\Box}$ and $C^{\Box}(P^+) \nvdash_L (\neg B)^{\Box}$

Example 2.4.6 Continuing Example 2.4.4, notice that $SLDNF(P, p) = \uparrow$. (This gives a counter-example to completeness with respect to Clark's completion, because $Comp(P) \vdash p$.) Let us see how Theorem 2.4.5 works for this program. According to the theorem in order to show that $SLDNF(P, p) = \uparrow$ it is enough to prove that in **sS5** $C^{\Box}(P^+) \nvdash \Box p$ and $C^{\Box}(P^+) \nvdash \Box\neg p$. For that we present a Kripke model for **sS5** that makes $C^{\Box}(P)$ true and $\Box p, \Box\neg p$ false. Let $M = \langle W, R, v\rangle$ where $W = \{w_1, w_2, w_3, w_4\}$, R is a total relation, and

$v(w_1, p) = false, \quad v(w_1, q) = false$

$v(w_2, p) = false, \quad v(w_2, q) = true$

[4] Recently we were informed that a particular case of this theorem, with **L** being the modal logic **K**, has been obtained by another author and submitted to the *Journal of Logic and Computation.*

$v(w_3, p) = true, \quad v(w_3, q) = false$

$v(w_4, p) = true, \quad v(w_4, q) = true.$

It is interesting to notice that there is freedom not only in choosing the logic **L** but also in taking $\equiv$ instead of $\leftarrow$ or $\rightarrow$ in the formulas of the modal completion. This corresponds to possibility of using $Comp(P^+)$ or just P^+ in the completeness theorem with respect to classical logic.

Definition 2.4.7 For a propositional program P its modal completion $C^{\Box}_{\equiv}(P)$ results by changing $\leftarrow$ to $\equiv$ and $\rightarrow$ to $\equiv$ in every formula of $C^{\Box}(P)$.

Theorem 2.4.8 (Soundness and completeness) Let **L** be any modal logic extending $\mathbf{CL} + \Box\top$ and contained in **sS5**. Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLDNF(P, B) = YES \iff C^{\Box}_{\equiv}(P) \vdash_L B^{\Box}$
2. $SLDNF(P, B) = NO \iff C^{\Box}_{\equiv}(P) \vdash_L (\neg B)^{\Box}$
3. $SLDNF(P, B) = \uparrow \iff C^{\Box}_{\equiv}(P) \nvdash_L B^{\Box}$ and $C^{\Box}_{\equiv}(P^+) \nvdash_L (\neg B)^{\Box}$

2.5 Gentzen-style characterization

In this section we propose a Gentzen-style system which may characterize the success and finite failure sets of propositional SLDNF-resolution. We conjecture its completeness. The system could be used as a convenient tool in reasoning about propositional SLDNF-computations.

Below we write e.g. $\Gamma,\ p \leftarrow d_1 \vee d_2$ to mean a propositional $\vee$-normal program with the set of clauses $\Gamma \cup \{p \leftarrow d_1 \vee d_2\}$, assuming that Γ doesn't contain any clause with the head p. d_1, d_2 denote disjunctions of conjunctions of literals; c_1, c_2 denote conjunctions of literals.

Definition 2.5.1 The system **G** consists of the following rules and axioms:

$$\Gamma,\ p \leftarrow d_1 \vee d_2 \vdash_G p \iff \Gamma,\ p \leftarrow d_1 \vdash_G p \text{ or } \Gamma,\ p \leftarrow d_2 \vdash_G p$$

$$\Gamma,\ p \leftarrow d_1 \vee d_2 \vdash_G \neg p \iff \Gamma,\ p \leftarrow d_1 \vdash_G \neg p \text{ and } \Gamma,\ p \leftarrow d_2 \vdash_G \neg p$$

$$\Gamma,\ p \leftarrow c_1 \wedge c_2 \vdash_G p \iff \Gamma,\ p \leftarrow c_1 \vdash_G p \text{ and } \Gamma,\ p \leftarrow c_2 \vdash_G p$$

$$\Gamma,\ p \leftarrow c_1 \wedge c_2 \vdash_G \neg p \iff \Gamma,\ p \leftarrow c_1 \vdash_G \neg p \text{ or } \Gamma,\ p \leftarrow c_2 \vdash_G \neg p$$

$$\Gamma,\ p \leftarrow q \vdash_G p \iff \Gamma,\ p \leftarrow q \vdash_G q$$

$$\Gamma,\ p \leftarrow q, q \leftarrow r \vdash_G p \iff \Gamma,\ p \leftarrow r, q \leftarrow r \vdash_G p$$

$$\Gamma,\ p \leftarrow \top \vdash_G p$$

Notice the exact form of the rules of the system **G** . A relaxed version of the first rule: $\Gamma,\ p \leftarrow d_1 \vee d_2 \vdash_G q \iff \Gamma,\ p \leftarrow d_1 \vdash_G q$ or $\Gamma,\ p \leftarrow d_2 \vdash_G q$ is not sound with respect to propositional SLDNF-resolution.

Proposition 2.5.2

$$\Gamma,\ p \leftarrow p \not\vdash_G p$$

$$\Gamma,\ p \leftarrow p \not\vdash_G \neg p$$

$$\Gamma,\ p \leftarrow \neg p \not\vdash_G p$$

$$\Gamma,\ p \leftarrow \neg p \not\vdash_G \neg p$$

$$\Gamma,\ p \leftarrow \top \not\vdash_G \neg p$$

$$\Gamma,\ p \leftarrow \bot \vdash_G \neg p$$

$$\Gamma,\ p \leftarrow \bot \not\vdash_G p$$

Proposition 2.5.3 It is decidable whether $P \vdash_G L$, where P is a propositional program and L is a literal in the language of P.

We conjecture that the system **G** characterizes the success and finite failure sets of propositional SLDNF-resolution.

Conjecture 2.5.4 (Soundness and completeness) Let P be a propositional $\vee$-normal program and let p be a propositional letter in P. Then:

1. $SLDNF(P,p) = YES \iff P^+ \vdash_G p$
2. $SLDNF(P,p) = NO \iff P^+ \vdash_G \neg p$
3. $SLDNF(P,p) = \uparrow \iff P^+ \not\vdash_G p$ and $P^+ \not\vdash_G \neg p$

Example 2.5.5 Let P be the following program:

$q \leftarrow q \vee p$

$p \leftarrow \neg p \vee q$

This program is troublesome if one considers Clark's completion: $Comp(P) \vdash p$ but

p does not belong to the SLDNF-success set of P. Let us see how this is resolved by the system **G**.

$q \leftarrow q \vee p,\ p \leftarrow \neg p \vee q \vdash_G p$

iff

$q \leftarrow q \vee p,\ p \leftarrow \neg p \vdash_G p$ or $q \leftarrow q \vee p,\ p \leftarrow q \vdash_G p$

iff

false or $q \leftarrow q \vee p,\ p \leftarrow q \vdash_G p$

iff

$q \leftarrow q \vee p,\ p \leftarrow q \vdash_G p$

iff

$q \leftarrow q \vee p,\ p \leftarrow q \vdash_G q$

iff

$q \leftarrow q,\ p \leftarrow q \vdash_G q$ or $q \leftarrow p,\ p \leftarrow q \vdash_G q$

iff

false or $q \leftarrow p,\ p \leftarrow q \vdash_G q$

iff

$p \leftarrow q,\ q \leftarrow q \vdash_G q$

iff

false

So, according to the conjecture above $SLDNF(P,p) \neq YES$. In a similar way one can use the system **G** to show that p doesn't belong to the SLDNF-finite failure set of this program either. Thus every computation of the SLDNF-resolution for this program

with the goal $\leftarrow p$ loops, assuming the conjecture.

3 ω-HERBRAND MODELS AND EQUALITY THEORY

In this chapter we introduce technical tools which will be used in the next chapter. We propose an alternative version of the notion of Herbrand models and investigate the formal theory of equality determined by such models. Algorithms related to the decision procedure for this theory will be applied to generate answers in first-order resolution systems. Use of ω-Herbrand models will be the key element in results stating that operators associated with positive programs (involving $\forall, \exists, =$), despite being not continuous, reach their least fix-points in ω steps.

3.1 First-order languages

This section explains the terminology and notation concerning first-order languages. Except for the definitions 3.1.1, 3.1.4, and 3.1.5, all other items are fairly standard, so after looking at the three definitions, the reader may move on to the next section, and come back to the rest of the material only in case of doubt.

Definition 3.1.1 By a *language* we understand any countable first-order language that contains finitely many individual constants or function symbols.

This is the only class of first-order languages that we consider in the rest of the paper. For brevity we do not repeat this assumption when formulating theorems, but the reader should be aware of it. Since in programming practice, programs are always

finite and do not require languages with infinite alphabets, such a class is sufficient for our purposes.

Now the terminology and notational conventions:

Individual constants are considered 0-ary function symbols; propositional letters are considered 0-ary predicate symbols. If a language $\mathcal{L}$ is specified in a context, by *Func* we denote the set of all its (individual constants and) function symbols.

Formula always means a formula of a (first-order) language, so if a big conjunction $\bigwedge_{i\in I}$ or a big disjunction $\bigvee_{i\in I}$ is used, the set I is assumed to be finite. A big conjunction over the empty set is understood as $\top$; a big disjunction over the empty set is understood as $\bot$. Recall also that $A \leftarrow B$ means $B \rightarrow A$, and that $\leftarrow B$ means $\bot \leftarrow B$, which is equivalent to $\neg B$.

If we refer to a formula as $B(x_1, \ldots, x_n)$, we assume that all the free variables of B are contained among $x_1, \ldots, x_n$, but we do not require each of the variables $x_1, \ldots, x_n$ actually occurs free in the formula. Sometimes we will refer to a formula without specyfying its variables, if we write B, the formula may contain free variables; by $var(B)$ we denote the set of all the free variables of the formula B.

Instead of $\exists_{y_1} \ldots \exists_{y_n} B$ we write $\exists_{y_1,\ldots,y_n} B$, or just $\exists_{\boldsymbol{y}} B$ where $\boldsymbol{y} = \langle y_1, \ldots, y_n \rangle$. In-

dependently of the notation we always admit n=0, in which case the formula reduces to B. Similar conventions are used for universal quantifiers.

Language $\mathcal{L}$ is called a *language with equality* if it contains a binary predicate symbol =, otherwise it is called a *language without equality.* If $\mathcal{L}$ is a language without equality $\mathcal{L}^{=}$ stands for the extension of $\mathcal{L}$ obtained by adjoining the symbol = to the alphabet. If we consider theories in languages with equality, we always carefully specify the assumed axioms of equality; typically we will be interested in axioms that impose on = conditions stronger than just being a congruence in models. The symbol $\neq$ is used as an abbreviation for the negation of = . Given a language with equality, by an *interpretation* we understand a normal interpretation, i.e. an interpretation in which the symbol = is interpreted as equality, not just as a congruence.

We say "every formula B in the class $\mathcal{C}$ is *effectively equivalent* in theory $\mathcal{T}$ to a formula in the class $\mathcal{C}'$", if there exists an algorithm which takes as input any $B \in \mathcal{C}$ and returns as output $B' \in \mathcal{C}'$, such that $\mathcal{T} \vdash B \equiv B'$. In the rest of the paper, when proving such statements, we always provide the algorithm. To illustrate the notion of effective equivalence, notice that every formula in the language of arithmetic is equivalent in $Th(\mathcal{N})$ either to $\top$ or to $\bot$, but as $Th(\mathcal{N})$ is undecidable this equivalence is not effective — there is no algorithm that could perform the task of transforming formulas in this way.

Definition 3.1.2 By a Δ_0-*formula* we understand any formula without quantifiers.

By a Σ_1-*formula* we understand any formula $\exists_{y_1,\ldots,y_m} B$ where $m \geq 0$ and B is a Δ_0-formula. By a Π_1-*formula* we understand any formula $\forall_{y_1,\ldots,y_m} B$ where $m \geq 0$ and B is a Δ_0-formula. By a Σ_{n+1}-*formula* we understand any formula $\exists_{y_1,\ldots,y_m} B$ where $m \geq 0$ and B is a Π_n-formula. By a Π_{n+1}-*formula* we understand any formula $\forall_{y_1,\ldots,y_m} B$ where $m \geq 0$ and B is a Σ_n-formula. The term Δ_ω-*formula* is a synonym for 'formula'.

By a Σ-*formula* we understand any formula with no negative occurrences of existential quantifiers, and no positive occurrences of universal quantifiers. By a Π-*formula* we understand any formula with no positive occurrences of existential quantifiers, and no negative occurrences of universal quantifiers.

Proposition 3.1.3

1. Every Δ_0-formula is also Σ_n and Π_n, for every n.
2. Every Σ_n-formula is also Σ_{n+1} and Π_{n+1}, for every n.
3. Every Σ_1-formula is a Σ-formula.
4. Every Σ-formula is effectively equivalent in the predicate calculus to a Σ_1-formula.
5. Every conjunction or disjunction of Σ_1-formulas

   is effectively equivalent in predicate calculus to a Σ_1-formula

Notice that formulas equivalent to Δ_0-formulas, but containing quantifiers, are not considered to be Δ_0; similarly for other classes of the hierarchy. As it is decidable whether a Δ_0-formula is a tautology, and as the predicate calculus is undecidable, there is no algorithm which, if given a formula, outputs correctly an equivalent Δ_0-

formula or the statement "The formula is not equivalent to any Δ_0-formula". The equivalence is not effective.

Definition 3.1.4 By a *positive formula* we understand any formula built from atomic formulas, including $\top$ and $\bot$, by means of $\wedge$, $\vee$, $\forall$, $\exists$.

By a *guard* we understand any formula containing no predicates except $=$.

By a *positive formula with guards* we mean any formula $B(S_1/p_1, \ldots, S_n/p_n)$ where $B(p_1, \ldots, p_n)$ contains the predicate variables $p_1, \ldots, p_n$ but not $=$, and $S_1, \ldots, S_n$ are guards. (Notice we do not require that $S_1, \ldots, S_n$ are positive!)

Notice that a positive formula with guards need not be a positive formula. Positive formulas with guards allow some, but not arbitrary, negative information in them.

It is also important to remark that formulas equivalent to those built only by means of $\wedge$, $\vee$, $\forall$, $\exists$, but containing $\neg$, are not considered positive. We will see later that it is decidable whether a positive formula is a tautology. This and the undecidability of the predicate calculus imply that there is no algorithm which, if given a formula as input, outputs correctly an equivalent positive formula or the statement "The formula is not equivalent to any positive formula". The equivalence is not effective.

By a *substitution* we understand any partial function from the set $Var = \{x_i \mid i < \omega\}$ of variables of language $\mathcal{L}$, into the set of terms of $\mathcal{L}$. Substitutions are written as sets of elementary substitutions, e.g. $\theta = \{t_i/x_i \mid i \in I\}$ where I is a set of natural

numbers. Any substitution $\{t_i/x_i \mid i \in I\}$ that is not total is identified with the following total substitution: $\{t_i/x_i \mid i \in I\} \cup \{x_i/x_i \mid i \in (\omega - I)\}$. A substitution $\theta = \{t_i/x_i \mid i \in I\}$ is called *finite* if the set $\{i \in I \mid t_i \text{ is distinct from } x_i\}$ is finite. The symbol ϵ stands for the *identity substitution*: $\epsilon = \{x_i/x_i \mid i < \omega\}$. Symbols like $\theta[f(x)/x_2, c/x_4]$ denote modifications of substitutions.

Given an interpretation of a language $\mathcal{L}$, by a *valuation* we always understand a total function from the set *Var* of variables of $\mathcal{L}$, into the domain of the interpretation. However, departing from formal style, we may sometimes specify only a finite fragment of a valuation, if the rest is irrelevant. This will be the case with the notion of satisfiability: $M \models B[\nu]$ depends only on the part of the valuation ν, that concerns variables occurring free in B. Symbols like $\nu[e_1/x_2, e_5/x_4]$ denote modifications of valuations.

Definition 3.1.5

1. If $\theta = \{t_i/x_i \mid i < \omega\}$ is a substitution, by $\tilde{\theta}$ we denote the sequence $\langle t_0, t_1, t_2, \ldots \rangle$.
2. If $\nu : \{x_i \mid < \omega\} \longrightarrow |M|$ is a valuation, by $\tilde{\nu}$ we denote the sequence $\langle x_0[\nu], x_1[\nu], x_2[\nu], \ldots \rangle$ of elements of $|M|$ substituted for consecutive variables.

Finally let us recall the concept of conservative extension of a theory.

Definition 3.1.6 Let $\mathcal{T}_1$, $\mathcal{T}_2$ be theories, respectively in languages $\mathcal{L}_1$, $\mathcal{L}_2$. Then:

1. $\mathcal{T}_2$ is called an *extension* of $\mathcal{T}_1$, if $\mathcal{L}_2$ is an extension of $\mathcal{L}_1$, and for every $B \in \mathcal{L}_1$, $\mathcal{T}_1 \models B$ implies $\mathcal{T}_2 \models B$.

2. $\mathcal{T}_2$ is called a *conservative extension* of $\mathcal{T}_1$, if $\mathcal{T}_2$ is an extension of $\mathcal{T}_1$, and for every $B \in \mathcal{L}_1$, $\mathcal{T}_2 \models B$ implies $\mathcal{T}_1 \models B$.

3. $\mathcal{T}_2$ is called a *linguistic extension* of $\mathcal{T}_1$, if $\mathcal{L}_2$ is an extension of $\mathcal{L}_1$, and the sets of axioms of $\mathcal{T}_1$, $\mathcal{T}_2$ are the same.

4. If $\mathcal{L}_2$ is obtained by adjoining a new predicate symbol p to the alphabet of $\mathcal{L}_1$, and there exists $B \in \mathcal{L}_1$ such that theory $\mathcal{T}_2$ is obtained by adjoining the formula $p(x) \equiv B(x)$ to the axioms of $\mathcal{T}_1$, then $\mathcal{T}_2$ is called a *definitional extension* of $\mathcal{T}_1$.

It is well known that linguistic or definitional extensions are conservative.

3.2 ω-Herbrand models

The notion of a Herbrand model is an important tool in the theory of logic programming. But one can see that proofs that use Herbrand models are not always smooth, and that some desired results on Herbrand models surprisingly turn out to be false. In this section we propose the notion of an ω-Herbrand model that corrects these deficiencies. ω-Herbrand models can be investigated using powerful methods of algebraic semantics for first-order classical logic.

Definition 3.2.1 (ω-Herbrand universe)

Let $\mathcal{L}$ be language and let $\mathcal{L}^{\mathcal{K}}$ result by adding a countable set $\mathcal{K} = \{k_i \mid i < \omega\}$ of

new individual constants to the alphabet of $\mathcal{L}$. By the *ω-Herbrand universe* $U_{\mathcal{L}}^{\omega}$ for $\mathcal{L}$ we understand the set of all ground terms of the language $\mathcal{L}^{\mathcal{K}}$. We refer to members of $U_{\mathcal{L}}^{\omega}$ as *elements.* Members of the set $\mathcal{K}$ will be called *free constants.*

Equipped with the notion of ω-Herbrand universe we define notions analogous to those for conventional Herbrand models.

- By the *ω-Herbrand base* $B_{\mathcal{L}}^{\omega}$ we understand the set of all ground atomic formulas in $\mathcal{L}^{\mathcal{K}}$.

- By an *ω-Herbrand interpretation* for $\mathcal{L}$ we understand any nonempty subset of $B_{\mathcal{L}}^{\omega}$.

- By an *ω-Herbrand model* for a set of formulas Γ we understand any ω-Herbrand interpretation that is a model for Γ.

- ω-Herbrand interpretations for $\mathcal{L}$ are ordered by inclusion. If among ω-Herbrand models for Γ there is the least model in this order, it is denoted M_P^{ω} and is called the *least ω-Herbrand model* for Γ.

- For a definite program P, the operator $T_P^{\omega} : 2^{B_P^{\omega}} \longrightarrow 2^{B_P^{\omega}}$ on the lattice of ω-Herbrand interpretations is defined as follows: For any $I \subseteq B_{\mathcal{L}}^{\omega}$,

  $T_P^{\omega}(I) = \{A[\nu] \in B_{\mathcal{L}}^{\omega} \mid (A \leftarrow B) \in P \text{ and } I \models B[\nu]\}$.

- If P is a definite program in language $\mathcal{L}$, then by the *SLD-ω-success set of P* we understand the set $\{A \in B_{\mathcal{L}}^{\omega} \mid P \cup \{\leftarrow A\}$ has an SLD-refutation $\}$.

In this new setting we still have results analogous to those for Herbrand models: For any definite program P,

$$M_P^\omega = lfpT_P^\omega = T_P^\omega\uparrow\omega = \{A \in B_P^\omega \ : P \vdash A\} = \text{SLD-}\omega\text{-success set of } P.$$

Cf. [44].

Example 3.2.2 Let $\mathcal{L}$ be a language without equality and let P be a set of definite clauses in $\mathcal{L}$. If P has a model, then P has a Herbrand model. P is unsatisfiable iff P has no Herbrand model. The straightforward counterexample with P being $\{p(c), \exists_x(\neg p(x))\}$ shows that in these propositions one cannot drop the assumption that formulas in P are definite clauses. Cf. [44] p. 17.

The next theorem shows that the situation is very different for ω-Herbrand models.

Theorem 3.2.3 (Completeness with respect to ω-Herbrand models)
Let $\Gamma \cup \{B\}$ be a set of formulas in a language $\mathcal{L}$ without equality. Then $\Gamma \not\vdash B$ implies that there exists an ω-Herbrand interpretation $I \subseteq B_P^\omega$ such that $I \models \Gamma$ and $I \not\models B$.

As corollaries to this theorem we obtain desirable strengthenings of propositions mentioned in example 3.2.2:
Let $\mathcal{L}$ be a language without equality and let Γ be a set of formulas in $\mathcal{L}$. If P has a model, then P has an ω-Herbrand model. P is unsatisfiable iff P has no ω-Herbrand model.

Example 3.2.4 Consider a language $\mathcal{L}$ without equality. For definite programs P and ground atomic formulas A, the *Herbrand rule* is defined as (cf. [44] p. 100):

If $Comp(P) \cup \{A\}$ has no Herbrand model, infer $\neg A$.

The Herbrand rule is strictly weaker than the Closed World Assumption Rule CWA. With ω-Herbrand interpretations we may define an *ω-Herbrand rule*:

If $Comp(P) \cup \{A\}$ has no ω-Herbrand model, infer $\neg A$.

Now, by theorem 3.2.3, for definite programs and ground atomic formulas, the ω-Herbrand rule coincides with CWA.

Example 3.2.5 Let P be a definite program and let A be a *ground* atomic formula in a language without equality, then: $P \vdash A$ iff $M_P \models A$. The straightforward counterexample with P being $\{p(a) \leftarrow \top\}$ and A being $p(x)$ shows that one can not drop the assumption that A is ground. Cf. [44] p. 39.

The next theorem shows that the situation changes dramatically if we allow ω-Herbrand models.

Theorem 3.2.6 Let P be a definite program and A an atomic formula in a language without equality. Then: $P \vdash A$ iff $M_P^\omega \models A$.

The problem mentioned in Example 3.2.5 causes some complications in the standard proof of completeness of SLD-resolution. Theorem 3.2.6 overcomes this problem and therefore it is a key to a concise proof. This idea will be demonstrated in the next chapter, while proving completeness of some extensions of SLD-resolution.

Another interesting property of M_P^ω, which does not have a counterpart with conventional Herbrand models is given in the next theorem.

Theorem 3.2.7 (Existence property)

Let P be a definite program and let A be an atomic formula in a language without equality. Then the following conditions are equivalent.

1. $M_P^\omega \models \exists_{\boldsymbol{x}} A(\boldsymbol{x})$
2. $P \vdash \exists_{\boldsymbol{x}} A(\boldsymbol{x})$
3. There exists a sequence of terms $\boldsymbol{t}$ such that $P \vdash A(\boldsymbol{t})$.

All the theorems in this section are proved using methods of algebraic semantics. The methods, developed for first-order logic by H. Rasiowa and R. Sikorski, constitute a powerful tool and one can expect that further algebraic analysis of ω-Herbrand models will bring new interesting results. An algebraic approach suits our needs especially well, because it turns out that universes of so called "canonical algebraic models" are exactly ω-Herbrand universes.

3.3 Theory of equality

In this section we introduce equality theory $CET_{\mathcal{L}}^\omega$. The theory guaranties not only that $=$ is a congruence, but also that function symbols are interpreted in a "free" way and that "free constants" exist in the models. Later it will be shown that $CET_{\mathcal{L}}^\omega$ is a complete and decidable description of the ω-Herbrand universe $U_{\mathcal{L}}^\omega$.

Definition 3.3.1 Let $\mathcal{L}$ be a language with equality. Clark's Equality Theory $CET_{\mathcal{L}}$ is based on the following axioms:

1. $x{=}x$

2. $x{=}x' \rightarrow x'{=}x$

3. $x{=}x' \wedge x'{=}x'' \rightarrow x{=}x''$

4. $x_1{=}x'_1 \wedge \ldots \wedge x_n{=}x'_n \rightarrow f(x_1,\ldots,x_n){=}f(x'_1,\ldots,x'_n)$ for any function symbol f.

5. $x_1{=}x'_1 \wedge \ldots \wedge x_n{=}x'_n \wedge p(x_1,\ldots,x_n) \rightarrow p(x'_1,\ldots,x'_n)$ for any predicate symbol p.

6. $f(x_1,\ldots,x_n){\neq}g(x'_1,\ldots,x'_m)$ for any pair of different function symbols f and g.

7. $f(x_1,\ldots,x_n){=}f(x'_1,\ldots,x'_n) \rightarrow x_1{=}x'_1 \wedge \ldots \wedge x_n{=}x'_n$ for any function symbol f.

8. $t{\neq}x$ for any term t containing x, but not identical with x.

Axioms 1-5 state that $=$ is a congruence in models, and this is what is assumed in so called first-order theories with equality. Axioms 6-8 assure that function symbols are interpreted in models in a 'free' way. Notice that the axioms are not independent: 1 and 5 imply 2 and 3. Using the semantics one can easily show that 8 is not a consequence of 1-7.

Our main task in this section is to give a syntactical description of the ω-Herbrand universe $U^{\omega}_{\mathcal{L}}$. As $U^{\omega}_{\mathcal{L}} \models CET_{\mathcal{L}}$, the theory $CET_{\mathcal{L}}$ is a correct description, but it is not complete, even if the language $\mathcal{L}$ does not contain any predicate symbols except $=$. Consider, for example, the language with one individual constant c, the predicate symbol $=$, and with no other symbols. Using semantics, one can see that neither $CET_{\mathcal{L}} \vdash \exists_x(x{\neq}c)$ nor $CET_{\mathcal{L}} \vdash \neg\exists_x(x{\neq}c)$. As for languages that do not contain any predicate symbols except $=$ we look for a complete description of $U^{\omega}_{\mathcal{L}}$, we have to add to $CET_{\mathcal{L}}$ axioms guaranteeing that in any model there exist infinitely many elements that are neither interpretations of individual constants nor values of functions. This is done in the next definition.

Definition 3.3.2 (Equality theory $CET^{\omega}_{\mathcal{L}}$)

Let $\mathcal{L}$ be a language with equality. By $CET^{\omega}_{\mathcal{L}}$ we understand the extension of $CET_{\mathcal{L}}$ obtained by adjoining the following axioms for every $n < \omega$:

$$\exists_{x_1,\ldots,x_n} \bigwedge_{n \geq i \neq j \leq n} x_i \neq x_j \wedge \bigwedge_{i \leq n} \bigwedge_{f \in Func} \neg is_f(x_i)$$

where $is_f(x)$ is an abbreviation for $\exists_{y_1,\ldots,y_m}(x=f(y_1,\ldots,y_m))$,

(for an individual constants i.e. 0-ary function symbol, $is_c(x)$ means $x=c$).

Recall that by a language we understand any first-order language whose set of individual constants and function symbols is finite. The definition above makes sense only if this restriction is assumed. Notice that $CET^{\omega}_{\mathcal{L}}$ does not have any finite models. Notice also that every ω-Herbrand model with the universe $U^{\omega}_{\mathcal{L}}$ is a model for $CET^{\omega}_{\mathcal{L}}$. We will use symbol $U^{\omega}_{\mathcal{L}}$ to denote not only the set but also the ω-Herbrand interpretation in which all predicates except $=$ are interpreted as $\emptyset$. We have $U^{\omega}_{\mathcal{L}} \models CET^{\omega}_{\mathcal{L}}$ and this shows that $CET^{\omega}_{\mathcal{L}}$ is a correct description of $U^{\omega}_{\mathcal{L}}$. $CET^{\omega}_{\mathcal{L}}$ is also a more accurate description than $CET_{\mathcal{L}}$ — for instance $CET_{\mathcal{L}}$ does not exclude the conventional Herbrand universe $U_{\mathcal{L}}$ as a model, $CET^{\omega}_{\mathcal{L}}$ does. We will come back to the problem of how well $CET^{\omega}_{\mathcal{L}}$ describes $U^{\omega}_{\mathcal{L}}$ in section 3.5.

Now we establish now two basic facts concerning extensions of theories and involving $CET^{\omega}_{\mathcal{L}}$.

Proposition 3.3.3 (Extensions with equality)

1. Let $\mathcal{L}$ be a language without equality. Let $\mathcal{L}^{=}$ be the extension of $\mathcal{L}$ obtained

by adjoining the symbol = to the alphabet. Then for any theory $\mathcal{T}$ in $\mathcal{L}$, $\mathcal{T} \cup CET^{\omega}_{\mathcal{L}=}$ is a conservative extension of $\mathcal{T}$.

2. Let $\mathcal{L}$ and $\mathcal{L}'$ be languages with equality, such that $\mathcal{L} \subseteq \mathcal{L}'$. Then $CET^{\omega}_{\mathcal{L}'}$ is a conservative extension of $CET^{\omega}_{\mathcal{L}}$.

Thanks to the second part of this proposition it is correct to say e.g. "α_1 is equivalent in CET^{ω} to α_2" without specifying the language. Assertions $CET^{\omega}_{\mathcal{L}} \vdash \alpha_1 = \alpha_2$ and $CET^{\omega}_{\mathcal{L}'} \vdash \alpha_1 = \alpha_2$ are equivalent for any languages $\mathcal{L}$ and $\mathcal{L}'$ containing α_1 and α_2.

Finally we may formulate a strengthening of theorem 3.2.3.

Theorem 3.3.4 (Completeness with respect to ω-Herbrand models II) Let $\Gamma \cup \{B\}$ be a set of formulas in a language $\mathcal{L}$ with equality. Then $CET^{\omega}_{\mathcal{L}} \cup \Gamma \not\vdash B$ implies that there exists an ω-Herbrand interpretation $I \subseteq B^{\omega}_{\mathcal{P}}$ such that $I \models CET^{\omega}_{\mathcal{L}} \cup \Gamma$ and $I \not\models B$.

3.4 Normal forms

In this section we consider languages $\mathcal{L}$ containing no predicate symbols other than =, and introduce normal forms of formulas of $CET^{\omega}_{\mathcal{L}}$. Normal forms will be used as the main technical tool in the proofs of decidability and completeness of this theory. Still further, transformations leading to normal forms will be used in algorithms incorporated in resolution systems.

Assumption In this section we consider only languages containing no predicate symbols except $=$.

The following proposition may be used to establish normal forms of positive Δ_0 - formulas in $CET^{\omega}_{\mathcal{L}}$.

Proposition 3.4.1 Any equality $t'=t''$ of terms is effectively equivalent in CET^{ω} to a formula of (at least) one of the following types:

1. $\top$
2. $\bot$
3. $\bigwedge_i x_i=t_i$, where any x_i are variable and any t_i is a term.

Example 3.4.2 In CET^{ω}:

1. $(f(x_1, g(x_2, c))=f(h(c), x_3)) \equiv (x_1=h(c) \wedge x_3=g(x_2, c))$
2. $(x=x) \equiv \top$
3. $(f(x_1, f(x_2, x_3))=f(x_2, x_1)) \equiv \bot$ and also:

   $(f(x_1, f(x_2, x_3))=f(x_2, x_1)) \equiv (x_1=x_2 \wedge x_1=f(x_2, x_3))$.

It turns out that in general, formulas of $CET^{\omega}_{\mathcal{L}}$ are not equivalent to Δ_0-formulas. Normal forms will require using existential quantifiers and negated predicates is_f (which contain a hidden universal quantifier).

Definition 3.4.3 (Canonical formulas in CET^{ω})

By a *canonical formula* we understand the formula $\top$ or any formula

$$\exists y(\bigwedge_{i\in I} x_i=t_i \wedge \bigwedge_{j\in J} y_j\neq t'_j \wedge \bigwedge_{(f,k)\in K} \neg is_f(y_k))$$

where:

1. $y=\langle y_1,\ldots,y_m\rangle$, $m \geq 0$.

2. $y_j \notin \mathrm{var}(t'_j)$ for $j=1,\ldots,m$.

3. $\mathrm{var}(\bigwedge_{j\in J} y_j \neq t'_j \wedge \bigwedge_{(f,k)\in K} \neg is_f(y_k)) \subseteq \bigcup_{i\in I} \mathrm{var}(t_i) = \{y_1,\ldots,y_m\}$.

   (This implies that x_i are all the free variables in the formula

   and no x_i occurs in one of t_i or t'_j).

By a *positive canonical formula* we understand the formula $\top$ or a formula like the one above with $J = K = \emptyset$.

By a *super-canonical formula* we understand the formula $\top$ or a formula like the one above (satisfying 1,2,3), with each t'_j being one of the variables $y_1,\ldots,y_m$.

Notice that every canonical formula is satisfiable in $U^{\omega}_{\mathcal{L}}$.

Theorem 3.4.4 Every inequality $t' \neq t''$ of terms is effectively equivalent in CET^{ω} either to $\bot$ or to a disjunction of *super*-canonical formulas whose free variables are contained among the free variables of $t' \neq t''$.

Example 3.4.5 In CET^{ω}

$$f(x_1, g(x_2)) \neq f(x_2, g(f(x_3, c)))$$

is equivalent to:

$$x_1 \neq x_2 \vee x_2 \neq f(x_3, c)$$

is equivalent to:

$$\exists_{y_1,y_2}(x_1=y_1 \wedge x_2=y_2 \wedge y_1 \neq y_2) \vee$$

$$\exists_{y_1,y_2}(x_1=y_1 \wedge x_2=y_2 \wedge \neg is_f(y_2)) \vee$$

$\exists_{y_1,y_2,y_3}(x_1{=}y_1 \wedge x_2{=}f(y_2,y_3) \wedge x_3{=}y_2 \wedge y_1{\neq}y_2) \vee$

$\exists_{y_1,y_2,y_3}(x_1{=}y_1 \wedge x_2{=}f(y_2,y_3) \wedge x_3{=}y_2 \wedge \neg is_c(y_3))$.

We leave it to the reader to check that theorem 3.4.4 does not generalize to arbitrary Δ_0-formulas. For instance formulas $x_1{\neq}x_2 \wedge x_1{\neq}f(x_2)$ or $x_1{\neq}f(x_2) \wedge x_2{\neq}f(x_1)$ can not be represented as (finite) disjunctions of super-canonical formulas. The situation would change if we allowed countable disjunctions. This can not be done in a first-order language. However in models we can consider countable unions of sets. We will come back to this idea in section 3.6.

Theorem 3.4.6 (Normal form theorem)

Every formula B containing no predicate symbols except $=$ is effectively equivalent in CET^{ω} either to $\perp$ or to a disjunction of canonical formulas whose free variables are contained among the free variables of B. Moreover, if B is positive then the formulas in the resulting disjunction are positive canonical formulas.

3.5 Decidability and completeness

This section completes our answer to the question of how well $CET^{\omega}_{\mathcal{L}}$ describes the ω-Herbrand universe $U^{\omega}_{\mathcal{L}}$. We prove that $CET^{\omega}_{\mathcal{L}}$ is complete and decidable but not categorical. Moreover we discuss basic properties of models of $CET^{\omega}_{\mathcal{L}}$.

The following theorem is the central theorem in our investigations of $CET^{\omega}_{\mathcal{L}}$. Its proof is related to the method of elimination of quantifiers. In typical elimination

of quantifiers, one shows that every formula is equivalent to a Δ_0-formula. This is not true in the case of $CET^{\omega}_{\mathcal{L}}$, and instead, we show that in a certain conservative extension of $CET^{\omega}_{\mathcal{L}}$, every formula is equivalent to a Σ_1-formula.

Theorem 3.5.1 (Decidability and completeness of $CET^{\omega}_{\mathcal{L}}$)

Let $\mathcal{L}$ be a language containing no predicate symbols except $=$. Then the following hold:

1. $CET^{\omega}_{\mathcal{L}}$ is decidable.
2. $CET^{\omega}_{\mathcal{L}}$ is complete.
3. $CET^{\omega}_{\mathcal{L}} = Th(U^{\omega}_{\mathcal{L}})$.

Notice that the assumption of the theorem is essential. If $\mathcal{L}$ contained predicate symbols different from $=$, $CET^{\omega}_{\mathcal{L}}$ restricted to formulas without $=$, would coincide with the predicate calculus, hence it would be neither complete nor decidable. As the next theorem shows, if we limit our attention to positive formulas with guards, we can admit predicate symbols other than $=$.

Theorem 3.5.2 Let $\mathcal{L}$ be a language with equality. Then for positive formulas with guards B in $\mathcal{L}$, it is decidable whether $CET^{\omega}_{\mathcal{L}} \vdash B$.

Example 3.5.3 Let $\mathcal{L}$ be a language with equality. According to part 3 of theorem 3.5.1, $CET^{\omega}_{\mathcal{L}}$ contains exactly those sentences that are true in $Th(U^{\omega}_{\mathcal{L}})$. This implies that $U^{\omega}_{\mathcal{L}}$ is a submodel of every normal model for $CET^{\omega}_{\mathcal{L}}$. Thus one may ask whether there are normal models for $CET^{\omega}_{\mathcal{L}}$, strictly bigger than $U^{\omega}_{\mathcal{L}}$. The answer is 'yes': consider a language $\mathcal{L}$ containing one unary function symbol f and the symbol

=. The interpretation consists of the ω-Herbrand universe $U^{\omega}_{\mathcal{L}}$ and countably many additional elements e_i , i being an integer (negative, 0, or positive), with f interpreted as $f(e_i)=e_{i+1}$. The interpretation defined in this way is a model for $CET^{\omega}_{\mathcal{L}}$. This shows that the theory $CET^{\omega}_{\mathcal{L}}$ is not categorical. In general, if we consider other languages possibly containing function symbols of greater arities, normal models for $CET^{\omega}_{\mathcal{L}}$ may contain, besides a standard $U^{\omega}_{\mathcal{L}}$ part, 'trees infinite in both directions'. Notice however that axiom 8 assures that no model for $CET^{\omega}_{\mathcal{L}}$ contains 'loops' like $e_0, e_1, \ldots, e_{n-1}$ with $f(e_i)=e_{(i+1)\mathrm{mod}(n)}$.

We end this section with a conjecture suggesting that some interesting model-theoretical results concerning $CET^{\omega}_{\mathcal{L}}$ may be obtained in future.

Conjecture 3.5.4 Let $\mathcal{L}$ be language containing no predicate symbols except = . Then:

1. $U^{\omega}_{\mathcal{L}}$ is a prime model for $CET^{\omega}_{\mathcal{L}}$.
2. $CET^{\omega}_{\mathcal{L}}$ is model-complete.

3.6 Guards and substitutions

In this section we consider substitutions guarded by conditions involving inequations, like:

$$\{f(x_3)/x_1,\ c/x_2,\ x_4/x_3\} \text{ WHERE } (x_3 \neq x_4 \wedge \forall_y x_3 \neq f(y)).$$

Forms of canonical substitutions are determined by forms of canonical formulas of $CET^{\omega}_{\mathcal{L}}$. We interpret substitutions as subsets of (a power of) the ω-Herbrand universe, and prove that every substitution is equivalent under this interpretation to a union of finitely many canonical substitutions.

Definition 3.6.1 (Canonical guarded substitutions)

By a *guard* we understand any formula that does not contain predicate symbols other than $=$. By a *guarded substitution* we understand any expression (θ WHERE S) such that θ is a substitution and S is a guard. By a *canonical guarded substitution* we understand any guarded substitution (θ WHERE S) such that:

1. θ is $\{t_i/x_i \mid i \in I\}$
2. S is either $\top$ or $\bigwedge_{j\in J} y_j \neq t'_j \wedge \bigwedge_{(f,k)\in K} \neg is_f(y_k)$
3. $y_j \notin \mathrm{var}(t'_j)$ for $j = 1, \ldots, m$
4. $\mathrm{var}(S) \subseteq \bigcup_{i\in I} \mathrm{var}(t_i)$.

By a *super-canonical guarded substitution* we understand any guarded substitution that is guarded either by $\top$ or by a formula satisfying the conditions above, with each t'_j being a variable.

Conventional substitutions can be composed and can be applied to formulas. The next definition extends these concepts to guarded substitutions.

Definition 3.6.2

1. The result (θ_1 WHERE S_1)(θ_2 WHERE S_2) of composing guarded substitutions (θ_1 WHERE S_1) and (θ_2 WHERE S_2) is defined as ($\theta_1\theta_2$ WHERE $((S_1\theta_2) \wedge S_2)$).
2. The result $B(\theta$ WHERE $S)$ of applying guarded substitution (θ WHERE S) to a formula B is defined as $B\theta \vee \neg S$.

Notice that $B(\theta$ WHERE $S)$ is equivalent to $S \rightarrow (B\theta)$; we did not use this formula in the definition of $B(\theta$ WHERE $S)$ because we wanted to achieve the fol-

lowing effect: If B is a positive formula with guards, so is $B(\theta \text{ WHERE } S)$. As $B(\theta \text{ WHERE } \top) \equiv B\theta$ we see that the notion of a guarded substitution generalizes that of a substitution. As $(\theta \text{ WHERE } \top)$ can be identified with just θ, all further results concerning guarded substitutions will apply to conventional substitutions as well. Notice however that, unlike conventional substitutions, guarded substitutions do not always commute with boolean operations: we have:

$(B_1 \wedge B_2)(\theta \text{ WHERE } S) \equiv B_1(\theta \text{ WHERE } S) \wedge B_2(\theta \text{ WHERE } S)$,

$(B_1 \rightarrow B_2)(\theta \text{ WHERE } S) \equiv B_1(\theta \text{ WHERE } S) \rightarrow B_2(\theta \text{ WHERE } S)$,

$(\neg B)(\theta \text{ WHERE } S) \not\equiv \neg(B(\theta \text{ WHERE } S))$,

$(B_1 \vee B_2)(\theta \text{ WHERE } S) \equiv B_1(\theta \text{ WHERE } S) \vee B_2(\theta \text{ WHERE } S)$.

Notice also that if the language $\mathcal{L}$ of B does not contain the symbol $=$, then $B(\theta \text{ WHERE } S)$ is a formula in the extended language $\mathcal{L}^{=}$.

Now we introduce notation which allows us to think of terms and substitutions as subsets of (a power of) the ω-Herbrand universe.

Definition 3.6.3 (Terms and substitutions as subsets of $U^\omega_\mathcal{L}$)

Let t, t_1, t_2 be terms of a language $\mathcal{L}$. Let $\theta = \{t_i/x_i \mid i < \omega\}$ be a substitution in $\mathcal{L}$, and let $\tilde{\theta} = \langle t_0, t_1, t_2, \ldots \rangle$. We define:

1. $[\![t]\!] = \{t[\nu] \in U^\omega_\mathcal{L} \mid \nu : Var \longrightarrow U^\omega_\mathcal{L}\}$.

2. $t_1 \leq t_2$ iff $[\![t_1]\!] \subseteq [\![t_2]\!]$.

3. $[\![\theta]\!] = \{\tilde{\theta}[\nu] \in (U^\omega_\mathcal{L})^\omega \mid \nu : Var \longrightarrow U^\omega_\mathcal{L}\}$.

4. $[\![\theta \text{ WHERE } S]\!] = \{\tilde{\theta}[\nu] \in (U^\omega_\mathcal{L})^\omega \mid U^\omega_\mathcal{L} \models S[\nu]\}$.

For instance: $[\![f(c_1, g(c_2))]\!] = \{f(c_1, g(c_2))\}$, $[\![x]\!] = U^\omega_\mathcal{L}$ and $[\![g(x)]\!] = \{g(e) \mid e \in U^\omega_\mathcal{L}\}$.

Sometimes we may write e.g. $[\![\theta_1]\!] \subseteq [\![\theta_2]\!]$ without specifying the language. Such a little departure from the formal style does not lead to any problems: if θ_1, θ_2 are substitutions in language $\mathcal{L}$ and if $\mathcal{L} \subseteq \mathcal{L}'$, then the relation $[\![\theta_1]\!] \subseteq [\![\theta_2]\!]$ does not depend on whether we interpret the substitutions as subsets of $U^\omega_\mathcal{L}$ or of $U^\omega_{\mathcal{L}'}$.

The choice of ω-Herbrand models in the definition above is essential. One can see that even in the case of a language with a single individual constant c and no function symbols, $[\![c]\!] \neq [\![x]\!]$. There wouldn't be any difference between $[\![c]\!]$ and $[\![x]\!]$ if one defined this notion using conventional Herbrand models.

Example 3.6.4

1. $t_1 \leq t_2$ iff there exists a substitution θ such that $t_1 = t_2\theta$.
2. If t is the result of a most general unification of t_1 and t_2, then $[\![t_1]\!] \cap [\![t_2]\!] = [\![t]\!]$.
3. $[\![\theta \text{ WHERE } \top]\!] = [\![\theta]\!]$.
4. $[\![\theta \text{ WHERE } S]\!] \subseteq [\![\theta]\!]$.
5. If $CET^\omega_\mathcal{L} \vdash S_1 \rightarrow S_2$ then $[\![\theta \text{ WHERE } S_1]\!] \subseteq [\![\theta \text{ WHERE } S_2]\!]$.
6. If $[\![\theta \text{ WHERE } S]\!]$ is a canonical guarded substitution then $[\![\theta \text{ WHERE } S]\!] \neq \emptyset$.
7. $[\![\theta_1]\!] \subseteq [\![\theta_2]\!]$ iff there exists a substitution θ such that $\theta_1 = \theta_2\theta$.

   (which means that θ_2 is a more general substitution than θ_1.)

Theorem 3.6.5 (Strong compactness)

1. If $[\![t]\!] \subseteq \bigcup_{i \in I} [\![t_i]\!]$ then there exists $i_0 \in I$ such that $[\![t]\!] \subseteq [\![t_{i_0}]\!]$.
2. If $[\![\theta]\!] \subseteq \bigcup_{i \in I} [\![\theta_i]\!]$ then there exists $i_0 \in I$ such that $[\![\theta]\!] \subseteq [\![\theta_{i_0}]\!]$.

One can see that this theorem does not generalize to guarded substitutions.

The following obvious lemma allows us to apply knowledge about normal forms in $CET^{\omega}_{\mathcal{L}}$ to guarded substitutions.

Lemma 3.6.6 Let $\theta=\{t_i/x_i \mid i \in I\}$ be a substitution and let S be a guard. Then:

$$[\![\theta \text{ WHERE } S]\!] = \{\tilde{\nu} \in (U^{\omega}_{\mathcal{L}})^{\omega} \mid U^{\omega}_{\mathcal{L}} \models \exists y(\bigwedge_{i\in I} x_i{=}t_i(y) \wedge S(y))[\nu]\}$$

Theorem 3.6.7 (Normal form theorem I)

For every guarded substitution (θ WHERE S), there exists a finite number of canonical guarded substitutions $(\theta_i \text{ WHERE } S_i) : i \leq k$ such that

$[\![\theta \text{ WHERE } S]\!] = \bigcup_{i\leq k}[\![\theta_i \text{ WHERE } S_i]\!]$. (If $k{=}0$ the right side of this equality is interpreted as $\emptyset$). Moreover substitutions $(\theta_i \text{ WHERE } S_i) : i \leq k$ can be effectively obtained from (θ WHERE S).

Corollary 3.6.8 It is decidable whether $[\![\theta \text{ WHERE } S]\!] \neq \emptyset$.

Theorem 3.6.9 (Normal form theorem II)

For every guarded substitution θ WHERE S, *with S being a positive formula*, there exists a finite number of substitutions $\theta_i : i \leq k$ such that $[\![\theta \text{ WHERE } S]\!] = \bigcup_{i\leq k}[\![\theta_i]\!]$ (If $k = 0$ the right side of this equality is interpreted as $\emptyset$). Moreover substitutions $\theta_i : i \leq k$ can be effectively obtained from (θ WHERE S).

Conjecture 3.6.10 For every canonical guarded substitution (θ WHERE S), there exists a finite or countable collection of *super*-canonical substitutions such that $[\![\theta \text{ WHERE } S]\!] = \bigcup_{i\in I}[\![\theta_i \text{ WHERE } S_i]\!]$. Moreover substitutions $(\theta_i \text{ WHERE } S_i) : i \in I$ can be effectively obtained from (θ WHERE S).

3.7 Computing guarded substitutions

The theorems of this subsection embody a preparatory step towards using the equality theory to generate computed answer substitutions in resolution systems. Applications will follow in the next chapter.

Theorem 3.7.1 (Computing substitutions I)

For positive formulas with guards it is decidable whether $CET^{\omega}_{\mathcal{L}} \vdash \exists_{\boldsymbol{x}} B(\boldsymbol{x})$. Moreover there exists an algorithm that takes as input a positive formula with guards $B(\boldsymbol{x})$, and returns as output either the statement: "$CET^{\omega}_{\mathcal{L}} \vdash \neg\exists_{\boldsymbol{x}} B(\boldsymbol{x})$" if it is correct, or otherwise returns a finite sequence $(\theta_i \text{ WHERE } S_i) : i \leq k$ of canonical guarded substitutions, such that if $CET^{\omega}_{\mathcal{L}} \vdash B(\boldsymbol{x})(\theta \text{ WHERE } S)$ then $[\![\theta \text{ WHERE } S]\!] \subseteq \bigcup_{i \leq k} [\![\theta_i \text{ WHERE } S_i]\!]$.

If we restrict our attention to positive formulas B we may obtain not only a more efficient algorithm but also a somewhat stronger result.

Theorem 3.7.2 (Computing substitutions II)

For *positive formulas* $B(\boldsymbol{x})$, it is decidable whether $CET^{\omega}_{\mathcal{L}} \vdash \exists_{\boldsymbol{x}} B(\boldsymbol{x})$. Moreover there exists an algorithm that takes as input a positive formula $B(\boldsymbol{x})$, and returns as output either the statement: "$CET^{\omega}_{\mathcal{L}} \vdash \neg\exists_{\boldsymbol{x}} B(\boldsymbol{x})$" if it is correct, or otherwise returns a finite sequence $\theta_i : i \leq k$ of substitutions, such that if $CET^{\omega}_{\mathcal{L}} \vdash B(\boldsymbol{x})\theta$ then $[\![\theta]\!] \subseteq [\![\theta_i]\!]$ for certain $i \leq k$.

The algorithms are given in the proofs, in section 7.7. Problems of efficiency

of algorithms, mentioned in the theorems above, will be discussed together with applications of these theorems in section 4.2.

3.8 Positive programs with guards

In this section we extend the syntax of logic programs, allowing positive formulas (with $\forall$ and $\exists$) and guards in clauses bodies. The hierarchy of such generalized programs will be studied in the next sections.

In Prolog programming practice one sometimes uses clauses like

$$p(x,y) \leftarrow x \neq y \wedge q(x,y)$$

calling the formula $x \neq y$ a guard in the clause. The following definition generalizes the concept of a clause as well as the concept of a guard. Another important notion will be that of a positive formula with guards — by that we understand a formula obtained from a positive formula by inserting in it (possibly negative) guards. So positive formulas with guards contain some (but not arbitrary) negative information. We will see that they share many properties with the class of conventional positive formulas.

Definition 3.8.1 (Programs with guards)

By a *guard* we understand any formula containing no predicates except $=$.

By a *positive formula with guards* we mean any formula $B(S_1/p_1, \ldots, S_n/p_n)$ where $B(p_1, \ldots, p_n)$ contains the predicate variables $p_1, \ldots, p_n$ but not $=$, and $S_1, \ldots, S_n$ are guards. (Notice we do not require that $S_1, \ldots, S_n$ are positive!)

The *hierarchy of positive formulas with guards* is defined by imposing restrictions on $B(p_1,\ldots,p_n)$ and on the guards $S_1,\ldots,S_n$. If B is a positive Π_1-formula and $S_1,\ldots,S_n$ are Δ_0-formulas, then the resultiong formula is called a $+\Pi_1$-*formula with* Δ_0-*guards.* If B is a positive Σ-formula and $S_1,\ldots,S_n$ are arbitrary guards, then the resultiong formula is called a $+\Sigma$-*formula with* Δ_ω-*guards.* If B is a positive formula and $S_1,\ldots,S_n$ are positive formulas, then the resultiong formula is called a $+\Delta_\omega$-*formula with* $+\Delta_\omega$-*guards.* Other levels of the hierarchy are defined in a similar way.

By a *statement with guards* we understand any formula $A \leftarrow B$ such that:

1. A is an atomic formula other than $\top, \bot$ or $t'=t''$,
2. B is arbitrary formula, possibly with $=$,
3. $var(B) \subseteq var(A)$.

By a *program with guards* we understand any finite set of statements with guards.

By a *positive statement with guards* we understand any statement $A \leftarrow B$ where B is a positive formula with guards. By a *positive program with guards* we understand any program with guards consisting of positive statements with guards.

A *hierarchy of positive statements with guards* is defined by imposing restrictions on bodies of the statements. If B is a $+\Pi_1$-formula with Δ_0-guards, then the statement $A \leftarrow B$ is called a $+\Pi_1$-*statement with* Δ_0-*guards.* If B is a $+\Sigma$-formula with Δ_ω-guards, then the statement $A \leftarrow B$ is called a $+\Sigma$-*statement with* Δ_ω-*guards.* If B is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards, then the statement $A \leftarrow B$ is called a $+\Delta_\omega$-*statement with* $+\Delta_\omega$-*guards.*

Other levels of the hierarchy, and the *hierarchy of positive programs with guards* are defined in the obvious way.

Allowing negative guards in otherwise positive clauses bodies is a way of incorporating negative information into logic programs. This is one of the reasons for introducing guards. Further discussion can be found in section 4.3.

Notice that the definite clause $p(x) \leftarrow q(x,y)$ is not considered a statement. It becomes a statement if we rewrite it as $p(x) \leftarrow \exists_y q(x,y)$.

The assumption of finiteness of program P, natural from the point of view of programming practice, will be essential in the next definition. However many results do not depend on this assumption.

Definition 3.8.2 Let P be a program with guards.

1. P is said to be in *Clark's form* if for every predicate p in in its language, program P contains exactly one clause defining this predicate: $p(\boldsymbol{x}) \leftarrow B$ such that $var(B) \subseteq var(p(\boldsymbol{x}))$.

2. *Clark's transformation* P^C of program P is constructed in the following way: Transform every statement: $p(\boldsymbol{t}) \leftarrow B$ in P to: $p(\boldsymbol{x}) \leftarrow \exists_{\boldsymbol{y}}((\boldsymbol{x} = \boldsymbol{t}) \wedge B)$, where all the variables in the sequence $\boldsymbol{x}$ are distinct, and where $\boldsymbol{y}$ is the sequence of all free variables that occur in $(\boldsymbol{x} = \boldsymbol{t}) \wedge B$ but do not occur in $p(\boldsymbol{x})$. Then for every predicate p, list all statements defining this predicate:

$$p(\boldsymbol{x}) \leftarrow B_1, \ldots, p(\boldsymbol{x}) \leftarrow B_n,$$

and form $p(\boldsymbol{x}) \leftarrow B_1 \vee \ldots \vee B_n$. If there are no clauses defining p, form $p(\boldsymbol{x}) \leftarrow \bot$.

Program P^C consists of the formulas obtained in this way.

3. *Clark's completion*[5] of program P is defined as:

$$Comp(P) = \{(p(\boldsymbol{x}) \equiv B) \mid (p(\boldsymbol{x}) \leftarrow B) \in P^C\}.$$

Notice that if the language of P does not contain the symbol $=$, then P^C is a positive program in the extended language $\mathcal{L}^=$.

Proposition 3.8.3 For any positive program P with guards:

1. P^C is in Clark's form.
2. P^C is a positive program with guards.
3. P and P^C are equivalent in classical logic.
4. $Comp(P)$ and $Comp(P^C)$ are equivalent in classical logic.
5. $T_P^\omega \uparrow \alpha = T_{PC}^\omega \uparrow \alpha$, for any ordinal α.

3.9 Fix-points

In this section we consider in detail two classes of generalized programs, namely the class of positive programs with positive guards and the class of positive existential programs with arbitrary guards. For programs of these classes we prove that the associated operators on lattices of ω-Herbrand interpretations, despite being not continuous, reach their least fix-points in ω steps. This indicates a great difference between ω-Herbrand models and conventional Herbrand models. The result constitutes a basis for completeness of

[5]Unlike in the original definition we do not assume that $Comp(P)$ contains CET.

generalizations of SLD-resolution, studied in the next chapter.

Definition 3.9.1 Let P be a program with guards. We define an operator $T_P^\omega : 2^{B_P^\omega} \longrightarrow 2^{B_P^\omega}$ on the lattice of ω-Herbrand interpretations. For any $I \subseteq B_P^\omega$:

$T_P^\omega(I) = \{A[\nu] \in B_P^\omega \mid (A \leftarrow B) \in P \text{ and } I \models B[\nu]\}$.

Proposition 3.9.2 (Monotonicity of T_P^ω)

Let P be a positive program with guards. Then:

1. T_P^ω is monotone: $I_1 \subseteq I_2$ implies $T_P^\omega(I_1) \subseteq T_P^\omega(I_2)$

2. $\alpha_1 \leq \alpha_2$ implies $T_P^\omega \uparrow \alpha_1 \subseteq T_P^\omega \uparrow \alpha_2$.

Example 3.9.3 For a positive program P the operator T_P^ω need not be either compact or continuous (even if P contains no guards). Let us consider the program P, consisting of one statement, in a language with predicate symbols p, q and one unary function symbol f:

$$q \leftarrow \forall_x p(x)$$

Recall that $k_i : i < \omega$ are free constants used to form ω-Herbrand universe. We define a directed family of interpretations. For every natural number n, let I_n be $\{p(f^m(k_i)) \in B_P^\omega \mid m < n, i < \omega\}$. We have $q \in T_P^\omega(\bigcup I_n)$ but $q \notin \bigcup T_P^\omega(I_n)$. So $T_P^\omega(\bigcup I_n) \neq \bigcup T_P^\omega(I_n)$, which shows that T_P^ω is not continuous. For monotone operators on lattices of subsets, the notions of continuity and compactness coincide, so T_P^ω is not compact either. This can also be seen directly. Let I be $\{p(f^m(k_i)) \in B_P^\omega \mid m < \omega, i < \omega\}$. We have $q \in T_P^\omega(I)$, but there is no finite $I_0 \subseteq I$ such that $q \in T_P^\omega(I_0)$.

A continuous operator on a complete lattice reaches its least fix point in ω steps. Since for positive programs (involving $\forall$) T_P^ω need not be continuous, this argument can not be applied. Before we approach the question of whether $lfpT_P^\omega = T_P^\omega\uparrow\omega$ let us look at the situation with the operator T_P on the lattice of conventional Herbrand interpretations.

Example 3.9.4 Consider the following positive program P (with no guards):

$p(c) \leftarrow \top$

$p(f(x)) \leftarrow p(x)$

$q \leftarrow \forall_x p(x)$

and the associated operator $T_P : 2^{B_\mathcal{L}} \longrightarrow 2^{B_\mathcal{L}}$ on the lattice of conventional Herbrand interpretations. We have $q \in lfpT_P = B_\mathcal{L}$ while $q \notin T_P\uparrow\omega = \{p(f^m(c)) \mid m < \omega\}$. So $lfpT_P \neq T_P\uparrow\omega$.

As the next theorem shows, the situation is very different for the operator T_P^ω on the lattice of ω-Herbrand interpretations.

Theorem 3.9.5 (Least fix-point theorem)

Assume one of the following:

(i) P is a $+\Delta_\omega$-program with $+\Delta_\omega$-guards in $\mathcal{L}$,

B is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards in $\mathcal{L}$.

(ii) P is a $+\Sigma$-program with Δ_ω-guards in $\mathcal{L}$,

B is a $+\Sigma$-formula with Δ_ω-guards in $\mathcal{L}$.

If $\mathcal{L}$ is a language with equality, then the following conditions are equivalent.

1. $T_P^\omega\uparrow\omega \models B$

2. There exists $n < \omega$ such that $T_P^\omega\uparrow n \models B$

3. $lfpT_P^\omega \models B$

4. $CET_{\mathcal{L}}^\omega \cup P \vdash B$

5. $CET_{\mathcal{L}}^\omega \cup Comp(P) \vdash B$.

If $\mathcal{L}$ is a language without equality, each of 1, 2, 3 is equivalent to either of:

4'. $P \vdash B$

5'. $Comp(P) \vdash B$.

Notice that the equivalence of conditions 1 and 3 implies that $lfpT_P^\omega = T_P^\omega\uparrow\omega$.

The following example shows that this theorem does not generalize to other levels in the hierarchy of positive programs with guards.

Example 3.9.6 Consider the following $+\Pi_1$-program with Δ_0-guards P:

$p(x) \leftarrow \forall_y q(x, f(y))$

$p(f(x)) \leftarrow p(x)$

$q(x, y) \leftarrow x \neq y$

$r \leftarrow \forall_x p(x)$

Consider the operator T_P^ω on the lattice of ω-Herbrand interpretations.

We have: $r \notin T_P^\omega\uparrow\omega$ and $r \in lfpT_P^\omega = T_P^\omega\uparrow\omega + 1$. So $lfpT_P^\omega \neq T_P^\omega\uparrow\omega$.

We have $T_P^\omega\uparrow\omega \models \forall_x p(x)$, but for no $n < \omega$, $T_P^\omega\uparrow n \models \forall_x p(x)$.

4 FIRST-ORDER RESOLUTION SYSTEMS

In this chapter we define SLPG-resolution — an extension of SLD-resolution, which is capable of handling positive programs with guards (involving $\forall, \exists, =, \neq$). We prove completeness of SLPG for two important classes of such programs. Then using the idea of constructive negation, we extend this resolution to SLPGCN, which can handle programs with negation. Finally we give an example suggesting that incompleteness of SLDNF-resolution is in part a consequence of certain inherent properties of the class of normal programs, and one should not expect that another resolution system, complete for normal programs, may be defined. Other reasons for the incompleteness of SLDNF-resolution can be corrected — just this is done in defining SLPGCN and the notion of constructive completion of a program.

4.1 Completeness versus lifting lemma

In this section we give new formulations of completeness and of the lifting lemma and discuss their relationship. The discussion points out some deficiencies of SLDNF-resolution and of Prolog's resolution, which are obstacles on the way to a completeness theorem.

In section 3.6 we introduced the notation $[\theta]$, allowing us to think about substitutions as subsets of (a power of) ω-Herbrand universe. Now, we will use this notion in an analysis of resolution systems. For SLD-resolution, definite program P and atomic formula A, we may define the *SLD-answer set* as: $SLD(P, A) = \bigcup_{i \in I} [\theta_i]$,

where $\theta_i : i \in I$ is the collection of all SLD-computed answers for $P \cup \{\leftarrow A\}$. Similarly for SLDNF-resolution, normal program P and literal L, the *SLDNF-answer set* is: $SLDNF(P, L) = \bigcup_{i \in I}[\theta_i]$, where $\theta_i : i \in I$ is the collection of all SLDNF-computed answers for $P \cup \{\leftarrow L\}$. Notice that notions of SLD- and SLDNF-resolution sets are finitary — the answer set is determined by a finite number of substitutions, each substitution being non-identical only on finitely many variables. The next definition generalizes these notions.

Definition 4.1.1 Let *Resolution* be a resolution system which, if given a program with guards P and a goal with guards $\leftarrow B$, returns a sequence of computed answer substitutions $(\theta_i \text{ WHERE } S_i) : i \in I$. Then by *Resolution-answer set* for $P \cup \{\leftarrow B\}$ we understand the set: $Resolution(P, B) = \bigcup_{i \in I}[\theta_i \text{ WHERE } S_i]$.

Soundness and completeness of SLD-resolution are usually[6] formulated as two separate implications.

Soundness:

If θ is an SLD-computed answer for $P \cup \{\leftarrow A\}$, then $Comp(P) \vdash A\theta$

Completeness:

If $Comp(P) \vdash A\theta$ then there exists a substitution θ', more general than θ, which is an SLD-computed answer for $P \cup \{\leftarrow A\}$.

These statements could not be combined into one equivalence because of a technical difficulty with the "substitution θ' that is more general than θ". Using answer sets

[6]perhaps using just P instead of $Comp(P)$

we may formulate soundness and completeness conveniently as:

$$SLD(P, A) \supseteq [\![\theta]\!] \quad \text{iff} \quad Comp(P) \vdash A\theta.$$

Remark 4.1.2 Notice that "substitution θ' that is more general than θ" means: "there exists θ'' such that $\theta = \theta'\theta''$" and can be expressed as: $[\![\theta]\!] \subseteq [\![\theta']\!]$. Now we can see that by strong compactness 3.6.5 (7.6.1), the statement:

$$Resolution(P, B) \supseteq [\![\theta]\!] \text{ iff } Completion(P) \vdash B\theta$$

is equivalent to:

If θ is a *Resolution*-computed answer for $P \cup \{\leftarrow B\}$ then $Completion(P) \vdash B\theta$.

If $P \vdash B\theta$ then there exists a substitution θ' more general than θ,

which is an SLD-computed answer for $P \cup \{\leftarrow B\}$.

Another important property of SLD-resolution is expressed by the lifting lemma:

$$SLD(P, A) \supseteq [\![\theta]\!] \quad \text{iff} \quad SLD(P, A\theta) \supseteq [\![\epsilon]\!].$$

Informally this can be understood as: "SLD-resolution is capable of computing answer θ, exactly when it accepts θ as appropriate". Indeed by 3.6.5 (7.6.1), $SLD(P, A) \supseteq [\![\theta]\!]$ means that θ can be computed, and $SLD(P, A\theta) \supseteq [\![\epsilon]\!]$ means that given $P \cup \{\leftarrow A\theta\}$ SLDNF returns the identity substitution, i.e. *YES*. We belive that this formulation of the lifting lemma is interesting for its own sake, independently of applications in proofs of other theorems. The next remark explains how this version is related to conventional one.

Remark 4.1.3 The lifting lemma is usually formulated as (cf. [44], lm. 8.2 p. 47):

1. If θ' is a *Resolution*-computed answer for $P \cup \{\neg B\theta\}$,

then there exists a substitution θ'' mare general than $\theta\theta'$,

which is a *Resolution*-computed answer for $P \cup \{\neg B\}$.

This can be reformulated as:

2. If $Resolution(P, B\theta) \supseteq [\![\theta']\!]$, then $Resolution(P, B) \supseteq [\![\theta\theta']\!]$

Implication 1 $\Rightarrow$ 2 is obvious, implication 2 $\Rightarrow$ 1 results from strong compactness 3.6.5. Our version:

3. $Resolution(P, B) \supseteq [\![\theta]\!]$ iff $Resolution(P, B\theta) \supseteq [\![\epsilon]\!]$

is stronger. 3 implies 2.

In this chapter we are interested resolution systems capable of handling programs more general then definite programs, say programs involving $\forall$ or $\neg$. If we look for a completeness result in a form analogous to that for SLD-resolution:

(*) $Resolution(P, B) \supseteq [\![\theta]\!]$ iff $Completion(P) \vdash_{\mathbf{L}} B\theta$

we expect to have a lifting lemma analogous to the one for SLD:

(**) $Resolution(P, B) \supseteq [\![\theta]\!]$ iff $Resolution(P, B\theta) \supseteq [\![\epsilon]\!]$

because (in any reasonable logic L) lifting lemma (**) is a consequence of completenss (*). Indeed:

$Resolution(P, B) \supseteq [\![\theta]\!]$ iff

$Completion(P) \vdash_{\mathbf{L}} B\theta$ iff

$Completion(P) \vdash_{\mathbf{L}} B\theta\epsilon$ iff

$Resolution(P, B\theta) \supseteq [\![\epsilon]\!]$.

A serious deficiency of SLDNF-resolution or of Prolog's resolution is that they do not

satisfy the lifting lemma (**). This is an obstacle on the way to completeness.

Example 4.1.4 The following program P shows that first order SLDNF-resolution does not satisfy the lifting lemma (**):

$p(x) \leftarrow \neg q(x)$

$q(a) \leftarrow \bot$

$q(b) \leftarrow \top$

Every derivation of SLDNF for $P, \leftarrow p(x)$ flounders (i.e. halts without returning an answer due to the safeness condition) because an open negative subgoal is selected. So $SLDNF(P, p(x)) = \emptyset$. On the other hand $SLDNF(P, p(a)) = [\epsilon] = YES$. The source of the problem is the lack of symmetry in the way SLDNF-resolution treats positive and negative goals; it makes substitutions in positive but not in negative goals. So to obtain completeness results we have to find a generalization of SLDNF that never flounders, can process negative subgoals making substitutions and, if it stops, always returns an answer.

Example 4.1.5 One might hope that some versions of SLDNF-resolution with a relaxed safeness condition may admit the lifting lemma. We repeat the following description after [44], (p. 94).

"It is possible to weaken the safeness condition a little and still obtain ... [soundness]. Consider the following weaker safeness condition. Non-ground negative subgoals are allowed to proceed. If the negative subgoal succeeds, then we proceed as before. However, if the negative subgoal fails, a check is made to make sure no bindings were made to any variables in the top-level goal of the corresponding refutation.

If no such binding was made, then the negative subgoal is allowed to fail and we proceed as before. But, if such a binding was made, then a different literal is selected and the negative subgoal is delayed in the hope that more of its variables will be bound later. [If we are left with only negative literals, neither of which can be processed, the system halts without returning an answer - flounders.]"

Call such a resolution SLDNF^{-}. SLDNF^{-} is a generalization of SLDNF-resolution: $SLDNF(P,L) \subseteq SLDNF^{-}(P,L)$. Unfortunately there is no chance for completeness (*) for SLDNF^{-}, either. The argument given in previous example applies to SLDNF^{-} as well, and shows that this system does not satisfy the lifting lemma (**).

Example 4.1.6 Finally let us see what happens if we drop the safeness condition entirely. Let SLDNF^{--} be a version of SLDNF-resolution without the safeness condition — a version in which open negative subgoals are processed as well as ground ones. SLDNF^{--} is an idealization of Prolog's resolution. (Notice however that in Prolog not only safeness is dropped but also, for pragmatic reasons, unification is done without the occurs check, and a deterministic depth first search is implemented. SLDNF^{--} doesn't make these further concessions — its unification is correct and selection of subgoals and input clauses in derivations is non-deterministic.) Consider the program P:

$p(x) \leftarrow \neg q(x)$

$q(a) \leftarrow \bot$

$q(b) \leftarrow \top$

$SLDNF^{--}(P,p(x)) = \emptyset = NO$ and $SLDNF^{--}(P,p(a)) = [\epsilon] = YES$. So the lifting

lemma (**) does not hold for SLDNF^{--} either. As we see, dropping safeness from SLDNF doesn't yield completeness. In fact it even has some severe side effects. It is known that SLDNF^{--} is not sound (cf. [44] p. 93). Moreover different selection functions may give incompatible answers. Consider P':

$p(x) \leftarrow r(y) \wedge \neg q(x,y)$

$r(a) \leftarrow \top$

$q(x,a) \leftarrow \bot$

$q(x,b) \leftarrow \top$

and query: $\leftarrow p(x)$. If in SLDNF^{--} resolution literal $r(y)$ is selected before $\neg q(x,y)$ the answer is $\theta = \{a/x\}$; if $\neg q(x,y)$ is selected before $r(x)$ the answer is $\emptyset = NO$.

4.2 Positive programs with guards and SLPG-resolution

In this section we consider programs consisting of statements with arbitrary positive formulas (involving $\forall, \exists$) with guards (i.e. inequations) in the bodies. We define SLPG-resolution capable of handling programs of this sort, which is an extension of SLD-resolution. For SLPG-resolution, we prove soundness and a lifting lemma.

Positive programs with guards were defined in section 3.8. Now we add to that definition only the notion of a goal.

Definition 4.2.1 By a *positive goal with guards* we understand any formula $\leftarrow B$, where B is a positive formula with guards.

The next two definitions embody the description of SLPG-resolution. i.e. SL-resolution for Positive programs with Guards.

Definition 4.2.2 Let $\leftarrow B'$ and $\leftarrow B''$ be positive goals with guards in $\mathcal{L}$ and let $C : p(\boldsymbol{x}) \leftarrow B$ be Clark's form of a positive statement with guards. Then a goal $\leftarrow B''$ is said to be *SLPG-derived* from $\leftarrow B'$ and C using an mgu θ, if the following holds:

1. There is an occurrence $p(\boldsymbol{t})$ of an atomic formula in B', called the *selected atom.*
2. B'' is $B'[B(\boldsymbol{t}/\boldsymbol{x})/p(\boldsymbol{t})]$.
3. θ is $\boldsymbol{t}/\boldsymbol{x}$.

Definition 4.2.3 Let P be a positive program with guards, and let $G : \leftarrow B$ be a positive goal with guards.

1. An *SLPG-derivation* for $P \cup \{G\}$ consists of a (finite or infinite) sequence $G_0 = G, G_1, \ldots$ of positive goals with guards, a sequence $C_1, C_2, \ldots$ of variants of clauses of P^C (Clark's transformation), and a sequence $\theta_1, \theta_2, \ldots$ of mgu's, such that each G_{i+1} is SLPG-derived from G_i and C_{i+1} using θ_{i+1}.
2. An *SLPG-refutation* of $P \cup \{\leftarrow B\}$ (or a *successful SLPG-derivation*) is a finite SLPG-derivation that has as the last goal $\leftarrow B'(\boldsymbol{x})$, such that $CET^{\omega}_{\mathcal{L}} \vdash \exists_{\boldsymbol{x}} B'(\boldsymbol{x})$. Then any guarded substitution $(\theta' \text{ WHERE } S')$ such that $CET^{\omega}_{\mathcal{L}} \vdash B'(\theta' \text{ WHERE } S')$, is called an *SLPG-computed answer* for $P \cup \{\leftarrow B\}$.
3. If $(\theta_i \text{ WHERE } S_i) : i \in I$ is the collection of all SLPG-computed answers for $P \cup \{\leftarrow B\}$, then the *SLPG-answer set* for $P \cup \{\leftarrow B\}$ is defined as $SLPG(P, B) =$

$\bigcup_i [\theta_i$ WHERE $S_i]$.

SLPG is an example of a resolution system that works on formulas from the inside, without decomposing them into conventional clauses. In conventional resolution systems one first turns formulas into prenex normal form, skolemizes and then applies the resolution rule. In [25] Fitting has shown that skolemization can be incorporated into special decomposition rules, so that the preparation steps of transforming the formula to prenex form and skolemizing are not necessary. As one uses decomposition rules and the resolution rule, one skolemizes on the way. SLPG-resolution, presented in this section, uses a similar idea. However as we consider, not arbitrary sets of formulas, but positive programs with guards, additional decomposition rules are not necessary. We delay decomposition till the very last step and then use the equality theory instead.

In section 3.7 we proved that there exists an algorithm that takes as input a positive formula B such that $CET^{\omega}_{\mathcal{L}} \vdash \exists_{\boldsymbol{x}} B(\boldsymbol{x})$, and returns as output a finite sequence $(\theta_i$ WHERE $S_i) : i \in I$ of guarded substitutions, such that $CET^{\omega}_{\mathcal{L}} \vdash B(\theta$ WHERE $S)$ implies $[\theta$ WHERE $S] \subseteq [\theta_i$ WHERE $S_i]$ for some $i \in I$. This algorithm can be used to obtain computed answers, when implementing SLPG-resolution. Notice also that if P is a positive program ($+\Delta_\omega$-program with $+\Delta_\omega$-guards) and if B is a positive formula ($+\Delta_\omega$-formula with $+\Delta_\omega$-guards) then in the refutation of $P \cup \{\leftarrow B'\}$ the last goal $\leftarrow B'$ is positive and then the stronger result 3.7.2 can be used in the implementation. (For a certain restricted class of positive programs it is possible to eliminate the use of equality theory from the procedure of obtaining SLPG-computed answers. This

idea will be developed elsewhere.)

Example 4.2.4

1. Let program P be: $p(x,y) \leftarrow x \neq y$, and let goal G be $\leftarrow p(f(x), f(f(y)))$. In SLPG we can derive in one step $\leftarrow f(x) \neq f(f(y))$. As $CET^{\omega} \vdash f(x) \neq f(f(y))(\epsilon \text{ WHERE } x \neq f(y))$ guarded substitution $(\epsilon \text{ WHERE } x \neq f(y))$ is an SLPG-computed answer for $P \cup \{G\}$. As we are heading towards a soundness theorem, notice that $CET^{\omega} \cup Comp(P) \vdash p(f(x), f(f(y)))(\epsilon \text{ WHERE } x \neq f(y))$.

2. Consider the following program P:

   $q(x) \leftarrow \forall_y r(x,y)$

   $r(x,y) \leftarrow x \neq y \vee \exists_z y = f(z)$.

   Given goal $G :\leftarrow q(x)$, in two steps we can derive $\leftarrow \forall_y(x \neq y \vee \exists_z y = f(z))$. As $CET^{\omega} \vdash \forall_y(x \neq y \vee \exists_z y = f(z))(f(v)/x)$, the substitution $\{f(v)/x\}$ is an SLPG-computed answer for $P \cup \{G\}$. Looking forward to a soundness theorem, notice that $CET^{\omega} \cup Comp(P) \vdash q(x)(f(v)/x)$.

As the next proposition shows, SLPG-resolution is an extension of SLD-resolution.

Proposition 4.2.5 Let P be a definite program and let B be a definite goal. Let $P^{\exists}$ be the program obtained from P by adding to the body of every clause an existential quantifier binding those variables occurring free in the body but not in the head. Then $SLPG(P^{\exists}, B) = SLD(P, B)$.

Theorem 4.2.6 (Soundness of SLPG-resolution)

Let P be a positive program with guards and let $\leftarrow B$ be a positive goal with guards. Then $SLPG(P,B) \supseteq [\![\theta \text{ WHERE } S]\!]$ implies $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B(\theta \text{ WHERE } S)$.

In this theorem we could use just P instead of $Comp(P)$, but using $Comp(P)$ is closer to the way programmers think.

According to the considerations of the previous section, the following result gives a chance for completeness of SLPG-resolution.

Theorem 4.2.7 (Lifting lemma for SLPG-resolution)

Let P be a positive program with guards and let $\leftarrow B$ be a positive goal with guards. Then: $SLPG(P,B) \supseteq [\![\theta \text{ WHERE } S]\!]$ iff $SLPG(P, B(\theta \text{ WHERE } S)) \supseteq [\![\epsilon]\!]$.

Let us end this section with a remark concerning implementation. One can broaden the notion of an SLPG-derived goal, allowing us not only to replace an atom $p(t)$ by a modified body of a suitable statement $p(x) \leftarrow B(x)$, but allowing us also to replace an arbitrary positive subformula of the goal $\leftarrow B'$ by $\bot$. Such an operation will correspond to selecting this or another definite clause in an SLD-derivation. For instance, with goal $\leftarrow p(x)$ and clauses $p(a) \leftarrow \top$ and $p(b) \leftarrow q$, using one or another clause, we could derive in SLD either $\leftarrow \top$ with substitution a/x, or $\leftarrow q$ with substitution b/x. SLPG-derivation deals with clauses in Clark's form, so the two clauses $p(a) \leftarrow \top$ and $p(b) \leftarrow q$ is expressed as a single one: $p(x) \leftarrow (x{=}a \wedge \top) \vee (x{=}b \wedge q)$. In SLPG-derivation with the the same goal, selecting the subformula $x{=}a \wedge \top$ and

replacing it by $\bot$, corresponds to to using $p(b) \leftarrow q$ in the SLD-derivation, and allows us to derive $x{=}b \wedge q$. Selecting the subformula $x{=}b \wedge q$ and replacing it by $\bot$, corresponds to to using $p(a) \leftarrow \top$ in the SLD-derivation, and allows us to derive $x{=}a \wedge \top$. This extension does not affect soundness but allows us to end a refutation with a formula that contains no predicate symbols except $=$. This will yield a better organization of backtracking mechanisms in an implementation.

Concerning the efficiency of implementation, notice that SLPG-resolution is an extension of SLD-resolution, that works not only for definite programs but for arbitrary positive programs with guards. It seems possible to implement SLPG so that for definite programs it is as efficient as SLD. Of course processing programs with complex guards will be a little time-consuming. It should be left to the user to decide whether to write with some effort a very efficient definite program, or whether to write effortlessly a less efficient program in a broader language.

4.3 Two completeness results

In this section we consider two classes of programs: positive programs with positive guards, and positive existential programs with arbitrary guards, and motivate their importance. We showed earlier that operators associated with such programs reach their least fix-points in ω steps. Now we use this result to obtain completeness for SLPG-resolution. We also give an example showing that for other levels of the hierarchy of programs, SLPG is not complete.

A hierarchy of programs was defined in section 3.8. Now we augment that definition by introducing the hierarchy of goals.

Definition 4.3.1 A *hierarchy of positive goals with guards* is defined by imposing restrictions on the formula B in $\leftarrow B$. If B is a $+\Pi_1$-formula with Δ_0-guards, then the goal $\leftarrow B$ is called a $+\Pi_1$-*goal with* Δ_0-*guards.* If B is a $+\Sigma$-formula with Δ_ω-guards, then the goal $\leftarrow B$ is called a $+\Sigma$-*goal with* Δ_ω-*guards.* If B is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards, then the goal $\leftarrow B$ is called a $+\Delta_\omega$-*goal with* $+\Delta_\omega$-*guards.* Other levels of the hierarchy are defined analogously.

Before we start analyzing $+\Delta_\omega$-programs with $+\Delta_\omega$-guards and $+\Sigma$-programs with Δ_ω-guards let us consider the importance of these two classes for the theory and practice of logic programing.

Much research in the theory of logic programming has been devoted to analyzing completeness issues for SLDNF-resolution and normal programs i.e. programs with negative literals in clauses bodies. We argue that before such a task is undertaken, it is reasonable to investigate positive (quantified) programs.

Existential quantifiers are implicitly present in bodies of definite clauses:

$A(x) \leftarrow B(x, y)$

is interpreted as:

$A(x) \leftarrow \exists_y B(x, y)$.

If one allows normal programs, then $\neg$ together with $\exists$ gives rise to (implicit) universal

quantifiers. This can be fully justified only if we decide how to interpret normal clauses. As shown in Chapter 2, an interpretation that uses Clark's completion is too strong, but the interpretation with constructive negation seems to be appropriate. Consider the following program:

$$p \leftarrow \neg q(x)$$

$$q(x) \leftarrow \neg r(x,y)$$

$$r(x,y) \leftarrow \neg s(x,y,z)$$

With constructive negation the program is interpreted as:

$$p \leftarrow \exists_x \overline{q}(x) \qquad \overline{p} \leftarrow \forall_x q(x)$$

$$q(x) \leftarrow \exists_y \overline{r}(x,y) \qquad \overline{q}(x) \leftarrow \forall_y r(x,y)$$

$$r(x,y) \leftarrow \exists_z \overline{s}(x,y,z) \qquad \overline{r}(x,y) \leftarrow \forall_z s(x,y,z)$$

and the following formulas follow:

$$p \leftarrow \exists_x \forall_y \exists_z s(x,y,z) \qquad \overline{p} \leftarrow \forall_x \exists_y \forall_z \overline{s}(x,y,z).$$

One can see that arbitrarily complicated strings of quantifiers can be built in this way. Using $\wedge$ is allowed in clauses bodies. Also an implicit $\vee$ can be constructed by introducing several clauses with the same head and bodies corresponding to intended disjuncts. Thus, once we admit negation in clauses bodies, we in fact are programmming with statements $A \leftarrow B$, where B is an arbitrary quantified formula. Before completeness for such general programs is considered, one should be able to prove completeness for the simpler case of positive (quantified) programs. Therefore we will consider $+\Delta_\omega$-programs with $+\Delta_\omega$-guards.

The second class of programs, $+\Sigma$-programs with Δ_ω-guards, is a natural extension of the class of definite programs, which is of basic importance for programming practice. In Prolog one often uses clauses like:

$$p(x,y) \leftarrow x \neq y \wedge B(x,y)$$

calling $x \neq y$ a guard. Using guards is a way of incorporating negative information into an otherwise positive clause body. Guards have some technical applications. Consider the following program P:

$$p(x,x) \leftarrow B_1(x)$$

$$p(x,y) \leftarrow B_2(x,y).$$

A goal $p(t,t)$ can be resolved using either clause. Sometimes it is convenient to deal with generalized programs for which every goal can be resolved with at most one clause. To achieve such a situation we would like to rewrite P as:

$$p(x,x) \leftarrow B_1(x) \vee B_2(x,x)$$

$$p(x,y) \leftarrow x \neq y \wedge B_2(x,y).$$

Another example:

$$q(f(x)) \leftarrow B_3(x)$$

$$q(x) \leftarrow B_4(x)$$

could be transformed to:

$$q(f(x)) \leftarrow B_3(x) \vee B_4(f(x))$$

$$q(x) \leftarrow \forall_y(x \neq f(y)) \wedge B_4(x).$$

Guards like $\forall_y(x \neq f(y))$ go beyond the syntax of Prolog. Notice that $x \neq y$ means $\neg(x=y)$ and it is a *negative* literal; $\forall_y(x \neq f(y))$ is not a literal at all. Therefore

$p(x,y) \leftarrow x \neq y \wedge B_2(x,y)$

is not a definite clause;

$q(x) \leftarrow \forall_y(x \neq f(y)) \wedge B_4(x)$

is not a clause at all. Therefore conditions like $x \neq y$ or $\forall_y(x \neq f(y))$ deserve special treatment.

Theorem 4.3.2 (Soundness and completeness of SLPG-resolution)
Assume one of the following:

(i) P is a $+\Delta_\omega$-program with $+\Delta_\omega$-guards, and B is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards.

(ii) P is a $+\Sigma$-program with Δ_ω-guards, and B is a $+\Sigma$-formula with Δ_ω-guards.

Then:

$SLPG(P,B) \supseteq [\theta \text{ WHERE } S]$ iff $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B(\theta \text{ WHERE } S)$.

In this theorem we could use just P instead of $Comp(P)$, but using $Comp(P)$ is closer to the way programmers think.

It turns out that SLPG-resolution is complete for no levels of the hierarchy of positive programs with guards other than those considered in Theorem 4.3.2. This can be demonstrated by the same program that was used in Example 3.9.6.

Example 4.3.3 Consider the following $+\Pi_1$-program with Δ_0-guards P:

$p(x) \leftarrow \forall_y q(x, f(y))$

$p(f(x)) \leftarrow p(x)$

$q(x,y) \leftarrow x \neq y$

$r \leftarrow \forall_x p(x)$

We have: $CET_{\mathcal{L}}^{\omega} \cup Comp(P) \vdash r$ and $CET_{\mathcal{L}}^{\omega} \cup P \vdash r$ but $SLPG(P, r) \not\supseteq [\epsilon]$.

4.4 Programs with negation and SLPGCN-resolution

The characterization of propositional SLDNF-resolution in classical logic, presented in section 2.2, suggests the following alternative: instead of performing SLDNF-resolution on a program with negation P, using the idea of constructive negation transform P to a positive program and then apply SLD-resolution. In this section we generalize this idea to the first order level. We define SLPGCN-resolution and obtain completeness for a certain class of programs with negation. We also give an example suggesting that incompleteness of SLDNF-resolution is in part a consequence of inherent properties of the class of normal programs, and one should not expect that another resolution system, complete for normal programs, can be defined.

Definition 4.4.1 We define a *deMorgan operation* which transforms formula B into formula B^{deM}:

$A^{deM} = A$, for atomic formula A

$(\neg A)^{deM} = \neg A$, for atomic formula A different from $\top, \bot$

$(\neg\top)^{deM} = \bot$

$(\neg\bot)^{deM} = \top$

$(\neg\neg B)^{deM} = B^{deM}$

$(B_1 \wedge B_2)^{deM} = B_1^{deM} \wedge B_2^{deM}$

$(\neg(B_1 \wedge B_2))^{deM} = (\neg B_1)^{deM} \vee (\neg B_2)^{deM}$

$(B_1 \vee B_2)^{deM} = B_1^{deM} \vee B_2^{deM}$

$(\neg(B_1 \vee B_2))^{deM} = (\neg B_1)^{deM} \wedge (\neg B_2)^{deM}$

$(B_1 \rightarrow B_2)^{deM} = (\neg B_1)^{deM} \vee B_2^{deM}$

$(\neg(B_1 \rightarrow B_2))^{deM} = B_1^{deM} \wedge (\neg B_2)^{deM}$

$(\forall_x B)^{deM} = \forall_x B^{deM}$

$(\neg\forall_x B)^{deM} = \exists_x (\neg B)^{deM}$

$(\exists_x B)^{deM} = \exists_x B^{deM}$

$(\neg\exists_x B)^{deM} = \forall_x (\neg B)^{deM}$

For any set Γ of formulas, we define $\Gamma^{deM} = \{B^{deM} \mid B \in \Gamma\}$.

Notice that in B^{deM} negations occur only at the atomic level, and that

$\vdash B^{deM} \equiv B$.

Definition 4.4.2 By a *statement* we understand any formula $A \leftarrow B$ where A is an atomic formula other than $\top, \bot$ or $t' = t''$, and where B is a formula, possibly containing $=$, such that $var(B) \subseteq var(A)$. By a *program* we understand any finite set of statements. By a *goal* we understand any formula $\leftarrow B$, possibly containing $=$.

Definition 4.4.3 let $\mathcal{L}'$ be an extension of a language $\mathcal{L}$ obtained by adding for each predicate symbol p (other than $=$), a new predicate symbol $\overline{p}$ of the same arity. For any program P and any goal $\leftarrow B$ in $\mathcal{L}$ we define the *constructive version* P^+ of P, and *constructive version* $(\leftarrow B)^+$ of $\leftarrow B$.

P^+ is obtained in the following way. Form

$$P' = \{A \leftarrow B^{deM} \mid (A \leftarrow B) \in P^C\}$$

$$P'' = \{\neg A \leftarrow (\neg B)^{deM} \mid (A \leftarrow B) \in P^C\}$$

(where P^C is Clark's transformation of P). Then replace every negative literal $p(t)$ in $P' \cup P''$ by $\overline{p}(t)$.

$\leftarrow B^+$ is obtained from $\leftarrow B^{deM}$ by replacing every negative literal $p(t)$ by $\overline{p}(t)$.

Notice that P^+ is a positive program with guards and $\leftarrow B^+$ is a positive goal with guards.

In the analysis of completeness we will use the *constructive completion* of P which is defined as $Comp(P^+)$. Notice that $Comp(P^+)$ is always consistent in classical logic. Similarly as at the propositional level, we can see that $Comp(P)$ is stronger than $Comp(P^+)$ (and too strong to obtain completeness). Roughly speaking:

$$Comp(P) = Comp(P^+) + \{\overline{p}(x) \equiv p(x) \mid p \textit{ is a predicate symbol }\}.$$

The relationship between the two notions of completion is expressed more precisely in the following (given without a proof) proposition.

Proposition 4.4.4 Let P be a program. Then (in classical logic):

1. $Comp(P) \cup \{\neg p(x) \equiv \overline{p}(x) \mid p \textit{ is a predicate symbol }\}$ is equivalent to

   $Comp(P^+) \cup \{\neg p(x) \equiv \overline{p}(x) \mid p \textit{ is a predicate symbol }\}$.

2. $Comp(P) \cup \{\neg p(x) \equiv \overline{p}(x) \mid p \textit{ is a predicate symbol }\}$

   is a conservative extension of $Comp(P)$.

Now we define SLPGCN-resolution, i.e. SLPG-resolution with Constructive Negation. SLPGCN takes as input a program P and a goal $\leftarrow B$, transforms them into

P^+ and $\leftarrow B^+$, and then employs SLPG-resolution.

Definition 4.4.5 Let P be a program and let $\leftarrow B$ be a goal. We define the *SLPGCN-answer set* for $P \cup \{\leftarrow B\}$ as: $SLPGCN(P, B) = SLPG(P^+, B^+)$.

The following statements describe relationship of SLPGCN-resolution to other resolution systems.

Proposition 4.4.6 Let P be a positive program with guards, and let $\leftarrow B$ be a positive goal with guards. Then: $SLPGCN(P, B) = SLPG(P, B)$.

Conjecture 4.4.7 Let P be a normal program and let $\leftarrow B$ be a normal goal. Let $P^{\exists}$ be the program obtained from P by adding in the body of every clause an existential quantifier binding variables occurring free in the body but not in the head. Then:

$$SLPGCN(P^{\exists}, B) \supseteq SLDNF^{-}(P, B) \supseteq SLDNF(P, B).$$

Example 4.4.8 To see that

$SLPGCN(P^{\exists}, B) \not\supseteq SLDNF^{-}(P, B)$

$SLPGCN(P^{\exists}, B) \not\supseteq SLDNF(P, B)$

consider the following program:

$p \leftarrow \neg q(x)$

$q(x) \leftarrow \bot$

We have:

$SLDNF(P, p)$ flounders,

$SLDNF^{-}(P,p)$ flounders,

$SLPGCN(P,p) = [\epsilon]$.

Theorem 4.4.9 (Soundness of SLPGCN-resolution)

Let P be a program and let $\leftarrow B$ be a goal. Then:

$SLPGCN(P,B) \supseteq [\theta \text{ WHERE } S]$ implies $CET^{\omega}_{\mathcal{L}} \cup Comp(P^{+}) \vdash B^{+}(\theta \text{ WHERE } S)$.

Theorem 4.4.10 (Completeness of SLPGCN-resolution)

Let P be a program and let $\leftarrow B$ be a goal such that one of the following holds:

(i) P^{+} is a $+\Delta_{\omega}$-program with $+\Delta_{\omega}$-guards, and

B^{+} is a $+\Delta_{\omega}$-formula with $+\Delta_{\omega}$-guards.

(ii) P^{+} is a $+\Sigma$-program with Δ_{ω}-guards, and

B^{+} is a $+\Sigma$-formula with Δ_{ω}-guards.

Then: $SLPGCN(P,B) \supseteq [\theta \text{ WHERE } S]$ iff $CET^{\omega}_{\mathcal{L}} \cup Comp(P^{+}) \vdash B^{+}(\theta \text{ WHERE } S)$.

From this theorem one can derive completeness for the class of normal programs consisting of clauses $A \leftarrow B$, where $var(B) \subseteq var(A)$.

In this theorem we could use P^{+} instead of $Comp(P^{+})$ but using $Comp(P^{+})$ is closer to the way programmers think.

One can strengthen the theorem on soundness and completeness, looking at the programs which are not necessarily in the classes considered, but are logically and

operationally (declaratively and procedurally) equivalent to programs in those classes. A direct description of classes of programs which satisfy this condition may be found in further research.

SLPGCN-resolution employs the engine of SLPG-resolution. We already know that SLPG is not complete for the class of $+\Pi_1$-programs with Δ_0-guards. Moreover, it seems that incompleteness of SLPG is an inherent property of this class of programs — we do not expect that a different resolution system (and notion of completion) could be defined so that they yield completeness, cf. 3.9.6 The following example shows that there are normal programs that translate under the constructive negation interpretation into troublesome $+\Pi_1$-programs with Δ_0-guards. This suggests that the incompleteness of SLPGCN-resolution is not the fault of the resolution but rather it is the inherent property of the class of normal programs. If so, fully declarative programming with normal programs cannot be achieved. We hope that further research will bring better understanding of this issue.

Example 4.4.11 Consider the following normal program P:

$p(f(x)) \leftarrow p(x)$

$q \leftarrow p(x)$

We have $SLDNF(P, \neg q) = \uparrow$.

The constructive version $(P^{\exists})^{+}$ of the program $P^{\exists}$ is the following:

$p(y) \leftarrow \exists_x(y{=}f(x) \wedge p(x))$ $\qquad$ $\overline{p}(y) \leftarrow \forall_x(y{\neq}f(x) \vee \overline{p}(x))$

$q \leftarrow \exists_x p(x)$ $\qquad$ $\overline{q} \leftarrow \forall_x \overline{p}(x)$

The constructive version of the goal $\leftarrow \neg q$ is $\overline{q}$. We have:

$$SLPGCN(P^{\exists}, \neg q) = SLPG((P^{\exists})^{+}), \overline{q}) = \uparrow$$

while:

$$CET^{\omega}_{\mathcal{L}} \cup Comp((P^{\exists})^{+}) \vdash \overline{q}.$$

The problem seems to be inherent in the program itself because the operator on lattice of ω-Herbrand interpretations, associated with the program $(P^{\exists})^{+}$ requires more than ω steps to reach its least fix-point.

5 CONCLUSION

Here is a list of issues that should be considered either in connection with the implementation or in further theoretical research.

Implementation.

Resolution systems discussed in this paper can be implemented as the basis for a logic programming language. Completeness theorems guarantee that programming will be fully declarative. As argued in section 4.2 implementation can be as efficient as implementations of SLD-resolution.

Direct characterization of complete classes of programs with negation.

It was shown that SLPGCN is complete for programs with negation which translate into positive programs equivalent to $+\Delta_\omega$-programs with $+\Delta_\omega$-guards or to $+\Sigma$-programs with Δ_ω-guards. One can find direct characterizations of classes of programs with negation for which SLPGCN is complete.

Incompleteness as inherent property of the class of normal programs.

We attach special importance to example 4.4.11. Further research should elucidate this issue.

Developing the constructive negation interpretation.

SLPGCN-resolution takes as input a program with negation, uses the deMorgan operation to push negations to the atomic level, and then replaces negative literals $\neg p(t)$ by $\overline{p}(t)$. One can modify the process so that the deMorgan oper-

ation won't be necessary. Introduce "constructive versions" $\overline{\alpha}$ for any formula α, where α is possibly more complicated than atomic. If this idea is worked out theoretically, it may allow increasing the efficiency of SLPGCN.

Model-theoretic analysis of equality theory $CET^{\omega}_{\mathcal{L}}$.

Such an analysis may bring interesting results. Besides conjectures formulated in section 3.5, one can consider theories containing $CET^{\omega}_{\mathcal{L}}$ with the following analog of the ω-rule: If $B(t)$ for every closed term t, deduce $\forall_x B(x)$.

Modal analysis of the negation as failure rule.

It seems that formula $\Box\neg p \equiv \Box\neg\Box p$ expresses the essential idea of the NF-rule. It may be possible to formulate a propositional modal system with axioms of this form.

Completeness of first-order resolutions using nonclassical logics.

There is a chance of generalizing results of Chapter 2 to the first order level.

Combining programs from different completeness classes.

We proved that SLPG-resolution is complete for $+\Delta_\omega$-programs with $+\Delta_\omega$-guards and $+\Sigma$-programs with Δ_ω-guards, but not for other levels of the hierarchy. For every other class there *exists* a program which leads to incompleteness. It does not mean that *every* program outside 'good' classes is 'bad'. One can investigate ways in which 'good' programs can be combined, resulting in a 'good' program (possibly in a 'bad' class).

Eliminating use of equality theory from the mechanisms of resolutions.

SLPG-resolution is a generalization of SLD-resolution, and as SLD works for definite programs without using any serious equality theory (except unification and composing substitutions), one can investigate classes broader than definite programs, for which the use of $CET^{\omega}_{\mathcal{L}}$ can be eliminated from SLPG. The same applies to SLPGCN.

6 PROOFS FOR CHAPTER 2

In the proofs given in this chapter we use the following conventions:

- If L is a negative literal $\neg A$, then $\neg L$ is identified with A.
- For any propositional letter q,

  $\langle q \rangle$ stands for q,

  $\langle \overline{q} \rangle$ stands for $\neg q$

  $\langle \neg q \rangle$ stands for $\overline{q}$.

6.1 Proofs for section 2.1

Proposition 6.1.1 (2.1.6) Let P be a propositional program and let p be a propositional letter in P. Then:

1. p belongs to SLDNF-success set of P iff $SLDNF(P,p) = YES$
2. p belongs to SLDNF-finite failure set of P iff $SLDNF(P,p) = NO$

Proof. Immediately from the definitions of SLDNF-sucess set and SLDNF-finite faiure set.

Proposition 6.1.2 (2.1.7) Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then exactly one of the following conditions holds:

$$SLDNF(P,B) = YES,\ \ SLDNF(P,B) = NO,\ \ SLDNF(P,B) = \uparrow.$$

Proof. It is clear that at least one of these conditions holds. Indeed, by definition $SLDNF(P,B) = \uparrow$ if neither $SLDNF(P,B) = YES$ nor $SLDNF(P,B) = NO$. For the

same reason we can see that conditions $SLDNF(P,B) = YES$ and $SLDNF(P,B) = \uparrow$ are disjoint, aa well as $SLDNF(P,B) = NO$ and $SLDNF(P,B) = \uparrow$ are disjoint. It remains to show that $SLDNF(P,B) = YES$ and $SLDNF(P,B) = NO$ are disjoint. Restricting theorem 9.2 (p. 51) in [44] to propositional case and allowing (arbitrary) programs, we obtain: If there is a successful derivation for $P; \leftarrow B$, then for every computation rule R there is a successful derivation for $P; \leftarrow B$ via R. So, if there is a successful derivation for $P; \leftarrow B$, then every SLDNF-tree for $P; \leftarrow B$ contains a successful branch, and there are no finite failure trees. This ends the proof.

Proposition 6.1.3 (2.1.8) Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

$$SLDNF(P,B) = YES \iff SLDNF(P,\neg B) = NO$$

$$SLDNF(P,B) = NO \iff SLDNF(P,\neg B) = YES$$

$$SLDNF(P,B) = \uparrow \iff SLDNF(P,\neg B) = \uparrow$$

Proof. Assume that disjunctive normal form of B is $\bigvee_{i \in I} B_i$ and disjunctive normal form of $\neg B$ is $\bigvee_{j \in J} B'_j$. Notice that if literal L occurs in a certain B_i ($i \in I$), then $\neg L$ occurs in every B_j ($j \in J$), and vice versa. Now let us prove the first equivalence:

$SLDNF(P,B) = YES$ iff

there is $i \in I$ such that $P; \leftarrow B_i$ has a successful SLDNF-derivation iff

there is $i \in I$ such that for every conjunct L in B_i,

$P; \leftarrow L$ has a successful SLDNF-derivation iff

there is $i \in I$ such that for every conjunct L in B_i,

$P; \leftarrow \neg L$ has a finitely failed SLDNF-tree iff

for every $j \in J$ there is a literal $\neg L$ in B_j such that

$P; \leftarrow \neg L$ has a finitely failed SLDNF-tree iff

for every $j \in J$ there is a finitely failed SLDNF-tree for $P; \leftarrow B_j$ iff

$SLDNF(P, \neg B) = NO$.

The second equivalence in the proposition is a reformulation of the first one. The third equivalence follows from the first and from the second by 2.1.7 (6.1.2).

6.2 Proofs for section 2.2

In this section we prove completeness theorem 2.2.3. The idea is this: show that $SLDNF(P, B) = SLD(P^+, B^+$, and then use the completeness theorem for SLD-resolution. Inductions used in the proof are straightforward but strenuous.

Lemma 6.2.1 Let P be a propositional normal program, and let p be a propositional letter. Then:

1. $SLD(P^+, p) = YES$ implies $SLD(P^+, \overline{p}) = NO$.
2. $SLD(P^+, \overline{p}) = YES$ implies $SLD(P^+, p) = NO$.

Proof. We prove both statements in parallel by induction on the length of successful SLD-derivation for $P^+; \leftarrow p$ or $P^+; \leftarrow \overline{p}$.

$SLD(P^+, p) = YES$ derived in 1 step.

Then the (unique) statement defining p in P^+ is of the form $p \leftarrow \mathsf{T} \vee B$, where B is a disjunction of conjunctions of atoms. Thus the (unique) statement defining

$\overline{p}$ in P^+ is of the form $\overline{p} \leftarrow \bigvee_{i \in I}(\bot \wedge B_i)$, where each B_i is a conjunction of atoms. This is interpreted as a collection $\overline{p} \leftarrow \bot \wedge B_i \quad : i \in I$ of statements in a definite program. If we start from $\leftarrow \overline{p}$, use each of these rules, and select atom $\bot$ in every derived goal, we obtain a finite failure tree for $P^+; \leftarrow \overline{p}$, so $SLD(P^+, \overline{p}) = NO$.

$SLD(P^+, \overline{p}) = YES$ derived in 1 step.

Similar to the case above.

$SLD(P^+, p) = YES$ derived in $n + 1$ steps.

Let $p \leftarrow L_1 \wedge \ldots \wedge L_k$ be the clause in (the normal version of) P^+, with which the successful derivation for $P^+; \leftarrow p$ starts. By insuction hypothesis, for each $L_i : i \leq k$ there is a finitely failed SLD-tree for $P; \leftarrow \overline{L}$. (If L is $\overline{q}$, by $\overline{L}$ we understand just q). The (unique) statement defining p in P^+ is of the form $p \leftarrow (L_1 \wedge \ldots \wedge L_k) \vee B$ where B is a disjunction of conjunctions of atoms. Thus every clause defining $\overline{p}$ in the normal version of P^+ contains one of $\overline{L_i} : i \leq k$ as a conjunct in its body: $p \leftarrow \ldots \wedge \overline{L_i} \wedge \ldots$. If we start fro $\leftarrow \overline{p}$ and use each of these clauses, selecting in each of derived goals the corresponding atom $\overline{L_i}$ and constructing forther finite failure trees for these atoms, we will obtain a filnitely failed SLD-tree for $P; \leftarrow \overline{p}$. So $SLD(P^+, \overline{p}) = NO$.

$SLD(P^+, \overline{p}) = YES$ derived in $n + 1$ steps.

Similar to the case above.

Lemma 6.2.2 Let P be a propositional normal program, and let p be a propositional letter. Then:

1. $SLD(P^+, p) = NO$ implies $SLD(P^+, \overline{p}) = YES$.

2. $SLD(P^+, \overline{p}) = NO$ implies $SLD(P^+, p) = YES$.

Proof. We prove both statements in parallel by induction on the depth of finitely failed SLD-tree for $P^+; \leftarrow p$ or $P^+; \leftarrow \overline{p}$.

Finitely failed SLD-tree of depth 1 for $P; \leftarrow p$.

Then the (unique) statement defining p in P^+ is of the form $p \leftarrow \bigvee_i(\bot \wedge B_i)$, where each B_i is a conjunction of of atoms. Thus the unique statement defining $\overline{p}$ in P^+ is of the form $\overline{p} \leftarrow \top \vee B'_j$, where each B'_j is a disjunction of conjunctions of atoms. Using the clause $\overline{p} \leftarrow \top$ from the definite version of P^+, we obtain a successful derivation for $P; \leftarrow \overline{p}$.

Finitely failed SLD-tree of depth 1 for $P; \leftarrow \overline{p}$.

Similar to the case above.

Finitely failed SLD-tree of depth $n+1$ for $P; \leftarrow p$.

Let $p \leftarrow \ldots \wedge L_1 \wedge \ldots, \quad \ldots p \leftarrow \ldots \wedge L_k \wedge \ldots$

be the collection of all clauses in the definite version of P^+, with $L_1, \ldots, L_k$ being the literals sellected in the construction of finitely failed SLD-tree of depth $n+1$ for $P; \leftarrow p$. By induction hypothesis, for each $i \leq k$ there is a successful derivation for $P; \leftarrow \overline{L_i}$. Un the definite version of P^+ there has to be a clause $\overline{p} \leftarrow \overline{L_1} \wedge \ldots \wedge \overline{L_k}$. Using this clause, and then performing steps from

successful derivations for $P; \leftarrow \overline{L_i}$ for each i, we obtain a successful derivation for $P^+; \overline{p}$. So $SLD(P, \overline{p}) = YES$.

Finitely failed SLD-tree of depth $n+1$ for $P; \leftarrow \overline{p}$.

Similar to the case above.

Lemma 6.2.3 Let P be a propositional normal program, and let p be a propositional letter. Then:

1. $SLDNF(P, p) = YES$ implies $SLD(P^+, p) = YES$.
2. $SLDNF(P, p) = NO$ implies $SLD(P^+, p) = NO$.

Proof. We prove both statements in parallel by induction on the rank of the sucessful SLDNF-derivation for $P; \leftarrow p$ or rank of finitely failed tree for $P; \leftarrow p$.

Successful SLDNF-derivation of rank 0 for $P; \leftarrow p$.

The derivation is of rank 0, which means that no negative subgoals sellected were selected. By replacing every negative literal $\neg q$ in the derivation by $\overline{q}$ we obtain a sucessful SLD-derivation for $P^+; p$. So $SLD(P^+, p) = YES$.

Finitely failed SLDNF-tree of rank 0 for $P; \leftarrow p$.

The tree is of rank 0, which means that no negative subgoals were selected. Replacing every negative literal $\neg q$ in the tree by $\overline{q}$ we obtain a finitely failed SLD-tree for $P^+; p$. So $SLD(P^+, p) = NO$.

Sucessful SLDNF-derivation of rank $n+1$ for $P; \leftarrow p$.

Consider each negative subgoal $\neg A$ selected in this derivation. $P; \leftarrow A$ has a

finitely failed SLDNF-tree of rank $\leq n$, and by induction hypothesis, there exists a finitely failed SLD-tree for $P^+; \leftarrow A$, then by 6.2.2, there exists a successful SLD-derivation for $P^+; \leftarrow \overline{A}$. If we replace each selected negative subgoal $\neg A$ in the successful SLDNF-derivation for $P; \leftarrow p$ by the corresponding successful SLD-derivation for $P^+; \leftarrow \overline{A}$, and also each remaining negative literal $\neg q$ by $\overline{q}$, we will obtain a successful SLD-derivation for $P^+; \leftarrow p$. So $SLD(P^+, p) = YES$.

Finitely failed SLDNF-tree of rank $n+1$ for $P; \leftarrow p$.

Consider each $\neg A$ selected in a goal which is not the last goal on a branch of the tree. $P; \leftarrow \neg A$ has a sucessful SLDNF-derivation of rank $\leq n$, so by 2.1.8, (6.1.3) $P; \leftarrow A$ has a finitely failed SLDNF-tree (and the tree constructed in the proof is of rank $\leq n$. By induction hypothesis there is a finitely failed SLD-tree for $P^+; \leftarrow A$ and then by 6.2.2 there is a sucessful SLD-derivation for $P^+; \leftarrow \overline{A}$. Replace each such goal by the corresponding sucessful SLD-derivation for $P^+; \leftarrow \overline{A}$. Consider each $\neg A$ selected in a goal which is the last goal on a branch of the tree. $P; \leftarrow A$ has a sucessful SLDNF-derivation of rank $\leq n$, and by induction hypothesis there is a successful SLD-derivation for $P^+; \leftarrow A$, then by 6.2.1 there exists a finitely failed SLD-tree for $P^+; \leftarrow \overline{A}$. Replace each such goal in the finitely failed SLDNF-tree for $P; \leftarrow A$ by the corresponding finitely failed SLD-tree for $P^+; \leftarrow \overline{A}$, then replace each remaining negative literal $\neg q$ in the tree by $\overline{q}$. We will obtain a finitely failed SLD-tree for $P^+; \leftarrow p$. So $SLD(P^+, p) = NO$.

Last proof shows that every SLDNF-derivation for program P translates directly into an SLD-drivation for program P^+. The converse translation is more complex, and not all SLD-derivations for P^+ correspond *directly* to SLDNF-derivations for P. Recall that in SLDNF-derivations negative subgoals are treated as "lemmas" which have to be completed before other literals can be sellected and resolved. In an SLD-derivation for P^+, given a goal $\leftarrow q_1 \wedge \overline{q}_2$, we could select atom $\overline{q}_2$, replace it by (say) q_3 obtaining $\leftarrow q_1 \wedge q_3$ and then instead of selecting q_3, we could select q_1, etc. As the "lemma" starting from $\overline{q}_2$ has not been completed, such a derivation does not translate directly into an SLDNF-derivation.

To deal with this problem we define *systematic drivations* for P^+, as those in which "lemmas" are treated appropriately; we prove that there is sufficiently many such derivations, and that they translate directly into SLDNF-derivations.

Definition 6.2.4

1. Let $G :\leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A_k \wedge A_{k+1} \wedge \ldots A_m$

   and $G' :\leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A'_1 \wedge \ldots \wedge A'_n \wedge \wedge A_{k+1} \wedge \ldots A_m$

   be two consecutive goals in an SLD-derivation.

   Then atoms $A'_1, \ldots, A'_n$ in G are called *children* of A_k in G'.

   Each A_i $(1 \leq i \leq m, i \neq k)$ in G' is called a *copy* of the corresponding atom A_i in G. Copy of a copy is also called a copy.

   By a *descendant* we understand either a child, or a copy, or a descendant of a descendant.

   If A'' is a descendant of A', then A' is called an *ancestor* of A''.

2. An SLD-derivation is called *systematic* if for every goal which is not the last goal, one of the following holds:
   - the sellected atom has children and in the next step one of its children is selected,
   - the selected atom is the only atom in the goal and it doas not have any children (so the next goal is $\leftarrow \emptyset$),
   - the selected atom A does not have any children but there are other atoms in this goal; in the next step one of those atoms is selcted, which is a copy of the closest ancestor of A.
3. A finitely failed SLD-tree is called *systematic* if its every branch is a systematic derivation.

Remark 6.2.5

1. Every SLD-derivation via the computation rule "select first from the left" is systematic. (This is the rule that is used in Prolog).
2. It is known that: If $P; \leftarrow p$ has a sucessful SLD-derivation, then it has a sucessful SLD-derivation via the rule "select first from the left". (cf. Thm. 9.2, p. 51, [44].) The same is not true for finitely failed trees. Consider program P:

   $p \leftarrow q \wedge r$

   $r \leftarrow \bot$

   $q \leftarrow q$

   Then $P; \leftarrow p$ has a finitely failed SLD-tree, but the tree constructed using the

rule "select first from the left" is not finite.

3. Accoring to [44], a *computation rule* in a derivation, is a function from a set of definite goals to a set of atoms, such that the value of the function for a goal in the derivation is the atom that gets selected. With this definition there are derivations, for which there is no rule R such that the derivation is via R. Indeed, consider the following program P:

   $p \leftarrow \top$

   $q \leftarrow p \wedge q$

   $q \leftarrow \top$

   and the derivation (selected atoms are underlined):

   $\leftarrow \underline{p} \wedge q$

   $\leftarrow \underline{q}$

   $\leftarrow p \wedge \underline{q}$

   $\leftarrow \underline{p}$

   $\leftarrow \emptyset$

Definition 6.2.6 Let P be a propositional normal program. Let $\mathcal{D}$ be a systematic SLD-derivation for P^+.

1. By a *lemma* in $\mathcal{D}$ we understand a fragment of the derivation which starts with goal $G :\leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A_k \wedge A_{k+1} \wedge \ldots A_m$, and

   ends with $G' :\leftarrow A_1 \wedge \ldots \wedge A_{k-1} \wedge A_{k+1} \wedge \ldots A_m$,

   such that each A_i in G' is a descendant of the corresponding A_i in G.

2. If $\mathcal{D}$ is a failed derivation, by a *failed lemma* we understand any final fragment of $\mathcal{D}$ which starts with a certain goal G and does not contain any lemma starting with G.

3. By a *generaized lemma* we understand either a lemma or a failed lemma.

4. Generalized lemma is *positive* if it starts with a goal in which a "positive" atom q is selected. Generalized lemma is *negative* if it starts with a goal in which a "negative" atom $\overline{q}$ is selected.

5. Consider maximal sequencess of nested generalized lemmas $\mathcal{D}$. For each such sequence S count number n_S of changes in the sequence of signs of the lemmas. *Rank* of the derivation $\mathcal{D}$ is defined as: $\max_S(n_S)$.

6. By a *rank* of a systematic finitely failed SLD-tree we understand maximum of ranks of the derivations which are branches of the tree.

Example 6.2.7 Consider the following program:

$p \leftarrow \top$

$q \leftarrow \bot$

and derivation:

$\leftarrow p \wedge q \wedge r$

$\leftarrow q \wedge r$

$\leftarrow \bot \wedge r$

This derivation is systematic. The part consisting of the first two goals is apositive lemma. The part consisting of the second and third goal is a positive gailed lemma.

The segment consisting of all the three goals is neither a lemma nor a failed lemma. Rank of the derivation is 0.

Lemma 6.2.8 Let P be a propositional normal program and let p be a propositional letter. Then:

1. If $P^+; \leftarrow p$ has a sucessful SLD-derivation, then it has a systematic one. 2. If $P^+; \leftarrow p$ has a finitely failed SLD-tree, then it has a systematic one.

Proof.

1. By thoerem 9.2, p. 51 in [44], every sucessful SLD-drivation can be replaced by a sucessful SLD-derivation via the rule "select first from the left". Such a derivation is systemstic.

2. (As mentioned in the last remark, the rule "select first from the left" is of no use for finitely failed trees - we have to proceed differently.) Assume that $P^+; \leftarrow p$ has a finitely failed SLD-tree. Then by lemma 6.2.2, $P^+; \leftarrow \overline{p}$ has a successful SLD-derivation $\mathcal{D}$ which by point 1 can be assumed to be systematic. Using construction from the proof of lemma 6.2.1, translate $\mathcal{D}$ into a finitely failed SLD-tree for $P^+; \leftarrow p$. The resulting tree is systematic.

The lemma above holds not only for programs of the form P^+ but for arbitrary definite programs. This can be shown by using another construction for point 2.

Lemma 6.2.9 let P be a propositional norma program and let p be a propositional letter. Then:

1. There exists a successful systematic SLD-derivation of rank n for $P^+; \leftarrow p$ iff there exists a systematic finitely failed SLD-tree of rank n for $P^+; \leftarrow \overline{p}$.

2. There exists a successful systematic SLD-derivation of rank n for $P^+; \leftarrow \overline{p}$ iff there exists a systematic finitely failed SLD-tree of rank n for $P^+; \leftarrow p$.

Proof. Repeat the constructions from lemmas 6.2.1, 6.2.2 and check that the resulting derivation or tree, is systematic and of the required rank.

Lemma 6.2.10 let P be a propositional normal program and let p be a propositional letter. Then:

1. $SLD(P^+, p) = YES$ implies $SLDNF(P, p) = YES$.
2. $SLD(P^+, p) = NO$ implies $SLDNF(P, p) = NO$.

Proof. BY lemma 6.2.8, it is enough to prove the following two statements:

1. If there is a systematic finitely failed SLD-tree of rank n for $P^+; \leftarrow p$, then there is a successful SLDNF-derivation for $P; \leftarrow p$.

2. If there is a successful systematic SLD-derivation of rank n for $P^+; \leftarrow p$, then there is a finitely failed SLDNF-derivation for $P; \leftarrow p$.

We prove both statements in parallel, by induction on n.

Successful SLD-derivation for $P^+; \leftarrow p$ of rank 0.

No "negative" subgoals (like $\overline{q}$) were selected in this derivation. Therefore if we replce every $\overline{q}$ by $\neg q$, we will obtain a successful SLDNF-derivation for $P; \leftarrow p$.

Finitely failed SLD-tree for $P^+; \leftarrow p$ of rank 0.

No "negative" subgoals (like $\overline{q}$) were selected in the derivations of the tree. Therefore if we replce every $\overline{q}$ by $\neg q$, we will obtain a finitely failed SLDNF-tree for $P; \leftarrow p$.

Successful systematic SLD-derivation for $P^+; \leftarrow p$ of rank $n+1$.

Consider a negative maximal lemma in this derivation. Rank of the lemma is $\leq n$. If the lemma starts with $\leftarrow \ldots \wedge \overline{q} \wedge \ldots$, where $\overline{q}$ is the selected atom, by 6.2.9 there is a systematic finitely failed SLD-tree for $P; \leftarrow q$ of rank $\leq n$, and by induction hypothesis there is a finitely failed SLDNF-tree for $P \leftarrow q$. Therefore by cutting out all negative maximal lemmas from the SLD-derivation for $P^+; \leftarrow p$ and by replacing all the remining "negative" atoms $\overline{q}$ by $\neg q$, we obtain a successful SLDNF-derivation for $P; \leftarrow p$.

Finitely failed systematic SLD-tree for $P^+; \leftarrow p$ of rank $n+1$.

In this case we have to deal with lemmas and with failed lemmas.

Consider a node in the tree, at which a branch with a negative maximal lemma starts: $\leftarrow \ldots \wedge \overline{q} \ldots$ with the selected atom $\overline{q}$. The lemma is of rank $\leq n$, so by 6.2.9, there is a finitely failed systematic SLD-tree for $P^+; \leftarrow q$ of rank $\leq n$, and by induction hypothesis there is a finitely failed SLDNF-tree for $P; \leftarrow q$. Delete all the branches comminig out of $\leftarrow \ldots \wedge \overline{q} \ldots$ except the one containing the lemma; in this branch cut out the lemma. Do that for all negative maximal lemmas.

Now consider a node where a negative maximal failed lemma starts: $\leftarrow \ldots \wedge \overline{q'} \ldots$. Notice that as we scooped all lemmas in the previous step, every branch comming out of this node is a failed lemma. The tree from this node down, is of rank $\leq n$. By 6.2.9 there exists a successful systematic SLD-derivation for $P; \leftarrow q'$ of rank $\leq n$, and by induction hypothesis there exists a successful SLDNF-derivation for $P; \leftarrow q'$. By 2.1.8 (6.1.3) there exists a finitely failed SLDNF-tree for $P; \leftarrow \neg q$; replace the tree starting at $\leftarrow \ldots \wedge \overline{q'} \ldots$ by this tree.

Finally replace all the remaining negative atoms $\overline{q}$ by $\neg q$. Resulting tree is a finitely failed SLDNF-tree for $P; \leftarrow p$.

Corollary 6.2.11 Let P be a propositional normal program and let p be a propositional letter. Then:

1. $SLDNF(P,p) = YES$ iff $SLD(P^+) = YES$
2. $SLDNF(P,p) = NO$ iff $SLD(P^+) = NO$
3. $SLDNF(P,p) = \uparrow$ iff $SLD(P^+) = \uparrow$

Proof. Points 1 and 2 result immediately from lemmas 6.2.10 and 6.2.3. Point 3 follows from 1 and 2 by 2.1.7, (6.1.2) and 6.2.1, 6.2.2.

Theorem 6.2.12 (2.2.3) Soundness and completeness

Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLDNF(P,B) = YES$ iff $Comp(P^+) \vdash B^+$
2. $SLDNF(P,B) = NO$ iff $Comp(P^+) \vdash (\neg B)^+$
3. $SLDNF(P,B) = \uparrow$ iff $Comp(P^+) \nvdash B^+$ and $Comp(P^+) \nvdash (\neg B)^+$

Proof.

1. Notice that for any propositional letter p,

   $SLDNF(P,p) = YES$ iff

   $T_{P+} \uparrow \omega \models p$,

   $SLDNF(P,\neg p) = YES$ iff

   $T_{P+} \uparrow \omega \models \overline{p}$.

   Indeed:

   $SLDNF(P,p) = YES$ iff (by 6.2.11)

   $SLD(P^+,p) = YES$ iff

   $T_{P+} \uparrow \omega \models p$

   and:

   $SLDNF(P,\neg p) = YES$ iff (by 2.1.8)

   $SLDNF(P,p) = NO$ iff (by 6.2.11)

   $SLD(P^+,p) = NO$ iff (by ??)

   $SLD(P^+,\overline{p}) = YES$ iff

   $T_{P+} \uparrow \omega \models \overline{p}$.

   Now consider arbitrary formula B. Assume that disjunctive normal form of B is $\bigvee_i \bigwedge_j L_{i,j}$. We have:

   $SLDNF(P,B) = YES$ iff

   $SLDNF(P,\bigvee_i \bigwedge_j L_{i,j}) = YES$ iff

there is i such that $SLDNF(P, \bigwedge_j L_{i,j}) = YES$ iff

there is i such that for all j, $SLDNF(P, L_{i,j}) = YES$ iff

there is i such that for all j, $T_{P+} \uparrow \omega \models \langle L_{i,j} \rangle$ iff

$T_{P+} \uparrow \omega \models \bigvee_i \bigwedge_j \langle L_{i,j} \rangle$ iff

$Comp(P^+) \vdash \bigvee_i \bigwedge_j \langle L_{i,j} \rangle$ iff

$Comp(P^+) \vdash B^+$.

2. $SLDNF(P, B) = NO$ iff (by 2.1.8)

   $SLDNF(P, \neg B) = YES$ iff (by 1)

   $Comp(P^+) \vdash (\neg B)^+$.

3. From 1 and 2 by 2.1.7 (6.1.2).

Proposition 6.2.13 (2.2.6) Let P be a propositional program in a language with the set *Prop* of propositional letters. Then (in classical logic):

1. $Comp(P) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$ is equivalent to

   $Comp(P^+) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$.

2. $Comp(P) \cup \{\neg p \equiv \overline{p} \mid p \in Prop\}$ is a conservative extension of $Comp(P)$.

Proof.

1. Straightforward.

2. As this extension is definitional, it is conservative.

Proposition 6.2.14 (2.2.7) Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLD(P^+, B^+) = YES$ iff $SLD(P^+, (\neg B)^+) = NO$

2. $SLD(P^+, B^+) = NO$ iff $SLD(P^+, (\neg B)^+) = YES$

3. $SLD(P^+, B^+) = \uparrow$ iff $SLD(P^+, (\neg B)^+) = \uparrow$

Proof.

1. By lemmas 6.2.1, 6.2.2, for every propositional letter we obtain:

   $SLD(P^+, p) = YES$ iff $SLD(P^+, \overline{p}) = NO$

   $SLD(P^+, \overline{p}) = YES$ iff $SLD(P^+, p) = NO$

   so the equivalence 1 is true at the atomic level. Consider now arbiterary formula B, and assume that $\bigvee_i \bigwedge_j L_{i,j}$ is the disjunctive normal form of B. (In the following we identify negative literal $L = \neg q$ with $\overline{q}$, and also $\overline{L}$ with just q.) We have:

   $SLD(P^+, B^+) = YES$ iff

   $SLD(P^+, \bigvee_i \bigwedge_j L_{i,j}) = YES$ iff

   there is i such that $SLD(P^+, \bigwedge_j L_{i,j}) = YES$ iff

   there is i such that for all j, $SLD(P^+, L_{i,j}) = YES$ iff

   there is i such that for all j, $SLD(P^+, \overline{L}_{i,j}) = NO$ iff

   there is i such that $SLD(P^+, \bigvee_j \overline{L}_{i,j}) = NO$ iff

   $SLD(P^+, \bigwedge_i \bigvee_j \overline{L}_{i,j}) = NO$ iff

   $SLD(P^+, (\neg B)^+) = NO$.

2. Similar to 1.

3. By complementing 1 and 2.

6.3 Proofs for section 2.3

Theorem 6.3.1 (2.3.1) Soundness and completeness

Let **L** be any intermediate logic between intuitionistic logic and classical logic (including the two). Let P be a propositional program and let $\leftarrow B$ ba a propositional goal. Then:

1. $SLDNF(P, B) = YES$ iff $Comp(P^+) \vdash B^+$
2. $SLDNF(P, B) = NO$ iff $Comp(P^+) \vdash (\neg B)^+$
3. $SLDNF(P, B) = \uparrow$ iff $Comp(P^+) \nvdash_{\mathbf{L}} B^+$ and $Comp(P^+) \nvdash_{\mathbf{L}} (\neg B)^+$.

Proof. By completeness theorem in classical logic 2.2.3 (6.2.12) it is enough to prove:

$Comp(P^+) \vdash\cdot B^+$ iff $Comp(P^+) \vdash_{\mathbf{IL}} B^+$.

Implication to the left is obvious. To prove the implication to the right assume $Comp(P^+) \nvdash_{\mathbf{IL}} B^+$. Then by completeness theorem for intuitionistic logic, there is a Kripke model $\mathcal{G}$ with a forcing relation $\models_{\mathcal{G}}$, such that for any node: $G \models_{\mathcal{G}} Comp(P^+)$, and for certain node: $G_0 \not\models_{\mathcal{G}} B^+$. Every node of $\mathcal{G}$ can be considered a model for classical propositional calculus (boolean valuation): for propositional letter p: we assume $G(p) = true$ if $G \models_{\mathcal{G}} p$, and $G(p) = false$ otherwise. We claim that if G_0 is treated as a classical model: $G_0 \models Comp(P^+)$ and $G_0 \not\models B^+$. Indeed, B^+ is of the form $\bigvee_i \bigwedge_j p_{i,j}$ and $Comp(P^+)$ is a collection of formulas of the form $p \equiv \bigvee_i \bigwedge_j p_{i,j}$,

where p and $p_{i,j}$ are propositional letters. In any node of any Kripke model, we have:

$G' \models_{\mathcal{G}'} \bigvee_i \bigwedge_j p_{i,j}$ iff $G' \models \bigvee_i \bigwedge_j p_{i,j}$ and

$G' \models_{\mathcal{G}'} p \equiv \bigvee_i \bigwedge_j p_{i,j}$ implies $G' \models p \equiv \bigvee_i \bigwedge_j p_{i,j}$.

6.4 Proofs for section 2.4

Definition 6.4.1 Let P be a set of definite clauses. If $(q \leftarrow \top) \in P$ then the sequence consisting of this one clause is called a *pure detachment proof of rank 0 of q from P*. If $(q \leftarrow p_1 \wedge \ldots \wedge p_m) \in P$ and $\mathcal{P}_1, \ldots, \mathcal{P}_m$ are pure detachment proofs of ranks $\leq n$ of, respectively, $p_1, \ldots, p_m$ fro P, then the sequence

$\mathcal{P}_1 \frown \ldots \frown \mathcal{P}_m \frown \langle q \leftarrow p_1 \wedge \ldots \wedge p_m \rangle$ is called a *pure detachment proof of rank $n+1$ of q from P*.

Notice that by inserting some tautiologies classical propositional calculus, any pure detachment proof can be turned into a formal proof. Therefore we will treat pure detachment proofs as formal proofs.

Lemma 6.4.2 Let P be a definite program, and let p be a propositional letter. If $Comp(P) \vdash p$ then p has a pure detachment proof from P.

Proof. If $Comp(P) \vdash p$, then $T_P \uparrow \omega \vdash p$. Thus for a certain $n < \omega$, we have $T_P \uparrow n \vdash p$. Now it is enough to notice that for any propositional letter q,

$T_P \uparrow n \vdash q$ iff q has a pure detachment proof of rank n from P.

Lemma 6.4.3 Let **L** be a propositional modal logic extending CL+$\Box\top$ and contained in **sS5**. Let P be a propositional program and let $\leftarrow B$ be a propositional

goal. Then:

1. $SLDNF(P, B) = YES$ implies $C^{\Box}(P) \vdash_{\mathbf{L}} B^{\Box}$.

2. $SLDNF(P, B) = NO$ implies $C^{\Box}(P) \vdash_{\mathbf{L}} (\neg B)^{\Box}$.

Proof.

1. By completeness of propositional SLDNF-resolution in classical logic 2.2.3 , (6.2.12), it is enough to show that

   $Comp(P^+) \vdash B^+$ implies $C^{\Box}(P) \vdash B^{\Box}$.

   First let us establish this implication for B being a propositional letter p. Assume $Comp(P^+) \vdash p$. By 6.4.2 p has a pure detachment proof $\mathcal{P}$ from P^+. By changing "negative" atoms $\overline{q}$ to $\neg q$ and by prefixing every literal by $\Box$, in the formulas of $\mathcal{P}$, we obtain a sequence $\mathcal{P}^{\Box}$ of formulas which extends to a formal proof of $\Box p$ from $C^{\Box}(P)$. (Notice that in this reasoning we need tha axiom $\Box\top$, and that to interpret $\mathcal{P}^{\Box}$ as a formal proof from $C^{\Box}(P)$ we have to to turn some formulas of $C^{\Box}(P)$ to contrapositive form, changing also $\neg\Diamond$ to $\Box\neg$; for instance instead of $\Diamond p \rightarrow (\Diamond q \wedge \Diamond\neg r) \vee \Diamond\neg s$ we need $\Box\neg p \leftarrow (\Box\neg q \wedge \Box s) \vee (\Box r \wedge \neg s)$ or $\Box\neg p \leftarrow \Box\neg q \wedge \Box s, \quad \Box\neg p \leftarrow \Box r \wedge \neg s$.

   This shows that our implication is true for B being a propositional letter p. Similarly the implication can be proved for any "negative" propositional letter $\overline{p}$.

   Now, consider arbitrary formula B. B^+ is of the form $\bigvee_i \bigwedge_j L_{i,j}$, where $L_{i,j}$ are atoms. In the following $\langle q \rangle$ means q, and $\langle \overline{q} \rangle$ means $\neg q$. We have:

$Comp(P^+) \vdash B$ iff

$T_{P^+} \uparrow \omega \models B$ iff

there is i such that for all j, $T_{P^+} \uparrow \omega \models L_{i,j}$ iff

there is i such that for all j, $Comp(P^+) \vdash L_{i,j}$ iff

there is i such that for all j, $C^{\Box}(P) \vdash_{\mathbf{L}} \Box\langle L_{i,j}\rangle$ iff

$C^{\Box}(P) \vdash_{\mathbf{L}} B^{\Box}$.

2. By completeness of propositional SLDNF-resolution in classical logic 2.2.3 , (6.2.12), it is enough to show that

   $Comp(P^+) \vdash (\neg B)^+$ implies $C^{\Box}(P) \vdash (\neg B)^{\Box}$.

   This follows from 1, by taking $\neg B$ for B.

Lemma 6.4.4 Let **L** be a propositional modal logic extending CL+$\Box\top$ and contained in **sS5**. Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $C^{\Box}_{\equiv}(P) \vdash_{\mathbf{L}} B^{\Box}$ implies $SLDNF(P, B) = YES$.
2. $C^{\Box}_{\equiv}(P) \vdash_{\mathbf{L}} (\neg B)^{\Box}$ implies $SLDNF(P, B) = NO$.

Proof.

1. By completeness of propositional SLDNF-resolution in classical logic 2.2.3 , (6.2.12) and by assumption that **L** is contained in **sS5**, it is enough to show that

$Comp(P^+) \not\vdash B^+$ implies $C^{\square}_{\equiv}(P) \not\vdash_{\mathbf{L}} B^{\square}$.

Assume $Comp(P^+) \not\vdash B^+$. Consider the least fix-point of T_{P+} and denote it by v_{min}. Recall that:

$v_{min} = T_{P+} \uparrow \omega = \{A \mid Comp(P^+) \vdash A\} = \{A \mid P^+ \vdash A\} = \bigcap\{v \mid v \models P^+\}$.

v_{min} can be treated as a Boolean valuation which makes q true if $q \in v_{min}$ ($\overline{q}$ true if $\overline{q} \in v_{min}$) and makes it false otherwise.

We claim that:

for no propositioanl letter q, both $v_{min}(q) = true$ and $v_{min}(\overline{q}) = true$.

Indeed, if we assume the contrary then in every model v for P^+, $v(q) = v(\overline{q}) = true$, which implies $P^+ \vdash q$ and $P^+ \vdash \overline{q}$, which implies $SLD(P^+, q) = YES$ and $SLD(P^+, \overline{q}) = YES$. contradiction with 6.2.1 and 6.2.2.

Let us call a valuation *consistent* if for every letter q, $v(q) \neq v(\overline{q})$. Define a set W of Boolean valuations: $v \in W$ iff

- v agrees with v_{min} on all propositional letters $q, \overline{q}$, for which $v_{min}(q) \neq v_{min}(\overline{q})$.
- v is consistent.

(For instance if

v_{min} makes $q, \overline{r}$ true and $\overline{q}, r, s, \overline{s}, t, \overline{t}$ false

then $W = \{v_1, v_2, v_3, v_4\}$, where

v_1 makes $q, \overline{r}, s, t$ true and $\overline{q}, r, \overline{s}, \overline{t}$ false,

v_2 makes $q, \overline{r}, s\overline{t}$ true and $\overline{q}, r, \overline{s}, t$ false,

v_3 makes $q, \overline{r}, \overline{s}, t$ true and $\overline{q}, r, s, \overline{t}$ false,

v_4 makes $q, \overline{r}, \overline{s}, \overline{t}$ true and $\overline{q}, r, s, t$ false.)

Consider standard Kripke model $M = \langle W, R, val \rangle$, where R is total relation (for any w, w', wRw') and for $v \in W$, $val(q, v) = true$ iff $v(q) = true$. M is a model for **sS5**. We will show that $M \not\models B^{\Box}$ and $M \models C^{\Box}_{\equiv}(P)$.

To show that $M \not\models B^{\Box}$, note first that for any propositional letter q,

if $v_{min}(q) = false$ then $M \not\models \Box q$.

Indeed, if $v_{min}(q) = false$ then there is $v \in W$ such that $v(q) = false$, then $M, v \not\models q$ and $M \not\models q$. Also:

if $v_{min}(\overline{q}) = false$ then $M \not\models \Box \neg q$.

Indeed, if $v_{min}(\overline{q}) = false$ then then there is $v \in W$ such that $v(q) = true$, then $M, v \not\models \neg q$ and $M \not\models \neg q$.

Using these facts we, one can easily show for every formula B_1 of the form $\bigvee_i \bigwedge_j A_{i,j}$ that:

if $v_{min}(B_1^+) = false$ then $M \not\models B_1^{\Box}$.

As $v_{min}(B^+) = false$, we can infer that $M \not\models B^{\Box}$.

It remains to show $M \not\models C^{\Box}_{\equiv}(P)$. Consider statement:

$p \leftarrow (l_1^1 \wedge l_2^1 \wedge \ldots) \vee \ldots \vee (l_1^k \wedge l_2^k \wedge \ldots)$

belonging to P^{Cd}. We have to prove:

a) $M \models \Box p \leftarrow (\Box l_1^1 \wedge \Box l_2^1 \wedge \ldots) \vee \ldots \vee (\Box l_1^k \wedge \Box l_2^k \wedge \ldots)$

b) $M \models \Box p \rightarrow (\Box l_1^1 \wedge \Box l_2^1 \wedge \ldots) \vee \ldots \vee (\Box l_1^k \wedge \Box l_2^k \wedge \ldots)$

c) $M \models \Diamond p \leftarrow (\Diamond l_1^1 \wedge \Diamond l_2^1 \wedge \ldots) \vee \ldots \vee (\Diamond l_1^k \wedge \Diamond l_2^k \wedge \ldots)$

d) $M \models \Diamond p \rightarrow (\Diamond l_1^1 \wedge \Diamond l_2^1 \wedge \ldots) \vee \ldots \vee (\Diamond l_1^k \wedge \Diamond l_2^k \wedge \ldots)$

Proof of a) is the following:

Assume the contrary: $M \not\models \Box p$ and for some i $M \models \Box l_1^i \wedge \Box l_2^i \wedge \ldots$.

As $M \not\models \Box p$, there is $v \in W$ such that $v(p) = false$, thus $v_{min}(p) = false$.

As $M \models \Box l_1^i \wedge \Box l_2^i \wedge \ldots$, $v_{min}(\langle l_1^i \rangle) = v_{min}(\langle l_2^i \rangle) = \ldots = true$, but $p \leftarrow \langle l_1^i \rangle \wedge \langle l_2^i \rangle \wedge \ldots$ is a clause in P^+, $v_{min}(P^+)$. Contradiction!

Proofs of b), c), d) are similar.

2. Analogous to 1.

Theorem 6.4.5 (2.4.5) Soundness and completeness Let **L** be any modal logic extending $\mathbf{CL} + \Box\top$ and contained in **sS5**. Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLDNF(P, B) = YES \iff C^{\Box}(P) \vdash_L B^{\Box}$
2. $SLDNF(P, B) = NO \iff C^{\Box}(P) \vdash_L (\neg B)^{\Box}$
3. $SLDNF(P, B) = \uparrow \iff C^{\Box}(P) \not\vdash_L B^{\Box}$ and $C^{\Box}(P^+) \not\vdash_L (\neg B)^{\Box}$

Proof. Notice that $C^{\Box}_{\equiv}(P) \vdash C^{\Box}(P)$, so points 1 and 2 follow from 6.4.3, 6.4.4. Point 3 follows from 1 and 2 by 2.1.7, (6.1.2).

Theorem 6.4.6 (2.4.8) Soundness and completeness Let **L** be any modal logic extending $\mathbf{CL} + \Box\top$ and contained in **sS5**. Let P be a propositional program and let

$\leftarrow B$ be a propositional goal. Then:

1. $SLDNF(P,B) = YES \iff C^{\square}_{\equiv}(P) \vdash_L B^{\square}$
2. $SLDNF(P,B) = NO \iff C^{\square}_{\equiv}(P) \vdash_L (\neg B)^{\square}$
3. $SLDNF(P,B) = \uparrow \iff C^{\square}_{\equiv}(P) \nvdash_L B^{\square}$ and $C^{\square}_{\equiv}(P^+) \nvdash_L (\neg B)^{\square}$

Proof. Notice that $C^{\square}_{\equiv}(P) \vdash C^{\square}(P)$, so points 1 and 2 follow from 6.4.3, 6.4.4. Point 3 follows from 1 and 2 by 2.1.7, (6.1.2).

Proposition 6.4.7 (2.2.7) Let P be a propositional program and let $\leftarrow B$ be a propositional goal. Then:

1. $SLD(P^+, B^+) = YES$ iff $SLD(P^+, (\neg B)^+) = NO$
2. $SLD(P^+, B^+) = NO$ iff $SLD(P^+, (\neg B)^+) = YES$
3. $SLD(P^+, B^+) = \uparrow$ iff $SLD(P^+, (\neg B)^+) = \uparrow$

Proof. From 6.2.1 and 6.2.2

7 PROOFS FOR CHAPTER 3

7.1 Proofs for section 3.1

Proposition 7.1.1 (3.1.3)

1.Every Δ_0-formula is also Σ_n and Π_n, for every n.

2. Every Σ_n-formula is also Σ_{n+1} and Π_{n+1}, for every n.

3. Every Σ_1-formula is a Σ-formula.

4. Every Σ-formula is effectively equivalent in the predicate calculus to a Σ_1-formula.

Proof. Points 1-3 result immediately from the definition. For the proof of poit 4 notice that the following formulas are tautologies of classical logis:

$$\alpha \wedge \exists_y \beta(y) \equiv \exists_y(\alpha \wedge \beta(y)) \quad \text{provided that } y \notin var(\alpha)$$

$$\alpha \vee \exists_y \beta(y) \equiv \exists_y(\alpha \vee \beta(y))$$

7.2 Proofs for section 3.2

Theorem 7.2.1 (3.2.3) Completeness with respect to ω-Herbrand models Let $\Gamma \cup \{B\}$ be a set of formulas in a language $\mathcal{L}$ without equality. Then $\Gamma \nvdash B$ implies that there exists an ω-Herbrand interpretation $I \subseteq B_{\mathcal{P}}^{\omega}$ such that $I \models \Gamma$ and $I \not\models B$.

Proof. Assume that $\Gamma \nvdash B$. Consider the canonical algebraic model M for Γ, whose universe is the set of terms of $\mathcal{L}$, with logical values in the Lindenbaum algebra $\mathcal{A}(\Gamma)$, of formulas of $\mathcal{L}$ modulo Γ. M is a model for Γ: for every $\gamma \in \Gamma$ and every valuation

ν, $\gamma_M[\nu] = \top$. As $\Gamma \not\vdash B$ we have $\gamma_M[\iota] \neq \top$, where ι is the identity valuation $\{x_i/x_i \mid i < \omega\}$. As $\mathcal{L}$ is countable, by Rasiowa-Sikorski lemma, there exists a Q-filter $\nabla \subseteq \mathcal{A}(\Gamma)$ such that $B_M[\iota] \notin \nabla$. Consider quotient $M' = M/\nabla$. M' is an algebraic realization in the set of terms of $\mathcal{L}$ and in the two-element Boolean algebra, such that $M' \models \Gamma$ and $M' \not\models B[\iota]$. By identifying each variable x_i with the free constant k_i, the set of terms of $\mathcal{L}$ can be identified with $U^\omega_\mathcal{L}$. This ends the proof.

Theorem 7.2.2 (3.2.6) Let P be a definite program and A an atomic formula in a language without equality. Then: $P \vdash A$ iff $M^\omega_P \models A$.

Proof. $P \vdash A$ - iff

$P \vdash \forall A$ - iff

$P \cup \{\neg \forall A\}$ does not have a model - iff (by 3.2.3 i.e. 7.2.1)

$P \cup \{\neg \forall A\}$ does not have an ω-Herbrand model - iff

$\neg \forall A$ is false in all ω-Herbrand models for P - iff

$\forall A$ is true in all ω-Herbrand models for P - iff

$\forall A$ is true in the intersection of all ω-Herbrand models for P - iff

$\forall A$ is true in M^ω_P - iff

$M^\omega_P \models A$

Before we formulate the next lemma, recall that $k_0, k_1, k_2, \ldots$ are new individual constants used to form ω-Herbrand universe $U^\omega_\mathcal{L}$. The lemma justifies the name "free constants", that is used for $k_0, k_1, k_2, \ldots$.

Lemma 7.2.3 Let P be a definite program and let A be an atomic formula in a language $\mathcal{L}$ without equality. Then:

$M_P^\omega \models A(\boldsymbol{x})[\boldsymbol{t}(k_1,\ldots,k_n)/\boldsymbol{x}]$ implies $P \vdash A(\boldsymbol{t}(x_1,\ldots,x_n))$.

Proof. Assume $M_P^\omega \models A(\boldsymbol{x})[\boldsymbol{t}(k_1,\ldots,k_n)/\boldsymbol{x}]$. Then in any ω-Herbrand model M for P, $M \models A(\boldsymbol{x})[\boldsymbol{t}(k_1,\ldots,k_n)/\boldsymbol{x}]$. As any ω-Herbrand model for P is a quotient of the canonical algebraic model for P modulo a Q-filter, this implies that $\| A(\boldsymbol{t}(x_1,\ldots,x_n)) \|$ belongs to every Q-filter in the Lindenbaum algebra $\mathcal{A}(P)$ of formulas of $\mathcal{L}$ modulo P. Therefore by Rasiowa-Sikorski lemma, $\| A(\boldsymbol{t}(x_1,\ldots,x_n)) \|$ is the top element in the algebra. Therefore $P \vdash A(\boldsymbol{t}(x_1,\ldots,x_n))$.

Theorem 7.2.4 (3.2.7) Existence Property

Let P be a definite program and let A be an atomic formula in a language without equality. Then the following conditions are equivalent.

1. $M_P^\omega \models \exists_{\boldsymbol{x}} A(\boldsymbol{x})$
2. $P \vdash \exists_{\boldsymbol{x}} A(\boldsymbol{x})$
3. There exists a sequence of terms $\boldsymbol{t}$ such that $P \vdash A(\boldsymbol{t})$.

Proof. Notice that $3 \Rightarrow 2 \Rightarrow 1$. It remains to prove that $1 \Rightarrow 3$. Assume $M_P^\omega \models \exists_{\boldsymbol{x}} A(\boldsymbol{x})$. Then there exists a sequence $\boldsymbol{t}$ of terms and free constants $k_1,\ldots,k_n$ such that $M_P^\omega \models A(\boldsymbol{x})[\boldsymbol{t}(k_1,\ldots,k_n)/\boldsymbol{x}]$. By lemma 7.2.3 we have $P \vdash A(\boldsymbol{t})$.

7.3 Proofs for section 3.3

Proposition 7.3.1 (7.3.1) Extensions with equality

1. Let $\mathcal{L}$ be a language without equality. Let $\mathcal{L}^{=}$ be the extension of $\mathcal{L}$ obtained by adjoining the symbol $=$ to the alphabet. Then for any theory $\mathcal{T}$ in $\mathcal{L}$, $\mathcal{T} \cup CET^{\omega}_{\mathcal{L}^{=}}$ is a conservative extension of $\mathcal{T}$.

2. Let $\mathcal{L}$ and $\mathcal{L}'$ be languages with equality, such that $\mathcal{L} \subseteq \mathcal{L}'$. Then $CET^{\omega}_{\mathcal{L}'}$ is a conservative extension of $CET^{\omega}_{\mathcal{L}}$.

Proof.

1. Let B be a formula such that $B \in \mathcal{L}$ and $\mathcal{T} \not\vdash B$. Then $\mathcal{T} \cup \{\neg\forall B\}$ is consistent and by 3.2.3 (7.2.1) there exists an ω-Herbrand model M, with the universe $U^{\omega}_{\mathcal{L}}$, such that $M \models \mathcal{T} \cup \{\neg\forall B\}$. As every ω-Herbrand interpretation is a model for $CET^{\omega}_{\mathcal{L}}$ we have $M \models \mathcal{T} \cup CET^{\omega}_{\mathcal{L}}$ and $M \not\models B$. Thus $\mathcal{T} \cup CET^{\omega}_{\mathcal{L}} \not\vdash B$.

2. Let B be a formula such that $B \in \mathcal{L}$ and $\mathcal{T} \not\vdash B$. Then $\mathcal{T} \cup \{\neg\forall B\}$ is consistent and by 3.2.3 (7.2.1) there exists an ω-Herbrand model M, with the universe $U^{\omega}_{\mathcal{L}}$, such that $M \models CET^{\omega}_{\mathcal{L}} \cup \{\neg\forall B\}$. We will show that M can be extended to a model M' with the same universe and such that $M' \models CET^{\omega}_{\mathcal{L}'}$. Every predicate symbol which belongs to $\mathcal{L}'$ but does not belong to $\mathcal{L}$ will be interpreted in M' as $\emptyset$. If all the function symbols of $\mathcal{L}'$ belong also to $\mathcal{L}$, this ends the construction. Otherwise, build ω-Herbrand universe $U^{\omega}_{\mathcal{L}'}$ using the same set $\mathcal{K}=\{k_n \mid n < \omega\}$ of free constants, that was used to build $U^{\omega}_{\mathcal{L}}$. As both sets are countable, there exists an injection

   $F : (U^{\omega}_{\mathcal{L}'} - U^{\omega}_{\mathcal{L}}) \longrightarrow \{k_{2n} \mid n < \omega\}$. Now we may give the interpretation in M' of function symbols which belong to $\mathcal{L}' - \mathcal{L}$. Assume that f is n-ary

function symbol ($n \geq 0$), then for elements $e_1, \ldots, e_n \in |M'| = U^{\omega}_{\mathcal{L}}$ (which are terms in $\mathcal{L} + \mathcal{K}$). We set $f^{M'}(e_1, \ldots, e_n)$ to $F(f(e_1, \ldots, e_n))$. One can see that $M' \models CET^{\omega}_{\mathcal{L}'}$. As M' is an extension of M in the same universe, and with the same interpretations of symbols that belong to $\mathcal{L}$, we have $M' \models CET^{\omega}_{\mathcal{L}} \cup \{\neg \forall B\}$. So there is a model for $CET^{\omega}_{\mathcal{L}'}$ which does not satisfy B. Thus $CET^{\omega}_{\mathcal{L}'} \not\vdash B$.

Theorem 7.3.2 (3.3.4)

Completeness with respect to ω-Herbrand models II

Let $\Gamma \cup \{B\}$ be a set of formulas in a language $\mathcal{L}$ with equality. Then $CET^{\omega}_{\mathcal{L}} \cup \Gamma \not\vdash B$ implies that there exists an ω-Herbrand interpretation $I \subseteq B^{\omega}_{P}$ such that $I \models CET^{\omega}_{\mathcal{L}} \cup \Gamma$ and $I \not\models B$.

Proof. It is enough to prove the case when $\Gamma \cup \{B\}$ is a set of sentences. Assume $CET^{\omega}_{\mathcal{L}} \cup \Gamma \not\vdash B$. Then $CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\}$ is consistent. Consider canonical algebraic model M for $CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\}$, whose universe is the set of terms of $\mathcal{L}$, with logical values in the Lindenbaum algebra $\mathcal{A}(CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\})$. For every $\gamma \in CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\}$ and every valuation ν we have $\gamma_M[\nu] = \top$. The algebra $\mathcal{A}(CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\})$ is not a two-element Boolean algebra; for instance the equivalence classes $\| x_i = x_j \|$ where $i \neq j$ and $\| x_i = f(x_j) \|$ where $i \neq j$ are distinct from $\top$, $\bot$ and distinct from each other. as $\mathcal{L}$ is countable, by Rasiowa-Sikorski lemma, there exists a Q-filter ∇ in the algebra. Consider the quotient $M_0 = M/\nabla$. M_0 is a model for $CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\}$ whose universe is the set of terms of $\mathcal{L}$ and with logical values in the two-element Boolean algebra. Symbol $=$ is interpreted in M_0 as a congruence. Consider quotient

$M_1 = M_0/_{=}$. M_1 is a normal model for $CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\}$. Let $\mathcal{L}'$ be the language with the same function symbols as $\mathcal{L}$, in which the only predicate symbol is $=$. As $M_1 \models CET^{\omega}_{\mathcal{L}}$, the ω-Herbrand universe $U^{\omega}_{\mathcal{L}}$ can be embedded into $M_1\rceil_{\mathcal{L}'}$. the universe $\mid M_1 \mid$ of $M_1\rceil_{\mathcal{L}'}$ may be however strictly bigger then $U^{\omega}_{\mathcal{L}}$, it may contain for instance "infinite chains" of elements, like e_i with i being (positive , 0 or negative) integer, and a function symbol interpreted as $f(e_i) = e_{i+1}$. Therefore M_1 is not yet the model we need for the proof of the theorem. Thanks to the mentioned embedding we may assume that $U^{\omega}_{\mathcal{L}}$ *is* a subset of $\mid M_1 \mid$. Recall that elements of $\mid M_1 \mid$ are sets of terms which are equivalence classes of $=_{M_0}$. Consider elements of $\mid M_1 \mid$ that correspond to free constants $k_i : i < \omega$, and chose a representant from the equivalence class representing every such element. This yields a countable collection of terms. The terms must however be just variables, because they represent free constants, about which $CET^{\omega}_{\mathcal{L}}$ asserts that they are not more complex terms. So we obtained collection $x_{i_j} : j < \omega$ of variables of $\mathcal{L}$. As the variables belong to equivalence classes representing free constants, we have $\| x_{i_j} \neq t \| \in \nabla$ for any $j < \omega$ and any term t distinct from x_{i_j}. Consider language $\mathcal{L}''$ of the same signature as $\mathcal{L}$ but whose variables ere exactly $x_{i_j} : j < \omega$. Consider the Lindenbaum algebra $\mathcal{A}''(CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\})$ of formulas of $\mathcal{L}''$ modulo $CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\}$. $\mathcal{A}''$ is a subalgebra of $\mathcal{A}$. $\nabla'' = \nabla \cap \mathcal{A}''$ is a Q-filter in $\mathcal{A}''$. Consider canonical algebraic model M'' for $CET^{\omega}_{\mathcal{L}} \cup \Gamma \cup \{B\}$ whose universe is the set of terms of $\mathcal{L}''$ and with logical values in $\mathcal{A}''$. For the proof of the theorem take the quotient $I = M''/\nabla''$.

7.4 Proofs for section 3.4

Proposition 7.4.1 (3.4.1) Any equality $t'=t''$ of terms effectively equivalent in CET^{ω} to a formula of (at least) one of the following types:

1. $\top$

2. $\bot$

3. $\bigwedge_i x_i=t_i$, where any x_i are variable and any t_i is a term.

Proof. If t' and t'' are identical, output $\top$. If one of t', t'' is a variable, say t' is x, output $x=t''$. If the main function symbols in t', t'' are different, output $\bot$ (individual constants are considered 0-ary function symbols). If main function symbols in t', t'' are the same: $t'=f(t'_1,\ldots,t'_n)$ and $t''=f(t''_1,\ldots,t''_n)$ apply the procedure repeatedly to equalities $t'_1=t''_1,\ldots,t'_n=t''_n$ eventually joining the outputs into a conjunction, and making the obvious simplifications if some of the conjuncts are $\top$ or $\bot$. (Notice that the only axioms used in the proof are 1-8.)

Definition 7.4.2 For any first-order language $\mathcal{L}$, by $\mathcal{L}^{is}$ we denote the language obtained by extending the alphabet of $\mathcal{L}$ by adding new unary relational symbol is_f for every function symbol f in $\mathcal{L}$ (Recall that individual constants are considered 0-ary function symbols). By $CET^{\omega}_{\mathcal{L}}$+IS we denote extension of $CET^{\omega}_{\mathcal{L}}$ in the language $\mathcal{L}^{is}$, obtained by adjoining for every function symbol f the axiom: $is_f(x) \equiv \exists_y x=f(y)$.

Notice that $CET^{\omega}_{\mathcal{L}}$+IS is a definitional extension of $CET^{\omega}_{\mathcal{L}}$, hence it is a conservative extension.

Language $\mathcal{L}^{is}$ will be used while referring to some classes of formulas; for instance $is_f(x)$ is a Δ_0-formula in $\mathcal{L}^{is}$ while $\exists y(x{=}f(y))$ is not a Δ_0-formula in $\mathcal{L}$ - this is the reason for introducing this definition.

One more definition will be useful in further considerations.

Definition 7.4.3 We define function *depth* that assigns to any term or to any equality of terms, a natural number.

$depth(x) = 1$ for any variable x

$depth(c) = 1$ for any individual constant c

$depth(f(t_1, \ldots, t_n) = 1 + \max(depth(t_1), \ldots, depth(t_n))$

$depth(t_1{=}t_2) = \max(depth(t_1), depth(t_2))$.

For instance $depth(f(c, g(b, x)) = c){=}3$.

Lemma 7.4.4 Let $\mathcal{L}$ be a language containing no predicates except $=$. Then every Σ_1-formula B in $\mathcal{L}^{is}$ is effectively equivalent in $CET^{\omega}_{\mathcal{L}}$+IS either to $\bot$ or to a disjunction of canonical formulas whose free variables are contained among the free variables of B. Moreover if B is positive then the formulas in the obtained disjunction are positive canonical formulas.

Proof. Take any Σ_1-formula, assume that it is written as:

$$\exists_{y_1,\ldots,y_m} B(y_1, \ldots, y_m, x_1, \ldots, x_n) \ (m, n \geq 0)$$

and perform the following steps.

1. Introduce new variables $y'_1, \ldots, y'_n$ and replace the formula by

$$\exists_{y_1,\ldots,y_m,y'_1,\ldots,y'_n}(\wedge_{i\leq n} x_i{=}y'_i \ \wedge \ B(y_1, \ldots, y_m, y'_1, \ldots, y'_n)).$$

2. Transform $B(y_1,\ldots,y_m,y'_1,\ldots,y'_n)$ to disjunctive normal form $\bigvee_j B_j$ and replace the formula by $\bigvee_j \exists_{y_1,\ldots,y_m,y'_1,\ldots,y'_n}(\bigwedge_{i\leq n} x_i{=}y'_i \wedge B_j)$.

3. With each disjunct perform the following steps, then replace the disjuncts in

$$\bigvee_j \exists_{y_1,\ldots,y_m,y'_1,\ldots,y'_n}(\bigwedge_{i\leq n} x_i{=}y'_i \wedge B_j)$$

by the obtained formulas. If the resulting disjunction contains formulas $\top$ or $\bot$, do the obvious simplifications. Return the obtained formula as output.

Now we deal with a formula that can be written as:

$$\exists_{y_1,\ldots,y_m}(\bigwedge_{i\in I} x_i{=}y_i \wedge B'(y_1,\ldots,y_m))$$

where $x_i : i \in I$ are all the free variables in the formula, and B' is a conjunction of formulas of the following types: $\top$, $\neg\top$, $\bot$, $\neg\bot$, $t'{=}t''$, $t'{\neq}t''$, $is_f(t)$, $\neg is_f(t)$. We will show that any such formula is effectively equivalent either to $\bot$ or to a disjunction of canonical formulas.

4. Replace every conjunct $is_f(f(...))$ by $\top$ and every conjunct $is_f(g(...))$, where f and g are distinct function symbols, by $\bot$. Then for every conjunct $is_f(y)$, add at the beginning of the formula an existential quantifier $\exists_{y'_1,\ldots,y'_n}$ with new variables $y'_1,\ldots,y'_n$, and replace $is_f(y)$ by $y{=}f(y'_1,\ldots,y'_n)$.

After this step is done we obtain formula, that can be written as:

$$\exists_{y_1,\ldots,y_m}(\bigwedge_{i\in I} x_i{=}y_i \wedge \bigwedge_{j\in J} t'_j(y_1,\ldots,y_m){=}t''_j(y_1,\ldots,y_m) \wedge B'(y_1,\ldots,y_m))$$

where B' is a conjunction of formulas of the following types: $\top$, $\neg\top$, $\bot$, $\neg\bot$, $t'{\neq}t''$, $\neg is_f(t)$. For *any* formula of this kind we define an equivalence relation $\sim$ on the set $\{t'_j \mid j \in J\} \cup \{t''_j \mid j \in J\}$:

$t' \sim_0 t''$ iff one of the conjuncts $t'=t''$ or $t''=t'$ occurs in $\bigwedge_{j\in J} t'_j=t''_j$,

$\sim$ is the reflexive, transitive closure of $\sim_0$.

5. Replace every conjunct $t'=t''$ in B, according to proposition 3.4.1 (7.4.1), either by $\top$, or by $\bot$, or by a conjunction of equalities of the form $y=t$.

6. If there are variables y', y'' and terms t', t'' in the formula, such that $y' \sim y''$, $y' \sim_0 t'$, $y'' \sim_0 t''$, $1 < depth(t') \leq depth(t'')$ and t', t'' are distinct (y', y'' may be the same) replace equality $y''=t''$, by the formula obtained from $t'=t''$ according to proposition 3.4.1 (7.4.1).

7. Repeat operations 5 or 6 as many times as they apply.

Notice that steps 5 or 6 replace equality $t'=t''$ by a finite number of equalities of smaller depth, so according to Kõnig's lemma the process of repeated applications of 5 or 6 will eventually terminate.

After it terminates we obtain a formula that can be written as:

$$\exists_{y_1,\ldots,y_m}(\bigwedge_{i\in I} x_i=y_i \ \wedge \ \bigwedge_{j\in J}\bigwedge_{k\in K_j} y_j=t_j(y_1,\ldots,y_m) \ \wedge \ B'(y_1,\ldots,y_m))$$

where no equivalence class of $\sim$ contains two terms that are not variables, and where B' is a conjunction of formulas of the following types: $\top$, $\neg\top$, $\bot$, $\neg\bot$, $t'\neq t''$, $\neg is_f(t)$.

8. Remove every conjunct of the form $y=y$.

9. If there is an equivalence class of $\sim$ consisting of $y_{j_1},\ldots,y_{j_n},t$ (where t is a variable or a more complex term) do the following: If t contains one of the variables $y_{j_1},\ldots,y_{j_n}$ return to point 3 with $\bot$. Otherwise, remove every conjunct

$y_{j_l}=t'$ or $y=y_{j_l}$ $(l=1,\ldots,n)$ and replace every occurrence of y_{j_1} or ... or y_{j_n} by t. Repeat this step as many times as it applies.

After this step is done, we obtain a formula that can be written as:

$$\exists_{y_1,\ldots,y_m}(\wedge_{i\leq n} x_i=t_i \ \wedge \ B'(y_1,\ldots,y_m))$$

where B' is a conjunction of formulas of the following types: $\top$, $\neg\top$, $\bot$, $\neg\bot$, $t'\neq t''$, $\neg is_f(t)$.

10. Replace every conjunct $\neg is_f(f...))$ by $\bot$ and every conjunct $\neg is_f(g(..))$, where f and g are distinct symbols, by $\top$.

11. For every conjunct of the form $t'\neq t''$, represent $t'=t''$ according to proposition 3.4.1 (7.4.1) as $\top$ or $\bot$ or $\wedge_{j\in J} y_j=t_j$ and replace the conjunct by, respectively, $\bot$ or $\top$ or $\vee_{j\in J} y_j\neq t_j$.

12. Transform the formula to the disjunctive normal form $\exists_{y_1,\ldots,y_m} \vee_i B_i$ and then to $\vee_i \exists_{y_1,\ldots,y_m} B_i$.

13. With each disjunct perform the following steps, then replace the disjuncts in $\vee_i \exists_{y_1,\ldots,y_m} B_i$ by obtained formulas. Return with the resulting formula to point 3.

Now we deal with a formula that can be written as:

$$\exists_{y_1,\ldots,y_m}(\wedge_{i\in I} x_i=t_i \ \wedge \ \wedge_{j\in J} y_j\neq t'_j \ \wedge \ \wedge_{(f,k)\in K} \neg is_f(t_k) \wedge B')$$

where B' is a conjunction of formulas of the following types: $\top$, $\neg\top$, $\bot$, $\neg\bot$.

14. If there is a conjunct $y_j \neq t'_j$ such that t'_j is identical with y_j, replace the conjunct by $\bot$. If there is a conjunct $y_j \neq t'_j$ such that $y_j \in var(t'_j)$ and t'_j is is not identical with y_j, replace the conjunct by $\top$. Repeat this operation as many times as it applies.

15. Replace every conjunct $\neg is_f(f...))$ by $\bot$ and every conjunct $\neg is_f(g(..))$, where f and g are distinct symbols, by $\top$.

16. If the conjunction contains formulas $\bot$ or $\neg\top$ return to point 13 with $\bot$, otherwise remove all the conjuncts of the forms $\top$ or $\neg\bot$ from the formula.

17. If among the conjuncts $y_j \neq t'_j$ $(j \in J)$ and $\neg is_f(y_k)$ $(\langle f, k\rangle \in K)$ there is one containing a variable y, such that $y \notin \bigcup_{i\in I} var(t_i)$, remove this conjunct. Repeat this step as many times as it applies.

One can see that the formula resulting after step 17 is equivalent in $CET^{\omega}_{\mathcal{L}}$ to the previous one. This is the only place in this proof, where we use the axioms of $CET^{\omega}_{\mathcal{L}}$ that guarantee existence of infinitely many free constants.

18. Remove all quantifiers $\exists_y$ with variables y that do not occur in the matrix of the formula.

One can see that after each step the resulting formula is equivalent in $CET^{\omega}_{\mathcal{L}}$+**IS** to the previous one. Thus after all the steps are done we obtain either $\bot$ or a disjunction of canonical formulas, equivalent to the initial formula.

It is straightforward to check that if the initial formula is positive, then the formulas in the resulting disjunction are positive canonical formulas.

Example 7.4.5 In $CET^{\omega}_{\mathcal{L}}$

1. $\forall_{y_1,y_2,y_3} x_0 \neq f(g(y_1), f(g(y_2), y_3))$

   is equivalent to:

   $\exists_{y_0}(x_0 = y_0 \wedge \neg is_f(y_0)) \vee$

   $\exists_{y_1,y_2}(x_0 = f(y_1, y_2) \wedge \neg is_g(y_1)) \vee$

   $\exists_{y_1,y_2}(x_0 = f(y_1, y_2) \wedge \neg is_f(y_2)) \vee$

   $\exists_{y_1,y_2,y_3}(x_0 = f(y_1, f(y_2, y_3)) \wedge \neg is_g(y_2))$.

2. $x_0 \neq f(g(x_1), f(g(x_2), x_1))$

   is equivalent to:

   $\exists_{y_1,y_2,y_3}(x_0 = f(g(y_1), f(g(y_2), y_3)) \wedge y_1 \neq x_1 \wedge y_2 \neq x_2 \wedge y_3 \neq x_1) \vee$

   $\forall_{y_1,y_2,y_3} x_0 \neq f(g(y_1), f(g(y_2), y_3))$

   is equivalent to:

   $\exists_{y_1,y_2,y_3}(x_0 = f(g(y_1), f(g(y_2), y_3)) \wedge y_1 \neq x_1 \wedge y_2 \neq x_2 \wedge y_3 \neq x_1) \vee$

   $\exists_{y_0}(x_0 = y_0 \wedge \neg is_f(y_0)) \vee$

   $\exists_{y_1,y_2}(x_0 = f(y_1, y_2) \wedge \neg is_g(y_1)) \vee$

   $\exists_{y_1,y_2}(x_0 = f(y_1, y_2) \wedge \neg is_f(y_2)) \vee$

   $\exists_{y_1,y_2,y_3}(x_0 = f(y_1, f(y_2, y_3)) \wedge \neg is_g(y_2))$.

The next lemma generalizes this example.

Lemma 7.4.6 Let $\mathcal{L}$ be a language with equality. Then

1. Any formula $\forall_{y_1,\ldots,y_n}(x_0 \neq t(y_1, \ldots, y_n))$ where t is a term being not a variable, and not containing two occurrences of the same variable, is effectively equivalent

in $CET^{\omega}_{\mathcal{L}}$ to a disjunction of formulas of the form:

$$\exists_{y_1,\ldots,y_j,\ldots,y_m}(x_0{=}t'(y_1,\ldots,y_j,\ldots,y_m) \wedge \neg is_f(y_j)).$$

2. Any inequality $x_0{\neq}t(x_1,\ldots,x_n)$ is effectively equivalent in $CET^{\omega}_{\mathcal{L}}$ to a formula:

$$\exists_{y_1,\ldots,y_j,\ldots,y_m}(x_0{=}t'(y_1,\ldots,y_j,\ldots,y_m) \wedge \bigvee_{j\leq m} y_j{\neq}x_{i_j}) \vee$$

$$\bigvee_k \exists_{y_1,\ldots,y_{j_k},\ldots,y_{m_k}}(x_0{=}t'_k(y_1,\ldots,y_{j_k},\ldots,y_{m_k}) \wedge \neg is_{f_k}(y_{j_k})).$$

Proof.

1. Induction on t.

   If t is $f(y_1,\ldots,y_k,t_1(y'),\ldots,t_m(y'))$ where $t_1(y'),\ldots,t_m(y')$ are not variables (to avoid complicated notation we assume that arguments of f are ordered so that more complex terms occur after variables, but the same reasoning will apply to other situations as well), then $\forall_{y_1,\ldots,y_k,y'}(x_0{\neq}t)$ is equivalent to:

$$\exists_z(z_0{=}z \wedge \neg is_f(z)) \vee$$

$$\bigvee_{i\leq m} \exists_{z_1,\ldots,z_{k+m}}(x_0{=}f(z_1,\ldots,z_{k+i},\ldots,z_{k+m}) \wedge \forall_{y'}(z_{k+i}{\neq}t_i(y')).$$

   By the induction hypothesis each formula $\forall_{y'}(z_{k+i}{\neq}t_i(y')$ can be replaced by a disjunction: $\bigvee_{j\in J_i} \exists_{y_1,\ldots,y_{m_j}}(z_{k+i}{=}t'_{i,j}(y_1,\ldots,y_{l_j},\ldots,y_{m_j}) \wedge \neg is_{f_{i,j}}(y_{l_j}))$.

   Then the disjunct $\exists_{z_1,\ldots,z_{k+m}}(x_0{=}f(z_1,\ldots,z_{k+i},\ldots,z_{k+m}) \wedge \forall_{y'}(z_{k+i}{\neq}t_i(y'))$ can be replaced by

$$\bigvee_{j\in J_i} \exists_{z_1,\ldots,z_{k+m},y_1,\ldots,y_{m_j}}(x_0{=}f(z_1,\ldots,t'_{i,j}(y_1,\ldots,y_{l_j},\ldots,y_{m_j}),\ldots,z_{k+m}) \wedge$$

$$\neg is_{f_{i,j}}(y_{l_j})).$$

2. If f is a variable x_i then $x_0{\neq}t$ is equivalent to $\exists_y(x_0{=}y \wedge y{\neq}x_i)$. If t is a more complex term, let $t'(y_1,\ldots,y_{n'})$ be a term such that no variable occurs twice in

t' and such that for certain substitution θ, $t(x_1,\ldots,x_n)=t'(y_1,\ldots,y_{n'})\theta$. The inequality $x_0\neq t(x_1,\ldots,x_n)$ is then equivalent in $CET^{\omega}_{\mathcal{L}}$ to:

$$\exists_{y_1,\ldots,y_{n'}} x_0=t'(y_1,\ldots,y_{n'}) \wedge \bigvee_{j\leq n'} y_j\neq(y_j\theta) \vee \forall_{y_1,\ldots,y_{n'}} \neg(x_0=t'(y_1,\ldots,y_{n'}))$$

and according to the first part of the lemma the last disjunct can be replaced by a disjunction satisfying the required conditions.

Theorem 7.4.7 (3.4.4) Every inequality $t'\neq t''$ of terms is effectively equivalent in CET^{ω} either to $\bot$ or to a disjunction of *super*-canonical formulas whose free variables are contained among the free variables of $t'\neq t''$.

Proof. By proposition 3.4.1 (7.4.1) $t'(x_1,\ldots,x_n)=t''(x_1,\ldots,x_n)$ is effectively equivalent either to $\top$ or to $\bot$ or to a conjunction $\bigwedge_i x_i=t_i$. Therefore $t'(x_1,\ldots,x_n)\neq t''(x_1,\ldots,x_n)$ is effectively equivalent, respectively, either to $\bot$ or to $\top$ or to a disjunction $\bigvee_i x_i\neq t_i$. By lemma 7.4.6 each $x_i\neq t_i$ is equivalent to $\exists_{y^i_1,\ldots,y^i_{m_i}} \bigvee_{j\in J_i} B^i_j$. Then $t'(x_1,\ldots,x_n)=t''(x_1,\ldots,x_n)$ is effectively equivalent to

$$\exists_{y^1_1,\ldots,y^1_{m_1} y^k_1,\ldots,y^k_{m_k}} \bigvee_{i\leq k} \bigvee_{j\in J_i} B^i_j.$$

(The proof required only axioms 1-8 of $CET^{\omega}_{\mathcal{L}}$+**IS**.)

Lemma 7.4.8 Let $\mathcal{L}$ be a language with equality. Then any formula:

$$\forall_{y_1,\ldots,y_m} B(x_1,\ldots,x_n,y_1,\ldots,y_m)$$

where B is a disjunction of formulas of the form $y_i\neq x_i$, is effectively equivalent in $CET^{\omega}_{\mathcal{L}}$ either to $\bot$ or to a disjunction of formulas of the form $x_i\neq x_j$.

Proof. For i=1,...,m, let $X(i) = \{j \mid$ there is a disjunct $y_i \neq x_j$ in $B\}$. Than as $CET^{\omega}_{\mathcal{L}}$ postulates existence of infinitely many elements,

$\forall_{y_1,\ldots,y_m} B(x_1,\ldots,x_n,y_1,\ldots,y_m)$ is equivalent to $\bigvee_{i\leq m} \bigvee_{j,k\in X(i),j\neq k} x_j \neq x_k$.

Lemma 7.4.9 Let $\sim$ be a binary predicate symbol in the language of a first-order theory $\mathcal{T}$, such that $\mathcal{T}$ proves that $\sim$ is a congruence. Let $B_i(\boldsymbol{x},\boldsymbol{y})$ (i=1,...,n) be formulas such that $\forall_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{y}'} B_i(\boldsymbol{x},\boldsymbol{y}) \wedge B_i(\boldsymbol{x},\boldsymbol{y}') \rightarrow \boldsymbol{y} \sim \boldsymbol{y}'$ (where $\boldsymbol{x},\boldsymbol{y},\boldsymbol{y}'$ are pairwise disjoint sequences of variables, and where $\langle y_1,\ldots,y_n\rangle \sim \langle y'_1,\ldots,y'_n\rangle$ means $\bigwedge_{i\leq n}(y_i \sim y'_i)$). Then for any formulas $C(\boldsymbol{y})$, $C_i(\boldsymbol{y},\boldsymbol{z})$: i=1,...,n (such that $\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$ are pairwise disjoint sequences of variables) the following equivalences are theorems of $\mathcal{T}$:

$$\neg\exists \boldsymbol{y}(B_1(\boldsymbol{x},\boldsymbol{y}) \wedge C(\boldsymbol{y})) \equiv \forall \boldsymbol{y}\neg B_1(\boldsymbol{x},\boldsymbol{y}) \vee \exists \boldsymbol{y}(B_1(\boldsymbol{x},\boldsymbol{y}) \wedge \neg C(\boldsymbol{y}))$$

and

$$\forall \boldsymbol{z} \bigvee_{i\leq n} \exists \boldsymbol{y}(B_i(\boldsymbol{x},\boldsymbol{y}) \wedge C_i(\boldsymbol{y},\boldsymbol{z})) \equiv \bigvee_{\emptyset\neq I\subseteq\{1,\ldots,n\}} \exists \boldsymbol{y}(\bigwedge_{i\in I} B_i(\boldsymbol{x},\boldsymbol{y}) \wedge \forall \boldsymbol{z} \bigvee_{i\in I} C_i(\boldsymbol{y},\boldsymbol{z})).$$

Proof. As $\mathcal{T}$ proves that $\sim$ is a congruence, every sentence which is not a theorem of $\mathcal{T}$ can be refuted in a model of $\mathcal{T}$ in which $\sim$ is interpreted as equality. Therefore it is enough to see that the equivalences given in the lemma hold in all models M where $M \models B_i(\boldsymbol{x},\boldsymbol{y}) \wedge B_i(\boldsymbol{x},\boldsymbol{y}')[v]$ implies $\boldsymbol{y}[v]=\boldsymbol{y}'[v]$.

The following lemma, which is a straightforward consequence of axiom 7 of $CET^{\omega}_{\mathcal{L}}$, states that symbol = in $CET^{\omega}_{\mathcal{L}}$ satisfies the assumption of the lemma 7.4.9.

Lemma 7.4.10 $CET^{\omega}_{\mathcal{L}} \vdash \bigwedge_{i\in I} x_i=t_i(y_1,\ldots,y_n) \wedge \bigwedge_{i\in I} x_i=t_i(y'_1,\ldots,y'_n) \rightarrow \bigwedge_{i\in I} y_i=y'_i$.

Lemma 7.4.11 Let $\mathcal{L}$ be a language containing no predicate symbols except $=$. Then the negation of any canonical formula is effectively equivalent in $CET^{\omega}_{\mathcal{L}}$ either to $\bot$ or to a disjunction of canonical formulas.

Proof. Take a canonical formula: $\exists y(\bigwedge_{i\in I} x_i{=}t_i \wedge \bigwedge_{j\in J} y_j{\neq}t'_j \wedge \bigwedge_{(f,k)\in K} \neg is_f(y_k))$. By lemma 7.4.9-part 1, and 7.4.10 (notice that condition 3 of the definition of a canonical formula is crucial here) negation of this formula is equivalent to:

$$\forall y(\neg \bigwedge_{i\in I} x_i{=}t_i) \ \vee \ \exists y(\bigwedge_{i\in I} x_i{=}t_i \wedge \neg(\bigwedge_{j\in J} y_j{\neq}t'_j \wedge \bigwedge_{(f,k)\in K} \neg is_f(y_k)).$$

The second disjunct is a Σ_1-formula in $\mathcal{L}^{is}$ so by lemma 7.4.4 it can be transformed to a disjunction of canonical formulas. The first disjunct can be rewritten as $\forall y \neg \bigvee_{i\in I} x_i{\neq}t_i$ and then by lemma 7.4.6-part 1, as: $B \vee \bigvee_i B_i$ where B is a Δ_0 formula in $\mathcal{L}^{is}$ and each B_i is of the form: $\exists_{y_1,\ldots,y_j,\ldots,y_m}(x_i{=}t'(y_1,\ldots,y_j,\ldots,y_m) \wedge \bigvee_{j\leq m} y_j{\neq}x_{i_j})$. As B is a Σ_1-formula it is equivalent to a disjunction of canonical formulas, so it remains to consider $\bigvee_i B_i$. By lemmas 7.4.10 and 7.4.9-part 2, $\bigvee_i B_i$ is equivalent to a disjunction of formulas of the form: $\exists y(\bigwedge_{i\in I} x_i{=}t'(y) \wedge \forall z \bigvee_{j\in J} C(y,z)$, where each C_j is a disjunction of formulas of type $z_k{\neq}z_l$. By lemma 7.4.8 we may transform $\forall z \bigvee_{j\in J} C(y,z)$ to a Δ_0-formula, each disjunct will be then a Σ_1-formula and can be further replaced by a disjunction of canonical formulas.

Theorem 7.4.12 (3.4.6) Normal form theorem

Every formula B containing no predicate symbols except $=$ is effectively equivalent in CET^{ω} either to $\bot$ or to a disjunction of canonical formulas whose free variables are contained among the free variables of B. Moreover, if B is positive then the formulas

in the resulting disjunction are positive canonical formulas.

Proof. It is enough to show that for every $n < \omega$, every Σ_n formula in $\mathcal{L}^{is}$ is effectively equivalent in $CET^{\omega}_{\mathcal{L}}+\mathbf{IS}$ either to $\bot$ or to a disjunction of canonical formulas. We perform induction on n.

$n = 1$

Every Σ_1-formula in $\mathcal{L}^{is}$ is equivalent to $\bot$ or to a disjunction of canonical formulas, by lemma 7.4.4.

$n = k + 1$

Take any Σ_n, use: lemma 7.4.4, lemma 7.4.11, the fact that any canonical formula is Σ_1 in $\mathcal{L}^{is}$, and the fact that conjuction or disjunction of Σ_1-formulas is equivalent to a Σ_1-formula, and transform the formula as follows:

Σ_{k+1} is $\exists\forall\Sigma_k$ and can be transformed to:

$\exists\forall\bigvee$canonical - can be transformed to:

$\exists\neg\exists\neg\bigvee$canonical - can be transformed to:

$\exists\neg\exists\bigwedge\neg$canonical - can be transformed to:

$\exists\neg\exists\bigwedge\bigvee$canonical - is $\exists\neg\exists\bigwedge\bigvee\Sigma_1$ - and can be transformed to:

$\exists\neg\exists\Sigma_1$ - is - $\exists\neg\Sigma_1$ - and can be transformed to:

$\exists\neg\bigvee$canonical - can be transformed to:

$\exists\bigwedge\neg$canonical - can be transformed to:

$\exists\bigwedge\bigvee$canonical - is $\exists\bigwedge\bigvee\Sigma_1$ and can be transformed to:

$\exists\Sigma_1$ - is Σ_1 and can be transformed to:

$\bigvee$canonical.

The proof of theorem 3.4.6 made use of lemma 7.4.4, lemma 7.4.6, lemma 7.4.8, lemma 7.4.9, lemma 7.4.10 and lemma 7.4.11. It turns out that for positive formulas there is much simpler algorithm, transforming them into disjunctions of positive canonical formulas. This algorithm is given in the proof of the next theorem.

Theorem 7.4.13 Let $\mathcal{L}$ be a language containing no predicate symbols except $=$. Then every positive formula B in $\mathcal{L}$ is effectively equivalent in $CET^{\omega}_{\mathcal{L}}$ either to $\bot$ or to a disjunction of positive canonical formulas whose free variables are contained among the free variables of B.

Proof. By lemma 7.4.4, any positive Σ_1-formula in $\mathcal{L}^{is}$ is effectively equivalent to $\bot$ or to a disjunction of positive canonical formulas. Note that for positive formulas the algorithm given in the proof of that lemma can be considerably simplified: parts 10-15 and 17 are not necessary. Now for the proof of our lemma it is enough to show that the thesis is true for positive Π_2-formulas B with one universal quantifier. Indeed, given arbitrary positive formula, we may turn it to a prenex form, and by replacing innermost Π_2-formula with one universal quantifier by a Σ_1-formula ($\top$, $\bot$ or disjunction of canonical formulas), and repeating this process, we may eliminate all universal quantifiers, further we apply lemma 7.4.4 again.

Consider positive Π_2-formula B with one universal quantifier: $\forall_v \exists_{y_1,\ldots,y_n} B'$ where B' is Δ_0. If we turn B' into disjunctive normal form $B'_1 \vee \ldots \vee B'_k \vee B_1 \vee \ldots \vee B_m$ where $B'_1 \vee \ldots \vee B'_k$ do not contain occurrences of variable v, then the formula may

be further transformed into $\exists_{y_1,\ldots,y_n}(B'_1 \vee \ldots \vee B'_k) \vee \forall_v(\exists_{y_1,\ldots,y_n}B_1 \vee \ldots \vee \exists_{y_1,\ldots,y_n}B_m)$. It is enough to show that our lemma holds for:

$$(*) \qquad \forall_v(\exists_{y_1,\ldots,y_n}B_1 \vee \ldots \vee \exists_{y_1,\ldots,y_n}B_m)$$

where $\exists_{y_1,\ldots,y_n}B_1, \ldots, \exists_{y_1,\ldots,y_n}B_m$ are positive Σ_1-formulas. Let us replace each of them, according to lemma 7.4.4 either by $\bot$ or by a disjunction of positive canonical formulas. Then if one of the disjuncts is $\top$, $(*)$ is equivalent to $\top$. If all the disjuncts are $\bot$, then $(*)$ is equivalent to $\bot$. If neither of these cases holds we may remove all formulas $\bot$ from the disjunction and obtain a formula of the form:

$$(**) \qquad \forall_v B''_1 \vee \ldots \vee B''_k$$

where any B''_i is a positive canonical formula.

We will show the lemma is true for $(**)$.

For each $i \leq n$ do the following

- if B''_i can be written in the form $\exists_{y_1,\ldots,y_n}(v{=}y_1 \wedge x_2{=}t_2 \wedge \ldots \wedge x_m{=}t_m)$ where neither of terms $t_2, \ldots, t_m$ contains the variable y_1, set B'''_i to:

  $$\exists_{y_2,\ldots,y_n}(x_2{=}t_2 \wedge \ldots \wedge x_m{=}t_m).$$

  Notice that in this case $B'''_i \rightarrow \forall_v B''_i$ and $\neg B'''_i \rightarrow \forall_v \neg B''_i$ are true in $U^\omega_{\mathcal{L}}$.

- If B''_i can be written in the form $\exists_{y_1,\ldots,y_n}(v{=}t \wedge x_2{=}t_2 \wedge \ldots \wedge x_m{=}t_m)$ where t is not a variable, set B'''_i to $\bot$.

- if B''_i can be written in the form $\exists_{y_1,\ldots,y_n}(v{=}y_1 \wedge x_2{=}t_2 \wedge \ldots \wedge x_m{=}t_m)$ where one of the terms $t_2, \ldots, t_m$ contains variable y_1, set B'''_i to $\bot$.

We claim that in $CET^{\omega}_{\mathcal{L}}$, $(\forall_v(B''_1 \vee \ldots \vee B''_k) \equiv (B'''_1 \vee \ldots \vee B'''_k)$. The implication to the left is obvious. For the proof of the implication to the right it is enough to construct the syntactical counterpart of the following reasoning.

Assume that $U^{\omega}_{\mathcal{L}} \not\models B'''_1 \vee \ldots \vee B'''_k[\nu]$ where $\nu=\{e_1/x_1,\ldots,e_n/x_n\}$ is a valuation of free variables which occur in the formula. Notice also that with the same valuation $U^{\omega}_{\mathcal{L}} \models \forall_v(B''_1 \vee \ldots \vee B''_k)[\nu]$. Let k be a free constant that does not occur in $e_1,\ldots,e_n$. In particular we have $U^{\omega}_{\mathcal{L}} \models B''_1 \vee \ldots \vee B''_k[\nu[k/v]]$ so there is i_0 such that $U^{\omega}_{\mathcal{L}} \models B''_{i_0}[\nu[k/v]]$.

B''_{i_0} can not be of the form $\exists_{y_1,\ldots,y_n}(v=y_1 \wedge x_2=t_2 \wedge \ldots \wedge x_m=t_m)$ because $U^{\omega}_{\mathcal{L}} \models \neg B'''_{i_0}[\nu]$ and $U^{\omega}_{\mathcal{L}} \models \neg B'''_{i_0} \rightarrow \forall_v \neg B''_{i_0}$.

B''_{i_0} can not be of the form $\exists_{y_1,\ldots,y_n}(v=t \wedge x_2=t_2 \wedge \ldots \wedge x_m=t_m)$ where t is not a variable, because as k does not occur in valuation ν, we have $U^{\omega}_{\mathcal{L}} \not\models v=t[\nu[k/v]]$.

B''_{i_0} can not be of the form $\exists_{y_1,\ldots,y_n}(v=y_1 \wedge x_2=t_2 \wedge \ldots \wedge x_m=t_m)$ where t_i contains variable y_1, because as k odes not occur in valuation ν, $U^{\omega}_{\mathcal{L}} \not\models v=y_1 \wedge \ldots \wedge x_i=t_i \wedge \ldots \wedge x_m=t_m$.

This proves that $\forall(B''_1 \vee \ldots \vee B''_k)$ is equivalent to positive Σ_1-formula $B'''_1 \vee \ldots \vee B'''_k$.

This completes the proof of the lemma.

7.5 Proofs for section 3.5

Theorem 7.5.1 (3.5.1) Decidability and completeness of $CET^{\omega}_{\mathcal{L}}$

Let $\mathcal{L}$ be a language containing no predicate symbols except $=$. Then the following hold:

1. $CET^{\omega}_{\mathcal{L}}$ is decidable.

2. $CET^{\omega}_{\mathcal{L}}$ is complete.

3. $CET^{\omega}_{\mathcal{L}} = Th(U^{\omega}_{\mathcal{L}})$.

Proof.

1. Let B be a sentence in language $\mathcal{L}$. To decide whether $CET^{\omega}_{\mathcal{L}} \vdash B$ apply the algorithm from the theorem 3.4.6 (7.4.12) or 7.4.13. In general algorithm returns as output either $\top$ or $\bot$ or a disjunction of canonical formulas with variables contained among the variables of the input formula. As B does not have any free variables, the output can be only $\top$ or $\bot$. Respectively $CET^{\omega}_{\mathcal{L}} \vdash B$ or $CET^{\omega}_{\mathcal{L}} \nvdash B$.

2. Follows from 3.

3. As $U^{\omega}_{\mathcal{L}} \models CET^{\omega}_{\mathcal{L}}$ we have $CET^{\omega}_{\mathcal{L}} \subseteq Th(U^{\omega}_{\mathcal{L}})$. Part 1 of this proof shows that any sentence is equivalent in $CET^{\omega}_{\mathcal{L}}$ either to $\top$ or to $\bot$. Therefore $CET^{\omega}_{\mathcal{L}}$ is a maximal consistent theory and $CET^{\omega}_{\mathcal{L}} = Th(U^{\omega}_{\mathcal{L}})$.

Theorem 7.5.2 (3.5.2) For positive formulas with guards B in, it is decidable whether $CET^{\omega} \vdash B$.

Proof. A formula is a theorem of a theory, if its universal closure is, so it is enough to provide a decision procedure for sentences. So assume that B is a sentence for the beginning assume that it does not contain any predicates except $=$. To decide whether $CET^{\omega}_{\mathcal{L}} \vdash B$ apply the algorithm from theorem 3.4.6. In general algorithm returns as output either $\top$ or $\bot$ or a disjunction of canonical formulas with variables

contained among the variables of the input formula. As B does not have any free variables, the output can be only $\top$ or $\bot$. Respectively $CET^{\omega}_{\mathcal{L}} \vdash B$ or $CET^{\omega}_{\mathcal{L}} \nvdash B$.

Before we proceed to the case of a positive formula with guards B that contains predicates other than $=$, notice the following. $CET^{\omega}_{\mathcal{L}} \vdash B$ iff $CET^{\omega}_{\mathcal{L}} \vdash B^{\bot}$, where $B^{\bot}$ is obtained from B by replacing each predicate except $=$ by $\bot$. Indeed to establish the implication to the right, it is enough to consider a formal proof of B from $CET^{\omega}_{\mathcal{L}}$, and replace all occurrences of predicates different from $=$ in this proof by $\bot$. To establish the implication to the left notice that by positivity of B, $\models B^{\bot} \rightarrow B$. Now in order to decide whether $CET^{\omega}_{\mathcal{L}} \vdash B$ it is enough to decide whether $CET^{\omega}_{\mathcal{L}} \vdash B^{\bot}$ and this can be done, because $B^{\bot}$ does not contain any predicate symbols except $=$

7.6 Proofs for section 3.6

Theorem 7.6.1 (3.6.5) Strong compactness

1. If $[\![t]\!] \subseteq \bigcup_{i \in I} [\![t_i]\!]$ then there exists $i_0 \in I$ such that $[\![t]\!] \subseteq [\![t_{i_0}]\!]$.
2. If $[\![\theta]\!] \subseteq \bigcup_{i \in I} [\![\theta_i]\!]$ then there exists $i_0 \in I$ such that $[\![\theta]\!] \subseteq [\![\theta_{i_0}]\!]$.

Proof. We will only the second part of the theorem. Assume $[\![\theta]\!] \subseteq \bigcup_{i \in I} [\![\theta_i]\!]$. Consider valuation $\iota = \{k_i/x_i \mid i < \omega\}$. We have $\tilde{\theta}[\iota] \in \bigcup_{i \in I} [\![\theta_i]\!]$. Take i_0 such that $\tilde{\theta}[\iota] \in [\![\theta_{i_0}]\!]$. There exists substitution ν such that $\tilde{\theta}[\iota] = \tilde{\theta}_{i_0}[\nu]$ and there exists substitution θ' such that $\nu = \tilde{\theta'}[\iota]$. Thus $\tilde{\theta} = \tilde{\theta}_{i_0}(\theta')$ and $[\![\theta]\!] \subseteq [\![\theta_{i_0}]\!]$.

Theorem 7.6.2 (3.6.7) Normal form theorem I

For every guarded substitution (θ WHERE S), there exists a finite number of canonical guarded substitutions (θ_i WHERE $S_i : i \leq k$ such that

$[\![\theta \text{ WHERE } S]\!] = \bigcup_{i \leq k} [\![\theta_i \text{ WHERE } S_i]\!]$. (If k=0 the right side of this equality is interpreted as $\emptyset$). Moreover substitutions (θ_i WHERE S_i) : $i \leq k$ can be effectively obtained from (θ WHERE S).

Proof. Immediately from 3.4.6 (7.4.12) by 3.6.6.

Corollary 7.6.3 (3.6.8) It is decidable whether $[\![\theta \text{ WHERE } S]\!] \neq \emptyset$.

Proof. Immediately from the theorem 3.4.6 (7.4.12), by lemma 3.6.6.

Theorem 7.6.4 (3.6.9) Normal form theorem II

For every guarded substitution θ WHERE S, *with S being a positive formula*, there exists a finite number of substitutions $\theta_i : i \leq k$ such that $[\![\theta \text{ WHERE } S]\!] = \bigcup_{i \leq k} [\![\theta_i]\!]$ (If $k = 0$ the right side of this equality is interpreted as $\emptyset$). Moreover substitutions $\theta_i : i \leq k$ can be effectively obtained from (θ WHERE S).

Proof. Immediately from the theorem 3.4.6 (7.4.12) - case of positive formulas (7.4.13), by lemma 3.6.6.

7.7 Proofs for section 3.7

Theorem 7.7.1 (3.7.1) Computing substitutions I

For positive formulas with guards it is decidable whether $CET^{\omega}_{\mathcal{L}} \vdash \exists_{\boldsymbol{x}} B(\boldsymbol{x})$. Moreover there exists an algorithm that takes as input a positive formula with guards

$B(\boldsymbol{x})$, and returns as output either the statement: "$CET^{\omega}_{\mathcal{L}} \vdash \neg\exists_{\boldsymbol{x}} B(\boldsymbol{x})$" if it is correct, or otherwise returns a finite sequence $(\theta_i \text{ WHERE } S_i) : i \leq k$ of canonical guarded substitutions, such that if $CET^{\omega}_{\mathcal{L}} \vdash B(\boldsymbol{x})(\theta \text{ WHERE } S)$ then $[\![\theta \text{ WHERE } S]\!] \subseteq \bigcup_{i\leq k}[\![\theta_i \text{ WHERE } S_i]\!]$.

Proof. Take any positive formula with guards $B(x_1,\ldots,x_n)$ and apply the algorithm from the proof of the theorem 3.4.6 (7.4.12), transforming it either to $\top$ or to $\bot$ or to a disjunction of canonical formulas. If the equivalent form is $\top$, output the identity substitution $\{x_i/x_i \mid i < \omega\}$. If the equivalent form is $\bot$, it means $CET^{\omega}_{\mathcal{L}} \vdash B(x_1,\ldots,x_n) \equiv \bot$ i.e. $CET^{\omega}_{\mathcal{L}} \vdash \forall_{x_1,\ldots,x_n}\neg B(x_1,\ldots,x_n)$, so output "$CET^{\omega}_{\mathcal{L}} \vdash \neg\exists_{x_1,\ldots,x_n} B(x_1,\ldots,x_n)$". If the equivalent form is a disjunction of canonical formulas, the for every canonical formula $\exists_{\boldsymbol{y}}(\bigwedge_{i\leq n} x_i{=}t_i \wedge S)$ in the disjunction, output $(\{t_i/x_i,\ldots,t_n/x_n\} \text{ WHERE } S)$.

Theorem 7.7.2 (3.7.2) Computing substitutions II

For *positive formulas* $B(\boldsymbol{x})$, it is decidable whether $CET^{\omega}_{\mathcal{L}} \vdash \exists_{\boldsymbol{x}} B(\boldsymbol{x})$. Moreover there exists an algorithm that takes as input a positive formula $B(\boldsymbol{x})$, and returns as output either the statement: "$CET^{\omega}_{\mathcal{L}} \vdash \neg\exists_{\boldsymbol{x}} B(\boldsymbol{x})$" if it is correct, or otherwise returns a finite sequence $\theta_i : i \leq k$ of substitutions, such that if $CET^{\omega}_{\mathcal{L}} \vdash B(\boldsymbol{x})\theta$ then $[\![\theta]\!] \subseteq [\![\theta_i]\!]$ for certain $i \leq k$.

Proof. Proceed as in the proof of theorem 3.7.1 (7.7.1). For any positive formula $B(\boldsymbol{x})$ you will obtain either "$CET^{\omega}_{\mathcal{L}} \vdash \neg\exists_{\boldsymbol{x}} B(\boldsymbol{x})$" or a sequence of substitutions

$\theta_i : i \leq k$ (guarded by $\top$) such that if $CET^{\omega}_{\mathcal{L}} \vdash \exists_{\boldsymbol{x}} B(\boldsymbol{x})\theta$ then $[\![\theta]\!] \subseteq \bigcup_i [\![\theta_i]\!]$. Now by lemma 3.6.5, (7.6.1) for every such θ there is i such that $[\![\theta]\!] \subseteq [\![\theta_i]\!]$.

7.8 Proofs for section 3.8

Proposition 7.8.1 (3.8.3) For any positive program P with guards:

1. P^C is in Clark's form.
2. P^C is a positive program with guards.
3. P and P^C are equivalent in classical logic.
4. $Comp(P)$ and $Comp(P^C)$ are equivalent in classical logic.
5. $T^{\omega}_P \uparrow \alpha = T^{\omega}_{P^C} \uparrow \alpha$, for any ordinal α.

Proof. 1-4. Straightforward.

5. Induction on α.

7.9 Proofs for section 3.9

Lemma 7.9.1 Let B be a positive formula with guards in a first-order language $\mathcal{L}$, possibly with equality. Let I_1, I_2 be ω-Herbrand interpretations in the universe $U^{\omega}_{\mathcal{L}}$. Then for every valuation ν, $I_1 \subseteq I_2$ and $I_1 \models B[\nu]$ implies $I_2 \models B[\nu]$.

Proof. Straightforward induction on B.

Proposition 7.9.2 (3.9.2) Monotonicity of T^{ω}_P

Let P be a positive program with guards. Then:

1. T_P^ω is monotone: $I_1 \subseteq I_2$ implies $T_P^\omega(I_1) \subseteq T_P^\omega(I_2)$

2. $\alpha_1 \leq \alpha_2$ implies $T_P^\omega{\uparrow}\alpha_1 \subseteq T_P^\omega{\uparrow}\alpha_2$.

Proof.

1. We have to show that for any atomic formula A and any valuation ν, $T_P^\omega(I_1) \models A[\nu]$ implies $T_P^\omega(I_1) \models A[\nu]$. Assume that $T_P^\omega(I_1) \models A[\nu]$ where A is an atomic formula. By definition of T_P^ω there is a statement $A \leftarrow B$ in P such that $I_1 \models B \wedge S[\nu]$. Then by 7.9.1, $I_2 \models B[\nu]$ and therefore $T_P^\omega(I_2) \models A[\nu]$.

2. There exists α' such that $\alpha_2 = \alpha_1 + \alpha'$. By straightforward induction on β one can prove that $(T_P^\omega{\uparrow}\beta) \subseteq (T_P^\omega{\uparrow}\beta + \alpha')$. Then for the proof of the lemma take $\beta = \alpha_1$.

Lemma 7.9.3 Let $\mathcal{L}$ be a first-order language, possibly with equality. Let P be a program with guards in $\mathcal{L}$ and let I be an ω-Herbrand interpretation for $\mathcal{L}$. Then

1. $I \models P$ iff $T_P^\omega(I) \subseteq I$.

2. $I \models P$ implies $T_P^\omega{\uparrow}\omega \subseteq I$.

Proof.

1. Cf. proposition 6.4 [44] p.38.

2. Using point 1, one can show by a straightforward induction on α, that $I \models P$ implies $(T_P^\omega)^\alpha(I) \subseteq I$. By 3.9.2, (7.9.2), $T_P^\omega{\uparrow}\alpha \subseteq (T_P^\omega)^\alpha(I)$. Using both inclusions with α being ω, we obtain $T_P^\omega{\uparrow}\omega \subseteq I$.

Lemma 7.9.4 Let $\mathcal{L}$ be a language, possibly with equality. Let P be a $+\Delta_\omega$-program with $+\Delta_\omega$-guards in $\mathcal{L}$. Let n be a natural number and let $B(x_1,\ldots,x_m)$ be $+\Delta_\omega$-formula with $+\Delta_\omega$-guards. For any valuation ν, for any sequence $k_1,\ldots,k_l$ of free constants,

if:

1. $k_1,\ldots,k_l$ are all distinct,
2. $k_1,\ldots,k_l$ do not occur in $x_{l+1}[\nu],\ldots,x_m[\nu]$, provided that $l<m$,
3. $T_P^\omega\restriction n \models B[\nu(k_1/x_1,\ldots,k_l/x_l)]$,

then:

$T_P^\omega\restriction n \models \forall_{x_1,\ldots,x_l} B[\nu]$.

Proof. Induction on n.

$n=0$. Induction on B.

B is $\top$ or $\bot$.

Obvious.

B is $p(t_1,\ldots,t_i)$.

As $T_P^\omega\restriction 0 \not\models p(t_1,\ldots,t_i)[\nu[k_1/x_1,\ldots,k_l/x_l]]$ the thesis is true.

B is $t'=t''$.

Assume $T_P^\omega\restriction 0 \models (t'=t'')[\nu[k_1/x_1,\ldots,k_l/x_l]]$. Let $\theta=\nu[x_1/x_1,\ldots,x_l/x_l]$ be a substitution in the language $\mathcal{L}^\mathcal{K}$. By the assumption of this case, terms (in $\mathcal{L}^\mathcal{K}$) $t'\theta$ and $t''\theta$ are identical. Therefore

$T_P^\omega\restriction 0 \models (t'=t'')[\nu[x_1/x_1,\ldots,x_l/x_l]]$ and $T_P^\omega\restriction 0 \models (\forall_{x_1,\ldots,x_l} t'=t'')[\nu]$.

B is $B_1 \wedge B_2$.

Assume $T_P^\omega\uparrow 0 \models (B_1 \wedge B_2)[\nu[k_1/x_1, \ldots, k_l/x_l]]$. Then

$T_P^\omega\uparrow 0 \models B_1[\nu[k_1/x_1, \ldots, k_l/x_l]]$ and $T_P^\omega\uparrow 0 \models B_2[\nu[k_1/x_1, \ldots, k_l/x_l]]$.

By induction hypothesis:

$T_P^\omega\uparrow 0 \models (\forall_{x_1,\ldots,x_l} B_1)[\nu]$ and $T_P^\omega\uparrow 0 \models (\forall_{x_1,\ldots,x_l} B_2)[\nu]$.

Therefore $T_P^\omega\uparrow 0 \models (\forall_{x_1,\ldots,x_l} B_1 \wedge B_2)[\nu]$.

B is $B_1 \vee B_2$.

Assume $T_P^\omega\uparrow 0 \models (B_1 \vee B_2)[\nu[k_1/x_1, \ldots, k_l/x_l]]$. Then:

$T_P^\omega\uparrow 0 \models B_1[\nu[k_1/x_1, \ldots, k_l/x_l]]$ or $T_P^\omega\uparrow 0 \models B_2[\nu[k_1/x_1, \ldots, k_l/x_l]]$.

By induction hypothesis:

$T_P^\omega\uparrow 0 \models (\forall_{x_1,\ldots,x_l} B_1)[\nu]$ or $T_P^\omega\uparrow 0 \models (\forall_{x_1,\ldots,x_l} B_2)[\nu]$.

Therefore $T_P^\omega\uparrow 0 \models (\forall_{x_1,\ldots,x_l} B_1 \vee B_2)[\nu]$.

B is $\forall_x B_1(x, x_1, \ldots, x_m)$.

Assume $T_P^\omega\uparrow 0 \models \forall_x B_1(x, x_1, \ldots, x_m)[\nu[k_1/x_1, \ldots, k_l/x_l]]$.

Consider free constant k distinct from all $k_1, \ldots, k_l$ and such that k does not occur in elements that ν substitutes for $x_{l+1}, \ldots, x_m$, provided that $l < m$. By the assumption of this case

$T_P^\omega\uparrow 0 \models B_1(x, x_1, \ldots, x_m)[\nu[k/x, k_1/x_1, \ldots, k_l/x_l]]$.

By induction hypothesis $T_P^\omega\uparrow 0 \models \forall_{x_1,\ldots,x_l}(\forall_x B_1(x, x_1, \ldots, x_m))[\nu]$.

B is $\exists_x B_1(x, x_1, \ldots, x_m)$.

Assume $T_P^\omega\uparrow 0 \models \exists_x B_1(x, x_1, \ldots, x_m)[\nu[k_1/x_1, \ldots, k_l/x_l]]$.

Then there exists element e such that

$T_P^\omega\restriction 0 \models B_1(x, x_1, \ldots, x_m)[\nu[e/x, k_1/x_1, \ldots, k_l/x_l]]$.

Recall that e is a term in the language $\mathcal{L}^K$. There exist a term $t(x_1, \ldots, x_l, y_1, \ldots, y_i)$ in $\mathcal{L}$, and function ν' assigning to variables $y_1, \ldots, y_i$ free constants different from $k_1, \ldots, k_l$ such that:

e is $t[k_1/x_1, \ldots, k_l/x_l, y_1[\nu']/y_1, \ldots, y_i[\nu']/y_i]$. We have

$T_P^\omega\restriction 0 \models B_1(x, x_1, \ldots, x_m)[\nu[k_1/x_1, \ldots, k_l/x_l, y_1[\nu']/y_1, \ldots, y_i[\nu']/y_i]]$.

By induction hypothesis:

$T_P^\omega\restriction 0 \models \forall_{x_1,\ldots,x_l} B_1(t, x_1, \ldots, x_m)[\nu[y_1[\nu']/y_1, \ldots, y_i[\nu']/y_i]]$,

thus: $T_P^\omega\restriction 0 \models \exists_{y_1,\ldots,y_i}\forall_{x_1,\ldots,x_l} B_1(t(x_1, \ldots, x_l, y_1, \ldots, y_i), x_1, \ldots, x_m)[\nu]$.

Thus: $T_P^\omega\restriction 0 \models \forall_{x_1,\ldots,x_l}\exists_{y_1,\ldots,y_i} B_1(t(x_1, \ldots, x_l, y_1, \ldots, y_i), x_1, \ldots, x_m)[\nu]$, so

$T_P^\omega\restriction 0 \models (\forall_{x_1,\ldots,x_l}\exists_{x'_1,\ldots,x'_i}\exists_{y_1,\ldots,y_i} B_1(t(x'_1, \ldots, x'_l, y_1, \ldots, y_i), x_1, \ldots, x_m))[\nu]$.

Therefore $T_P^\omega\restriction 0 \models (\forall_{x_1,\ldots,x_l}\exists_x B_1(x, x_1, \ldots, x_m))[\nu]$.

$n > 0$. Induction on B. All the cases except the one with B being $p(t_1, \ldots, t_i)$ are analogous to the subcases of $n = 0$. We do not repeat them.

B is $p(t_1, \ldots, t_i)$.

Assume $T_P^\omega\restriction n \models p(t_1, \ldots, t_i)[\nu[k_1/x_1, \ldots, k_l/x_l]]$. Then by definition of T_P^ω there is a clause $p(t'_1, \ldots, t'_i) \leftarrow B'$ in P and a substitution θ such that $p(t'_1, \ldots, t'_i)\theta$ is $p(t_1, \ldots, t_i)$ and

$(T_P^\omega\restriction n - 1) \models \exists_{y_1,\ldots,y_j} B'\theta[\nu[k_1/x_1, \ldots, k_l/x_l]]$,

where $y_1, \ldots, y_j$ are all the free variables in $B'\theta$ that do not occur in $p(t_1, \ldots, t_i)$. By induction hypothesis (in the induction on n):

$(T_P^\omega\restriction n-1) \models \forall_{x_1,\ldots,x_l}\exists_{y_1,\ldots,y_j} B'\theta[\nu]$. So for any $e_1,\ldots,e_l$

$T_P^\omega\restriction n-1 \models \exists_{y_1,\ldots,y_j} B'\theta[\nu[e_1/x_1,\ldots,e_l/x_l]]$. Thus for any $e_1,\ldots,e_l$,

$T_P^\omega\restriction n \models p(t_1',\ldots,t_i')\theta[\nu[e_1/x_1,\ldots,e_l/x_l]]$.

Therefore $T_P^\omega\restriction n \models (\forall_{x_1,\ldots,x_l} p(t_1',\ldots,t_i'))[\nu]$.

It turns out that for languages without equality, by a straightforward induction on n, on can prove a simpler and stronger version of the lemma above:
For any positive program P, any positive formula B, any natural number n and any valuation ν, $T_P^\omega\restriction n \models B[\nu[k_0/x_0]]$ implies $T_P^\omega\restriction n \models (\forall_{x_0})B[\nu]$.

Lemma 7.9.5 Let $\mathcal{L}$ be a first order language, possibly with equality. Assume one of the following.

(i) P is a $+\Delta_\omega$-program with $+\Delta_\omega$-guards and B is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards in $\mathcal{L}$.

(ii) P is a $+\Sigma$-program with Δ_ω-guards and B is a $+\Sigma$-formula with Δ_ω-guards in $\mathcal{L}$.

Then:

$T_P^\omega\restriction\omega \models B$ implies that there exists $n<\omega$ such that $T_P^\omega\restriction n \models B$.

Proof. First let us give the proof based on the assumption (i). Induction on B.

B is $\top$ or $\bot$ or $t'=t''$.

Obvious, because if one does not consider predicates other than $=$, $T_P^\omega\restriction n$ and $T_P^\omega\restriction\omega$ coincide.

B is $p(t_1,\ldots,t_m)$.

Assume $T_P^\omega\restriction\omega \models p(t_1,\ldots,t_m)[\nu]$, i.e. $p(t_1,\ldots,t_m)[\nu] \subseteq \bigcup_{n<\omega} T_P^\omega\restriction n$. Then there

exists $n < \omega$ such that $T^{\omega}_{P}\restriction n \models p(t_1, \ldots, t_m)[\nu]$.

B is $B_1 \wedge B_2$.

Assume $T^{\omega}_{P}\restriction\omega \models (B_1 \wedge B_2)[\nu]$. Then $T^{\omega}_{P}\restriction\omega \models B_1[\nu]$ and $T^{\omega}_{P}\restriction\omega \models B_2[\nu]$. By induction hypothesis, there exist $n_1, n_2 < \omega$ such that $T^{\omega}_{P}\restriction n_1 \models B_1[\nu]$ and $T^{\omega}_{P}\restriction n_2 \models B_2[\nu]$. Then by 3.9.2 (7.9.2), $T^{\omega}_{P}\restriction n \models (B_1 \wedge B_2)[\nu]$, where $n = \max(n_1, n_2)$.

B is $B_1 \vee B_2$.

Assume $T^{\omega}_{P}\restriction\omega \models (B_1 \vee B_2)[\nu]$. Then $T^{\omega}_{P}\restriction\omega \models B_1[\nu]$ or $T^{\omega}_{P}\restriction\omega \models B_2[\nu]$. By induction hypothesis, there exist $n < \omega$ such that $T^{\omega}_{P}\restriction n \models B_1[\nu]$ or $T^{\omega}_{P}\restriction n \models B_2[\nu]$. Then $T^{\omega}_{P}\restriction n \models (B_1 \vee B_2)[\nu]$.

B is $\exists_x B_1$.

Assume $T^{\omega}_{P}\restriction\omega \models \exists_x B_1[\nu]$. Then there exists element $e \in U^{\omega}_{\mathcal{L}}$ such that $T^{\omega}_{P}\restriction\omega \models B_1[\nu[e/x]]$. By the induction hypothesis there exists $n < \omega$ such that $T^{\omega}_{P}\restriction n \models B_1[\nu[e/x]]$. For this n, $T^{\omega}_{P}\restriction n \models \exists_x B_1[\nu]$.

B is $\forall_x B_1$.

Assume $T^{\omega}_{P}\restriction\omega \models \forall_x B_1[\nu]$. Take a free constant k that does not occur in terms substituted by ν for free variables in $\forall_x B_1$. We have $T^{\omega}_{P}\restriction\omega \models B_1[\nu[k/x]]$. By induction hypothesis there exists $n < \omega$ such that $T^{\omega}_{P}\restriction n \models B_1[\nu[k/x]]$. By 7.9.4 we have $T^{\omega}_{P}\restriction n \models \forall_x B_1[\nu]$.

The proof based on the assumption (ii) is analogous with the following modifications. In the induction on B the first case should be changed to "B is a guard", and this

case is obvious with the same explanation as before. Moreover the troublesame case "B is $\forall_x B_1$" is not needed now.

Theorem 7.9.6 (7.9.6) Least fix-point theorem Assume one of the following:

(i) P is a $+\Delta_\omega$-program with $+\Delta_\omega$-guards in $\mathcal{L}$ and B is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards in $\mathcal{L}$.

(ii) P is a $+\Sigma$-program with Δ_ω-guards in $\mathcal{L}$ and B is a $+\Sigma$-formula with Δ_ω-guards in $\mathcal{L}$.

If $\mathcal{L}$ is a language with equality, then the following conditions are equivalent.

1. $T_P^\omega\restriction\omega \models B$

2. There exists $n < \omega$ such that $T_P^\omega\restriction n \models B$

3. $lfpT_P^\omega \models B$

4. $CET_{\mathcal{L}}^\omega \cup P \vdash B$

5. $CET_{\mathcal{L}}^\omega \cup P_{\equiv} \vdash B$.

If $\mathcal{L}$ is a language without equality, each of 1, 2, 3 is equivalent to either of:

4'. $P \vdash B$

5'. $P_{\equiv} \vdash B$.

Proof.

$1 \Rightarrow 2$.

By monotonicity 3.9.2, (7.9.2), T_P^ω is monotone. Therefore $T_P^\omega\restriction n \subseteq T_P^\omega\restriction\omega$. If $T_P^\omega\restriction n \models B[\nu]$ then by lemma 7.9.1, $T_P^\omega\restriction\omega \models B[\nu]$.

$2 \Rightarrow 1$.

By lemma 7.9.5.

$1 \Rightarrow 4$.

Assume the contrary: $CET^{\omega}_{\mathcal{L}} \cup P \not\vdash B$ for some positive formula B. Then by theorem on completeness with respect to ω-Herbrand models 3.3.4 (7.3.2), there exists an ω-Herbrand interpretation I for $\mathcal{L}$ such that $I \models CET^{\omega}_{\mathcal{L}} \cup P$ and $I \not\models B$. By lemma 7.9.3, this implies that $T^{\omega}_{P} \subseteq I$. By the contrapositive version of lemma 7.9.1, $T^{\omega}_{P}\uparrow\omega \not\models B$.

$4 \Rightarrow 5$.

Obvious.

$5 \Rightarrow 3$.

We have $lfpT^{\omega}_{P} \models P_{\equiv}$ and obviously $lfpT^{\omega}_{P} \models CET^{\omega}_{\mathcal{L}}$. Therefore if $CET^{\omega}_{\mathcal{L}} \cup P_{\equiv} \vdash B$ then $lfpT^{\omega}_{P} \models B$.

$3 \Rightarrow 1$.

It is enough to prove $(T^{\omega}_{P}\uparrow\omega + 1) \subseteq (T^{\omega}_{P}\uparrow\omega)$ because such an inclusion implies $lfpT^{\omega}_{P} \subseteq T^{\omega}_{P}\uparrow\omega)$. Assume $T^{\omega}_{P}\uparrow\omega + 1 \models A[\nu]$ where A is an atomic formula. Ten by the definition of T^{ω}_{P} there is a clause $A' \leftarrow B'$ in P and a substitution θ such that $A'\theta$ is A and $T^{\omega}_{P}\uparrow\omega \models B'\theta[\nu]$. By point 1 of this theorem, for some $n < \omega$, $T^{\omega}_{P}\uparrow n \models B'\theta[\nu]$ and therefore $T^{\omega}_{P}\uparrow n + 1 \models A'\theta[\nu]$ i.e. $T^{\omega}_{P}\uparrow n + 1 \models A[\nu]$. By 3.9.2 $T^{\omega}_{P}\uparrow\omega \models A[\nu]$.

Proofs given above work for both assumptions, for languages with equality. The case of language without equality is analogous, but instead of 3.2.3 (7.2.1) use 3.3.4 (7.3.2).

8 PROOFS FOR CHAPTER 4

8.1 Proofs for section 4.1

We hope the reader will enjoy the contents of this section. Unfortunately starting from 4.2 sections did contain some theorems so there will be more work with proofs in further sections of this chapter.

8.2 Proofs for section 4.2

Proposition 8.2.1 (4.2.5) Let P be a definite program and let B be a definite goal. Let $P^{\exists}$ be the program obtained from P by adding to the body of every clause an existential quantifier binding those variables occurring free in the body but not in the head. Then $SLPG(P^{\exists}, B) = SLD(P, B)$.

Proof. By straightforward induction on n one can prove the following two facts:

1. If there is an SLPG-refutation of $P^{\exists} \cup \{\leftarrow B\}$ of the length n, with computed answer θ WHERE S, then in the same refutation θ can be considered a computed answer.

2. There exists SLPG-refutation of $P^{\exists} \cup \{\leftarrow B\}$ of the length n, with computed answer θ iff there exists θ' more general than θ and there exists SLD-refutation of $P \cup \{\leftarrow B\}$ of the length n, with computed answer θ'.

From these facts the proposition follows.

Lemma 8.2.2 Let B_0 be a positive subformula orf B. Let B_1, B_2 be formulas such that $\mathrm{var}(B_1) \subseteq \mathrm{var}(B_0)$ and $\mathrm{var}(B_2) \subseteq \mathrm{var}(B)_0$. Then $\Gamma \vdash B_1 \to B_2$ implies $\Gamma \vdash B[B_1/B_0] \to B[B_2/B_0]$.

Proof. Straightforward induction on B.

Lemma 8.2.3 Let P be a positive program with guards and let $\leftarrow B'$, $\leftarrow B''$ be positive goals with guards, such that $\leftarrow B''$ is SLPG-derived from $\leftarrow B'$. Then for every guarded substitution (θ WHERE S), $CET^{\omega}_{\mathcal{L}} Comp(P) \vdash B''(\theta$ WHERE $S)$ implies $CET^{\omega}_{\mathcal{L}} Comp(P) \vdash B'(\theta$ WHERE $S)$

Proof. Assume that in driving $\leftarrow B''$ from $\leftarrow B''$, atom $p(\boldsymbol{t})$ was selected in B', statement $p(\boldsymbol{x}) \leftarrow B$ of P^C was used and B'' resulted as $B'[B(\boldsymbol{t}/\boldsymbol{x})/p(\boldsymbol{t})]$. We have $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B \to p(\boldsymbol{x})$ and $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B(\boldsymbol{t}/\boldsymbol{x}) \to p(\boldsymbol{t})$. As $p(\boldsymbol{t})$ is a positive subformula of B', by lemma ??, $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B'[B(\boldsymbol{t}/\boldsymbol{x})/p(\boldsymbol{t})] \to B'$ i.e. $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B'' \to B'$. As guarded substitutions commute with implication, $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B''(\theta$ WHERE $S) \to B'(\theta$ WHERE $S)$ and the thesis follows.

Theorem 8.2.4 (4.2.6) Soundness of SLPG-resolution

Let P be a positive program with guards and let $\leftarrow B$ be a positive goal with guards. Then $SLPG(P, B) \supseteq [\![\theta$ WHERE $S]\!]$ implies $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B(\theta$ WHERE $S)$.

One can proove also that if $P \cup \{\leftarrow B\}$ does not contain the symbol $=$, then $SLPG(P, B) \supseteq [\![\theta]\!]$ implies $P \vdash B\theta$.

Proof. Assume that (θ WHERE S) is an SLPG-computed answer for $P \cup \{\leftarrow B\}$. Let $\leftarrow B =\!\leftarrow B_0, \leftarrow B_1, \ldots, \leftarrow B_n =\!\leftarrow B'$ be an SLPG-refutation such that $CET^{\omega}_{\mathcal{L}} \vdash B'(\theta$ WHERE $S)$. Of course $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B'(\theta$ WHERE $S)$ and by repeated applications of lemma 8.2.3 we obtain $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B(\theta$ WHERE $S)$.

Theorem 8.2.5 (4.2.7) Lifting lemma for SLPG-resolution

Let P be a positive program with guards and let $\leftarrow B$ be a positive goal with guards. Then: $SLPG(P, B) \supseteq [\![\theta$ WHERE $S]\!]$ iff $SLPG(P, B(\theta$ WHERE $S)) \supseteq [\![\epsilon]\!]$.

Proof. By straightforward induction on n one can show that: There is an SLPG-refutation of $P \cup \{\leftarrow B(\theta$ WHERE $S)\}$ of the length n, with computed answer (θ' WHERE S') iff there is an SLPG-refutation of $P \cup \{plB\}$ of the length n, with computed answer (θ WHERE S)(θ' WHERE S'). Then taking unions of sets representing computed answers for $P \cup \{\leftarrow B(\theta$ WHERE $S)\}$ and those for $P \cup \{\leftarrow B\}$ we obtain the required equivalence.

8.3 Proofs for section 4.3

Lemma 8.3.1 Let B ba a $+\Delta_{\omega}$-formula with $+\Delta_{\omega}$-guards. Then $U^{\omega}_{\mathcal{L}} \models B[\iota]$ implies $U^{\omega}_{\mathcal{L}} \models B$.

Proof. Let $B^{\perp}$ be the formula resulting from B by replacing predicate symbols other than $=$ by $\perp$. $U^{\omega}_{\mathcal{L}} \models B[\iota]$, thus also $U^{\omega}_{\mathcal{L}} \models B^{\perp}[\iota]$. By induction on $B^{\perp}$ we will show that $U^{\omega}_{\mathcal{L}} \models B^{\perp}$ which will end the proof because $\models B^{\perp} \rightarrow B$.

$B^{\perp}$ is $\top$ or $\perp$.

Obvious.

$B^{\perp}$ is $t'{=}t''$.

Assume $U_{\mathcal{L}}^{\omega} \models t'{=}t''[\iota]$. Then terms t', t'' are identical and $U_{\mathcal{L}}^{\omega} \models t'{=}t''$.

$B^{\perp}$ is $B_1 \wedge B_2$.

Straightforward.

$B^{\perp}$ is $B_1 \vee B_2$.

Straightforward.

$B^{\perp}$ is $\forall_x B_1(x, x_1, \ldots, x_n)$.

Assume $U_{\mathcal{L}}^{\omega} \models \forall_x B_1(x, x_1, \ldots, x_n)[\iota]$. Then $U_{\mathcal{L}}^{\omega} \models B_1(x, x_1, \ldots, x_n)[\iota]$ and by induction hypothesis: $U_{\mathcal{L}}^{\omega} \models B_1(x, x_1, \ldots, x_n)$ so $U_{\mathcal{L}}^{\omega} \models \forall_x B_1(x, x_1, \ldots, x_n)$.

$B^{\perp}$ is $\exists_x B_1(x, x_1, \ldots, x_n)$.

Assume $U_{\mathcal{L}}^{\omega} \models \exists_x B_1(x, x_1, \ldots, x_n)[\iota]$. Then there exists $e \in U_{\mathcal{L}}^{\omega}$ such that $U_{\mathcal{L}}^{\omega} \models B_1(x, x_1, \ldots, x_n)[\iota[e/x]]$. Take term t such that $e = t[\iota]$. We have $U_{\mathcal{L}}^{\omega} \models B_1(t, x_1, \ldots, x_n)[\iota]$. By induction hypothesis $U_{\mathcal{L}}^{\omega} \models B_1(t, x_1, \ldots, x_n)$, so $U_{\mathcal{L}}^{\omega} \models \exists_x B_1(x, x_1, \ldots, x_n)$.

Lemma 8.3.2 Let P be $+\Delta_\omega$-program with $+\Delta_\omega$-guards. Then for any natural number n, $T_P^{\omega} \uparrow n \models B[\iota]$ implies that there exists an SLPG-refutation of $P \cup \{\leftarrow B\}$ with computed answer ϵ.

Proof. Induction on n:

$n = 0$.

Assume $T_P^\omega \uparrow 0 \models B[\iota]$ i.e. $U_{\mathcal{L}}^\omega \uparrow 0 \models B[\iota]$ and by lemma 8.3.1, $U_{\mathcal{L}}^\omega \models B$. Then by 3.5.1, (7.5.1), $CET_{\mathcal{L}}^\omega \vdash B$ i.e. $CET_{\mathcal{L}}^\omega \vdash B[\epsilon]$ and ϵ is a computed answer substitution in a 0-step refutation of $P \cup \{\leftarrow B\}$.

$n > 0$

Induction on B.

B is a guard.

Assume $T_P^\omega \uparrow n \models B[\iota]$. Then as B does not contain any predicate symbols other than $=$, $T_P^\omega \uparrow n-1 \models B[\iota]$ and by induction hypothesis $P \cup \{\leftarrow B\}$ has an SLPG-refutation with computed answer ϵ.

B is $p(t)$.

Assume $T_P^\omega \uparrow n \models p(t)[\iota]$. By definition of T_P^ω there is a statement $p(t') \leftarrow B'$ in P and an amu θ' of t and t' such that $T_P^\omega \uparrow n-1 \models (B'\theta')[\iota]$. by induction hypothesis there is a refutation of $P \cup \{\leftarrow B'\theta'\}$ with computed answer ϵ. Concatanating at the beginning of this derivation, a one step derivation from $\leftarrow (t)$ to $\leftarrow B'\theta'$, gives a refutation of $P \cup \{\leftarrow p(t)\}$ with computed answer ϵ.

B is $B_1 \wedge B_2$.

Assume $T_P^\omega \uparrow n \models B_1 \wedge B_2[\iota]$. Then $T_P^\omega \uparrow n \models B_1[\iota]$ and $T_P^\omega \uparrow n \models B_2[\iota]$. By induction hypothesis both $\leftarrow B_1$ and $\leftarrow B_2$ have refutations with computed

answers ϵ. Now, starting with $\leftarrow B_1 \wedge B_2$ and performing all the steps of the refutation of $\leftarrow B_1$, and after that all the steps of the refutation of $\leftarrow B_2$, we obtain a refutation of $B_1 \wedge B_2$ with the computed answer ϵ.

B is $B_1 \vee B_2$.

Assume $T_P^\omega \uparrow n \models B_1 \vee B_2[\iota]$. Then $T_P^\omega \uparrow n \models B_1[\iota]$ or $T_P^\omega \uparrow n \models B_2[\iota]$ and without loosing the generality we may assume that $T_P^\omega \uparrow n \models B_1[\iota]$. By induction hypothesis B_1 has a refutation ending with $\leftarrow B'$ such that $CET_{\mathcal{L}}^\omega \vdash B'[\epsilon]$. Now, starting with $\leftarrow B_1 \vee B_2$ perform the steps of this refutation. At the end $\leftarrow B' \vee B_2$ will be obtained and as $CET_{\mathcal{L}}^\omega \vdash (B' \vee B_2)[\epsilon]$, identity substitution ϵ is an SLPG-computed answer.

B is $\forall_x B_1(x, x_1, \ldots, x_n)$.

Assume $T_P^\omega \uparrow n \models \forall_x B_1(x, x_1, \ldots, x_m)[\iota]$. Then $T_P^\omega \uparrow n \models B_1(x, x_1, \ldots, x_m)[\iota]$. By induction hypothesis $P \cup \{B_1(x, x_1, \ldots, x_m)\}$ has an SLPG-refutation ending with $\leftarrow B_1'(x, x_1, \ldots, x_m)$ such that $CET_{\mathcal{L}}^\omega \vdash B_1'(x, x_1, \ldots, x_m)\epsilon$. Perform all the steps of this refutation starting with $\forall_x B_1(x, x_1, \ldots, x_m)$. At the end $\forall_x B_1'(x, x_1, \ldots, x_m)$ will be obtained and as $CET_{\mathcal{L}}^\omega \vdash \forall_x B_1'(x, x_1, \ldots, x_m)$, identity substitution ϵ is an SLPG-computed answer.

B is $\exists_x B_1(x, x_1, \ldots, x_n)$.

Assume $T_P^\omega \uparrow n \models \exists_x B_1(x, x_1, \ldots, x_m)[\iota]$. Then there exists $e \in U_{\mathcal{L}}^\omega$ such that $T_P^\omega \uparrow n \models B_1(x, x_1, \ldots, x_m)[\iota[e/x]]$. Take term t such that $e = t[\iota]$. We have $T_P^\omega \uparrow n \models B_1(t, x_1, \ldots, x_m)[\iota]$. By induction hypothesis $P \cup \{\leftarrow B_1(t, x_1, \ldots, x_m)\}$ has an SLPG-refutation with computed

answer substitution ϵ. By lifting lemma $P \cup \{\leftarrow B_1(x, x_1, \ldots, x_m)\}$ has a refutation with computed answer $\epsilon[t/x]$. Perform the steps of this refutation starting from $\exists_x B_1(x, x_1, \ldots, x_n)$. This will give an SLPG-refutation of $P \cup \{\leftarrow \exists_x B_1(x, x_1, \ldots, x_m)\}$ with computed answer ϵ.

Lemma 8.3.3

Let P be a $+\Delta_\omega$-program with $+\Delta_\omega$-guards, and let B be a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards.

Then $CET^\omega_{\mathcal{L}} \cup Comp(P) \vdash B(\theta \text{ WHERE } S)$ implies $SLPG(P, B) \supseteq [\![\theta \text{ WHERE } S]\!]$.

Proof. By theorem 3.6.7 (7.6.2), it is enough to consider canonical guarded substitution $(\theta \text{ WHERE } S)$. Assume $CET^\omega_{\mathcal{L}} \cup Comp(P) \vdash B(\theta \text{ WHERE } S)$. Then as S (in canonical guarded substitution) is if the form $\bigwedge y_j \neq t'_j \wedge \bigwedge \forall_z y_k \neq f(z)$, and $B(\theta \text{ WHERE } S)$ is $B\theta \vee \neg S$, we have that $B(\theta \text{ WHERE } S)$ is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards. Then by Theorem 7.9.6 (??), there exists $n < \omega$ such that $T^\omega_P \uparrow n \models B(\theta \text{ WHERE } S)$. In particular $T^\omega_P \uparrow n \models B(\theta \text{ WHERE } S)[\iota]$, and by Lemma 8.3.2 there is an SLPG-refutation of $P \cup \{\leftarrow B(\theta \text{ WHERE } S)\}$ with computed answer ϵ. So $SLPG(P, B(\theta \text{ WHERE } S)) \supseteq [\![\epsilon]\!]$. by lifting lemma 4.2.7 (8.2.5), $SLPG(P, B) \supseteq [\![\theta \text{ WHERE } S]\!]$.

Lemma 8.3.4 Let B be a positive formula with guards. If $U^\omega_{\mathcal{L}} \models B[\![\nu]\!]$ then there exists guarded substitution $(\theta \text{ WHERE } S)$ such that $CET^\omega_{\mathcal{L}} \vdash B(\theta \text{ WHERE } S)$ and $\tilde{\nu} \in [\![\theta \text{ WHERE } S]\!]$.

Proof. Let $B^{\perp}$ be the formula obtained from B by replacing all predicate symbols other than $=$ by $\perp$. Since predicate symbols other than $=$ are interpreted in $U^{\omega}_{\mathcal{L}}$ as $\emptyset$, $U^{\omega}_{\mathcal{L}} \models B[\nu]$ implies $U^{\omega}_{\mathcal{L}} \models B^{\perp}[\nu]$. Therefore $B^{\perp}$ cannot be equivalent in $CET^{\omega}_{\mathcal{L}}$ to fl and by Theorem 3.4.6, $B^{\perp}$ is equivalent to a disjunction $\bigvee_{j\in J} B_j$ of canonical formulas. Take j such that $U^{\omega}_{\mathcal{L}} \models B_j[\nu]$. (Canonical) formula B_j is of the form $\exists y \bigwedge_{i\in I} x_i{=}t_i \wedge S$, where $x_i : i \in I$ are all the variables in the formula, and x_i's do not occur either in t_i's or in S. Then $CET^{\omega}_{\mathcal{L}} \models B_j(\{t_i/x_i \mid i \in I\}$ WHERE $S)$ and $\tilde{\nu} \in [\![\{t_i/x_i \mid i \in I\}$ WHERE $S]\!]$. Then also $CET^{\omega}_{\mathcal{L}} \models B^{\perp}(\{t_i/x_i \mid i \in I\}$ WHERE $S)$ and as $\vdash B^{\perp} \rightarrow B$, we have $CET^{\omega}_{\mathcal{L}} \models B(\{t_i/x_i \mid i \in I\}$ WHERE $S)$.

Lemma 8.3.5 Let B be a $+\Sigma$-formula with Δ_ω guards and let P be a $+\Sigma$-program with Δ_ω guards. Then for any $n < \omega$, $T^{\omega}_{P} \uparrow n \models B[\nu]$ implies that there exists an SLPG-refutation of $P \cup \{\leftarrow B\}$ with computed answer $(\theta'$ WHERE $S')$, such that $\tilde{\nu} \in [\![\theta'$ WHERE $S']\!]$.

Proof. Induction on n.

$n = 0$.

Assume $T^{\omega}_{P} \uparrow 0 \models B[\nu]$. By Lemma 8.3.4, there exists guarded substitution $(\theta$ WHERE $S)$, such that $CET^{\omega}_{\mathcal{L}} \vdash B(\theta$ WHERE $S)$ and $\tilde{\nu} \in [\![\theta$ WHERE $S]\!]$. So the zero step refutation consisting of single goal $\leftarrow B$ yields the required computed answer.

$n > 0$.

Induction on B

B is a guard.

Assume $T_P^\omega \uparrow n \models B[\nu]$. Then as B does not contain any predicates except $=$, $T_P^\omega \uparrow n-1 \models B[\nu]$, and by induction hypothesis the thesis follows.

B is $p(t)$.

Assume $T_P^\omega \uparrow n \models p(t)[\nu]$. By Proposition 3.8.3, without loosing generality we may assume that P is in Clark's form. By definition of T_P^ω there exists statement $p(x) \leftarrow B'$ in P such that $T_P^\omega \uparrow n-1 \models B'(t/x)[\nu]$. By induction hypothesis there is a refutation of $P \cup \{\leftarrow B'(t/x)\}$ with computed answer (θ' WHERE S') such that $\tilde{\nu} \in [\![\theta' \text{ WHERE } S']\!]$. As $\leftarrow B'(t/x)$ can be derived from $\leftarrow p(t)$ in one step, by concatenating these derivations we obtain refutation of $\leftarrow p(t)$ with computed answer (θ' WHERE S') and we have $\tilde{\nu} \in [\![\theta' \text{ WHERE } S']\!]$.

B is $B_1 \wedge B_2$.

Assume $T_P^\omega \uparrow n \models (B_1 \wedge B_2)[\nu]$. Then $T_P^\omega \uparrow n \models B_1[\nu]$ and $T_P^\omega \uparrow n \models B_2[\nu]$. By induction hypothesis there exist refutations, of $P \cup \{\leftarrow B_1\}$ ending with $\leftarrow B_1'$ with computed answer (θ_1 WHERE S_1), and of $P \cup \{\leftarrow B_2\}$ ending with $\leftarrow B_2'$ with computed answer (θ_2 WHERE S_2), such that $\tilde{\nu} \in [\![\theta_1 \text{ WHERE } S_1]\!]$ and $\tilde{\nu} \in [\![\theta_2 \text{ WHERE } S_2]\!]$. Starting with $\leftarrow B_1 \wedge B_2$ and performing all the steps of the first of these refutations, and then all the steps of the second one, we obtain an SLPG-derivationending with $\leftarrow B_1' \wedge B_2'$. It remains to show that there exists (θ' WHERE S') such that $CET_{\mathcal{L}}^\omega \vdash B_1' \wedge B_2'(\theta' \text{ WHERE } S')$ and $\tilde{\nu} \in [\![\theta' \text{ WHERE } S']\!]$. let θ be an mgu

of $\tilde{\theta_1}$ and $\tilde{\theta_2}$. As (θ' WHERE S') take ($\theta_1'\theta$ WHERE $S_1\theta \wedge S_2\theta$).

B is $B_1 \vee B_2$.

Assume $T_P^\omega \uparrow n \models (B_1 \wedge B_2)[\nu]$. Then $T_P^\omega \uparrow n \models B_1[\nu]$ or $T_P^\omega \uparrow n \models B_2[\nu]$. Without loosing generality we may assume that the first disjunct holds. By induction hypothesis, there exists refutation of $P \cup \{\leftarrow B\}$, ending with $\leftarrow B'$ with computed answer (θ_1 WHERE S_1) such that $\tilde{\nu} \in$ [θ_1 WHERE S_1]. Starting with $B_1 \vee B_2$ and performing the same steps, we obtain derivation for $P \cup \{\leftarrow B_1 \vee B_2\}$, ending with $\leftarrow B_1' \vee B_2'$, and we have $CET_{\mathcal{L}}^\omega \vdash (B_1' \vee B_2')(\theta_1$ WHERE $S_1)$ and $\tilde{\nu} \in$ [θ_1 WHERE S_1]].

B is $\exists_x B_1(x, x_1, \ldots, x_m)$.

Assume $T_P^\omega \uparrow n \models \exists_x B_1(x, x_1, \ldots, x_m)[\nu]$. So there exists $e \in U_{\mathcal{L}}^\omega$ such that $T_P^\omega \uparrow n \models B_1(x, x_1, \ldots, x_m)[\nu[e/x]]$. By induction hypothesis there exists refutation of $P \cup \{\leftarrow B_1(x, x_1, \ldots, x_m))$ ending with $\leftarrow B_1'$ with computed answer (θ' WHERE S') such that $\nu[\tilde{e}/x] \in$ [[θ' WHERE S']]. Starting from $\exists_x B_1(x, x_1, \ldots, x_m)$ and performing the same steps we obtain derivation for $P \cup \{\leftarrow \exists_x B_1(x, x_1, \ldots, x_m))$, ending with $\leftarrow \exists_x B_1'$. Take variable x' whixh does not occur either in B_1' or in S'. We have $CET_{\mathcal{L}}^\omega \vdash \exists_x B_1'(\theta'[x'/x]$ WHERE $S')$ and $\tilde{\nu} \in$ [$\theta'[x'/x]$ WHERE S']].

Lemma 8.3.6 Let P be a $+\Sigma$-program with Δ_ω-guards, and let B be a $+\Sigma$-formula with Δ_ω-guards.

Then $CET_{\mathcal{L}}^\omega \cup Comp(P) \vdash B(\theta$ WHERE $S)$ implies $SLPG(P, B) \supseteq$ [θ WHERE S]].

Proof. Assume $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B(\theta \text{ WHERE } S)$. Notice that $B(\theta \text{ WHERE } S)$ is $B\theta \vee \neg S$ so it is a $+\Sigma$-formula with Δ_ω-guards. By theorem 7.9.6 there exists $n < \omega$ such that $T^{\omega}_{P} \uparrow n \models B(\theta \text{ WHERE } S)$, so for every ν we have $T^{\omega}_{P} \uparrow n \models B(\theta \text{ WHERE } S)[\nu]$. Thus by lemma 8.3.5 for every ν there exists SLPG-refutation of $P \cup \{\leftarrow B(\theta \text{ WHERE } S)\}$ with computed answer $(\theta' \text{ WHERE } S')$ such that $\tilde{\nu} \in [\![\theta' \text{ WHERE } S']\!]$. Thus $SLPG(P, B(\theta \text{ WHERE } S)) \supseteq [\![\epsilon]\!]$ and by lifting lemma 4.2.7 (??), we obtain $SLPG(P, B) \supseteq [\![\theta \text{ WHERE } S]\!]$.

Theorem 8.3.7 (4.3.2) Soundness and completeness of SLPG-resolution

Assume one of the following:

(i) P is a $+\Delta_\omega$-program with $+\Delta_\omega$-guards, and B is a $+\Delta_\omega$-formula with $+\Delta_\omega$-guards.

(ii) P is a $+\Sigma_1$-program with Δ_ω-guards, and B is a $+\Sigma$-formula with Δ_ω-guards.

Then:

$SLPG(P, B) \supseteq [\![\theta \text{ WHERE } S]\!]$ iff $CET^{\omega}_{\mathcal{L}} \cup Comp(P) \vdash B(\theta \text{ WHERE } S)$.

Proof. Immediately from Theorem 4.2.6 (8.2.4) and lemmas 8.3.3 and 8.3.6.

8.4 Proofs for section 4.4

Proposition 8.4.1 (4.4.6) Let P be a positive program with guards, and let $\leftarrow B$ be a positive goal with guards. Then: $SLPGCN(P, B) = SLPG(P, B)$.

Proof. $SLPGCN(P, B) = SLPG(P^+, B^+)$. As P is a positive program with guards $P^+ = P^C$, so we have $SLPGCN(P, B) = SLPG(P^C, B) = SLPG(P, B)$.

Theorem 8.4.2 (4.4.9) Soundness of SLPGCN-resolution

Let P be a program and let $\leftarrow B$ be a goal. Then:

$SLPGCN(P, B) \supseteq [\theta \text{ WHERE } S]$ implies $CET^{\omega}_{\mathcal{L}} \cup Comp(P^{+}) \vdash B^{+}(\theta \text{ WHERE } S)$.

Proof. Immediately from 4.2.6 (8.2.4).

Theorem 8.4.3 (4.4.10) Completeness of SLPGCN-resolution

Let P be a program and let $\leftarrow B$ be a goal such that one of the following holds:

(i) P^{+} is a $+\Delta_{\omega}$-program with $+\Delta_{\omega}$-guards, and

B^{+} is a $+\Delta_{\omega}$-formula with $+\Delta_{\omega}$-guards.

(ii) P^{+} is a $+\Sigma$-program with Δ_{ω}-guards, and

B^{+} is a $+\Sigma$-formula with Δ_{ω}-guards.

Then: $SLPGCN(P, B) \supseteq [\theta \text{ WHERE } S]$ iff $CET^{\omega}_{\mathcal{L}} \cup Comp(P^{+}) \vdash B^{+}(\theta \text{ WHERE } S)$.

Proof. Immediately from 4.3.2 (8.3.7).

References

[1] H. Abramson and M. H. Rogers (eds), *Meta-Programming in Logic Programming*, MIT Press, Cambridge 1989.

[2] K. R. Apt, H. A. Blair and A. Walker, Towards a Theory of Declarative knowledge, in [51], pp. 89-148.

[3] K. R. Apt and M. H. van Emden, Contributions to the Theory of Logic Programming, *J. ACM* 29, 3, July 1982, pp. 841-862.

[4] J. Barwise (ed.), *Handbook of Mathematical Logic*, Studies in Logic - vol. 90, North Holland, New York 1977.

[5] J. Barwise, An Introduction to First-order Logic, in [4], pp. 5-46.

[6] N. Bleuzen-Guernalec and B. Gil, A Semantical Approach of an Internal Rendering of the Negation in General Logic Programs, draft 198?.

[7] A. Bruffaerts and E. Henin, Negation as Failure: Proofs, Inference Rules and Meta-interpreters, in [1], pp. 169-190.

[8] L. Cavedon, Continuity, Consistency, and Completeness Properties for Logic Programs, in [42], pp. 571-584.

[9] L. Cavedon and J. W. Lloyd, A Completeness Theorem for SLDNF Resolution, *Journal of logic programming*, 1989, vol. 7, No 3, pp. 177-192.

[10] D. Chan, Constructive Negation Based on Completed Database, in [38], pp. 111-125.

[11] K. L. Clark, Negation as Failure, in *Logic and Databases*, H. Gallaire and J. Minker (eds.), Plenum Press, New York, 1978, 193-322.

[12] K. L. Clark, Predicate Logic as a Computational Formalism, *Research Report DOC 79/59*, Dept. of Computing, Imperial College, 1979.

[13] S. K. Debray and P. Mishra, Denotational and Operational Semantics for Prolog, *Journal of Logic Programming*, 1988, vol. 5, No 1, pp. 61-91.

[14] D. DeGroot and G. Lindstrom (eds.), *Logic Programming, Functions, Relations and Equations*, Prentice-Hall, New York 1986.

[15] P. M. Dung and K. Kanchanasut, On the Generalized Predicate Completion of Non-Horn Programs, in [47], pp. 587-603.

[16] P. M. Dung and K. Kanchanasut, A Fixpoint Approach to Declarative Semantics of Logic Programs, in [47], pp. 604-625.

[17] M. Falaschi, G. Levi, M. Martelli, C. Palamidessi, A New Declarative Semantics for Logic Languages, in [38], pp. 993-1005.

[18] M. C. Fitting, *Intuitionistic Logic, Model Theory and Forcing*, North Holland, New York 1969.

[19] M. Fitting, A Deterministic Prolog Fixpoint Semantics, *Journal of Logic Programming*, 1985, No 2, pp. 11-118.

[20] M. Fitting, A Kripke-Kleene Semantics for Logic Programs, *Journal of Logic Programming*, 1985, No 4, pp 295-312.

[21] M. Fitting, Partial Models and Logic Programming, *J. Theoretical Computer Science* 48 (1986), pp. 229-255.

[22] M. Fitting, Logic Programming Using A Compact Data Structure, *Proc. of the ASM SIGART International Sumposium on Methodologies for Intelligent Systems - ISMIS '86*, ACM 1986, pp. 247-255.

[23] M. Fitting, Negation as Refutation, Extended Abstract, *Proc.* 4^{th} *Annual Symposium on Logic in Computer Science - LICS '89*, Computer Society Press, 1989, pp. 63-70.

[24] M. Fitting, Bilattices in Logic Programming, *Proc. of* 20^{th} *International Symposium on Multiple-valued Logic*, pp. 238-246, IEEE Computer Society Press, 1990.

[25] M. Fitting, *First-order Logic and Automated Theorem Proving*, Springer Verlag, New York 1990.

[26] M. Fitting and M. Ben Jacob, Stratified and Three-valued Logic Programming Semantics, in [38], pp. 1054-1069.

[27] D. Gabbay, Modal Provability Foundations for Negation as Failure I, 4^{th} draft, Feb 1989, unpublished.

[28] M. Gelfond and V. Lifschitz, The Stable Model Semantics for Logic Programming, in [38], pp. 1070-1080.

[29] R. Goldblatt, *Logics of Time and Computation*, Lecture Notes No 7, Center for the Study of Language and Information (CSLI), 1987.

[30] J. Harland, A Kripke-like Model for Negation as Failure, in [47], pp 626-644.

[31] T. Hickey, Negation by Constraint Failure, draft 1987.

[32] J. Jaffar, J.-L. Lassez and J. W. Lloyd, Completeness of the negation as failure rule, *IJCAI-83*, Karlsruhe, 1983, pp. 500-506.

[33] J. Jaffar, J.-L. Lassez and M. J. Maher, A Logic Programming Language Scheme, in [14], pp. 441-468.

[34] *The Journal of Logic Programming*, North-Holland, published since 1982.

[35] R. A. Kowalski, Predicate Logic as a programming Language, *Information Processing '74*, Stockholm, North Holland, 1974, pp. 569-574.

[36] R. A. Kowalski, Algorithm = Logic + Control, *Comm. ACM 22*, 7, July 1979, pp. 424-436.

[37] R. A. Kowalski, The Relation Between Logic Progamming and Logic Specification, in C. A. R. Hoare and J. C. Shepherdson (eds.) *Mathematical Logic and*

Programming Languages, Prentice Hall, Englewood Cliffs, N.J. 1985, pp. 11-27.

[38] R. A. Kowalski and K. A. Bowen (eds.), *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, MIT Press, Cambridge 1988.

[39] K. Kunen, Some Remarks on the Completed Database, in [38], pp. 978-992.

[40] K. Kunen, Negation in Logic Programming, *Journal of logic programming* 1987, No 4, pp 289-308.

[41] K. Kunen, Signed Data Dependencies in Logic Programs, *Journal of logic programming*, 1989, vol. 7, No 3, pp.231-247.

[42] G. Levi and M. Martelli (eds.), *Logic Programming, Proceedings of the Sixth International Conference*, MIT Press, Cambridge 1989.

[43] V. Lifschitz, On the Declarative Semantics of Logic Programs with Negation, in [51], pp. 177-192.

[44] J. W. Lloyd, *Foundations of Logic Programming*, Second extended edition, Springer Verlag, 1987.

[45] J. Lobo, J. Minker and A. Rajasekar, Extending the Semantics of Logic Programs to Disjunctive Logic Programs, in [42], pp. 255.

[46] J. Lobo, J. Minker and A. Rajasekar, Weak Completion Theory for Non-Horn Programs, in [38], pp. 828-842.

[47] E. L. Lusk and R. A. Overbeek (eds.), *Logic Programming, Proceedings of the North American Conference 1989*, MIT Press, Cambridge 1989.

[48] J. Maluszynski, T. Näslund, Fail Substitutions for Negation as Failure, in [47], pp. 461-476.

[49] P. Mancarella, S. Martini, D. Pedreschi, Complete Logic Programs with Domain-Closure Axiom, *Journal of Logic Programming*, 1988, vol. 5, No 3, pp.263-276.

[50] L. T. McCarty, Clausal Intuitionistic Logic, Part I: Fixed-Point Semantics, Part II: Tableau Proof Procedures, *Journal of Logic Programming*, 1988, vol. 5, No 1: pp. 1-31, No 2: pp. 93-132.

[51] J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, 1988.

[52] J. Minker, Perspectives in Deductive Databases, *Journal of Logic Programming*, 1988, vol. 5, No 1, pp. 33-60.

[53] D. Pearce and G. Wagner, Reasoning with Negative Information I: Strong negation in Logic Programs, draft 1989.

[54] H. Przymusinska and T. Przymusinski, Weakly Perfect Model Semantics for Logic Programs, in [38], pp. 1106-1123.

[55] T. C. Przymusinski, On the Declarative Semantics of Deductive Databases and Logic Programs, in [51], pp. 193-216.

[56] T. C. Przymusinski, On Constructive Negation in Logic Programming, in [47], addendum.

[57] T. C. Przymusinski, On the Declarative and Procedural Semantics of Logic Programs, to appear in *Journal of Logic Programming.*

[58] A. Rajasekar and J. Minker, A Stratification Semantics for General Disjunctive programs, in [47], pp. 573-586.

[59] H. Rasiowa and R. Sikorski, *The Mathematics of Metamathematics*, third edition, PWN - Polish Scientific Publishers, Warsaw 1970.

[60] H. Rasiowa and R. Sikorski, A proof of Completeness Theorem of Gödel, *Fundamenta Mathematicae 37* (1950), pp. 193-200.

[61] H. Rasiowa and R. Sikorski, Algebraic treatment of the notion of satisfiability, *Fundamenta Mathematicae 40* (1953), pp. 62-95.

[62] J. C. Shepherdson, Negation in Logic Programming, in [51], pp. 19-88.

[63] J. R. Shoenfield, *Mathematical Logic*, Addison-Wesley, New York 1967.

[64] L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, 1986.

[65] G. Wagner, Algebraic Semantics of Propositional Logic Programs, draft 1989.

[66] G. Wagner, Vivid Reasoning with Negative Information, draft 1990.

[67] C. Walinsky, The semantics of logic programs that employ constructive negation, draft 1989.

[68] M. Wallace, A Computable Semantics for General Logic Programs, *Journal of Logic Programming* 1989, vol. 6, No 3, pp. 269-297.

[69] W. P. Weijland, Semantics for Logic Programs without Occur Check, draft 198?.

[70] D. A. Wolfram, M. J. Maher and J.-L. Lassez, A Unified Treatment of Resolution Strategies for Logic Programs, *Proc. Second Int. Conf. on Logic Programming*, Uppsala, 1984, pp. 263-276.

[71] A. Van Gelder, Negation as Failure Using Tight Derivations for General Logic Programs, *Journal of Logic Programming*, 1989, vol. 6, No 1 & 2, pp. 109-133.