Western University
Scholarship@Western

Digitized Theses

Digitized Special Collections

1986

Foundations Of Logic Programming With Equality

Kwok Hung Chan

Follow this and additional works at: https://ir.lib.uwo.ca/digitizedtheses

Recommended Citation

Chan, Kwok Hung, "Foundations Of Logic Programming With Equality" (1986). *Digitized Theses*. 1542.
https://ir.lib.uwo.ca/digitizedtheses/1542

This Dissertation is brought to you for free and open access by the Digitized Special Collections at Scholarship@Western. It has been accepted for inclusion in Digitized Theses by an authorized administrator of Scholarship@Western. For more information, please contact tadam@uwo.ca, wlswadmin@uwo.ca.

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming Every effort has been made to ensure the highest quality of reproduction possible

If pages are missing, contact the university which granted the degree

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R S C 1970, c C-30

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage Nous avons tout fait pour assurer une qualité supérieure de reproduction

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parveni une photocopie de qualité inférieure

Les documents qui font déjà l'objet d'un droit d'auteur (article de revue, examens publiés, etc) ne sont pas microfilmés

La reproduction, même partielle, de ce microfilm est soumis à la Loi canadienne sur le droit d'auteur, SRC 1970, c C-30

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL-339 (r 86/06)

Canadä

Foundations of Logic Programming with Equality

by
Kwok Hung Chan

Department of Philosophy

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario
September, 1986

© Kwok Hung Chan 1986

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-33037-6

Abstract

An obstacle to practical logic programming systems with equality is infinite computation. In the dissertation we study three strategies for eliminating infinite searches in Horn clause logic programming systems and develop an extension of Prolog that has the symmetry, transitivity and predicate substitutivity of equality built-in. The three strategies are:

1. Replacing logic programs with infinite search trees by equivalent logic programs with finite search trees;

2. Building into the inference machine the axioms that cause infinite search trees;

3. Detecting and failing searches of infinite branches.

The dissertation consists of two parts. General theories of the three strategies identified above are developed in Part I. In Part II we apply these strategies to the problem of eliminating infinite loops in logic programming with equality.

Part I. General Theories

We introduce the notion of *CAS-equivalent logic programs:* logic programs with identical correct answer substitutions. Fixpoint criteria for equivalent logic programs are suggested and their correctness is established. *Semantic reduction* is introduced as a means of establishing the soundness and completeness of extensions of SLD-resolution. The possibility of avoiding infinite searches by detecting infinite branches is explored. A class of SLD-derivations called *repetitive SLD-derivation* is distinguished. Many infinite derivations are instances of repetitive SLD-derivations. It is demonstrated that pruning repetitive SLD-derivations from SLD-trees does not cause incompleteness.

Part II. Extended Unification for Equality

An extension of SLD-resolution called *SLDEU-resolution* is presented. The symmetry, transitivity and predicate substitutivity of equality are built into SLDEU-resolution by extended unification. Extended unification, if unrestricted, also introduces infinite loops. We can eliminate some of these infinite loops by restricting SLDEU-resolution to non-repetitive right recursive SLDEU-resolution: this forbids extended unification of the first terms in equality subgoals and has a built-in mechanism for detecting repetitive derivations. The soundness and completeness of non-repetitive right recursive SLDEU-resolution are proved.

Acknowledgements

I would like to thank my advisor Professor W. Demopoulos for all his kindly assistance, encouragement and thorough guidance in completing this dissertation. I am very much indebted to Professor E.W. Elcock for the financial support obtained from his research funding that made the dissertation possible; his valuable criticisms are greatly appreciated. I would like to take the opportunity to thank Mr. P. Hoddinott who has offered a lot of helpful and constructive criticisms that led to many substantive improvements of the complete work. Moreover, I would like to thank the following examiners: Professor M.A. Brown, Professor M. Dawes, Professor A.N. Abdallah and Professor E.P. Stabler for their valuable suggestions which led to the final version of the dissertation. I am deeply grateful to my wife Shuk-fong for her forbearance, encouragement and support throughout my years at U.W.O. Her contribution in typing part of the manuscript is well taken. Last but not least, I would like to thank my parents for their strong personal support.

Table of Contents

List of Figures

The author of this thesis has granted The University of Western Ontario a non-exclusive license to reproduce and distribute copies of this thesis to users of Western Libraries. Copyright remains with the author.

Electronic theses and dissertations available in The University of Western Ontario's institutional repository (Scholarship@Western) are solely for the purpose of private study and research. They may not be copied or reproduced, except as permitted by copyright laws, without written authority of the copyright owner. Any commercial use or publication is strictly prohibited.

The original copyright license attesting to these terms and signed by the author of this thesis may be found in the original print version of the thesis, held by Western Libraries.

The thesis approval page signed by the examining committee may also be found in the original print version of the thesis held in Western Libraries.

Please contact Western Libraries for further information:
E-mail: libadmin@uwo.ca
Telephone: (519) 661-2111 Ext. 84796
Web site: http://www.lib.uwo.ca/

Chapter One
Introduction

1.1. Infinite Loops

We will consider those logic programming systems that construct refutations according to a predetermined algorithm. The construction of refutations involves a search through a tree of derivations. Logic programs with infinite search trees cause serious problems for practical logic programming systems. For systems which, like Prolog, adopt a depth first search strategy, infinite searches lead to *incompleteness*. This is because systems adopting a depth first search strategy follow a branch[1] to its tip before backtracking and trying other possibilities. Consequently, once a system has started searching an infinite branch, it can never finish the search and try other possibilities.

A desideratum of practical logic programming systems is a *finite response time:* given a program and a query, the machine should stop and give a correct answer in a finite time. Accordingly, it is very important for practical logical programming systems to avoid infinite searches of infinite trees.

Unfortunately, many commonly encountered theories have infinite search trees. For example, a relation P is *symmetric* if P satisfies the symmetry axiom:
sym: $P(x,y) \leftarrow P(y,x)$.
The search tree of programs containing the axioms *sym* have infinite branches.

[1] A branch here corresponds to a derivation.

Example 1.1:

$$\begin{array}{ll} \mathcal{P}: & P(a,b) \\ & P(x,y) \leftarrow P(y,x) \\ G_0: & \leftarrow P(c,d) \\ G_1: & \leftarrow P(d,c) \\ G_2: & \leftarrow P(c,d) \\ & \ldots \end{array}$$

Δ

As another example, consider the equality relation (=); the standard axioms for equality include a symmetry axiom, a transitivity axiom and a set of predicate substitutivity axioms. Thus the use of equality axioms may yield infinite search trees.

In this dissertation we study three strategies for eliminating infinite searches in Horn clause logic programming systems and develop an extension of Prolog that has the symmetry, transitivity and predicate substitutivity of equality built-in. The three strategies are:

1. Replacing logic programs with infinite search trees by equivalent logic programs with finite search trees;

2. Building into the inference machine the axioms that yield infinite search trees;

3. Detecting and failing searches of infinite branches.

The dissertation is organized as follows: In the rest of this chapter, we give a summary of SLD-resolution on which standard Prolog is based.

General theories of the three strategies identified above are developed in Part I. In *Chapter 2* we introduce the notion of *CAS-equivalent logic programs:* logic programs with identical correct answer substitutions. Fixpoint criteria for equivalent logic programs are suggested and their correctness is established. We define a program transformation $\mathcal{S}$ which transforms a program $\mathcal{P}$ to its symmetric extension $S(P)$. It is proved that $S(P)$ is CAS-equivalent to $P \cup \{sym\}$. In *Chapter 3* we extend SLD-resolution to *symmetric SLD-resolution*: this replaces the symmetry axiom by an inference rule. We introduce *semantic reduction* as a means of establishing the soundness and correctness of extensions of SLD-resolution such as symmetric SLD-resolution. In *Chapter 4* we explore the possibility of avoiding infinite searches by detecting infinite branches. A class of SLD-derivations called *repetitive SLD-derivation* is distinguished. Many infinite derivations are instances of repetitive SLD-derivations. It is demonstrated that pruning repetitive SLD-derivations from SLD-trees does not cause incompleteness.

In Part II of the dissertation we apply the methods developed in Part I to the problem of eliminating infinite loops in logic programming with equality. In *Chapter 5* an extension of SLD-resolution called *SLDEU-resolution* is presented. The symmetry, transitivity and predicate substitutivity of equality are built into SLDEU-resolution by extended unification. Extended unification, if unrestricted, also introduces infinite loops. We can eliminate some of these infinite loops by restricting SLDEU-resolution to *right recursive SLDEU-resolution*; this forbids extended unification of the first terms in equality subgoals. The soundness and completeness of SLDEU-resolution and right recursive SLDEU-resolution are established in *Chapter 6* by semantic reduction. In *Chapter 7* we restrict right recursive SLDEU-resolution further to *non-repetitive right recursive SLDEU-resolution*; this is right recursive SLDEU-resolution plus a mechanism for detecting repetitive derivations. The completeness of non-repetitive right recursive SLDEU-resolution is proved. A program transformation n is presented. n compiles a program P for non-repetitive right recursive SLDEU-resolution to an equivalent program P^n for Prolog. This contributes to the goal of building equality into Prolog without modifying Prolog. In *Chapter 8* we study the limitations of non-repetitive right recursive SLDEU-resolution and conclude the dissertation by a brief discussion of some interesting directions for further research.

1.2. Fixpoint Semantics of Definite Clause Logic Programs

We summarize briefly, in these sections, the fixpoint semantics of SLD-resolution [van Emden 76, Apt 82, Lloyd 84].

Let L be a first order language. An *atom* is a predication

$$P(t_1, \ldots, t_n)$$

where P is a predicate in L and $t_1, \ldots, t_n$ are terms in L. A *literal* is either an atom or the negation of an atom. Atoms are also called *positive literals* and negations of atoms, *negative literals*.

A *clause* is a set of literals in L. A *definite clause* is a clause with exactly one positive literal. A definite clause

$$C: \qquad \{A_0, \neg A_1, \neg A_2, \ldots, \neg A_m\}$$

is written as

$$A_0 \leftarrow A_1, \ldots, A_m.$$

C is interpreted as $\forall((A_1 \wedge \cdots \wedge A_m) \rightarrow A_0)$.[2] If $m=0$, C is called a *unit clause* and is written simply as A_0. A *program* in L is a finite set of definite clauses.

A *goal* (clause) is a clause containing only negative literals. A goal clause

$$G: \qquad \{\neg A_1, \ldots, \neg A_m\}$$

is written as

$$\leftarrow A_1, \ldots, A_m.$$

Each A_i, $1 \leq i \leq m$, is called a *subgoal* of G. G is interpreted as $\forall(\neg A_1 \vee \cdots \vee \neg A_m)$ which is logically equivalent to $\neg\exists(A_1 \wedge \cdots \wedge A_m)$.[3] Thus G is the negation of the *query* $\exists(A_1 \wedge \cdots \wedge A_m)$. A *Horn clause* is either a definite clause or a goal clause.

The *Herbrand Universe* U_L for L is the set of all ground terms (i.e., variable

[2] $\forall(C)$ is the universal closure of C.

[3] $\exists(C)$ is the existential closure of C.

free terms) in L. (If there is no constant symbol in L, we add one.) The *Herbrand base* B_L for L is the set of all ground atoms in L. A *Herbrand interpretation* I_L for L is a subset of the Herbrand base B_L. Ground atoms in I_L are considered to be true and ground atoms not in I_L are considered to be false. A *substitution* θ in L is a finite set of the form $\{t_1/x_1, \ldots, t_n/x_n\}$ where $x_1, \ldots, x_n$ are distinct variables and $t_1, \ldots, t_n$ are terms in L. θ is called a *ground substitution* iff each t_i is a ground term. Let E be an expression. $E\theta$ is the expression obtained by simultaneously replacing each occurrence of the variable x_i, $1 \leq i \leq n$, in E by the term t_i. A clause

C: $\quad A_0 \leftarrow A_1, \ldots, A_m$

is true in I_L iff for all ground substitutions θ, $A_0\theta$ is true in I_L if $A_1\theta, \ldots, A_m\theta$ are all true in I_L. I_L is called a *Herbrand model* of a set H of Horn clauses iff all clauses in H are true in I_L. H is satisfiable iff H has a Herbrand model. Hence we only need to consider Herbrand interpretations when we are considering the satisfiability of sets of (Horn) clauses. In the rest of the dissertation, reference to L will be omitted when the context makes it clear which language is intended.

A fixpoint operator T_P^L is associated with each program P in L.

Definition 1.1: (T_P^L)

Let P be a program in L. A function $T_P^L: 2^B \rightarrow 2^B$ is defined as follows. Let $I \subseteq B$. Then $T_P^L(I)$ is

$$\{A \in B \mid A \leftarrow A_1, \ldots, A_m \text{ is a ground instance of a clause in } P \text{ and } A_1, \ldots, A_m \in I\}.$$

△

The ordinal powers of T_P are defined as follows:

Definition 1.2: (Ordinal powers)

Let N be the set of natural numbers and ω be the first infinite ordinal number.

$$\begin{aligned} T_P \uparrow 0 &= \phi \\ T_P \uparrow i+1 &= T_P(T_P \uparrow i) \qquad \text{if } i \in N \\ T_P \uparrow \omega &= \cup\{T_P \uparrow i \mid i \in N\} \end{aligned}$$

△

Example 1.2: Let L be a language with a constant a and predicates P and Q. Let $\mathcal{P}$ be a program with the following clauses:

$P(a)$
$x=x$
$Q(x) \leftarrow P(x)$

The ordinal powers of $T_{\mathcal{P}}$ are as follows:

$T_{\mathcal{P}}\uparrow 0 = \phi.$
$T_{\mathcal{P}}\uparrow 1 = \{P(a).\ a=a\}.$
$T_{\mathcal{P}}\uparrow 2 = \{P(a).\ a=a.\ Q(a)\}.$
$T_{\mathcal{P}}\uparrow i = T_{\mathcal{P}}\uparrow 2$ for all $i \geq 3$, and
$T_{\mathcal{P}}\uparrow \omega = T_{\mathcal{P}}\uparrow 2.$

$\triangle$

Notation: (M_L)
Let $\mathcal{P}$ be a program in L. $M_L(\mathcal{P})$ denotes

$\cap\{\, I \subseteq B \mid I \text{ is a model of } \mathcal{P}\}.$

$M_L(\mathcal{P})$ is called the *least model* of $\mathcal{P}$.

$\triangle$

Proposition 1.1: [van Emden 76]
Let $\mathcal{P}$ be a program.

$T_{\mathcal{P}}\uparrow \omega = M(\mathcal{P}) = \{A \in B \mid \mathcal{P} \models A\}.$

$\triangle$

1.3. SLD-Resolution

SLD-resolution is Linear resolution for Definite clause programs with Selection function [Kowalski 71, Apt 82, Lloyd 84]. It is a refutation procedure. Let $\mathcal{P}$ be a program and let

Q: $\exists(A_1 \wedge \cdots \wedge A_m)$

be a query. First the negation

G: $\leftarrow A_1, \ldots, A_m$

of Q is formed. Then the system attempts to show the unsatisfiability of $\mathcal{P} \cup \{G\}$ by constructing a refutation from $\mathcal{P} \cup \{G\}$. The system will stop only when a refutation has been constructed or when it has tried all possibilities. There is only one inference rule: linear resolution. We need to introduce some notion before we define linear resolution.

Definition 1.3: (Selection rule)
A *selection rule* is a function mapping a goal G to an atom A in G. A is called the *selected atom*.

△

The selection rule used by Prolog is the leftmost rule: always select the leftmost atom of a goal.

Definition 1.4: (Composition of substitutions)
Let

θ: $\{r_1/x_1, \ldots, r_n/x_n, s_1/y_1, \ldots, s_m/y_m\}$

σ: $\{t_1/y_1, \ldots, t_m/y_m, u_1/z_1, \ldots, u_l/z_l\}$

be substitutions where $x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_l$ are distinct variables and $r_1, \ldots, r_n$, $s_1, \ldots, s_m$, $t_1, \ldots, t_m$, $u_1, \ldots, u_l$ are terms. The composition of θ and σ is

$$\theta\sigma = \{r_1\sigma/x_1, \ldots, r_n\sigma/x_n, s_1\sigma/y_1, \ldots, s_m\sigma/y_m, u_1/z_1, \ldots, u_l/z_l\}$$

△

Definition 1.5: Let θ and ρ be substitutions. We say that θ is more general than ρ, or equivalently, θ subsumes ρ ($\theta \geq \rho$), iff there is a substitution σ such that $\rho = \theta\sigma$.

△

Definition 1.6: ($\equiv$)
Let E_1 and E_2 be expressions in L. $E_1 \equiv E_2$ iff E_1 is syntactically identical to E_2.

△

Definition 1.7: (Unifier)
Let A and B be atoms. A *unifier* of A and B is a substitution for the variables in A and B such that $A\theta \equiv B\theta$. A unifier θ of A and B is called a *most general unifier* (mgu) iff all unifiers σ of A and B are subsumed by θ.

△

Definition 1.8: (Linear resolution)
Let R be a selection rule. Let

G: $\leftarrow A_1, \ldots, A_i, \ldots, A_m$

be a goal. Let

C: $A \leftarrow B_1, \ldots, B_q$

be a definite clause that shares no variables with G. If A_i is the atom selected by R and θ is an mgu of A and A_i, then the new goal

G': $\leftarrow(A_1, \ldots, A_{i-1}, B_1, \ldots, B_q, A_{i+1}, \ldots, A_m)$

is said to be *SLD-derived* from C and G via R using θ.

△

Definition 1.9: (Variant)
Let E_1 and E_2 be expressions. We say that E_1 and E_2 are *variants* iff there exist substitutions θ and σ such that $E_1 \equiv E_2\theta$ and $E_2 \equiv E_1\sigma$.

Δ

Definition 1.10: Let P be a program. An *input clause* from P is a variant of a clause in P.

Δ

Definition 1.11: (SLD-derivation)
Let P be a program, let G be a goal and let R be a selection rule. An *SLD-derivation* from $P \cup \{G\}$ is a sequence of triples

$T_0, T_1, T_2, \ldots$

satisfying the following conditions:

1. Each triple $T_i, i \geq 0$, is of the form $<G_i, C_i, \theta_i>$ where G_i is a goal, C_i is an input clause from P and θ_i is a substitution.

2. G_0 is G.

3. For each $i, i > 0$, G_i is SLD-derived from G_{i-1} and C_i via R using θ_i.

Δ

Since each clause C_i is universally quantified, occurrences of the same variable in different clauses are independent. It is required that the clauses

$G, C_1, C_2, \ldots$

in a derivation do not share their variables, i.e., variables in an input clause C_i do not occur in G nor in any other input clause $C_j, j \neq i$.

Definition 1.12: (SLD-refutation)
Let P be a program, let G be a goal and let R be a selection rule. An *SLD-refutation* from $P \cup \{G\}$ is a finite SLD-derivation

$<G_0, C_0, \theta_0>, <G_1, C_1, \theta_1>, \ldots, <G_k, C_k, \theta_k>$

from $P \cup \{G\}$ such that G_k is the empty clause $\square$. k is said to be the *length* of the refutation. $\theta_1\theta_2 \cdots \theta_k$ is said to be the *substitution of the refutation* and the substitution obtained by restricting $\theta_1\theta_2 \cdots \theta_k$ to the variables in G is said to be an *SLD-computed answer substitution* for $P \cup \{G\}$.

Δ

The empty clause $\square$ is interpreted as a contradiction. The existence of an SLD-refutation from $P \cup \{G\}$ demonstrates the unsatisfiability of $P \cup \{G\}$.

Definition 1.13: (Refutational completeness)
A logic programming system is said to be *refutationally complete* iff for all programs P and all goals G there is a refutation from $P \cup \{G\}$ if $P \cup \{G\}$ is unsatisfiable.

Δ

Besides unsatisfiability, we are also interested in correct substitutions for the variables in a query.

Definition 1.14: (Correct answer substitution)
Let P be a program in L and

G: $\leftarrow A_1, \ldots, A_m$

a goal. An *answer substitution* θ in L for $P \cup \{G\}$ is a substitution for the variables in G. θ is called a *correct answer substitution* iff

$$P \models \forall((A_1 \wedge \cdots \wedge A_m)\theta).$$

Δ

Definition 1.15: (CAS-completeness)
A logic programming system is said to be *CAS-complete* (complete with respect to correct answer substitutions) iff for all programs P and all goals G every correct answer substitution for $P \cup \{G\}$ is subsumed by a computed answer substitution for $P \cup \{G\}$.

Δ

SLD-resolution is both sound and complete:

Proposition 1.2: (Soundness and completeness, SLD-resolution) [Clark 79]
Let P be a program and let G be a goal.

1. **Soundness:** all computed answer substitutions of $P \cup \{G\}$ are correct answer substitutions.

2. **CAS-Completeness:** if θ is a correct answer substitution of $P \cup \{G\}$, then there is a computed answer substitution ρ and a substitution σ such that $\theta = \rho\sigma$.

Δ

1.4. Refutation Procedure

A *refutation procedure* is an algorithm for constructing refutations. We can organize the set of all SLD-derivations from $P \cup \{G\}$ into a search space called an *SLD-tree:*

Definition 1.16: (SLD-tree)

Let P be a program, let G be a goal and let R be a selection rule. Then the SLD-tree for $P \cup \{G\}$ via R is defined as follows:

1. Each node of the tree is a goal.

2. The root node is G.

3. Let

   N: $\leftarrow A_1, \ldots, A_i, \ldots, A_m$ $\quad (m \geq 1)$

   be a node and A_i be the atom selected by R. Then N has a descendant for each program clause

   $A \leftarrow B_1, \ldots, B_q$

   such that A and A_i are unifiable. Let θ be an mgu of A and A_i, the descendant is

   $\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_q, A_{i+1}, \ldots, A_m)\theta.$

4. Nodes which are the empty clause have no descendants.

Δ

A branch of an SLD-tree corresponds to an SLD-derivation. Branches corresponding to SLD-refutations are called *success branches*, other branches are called *failure branches*. A *search strategy* is a rule for searching SLD-trees to find SLD-refutations. Prolog uses a depth first backtracking search strategy which follows a branch to its tip before backtracking and trying other possibilities. The order in which program clauses are tried is the textual ordering of the clauses in the programs. The depth first strategy leads to incompleteness: once the system starts searching an infinite branch, it can never try other possibilities which might lead to the construction of a refutation. However, for most practical applications it is possible to avoid infinite loops by rearranging the order of the clauses in a program.

Part I
General Theories

Chapter Two
Equivalent Programs

A virtue of logic programming[1] is that the language of specification may be used as the language of implementation [van Emden 77, Elcock 81]. Hence we may use theories in first order languages that lucidly formulate our intuitive understanding of the problem domains as programs. However, the use of such programs frequently leads to logic programming systems with unacceptable performance. As we shown in Chapter 1, although the symmetry axiom

$sym.\quad P(x,y) - P(y,x)$

formulates our intuitive understanding of symmetry, the inclusion of *sym* in a logic program introduces infinite loops. A well known solution to this problem is that we begin with a logic program $\mathcal{P}$ that gives a lucid description of the problem domain in question; we then replace $\mathcal{P}$ by a program $\mathcal{P}'$ that is equivalent to $\mathcal{P}$ but with better performance.[2]

We study in this chapter the fixpoint foundations of equivalent programs. In Section 1 we introduce the notion of *CAS-equivalent logic programs*: logic programs with identical Correct Answer Substitutions. Examples are given which show that the notions CAS-equivalence, refutational equivalence and logical equivalence do not coincide in the case of definite clause logic programs. In Sections 2 and 3 fixpoint criteria for refutational and CAS-equivalence are suggested and their correctness is proved. In Section 4 a program transformation called *symmetric extension* is defined. It is shown that the symmetric extension of a program $\mathcal{P}$ is CAS-equivalent to $\mathcal{P} \cup \{sym\}$.

[1] *Logic programming* is used here as a general term which covers any system that uses a language of formal logic as its implementation language.

[2] See [Hogger 84, Chapter VII] for an exposition of formal program synthesis and for further references to the literature.

2.1. Equivalent Logic Programs

There is more than one way in which two programs $\mathcal{P}$ and $\mathcal{P}'$ may be equivalent to each other. Firstly, $\mathcal{P}$ and $\mathcal{P}'$ may be *logically equivalent*. Secondly, $\mathcal{P}$ and $\mathcal{P}'$ may be *refutationally equivalent* to each other in the following sense: for all goals G, $\mathcal{P} \cup \{G\}$ is unsatisfiable iff $\mathcal{P}' \cup \{G\}$ is unsatisfiable. If G is any sentence in the language, i.e., G is not required to be a negative clause, the two notions of *logical equivalence* and *refutational equivalence* coincide:

Proposition 2.1: Let L be a first order language. Let $\mathcal{P}$ and $\mathcal{P}'$ be two finite sets of sentences in L. $\mathcal{P}$ and $\mathcal{P}'$ are logically equivalent iff $\mathcal{P}$ and $\mathcal{P}'$ are refutationally equivalent.

△

Proof: ($\Rightarrow$) Straightforward.
($\Leftarrow$) Let G be the negation of the conjunction of the sentences in $\mathcal{P}$. $\mathcal{P} \cup \{G\}$ is unsatisfiable, hence $\mathcal{P}' \cup \{G\}$ is unsatisfiable and we have

$$\mathcal{P}' \models \neg G \models \mathcal{P}.$$

$\mathcal{P} \models \mathcal{P}'$ may be proved in a similar way.

Q.E.D.

However, the notions of *logical equivalence* and *refutational equivalence* do *not coincide* in the case of Horn clause logic programming where logic programs are sets of definite clauses and goals are negative clauses. There are definite clause logic programs which are refutationally equivalent but not logically equivalent. Consider the following programs:

Example 2.1:

$\mathcal{P}$: $P(a)$
$Q(b)$

$\mathcal{P}'$: $P(a)$
$Q(b)$
$Q(a) \leftarrow P(b)$

△

$\mathcal{P}' \models \mathcal{P}$, but $\mathcal{P} \not\models \mathcal{P}'$. Nevertheless, they are refutationally equivalent because they have the same least model: the set of ground atoms implied by $\mathcal{P}$ is the same as the set implied by $\mathcal{P}'$ (see §2.2). Note that if two programs $\mathcal{P}$ and $\mathcal{P}'$ are logically equivalent, then, of course, $\mathcal{P}$ and $\mathcal{P}'$ are refutationally equivalent.

In logic programming, we are more interested in computing answer substitutions than in just establishing the unsatisfiability of a goal relative to a program. Based on the notion of Correct Answer Substitution we introduce the notion of *CAS-equivalence:*

Definition 2.1: (CAS-equivalence)
Let $\mathcal{P}$ and $\mathcal{P}'$ be logic programs. $\mathcal{P}$ and $\mathcal{P}'$ are said to be *CAS-equivalent* (equivalent with respect to correct answer substitutions) iff for all goals G, the set of correct answer substitutions for $\mathcal{P} \cup \{G\}$ is identical to the set of correct answer substitutions for $\mathcal{P}' \cup \{G\}$.

△

Note that CAS-equivalent definite clause programs are refutationally equivalent. It is interesting to observe that definite clause logic programs with the same least model may not have the same set of correct answer substitutions. Consider the following programs in a language L with a as the only constant symbol.

Example 2.2:

$\mathcal{P}$: $P(a)$.
$Q(a)$.

$\mathcal{P}'$: $P(x)$.
$Q(a)$.

△

$\mathcal{P}$ and $\mathcal{P}'$ have the same least model: $\{P(a), Q(a)\}$. However, $\{\}$ is not a correct answer substitution for $\mathcal{P} \cup \{\leftarrow P(x)\}$, although it is a correct answer substitution for $\mathcal{P}' \cup \{\leftarrow P(x)\}$.

The notions of logical equivalence, CAS-equivalence and refutational equivalence are related as follows:

Proposition 2.2: Let $\mathcal{P}$ and $\mathcal{P}'$ be definite clause logic programs.

$\mathcal{P}$ and $\mathcal{P}'$ are logically equivalent
$\Rightarrow$ $\mathcal{P}$ and $\mathcal{P}'$ are CAS-equivalent
$\Rightarrow$ $\mathcal{P}$ and $\mathcal{P}'$ are refutationally equivalent.

△

In the rest of the dissertation we will concentrate on definite clause logic programs unless otherwise stated.

2.2. Refutational Equivalence

The notions of least model and refutational equivalence are related as follows.

Theorem 2.1: Let $\mathcal{P}$ and $\mathcal{P}'$ be programs. $\mathcal{P}$ and $\mathcal{P}'$ are refutationally equivalent iff $M(\mathcal{P}) = M(\mathcal{P}')$.

△

Proof: ($\Rightarrow$)

$$\begin{aligned} M(\mathcal{P}) &= \{A \in B \mid \mathcal{P} \cup \{\neg A\} \text{ is unsatisfiable } \} \\ &= \{A \in B \mid \mathcal{P}' \cup \{\neg A\} \text{ is unsatisfiable } \} \\ &= M(\mathcal{P}') \end{aligned}$$

($\Leftarrow$) Let G be a goal of the form

$$\leftarrow A_1, \ldots, A_m.$$

Suppose $\mathcal{P} \cup \{G\}$ is satisfiable. Then there is a Herbrand model I of $\mathcal{P}$ in which every ground instance

$$(\neg A_1 \vee \cdots \vee \neg A_m)\theta$$

of G is true. Hence there is an i, $1 \leq i \leq m$, such that $A_i\theta \notin I$. Since $M(\mathcal{P}) \subseteq I$, $A_i\theta \notin M(\mathcal{P})$. It follows that $M(\mathcal{P})$ is a model of G. Since $M(\mathcal{P}') = M(\mathcal{P})$, $M(\mathcal{P}')$ is also a model of G. Therefore $M(\mathcal{P}')$ is a model of $\mathcal{P}' \cup \{G\}$.

Q.E.D.

2.3. CAS-Equivalence

We illustrated in Example 2.2 that programs with identical least models may not be CAS-equivalent. Although $\mathcal{P}$ and $\mathcal{P}'$ have the same least model, they are not CAS-equivalent. The crucial point is that we cannot infer $\forall P(x)$ from the fact that every ground instance of $P(x)$ is in the least model of the program.

In studying the semantics of logic programs, it is a common practice to focus attention on the language $L_{\mathcal{P}}$ of the program $\mathcal{P}$ under consideration: the sets of constant, function and predicate symbols in $L_{\mathcal{P}}$ are precisely the sets of constant, function and predicate symbols in $\mathcal{P}$ (except in the case where $\mathcal{P}$ contains no constants). If we relax this restriction and consider *finite extensions* of $L_{\mathcal{P}}$, we have a fixpoint criterion for CAS-equivalence. The notion of a finite extension is defined as follows.

Definition 2.2: (Finite extension)
Let $\mathcal{P}$ be a program. A *finite extension* of $L_{\mathcal{P}}$ is a language L identical to $L_{\mathcal{P}}$ except that the sets of constant, function and predicate symbols in L are $\mathcal{C}_{\mathcal{P}} \cup \mathcal{C}$, $\mathcal{F}_{\mathcal{P}} \cup \mathcal{F}$ and $\mathcal{Q}_{\mathcal{P}} \cup \mathcal{Q}$ where $\mathcal{C}_{\mathcal{P}}$, $\mathcal{F}_{\mathcal{P}}$ and $\mathcal{Q}_{\mathcal{P}}$ are the sets of constant, function and predicate symbols in $\mathcal{P}$ and $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{Q}$ are finite sets of new constant, function and predicate symbols not occurring in $\mathcal{P}$.

Δ

Notation: ($[A]$)
Let A be an atom in a language L. $[A]$ denotes the set of all ground instances of A in L.

Δ

We have the following

Lemma 2.1: (Generalization)
Let $\mathcal{P}$ be a program. Let $A_1, \ldots, A_m$ be atoms in $L_{\mathcal{P}}$. Let $x_1, \ldots, x_n$ be the variables in $A_1, \ldots, A_m$. If there is a finite extension L of $L_{\mathcal{P}}$ with at least n new constants and $[A_1], \ldots, [A_m] \subseteq M_L(\mathcal{P})$, then $\mathcal{P} \models \forall(A_1 \wedge \cdots \wedge A_m)$.

Δ

Proof: (By *reductio ad absurdum*)
Let L be a finite extension of $L_{\mathcal{P}}$ with at least n new constants. Suppose $[A_1], \ldots, [A_m] \subseteq M_L(\mathcal{P})$. Assume that $\mathcal{P} \not\models \forall(A_1 \wedge \cdots \wedge A_m)$. Then $\mathcal{P} \cup \{\neg\forall(A_1 \wedge \cdots \wedge A_m)\}$ is satisfiable. $\neg\forall(A_1 \wedge \cdots \wedge A_m)$ is logically equivalent to $\exists(\neg A_1 \vee \cdots \vee \neg A_m)$. Let $\{c_1, \ldots, c_n, \ldots, c_k\}$ be the set of new constant symbols in L. Let θ be the substitution $\{c_1/x_1, \ldots, c_n/x_n\}$. $\mathcal{P} \cup \{(\neg A_1 \vee \cdots \vee \neg A_m)\theta\}$ is satisfiable.[3] Accordingly, there is a Herbrand model I_L of $\mathcal{P}$ in which $(\neg A_1 \vee \cdots \vee \neg A_m)\theta$ is true. It follows that there is a j, $1 \leq j \leq m$, such that $A_j\theta \notin I_L \supseteq M_L(\mathcal{P})$. Hence $A_j\theta \notin M_L(\mathcal{P})$, contradicting the assumption that $[A_j] \subseteq M_L(\mathcal{P})$.

Q.E.D.

Theorem 2.2: (CAS-equivalence)
Let $\mathcal{P}$ and $\mathcal{P}'$ be programs in $L_{\mathcal{P}} (= L_{\mathcal{P}'})$. $\mathcal{P}$ and $\mathcal{P}'$ are CAS-equivalent if for every non-negative integer k there is a finite extension L of $L_{\mathcal{P}}$ with k new constant, symbols such that

[3] See, e.g., [Loveland 78, §1.5] or any standard logic text on Skolem functions.

$$M_L(\mathcal{P}) = M_L(\mathcal{P}').$$

Δ

Proof: Let

$G: \quad \leftarrow A_1, \ldots, A_m$

be an arbitrary goal. Let $x_1, \ldots, x_n$ be the variables in G. Let L be a finite extension of $L_{\mathcal{P}}$ with n new constant symbols such that $M_L(\mathcal{P}) = M_L(\mathcal{P}')$. Let θ be a correct answer substitution for $\mathcal{P} \cup \{G\}$. Then

$$\mathcal{P} \models \forall((A_1 \wedge \cdots \wedge A_m)\theta).$$

Hence

$$[A_1\theta], \ldots, [A_m\theta] \subseteq M_L(\mathcal{P}) = M_L(\mathcal{P}').$$

By the Generalization Lemma,

$$\mathcal{P}' \models \forall(A_1\theta \wedge \cdots \wedge A_m\theta)$$
$$\models \forall((A_1 \wedge \cdots \wedge A_m)\theta).$$

Hence θ is a correct answer substitution for $\mathcal{P}' \cup \{G\}$. By the same argument, if θ is a correct answer substitution for $\mathcal{P}' \cup \{G\}$, then θ is a correct answer substitution for $\mathcal{P} \cup \{G\}$.

Q.E.D.

We have the following corollary justifying fixpoint proofs of CAS-equivalence.

Corollary 2.1: Let $\mathcal{P}$ and $\mathcal{P}'$ be programs in $L_{\mathcal{P}} (= L_{\mathcal{P}'})$. $\mathcal{P}$ and $\mathcal{P}'$ are CAS-equivalent if for every non-negative integer k there is a finite extension L of $L_{\mathcal{P}}$ with k new constant symbols such that

$$T_{\mathcal{P}}^{L} \uparrow \omega = T_{\mathcal{P}'}^{L} \uparrow \omega.$$

Δ

Proof: The corollary follows from Proposition 1.1 and Theorem 2.2.

Q.E.D.

Proofs of CAS-equivalence using $T_{\mathcal{P}}$ are relatively straightforward. This is because we need only consider least models and $T_{\mathcal{P}}$ determines, in a canonical way, the least model of $\mathcal{P}$.

2.4. Symmetric Extension

In this section we define a program transformation called *symmetric extension* and prove that the symmetric extension of a program $\mathcal{P}$ is CAS-equivalent to $\mathcal{P} \cup \{sym\}$.

Definition 2.3: (Symmetric extension)[4]
Let $\mathcal{P}$ be a program. The symmetric extension of $\mathcal{P}$ with respect to a predicate P is

$$S_P(\mathcal{P}) = \mathcal{P} \cup \{P(v,u) \leftarrow B_1, \ldots, B_q \mid P(u,v) \leftarrow B_1, \ldots, B_q \in \mathcal{P}\}.$$

(S_P is abbreviated as S when the predicate P can be determined from the context.)

Δ

Observe that $S(\mathcal{P})$ and $\mathcal{P} \cup \{sym\}$ are not logically equivalent. Although $\mathcal{P} \cup \{sym\}$ implies $S(\mathcal{P})$, $\mathcal{P} \cup \{sym\}$ is not implied by $S(\mathcal{P})$. Consider the counterexample where $\mathcal{P}$ is the empty set ϕ. $S(\phi)$ is ϕ and $\phi \not\equiv \phi \cup \{sym\}$. However, $S(\mathcal{P})$ and $\mathcal{P} \cup \{sym\}$ are CAS-equivalent, because $M_L(S(\mathcal{P}))$ equals $M_L(\mathcal{P} \cup \{sym\})$ for every finite extension L of $L_\mathcal{P}$.

Theorem 2.3: (Least model theorem for S)
Let $\mathcal{P}$ be a program. For all finite extensions L of $L_\mathcal{P}$

$$M_L(\mathcal{P} \cup \{sym\}) = M_L(S(\mathcal{P})).$$

Δ

Proof:
$\mathcal{P} \cup \{sym\}$ is abbreviated as $\mathcal{PS}$ in the proof.

$(\supseteq)$ $\quad M(\mathcal{PS}) \supseteq M(S(\mathcal{P}))$
It is straightforward to show that $\mathcal{PS} \models S(\mathcal{P})$. It follows from Proposition 1.1 that $M(\mathcal{PS}) \supseteq M(S(\mathcal{P}))$.

$(\subseteq)$ $\quad M(\mathcal{PS}) \subseteq M(S(\mathcal{P}))$
Consider an arbitrary finite extension L of $L_\mathcal{P}$. Since $M(\mathcal{PS}) = T_{\mathcal{PS}} \uparrow \omega$ and $M(S(\mathcal{P})) = T_{S(\mathcal{P})} \uparrow \omega$, it suffices to show that $T_{\mathcal{PS}} \uparrow \omega \subseteq T_{S(\mathcal{P})} \uparrow \omega$. We prove by induction that for all $n < \omega$, $T_{\mathcal{PS}} \uparrow n \subseteq T_{S(\mathcal{P})} \uparrow n$.

Base case: $n=0$.
$T_{\mathcal{PS}} \uparrow 0 = \phi \subseteq T_{S(\mathcal{P})} \uparrow 0$.

Inductive step.
Assume that the result holds for $n=k$.
Consider a ground atom $A \in T_{\mathcal{PS}} \uparrow k+1$. There is a ground instance

$$A \leftarrow B_1, \ldots, B_q$$

of a clause C in $\mathcal{PS}$ and $B_1, \ldots, B_q \in T_{\mathcal{PS}} \uparrow k$. Either $C \in \mathcal{P}$ or C is sym.

[4] The notion of a symmetric extension of a program is introduced in [Cox 85].

The case where $C \in \mathcal{P}$ is trivial because $\mathcal{P} \subseteq S(\mathcal{P})$ and $B_1, \ldots, B_q \in T_{\mathcal{P}S} \uparrow k$ by the inductive hypothesis.

Suppose C is *sym*. Then A is of the form $P(s,t)$.

$P(s,t) \leftarrow P(t,s)$

is an instance of *sym*, and $P(t,s) \in T_{\mathcal{P}S} \uparrow k$. By the inductive hypothesis $P(t,s) \in T_{S(\mathcal{P})} \uparrow k$. Hence there is a clause

C': $P(u,v) \leftarrow A_1, \ldots, A_q$

in $S(\mathcal{P})$ and a ground substitution θ such that $(P(u,v))\theta \equiv P(t,s)$ and $A_1\theta, \ldots, A_q\theta \in T_{S(\mathcal{P})} \uparrow k-1$. By the definition of $S(\mathcal{P})$

C'': $P(v,u) \leftarrow A_1, \ldots, A_q$

is also a clause in $S(\mathcal{P})$. Hence

$P(s,t) \leftarrow A_1\theta, \ldots, A_q\theta$

is a ground instance of C''. Since $A_1\theta, \ldots, A_q\theta \in T_{S(\mathcal{P})} \uparrow k-1$

$P(s,t) \in T_{S(\mathcal{P})} \uparrow k \subseteq T_{S(\mathcal{P})} \uparrow k+1.$

Q.E.D.

Corollary 2.2: (CAS-equivalence, symmetric extension)
Let $\mathcal{P}$ be a program. $\mathcal{P} \cup \{sym\}$ and $S(\mathcal{P})$ are CAS-equivalent.

Δ

Proof: The corollary follows from Theorems 2.2 and 2.3.

Q.E.D.

2.5. Conclusion

We have distinguished three notions of equivalence for logic programs: logical equivalence, refutational equivalence and CAS-equivalence. Fixpoint criteria for refutational equivalence and CAS-equivalence of logic programs have been suggested and their correctness proved. The fixpoint approach has been illustrated by a fixpoint proof of the CAS-equivalence of logic programs and their symmetric extensions. The transformation of a logic program to a CAS-equivalent program provides the foundation for *semantic reduction*: this is a general approach for reducing the semantics of extensions of SLD-resolution. Details of the semantic reduction approach are developed in the next chapter.

Chapter Three
Semantic Reduction

In the previous chapter we showed how the symmetry axiom for a predicate P can be subsumed by transforming a logic program $\mathcal{P}$ to its symmetric extension. In this chapter we consider an alternative solution: replace the symmetry axiom by an inference rule.

SYM: Let

G: $\leftarrow P(s,t), A_2, \ldots, A_m$

be a goal and let

C: $P(u,v) \leftarrow B_1, \ldots, B_q$

be an input clause. Then the new goal

G': $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta$

is derived from G and C using the unifier θ via the leftmost selection rule iff θ is an mgu of the pair $<P(s,t), P(u,v)>$ or of the pair $<P(s,t), P(v,u)>$.

We call an extension of SLD-resolution by the rule *SYM*, *symmetric SLD-resolution*. In this chapter, we develop a general approach, called *semantic reduction*, for reducing the semantics of such extensions of SLD-resolution to the semantics of SLD-resolution. We reduce the semantics of symmetric SLD-resolution by showing that the program transformation $\mathcal{S}$ *compiles* a program $\mathcal{P}$ for symmetric SLD-resolution to an equivalent program $\mathcal{S}(\mathcal{P})$ for SLD-resolution. Since $\mathcal{S}(\mathcal{P})$ and $\mathcal{P} \cup \{sym\}$ are CAS-equivalent, symmetric SLD-resolution is both sound and complete.

3.1. Symmetric SLD-Resolution

Symmetric SLD-resolution is like SLD-resolution except that subgoals with predicate P are processed according the the rule *SYM*. The notions of *selection rule*, *answer substitution*, *symmetric SLD-derivation*, *symmetric SLD-refutation* and *computed answer substitution*, are identical to those which occur in SLD-resolution. The notion of a *derivation step* is extended as follows:

Definition 3.1: (Derivation step, symmetric SLD-resolution for P)
Let P be a program and let

G: $\quad \leftarrow A_1, \ldots, A_m$

be a goal. Let

C: $\quad A \leftarrow B_1, \ldots, B_q$

be an input clause sharing no variables with G. Then the new goal

G': $\quad \leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta$

is *derived from* G and C using the unifier θ via the leftmost computation rule iff one of the following conditions holds:

1. A_1 and A are in the form $P(s,t)$ and $P(u,v)$, respectively, and θ is an mgu of the pair $<P(u,v), P(s,t)>$ or of the pair $<P(v,u), P(s,t)>$;

2. otherwise θ is an mgu of A and A_1.

Δ

3.2. Compilation Theorem

We can view an SLD-machine as a machine in which symmetric SLD-resolution is implemented and consider the program transformation S as a function that *compiles* a program P for symmetric SLD-resolution into a program $S(P)$ for the SLD-machine. The *compilation* is accomplished by simulating in $S(P)$ the effect of symmetric SLD-resolution on P. We have the following lemma for symmetric SLD-resolution:

Lemma 3.1: (Compilation, symmetric SLD-resolution)
Let P be a program and let G be a goal. There is a symmetric SLD-refutation from $P \cup \{G\}$ of length n with substitution θ iff there is an SLD-refutation from $S(P) \cup \{G\}$ of length n with the substitution θ.

Δ

Proof: (By induction on n.)

Base case: n=0.
G is □ and θ is $\{\}$. The result holds trivially.

Inductive case. Assume that the result holds for n.
Consider a sequence

$<G, \_, \_>,$[1] $<G_1, C_1, \theta_1>, \ldots, <G_{n+1}, C_{n+1}, \theta_{n+1}>$ (†)

($\Rightarrow$)

If sequence (†) is a symmetric SLD-refutation from $P \cup \{G\}$ of length $n+1$, then the subsequence

$<G_1, C_1, \theta_1>, \ldots, <G_{n+1}, C_{n+1}, \theta_{n+1}>$ (‡)

is a symmetric SLD-refutation from $P \cup \{G_1\}$ of length n with the substitution $\theta_2\theta_3 \cdots \theta_{n+1}$. By the inductive hypothesis, there is an SLD-refutation

$<G_1, \_, \_>, <G'_2, C'_2, \theta'_2>, \ldots, <G'_{n+1}, C_{n+1}, \theta'_{n+1}>$

from $S(P) \cup \{G_1\}$ of length n with substitution $\theta'_2\theta'_3 \cdots \theta'_{n+1}$, which is equal to $\theta_2\theta_3 \cdots \theta_{n+1}$.

Suppose G is $\leftarrow P(s,t), A_2, \ldots, A_n$; C_1 is $P(u,v) \leftarrow B_1, \ldots, B_q$; θ_1 is an mgu of $P(s,t)$ and $P(v,u)$; and G_1 is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta_1$. By the definition of S, the clause

$P(v,u) \leftarrow B_1, \ldots, B_q$

is in $S(P)$. The sequence

$<G, \_, \_>,$
$<G_1, P(v,u) \leftarrow B_1, \ldots, B_q, \theta_1>,$
$<G'_2, C'_2, \theta'_2>,$
$\ldots,$
$<G'_{n+1}, C'_{n+1}, \theta'_{n+1}>$

is an SLD-refutation from $S(P) \cup \{G\}$ of length $n+1$ with answer substitution $\theta_1\theta'_2\theta'_3 \cdots \theta'_{n+1}$, which is equal to $\theta_1\theta_2\theta_3 \cdots \theta_{n+1}$.

Otherwise

$<G, \_, \_>,$
$<G_1, C_1, \theta_1>,$
$<G'_2, C'_2, \theta'_2>,$
$\ldots,$
$<G'_{n+1}, C'_{n+1}, \theta'_{n+1}>$

is the required SLD-refutation.

($\Leftarrow$)

If sequence (†) is an SLD-refutation from $S(P) \cup \{G\}$, then subsequence (‡) is an SLD-refutation from $S(P) \cup \{G_1\}$ of length n with the substitution $\theta_2\theta_3 \cdots \theta_{n+1}$.

By the inductive hypothesis, there is a symmetric SLD-refutation

$<G_1, \_, \_>, <G'_2, C'_2, \theta'_2>, \ldots, <G'_{n+1}, C'_{n+1}, \theta'_{n+1}>$

[1] '_' is used as an anonymous variable.

from $\mathcal{P} \cup \{G_1\}$ of length n and the answer substitution $\theta_2'\theta_3' \cdots \theta_{n+1}'$, which is equal to $\theta_2\theta_3 \cdots \theta_{n+1}$.

Suppose G is $\leftarrow P(s,t), A_2, \ldots, A_n$; C_1 is $P(u,v) \leftarrow B_1, \ldots, B_q$; θ_1 is an mgu of $P(s,t)$ and $P(u,v)$; and G_1 is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta_1$. By the definition of $\mathcal{S}$ either

$$P(u,v) \leftarrow B_1, \ldots, B_q \qquad (a)$$

or

$$P(v,u) \leftarrow B_1, \ldots, B_q \qquad (b)$$

is in $\mathcal{P}$. Let C_1^* be clause (a) if (a) is in $\mathcal{P}$ otherwise let C_1^* be clause (b). The sequence

$<G, \_, \_>$,
$<G_1, C_1^*, \theta_1>$,
$<G_2', C_2', \theta_2'>$,
. . . .
$<G_{n+1}', C_{n+1}', \theta_{n+1}'>$

is a symmetric SLD-refutation from $\mathcal{P} \cup \{G\}$ of length $n+1$ with answer substitution $\theta_1\theta_2'\theta_3' \cdots \theta_{n+1}'$, which is equal to $\theta_1\theta_2\theta_3 \cdots \theta_{n+1}$.

Otherwise,

$<G, \_, \_>$,
$<G_1, C_1, \theta_1>$,
$<G_2', C_2', \theta_2'>$,
. . . .
$<G_{n+1}', C_{n+1}', \theta_{n+1}'>$

is the required symmetric SLD-refutation.

Q.E.D.

In general, the form of the Compilation Lemma is different for different extensions of SLD-resolution. We obtain the following theorem from the Compilation Lemma.

Theorem 3.1: (Compilation, symmetric SLD-resolution)
Let $\mathcal{P}$ be a program and let G be a goal. There is a symmetric SLD-refutation from $\mathcal{P} \cup \{G\}$ with computed answer substitution θ iff there is an SLD-refutation from $\mathcal{S}(\mathcal{P}) \cup \{G\}$ with computed answer substitution θ.

$\triangle$

The Compilation Theorem is weaker than the Compilation Lemma and we need the Compilation Theorem, rather than the Compilation Lemma, in subsequent steps in semantic reduction.

3.3. Soundness and Completeness

We have shown that the program transformation $\mathcal{S}$ compiles a program $\mathcal{P}$ for symmetric SLD-resolution to an equivalent program $\mathcal{S}(\mathcal{P})$ for SLD-resolution, and that $\mathcal{S}(\mathcal{P})$ is CAS-equivalent to $\mathcal{P} \cup \{sym\}$. We will now complete the semantic reduction of symmetric SLD-resolution by establishing the soundness and completeness of symmetric SLD-resolution. The results obtained in this section are completely general. They apply to any extension of SLD-resolution for which there is a program transformation with a Compilation Theorem and a CAS-equivalence Theorem. The notion of *correct answer substitution* is extended as follows.

Definition 3.2: ($\mathcal{A}$-correct answer substitution)
Let $\mathcal{A}$ be a set of definite clauses and let $\mathcal{P}$ be a program. Let

G: $\leftarrow A_1, \ldots, A_m$

be a goal. A substitution θ for the variables in G is said to be an *$\mathcal{A}$-correct answer substitution* for $\mathcal{P} \cup \{G\}$ if $\mathcal{P} \cup \mathcal{A} \models \forall((A_1 \wedge \cdots \wedge A_m)\theta)$.

$\triangle$

Theorem 3.2: ($\mathcal{A}$-correct answer substitution)
Let $\mathcal{A}$ be a set of definite clauses and $\mathcal{P}$ be a program. Let

G: $\leftarrow A_1, \ldots, A_m$

be a goal. A substitution θ is an $\mathcal{A}$-correct answer substitution for $\mathcal{P} \cup \{G\}$ iff θ is a correct answer substitution for $\mathcal{P} \cup \mathcal{A} \cup \{G\}$.

$\triangle$

Proof:

θ is an $\mathcal{A}$-correct answer substitution
iff $\mathcal{P} \cup \mathcal{A} \models \forall((A_1 \wedge \cdots \wedge A_m)\theta)$
iff θ is a correct answer substitution for $\mathcal{P} \cup \mathcal{A} \cup \{G\}$.

Q.E.D.

We now show the soundness and completeness of an extension of SLD-resolution based on the Compilation Theorem and the CAS-equivalence Theorem.

Theorem 3.3: (Soundness and completeness)
Let X be an extension of SLD-resolution that aims at the subsumption of a set $\mathcal{A}$ of definite clauses. Let τ be a program transformation such that (a) for all programs $\mathcal{P}$ and all goals G, a substitution θ is an X-computed answer substitution for $\mathcal{P} \cup \{G\}$ iff θ is an SLD-computed answer substitution for $\tau(\mathcal{P}) \cup \{G\}$

(compilation), and (b) for all programs $\mathcal{P}$, $\tau(\mathcal{P})$ and $\mathcal{P} \cup \mathcal{A}$ are CAS-equivalent. Then for all programs $\mathcal{P}$ and all goals G

1. every X-computed answer substitution for $\mathcal{P} \cup \{G\}$ is an $\mathcal{A}$-correct answer substitution (soundness);

2. for every $\mathcal{A}$-correct answer substitution θ of $\mathcal{P} \cup \{G\}$, there is an X-computed answer substitution ρ for $\mathcal{P} \cup \{G\}$ and a substitution σ such that $\theta = \rho\sigma$ (completeness).

$\triangle$

Proof:

1. Soundness

Suppose θ is an X-computed answer substitution of $\mathcal{P} \cup \{G\}$. By condition (a), θ is also an SLD-computed answer substitution of $\tau(\mathcal{P}) \cup \{G\}$.[2] By the soundness of SLD-resolution, θ is a correct answer substitution for $\tau(\mathcal{P}) \cup \{G\}$. By condition (b), θ is also a correct answer substitution for $\mathcal{P} \cup \mathcal{A} \cup \{G\}$. By Theorem 3.2 θ is an $\mathcal{A}$-correct answer substitution for $\mathcal{P} \cup \{G\}$.

2. Completeness

Suppose θ is an $\mathcal{A}$-correct answer substitution for $\mathcal{P} \cup \{G\}$. By Theorem 3.2 θ is a correct answer substitution for $\mathcal{P} \cup \mathcal{A} \cup \{G\}$. By condition (b) θ is also a correct answer substitution for $\tau(\mathcal{P}) \cup \{G\}$. By the completeness of SLD-resolution, there is an SLD-computed answer substitution ρ for $\tau(\mathcal{P}) \cup \{G\}$ and a substitution σ such that $\theta = \rho\sigma$. By condition (a), ρ is also an X-computed answer substitution for $\mathcal{P} \cup \{G\}$.

Q.E.D.

Corollary 3.1: Symmetric SLD-resolution is sound and complete with respect to *sym*.

$\triangle$

Proof: The result follows from Corollary 2.2, Theorem 3.1 and Theorem 3.3.

Q.E.D.

3.4. Conclusion

We have presented an approach for reducing the semantics of extensions of SLD-resolution to that of SLD-resolution. Given an extension X of SLD-resolution that builds in a set $\mathcal{A}$ of axioms, we need to show that there is a trans-

[2] In this proof we ignore the fact that θ may only be a variant of the SLD-computed answer substitution. It is straightforward to generalize the proof to cover such cases.

formation τ on definite clause programs such that (i) there is a refutation from a program $\mathcal{P}$ and a goal G according to X iff there is an SLD-refutation from the transformed program $\tau(\mathcal{P})$ and G; (ii) $\tau(\mathcal{P})$ and $\mathcal{P} \cup \mathcal{A}$ are CAS-equivalent. It then follows that X is sound and complete relative to $\mathcal{A}$.

Chapter Four
Loop Detection

The computational behavior of a logic programming system depends on two factors: the search space and the search strategy. The size and the structure of a search space is determined by the program and the selection rule. In Chapter 2 we introduced the notion of CAS-equivalent programs. A logic program with infinite search trees may be replaced by a CAS-equivalent program with finite search trees. Axioms causing infinite searches are deleted and the remaining program is transformed into a program which is CAS-equivalent to the original program. In Chapter 3 we studied the possibility of subsuming axioms causing infinite searches by extending the inference machine. Semantic reduction has been introduced as a general method for providing the semantics of such extensions of SLD-resolution. The methods presented in these two chapters aim at replacing infinite search spaces by finite search spaces.

In this chapter we will introduce a general strategy for searching infinite SLD-trees. We would need such a search strategy if infinite branches were unavoidable. We distinguish a type of SLD-derivations called *repetitive derivations* and prove that pruning repetition derivations from an SLD-tree does not lead to incompleteness.

4.1. Repetitive Derivations

Intuitively, an infinite loop is a computation in which a subgoal is generated repeatedly.

Example 4.1:

sym: $R(x,y) \leftarrow R(y,x)$

G_0: $\leftarrow R(a,b)$

G_1: $\leftarrow R(b,a)$

G_2: $\leftarrow R(a,b)$

. . .

△

In Example 4.1, the subgoal $R(a,b)$ is generated again and again, at every other step. We can prune derivations with recurring subgoals without losing completeness. This is because whatever substitutions obtainable from refutations with a recurring subgoal are subsumed by substitutions from refutations without recurring subgoals.

4.2. Non-Repetitive SLD-Resolution

We record the history of a subgoal by an ancestor list:

Definition 4.1: (Ancestor list)
Let P be a program and let G be a goal. Ancestor lists of subgoals in an SLD-derivation from $P \cup \{G\}$ are defined as follows:

1. Subgoals in $G_0(\equiv G)$ have the empty list [] as their ancestor list..

2: Let
G_k: $\leftarrow A_1, \ldots, A_i, \ldots, A_m$
be a goal with A_i as the selected atom. Let
C_{k+1}: $A \leftarrow B_1, \ldots, B_q$
be an input clause. Let θ_{k+1} be an mgu of A and A_i. Then the goal
G_{k+1}: $\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_q, A_{i+1}, \ldots, A_m)\theta_{k+1}$
is SLD-derived from G_k and C_{k+1} using θ_{k+1}. Let L_j, $1 \le j \le m$, be the ancestor list of the subgoal A_j in G_k. The ancestor list of the new subgoal $B_p\theta_{k+1}$, $1 \le p \le q$, in G_{k+1} is $[A_i \mid L_i]\theta_{k+1}$.[1] The ancestor list of the inherited subgoal $A_j\theta_{k+1}$, $1 \le j \le i-1$ or $i+1 \le j \le m$, in G_{k+1} is $L_j\theta_{k+1}$.

△

Definition 4.2: (Repetitive subgoal)
A subgoal A in an SLD-derivation is said to be *repetitive* iff A is syntactically identical to a member in its ancestor list.

△

[1] $[H|T]$ is a list with H as its first element and T denotes the sublist obtained by deleting H from L.

Definition 4.3: (Repetitive SLD-derivation)
An SLD-derivation is said to be *repetitive* iff it contains repetitive subgoals.

Δ

In Example 4.1, the ancestor list of the subgoal $R(a,b)$ in G_0 is $[\,]$ and the ancestor list of the subgoal $R(b,a)$ in G_1 is $[R(a,b)]$. The ancestor list of the recurring subgoal $R(a,b)$ in G_2 is $[R(b,a),R(a,b)]$. Accordingly the subgoal $R(a,b)$ in G_2 is repetitive and any SLD-derivation of the form

S: $\quad <\leftarrow R(s,t),\ \_,\ \_>,\ <\leftarrow R(t,s),\ \_,\ \_>,\ <\leftarrow R(s,t),\ \_,\ \_>,\ . . .$

is a repetitive derivation.

Suppose there is a repetitive SLD-refutation of the form S. Then there is an SLD-refutation without the repetitive subgoal.

Example 4.2:

P: $\quad R(a,b)$ (1)

$\quad\quad R(x,y) \leftarrow R(y,x)$ (2)

G_0: $\quad \leftarrow R(a,b)$

$$\boxed{\begin{array}{ll} G_1: & \leftarrow R(b,a) \\ G_2: & \leftarrow R(a,b) \end{array}}$$

G_3: $\quad \square$ (by 1)

Δ

We can shorten the refutation in this example by deriving G_3 directly from G_0 and clause 1.

The following search rule restricts the system so that only non-repetitive derivations are constructed.

Rule 4.1: (Non-repetitive SLD-resolution)
Fail all repetitive subgoals.

Δ

SLD-resolution restricted by Rule 4.1 is called *non-repetitive SLD-resolution*. A system that adopts Rule 4.1 does not construct repetitive derivations such as S.

4.3. Completeness of Non-repetitive SLD-Resolution

Non-repetitive SLD-Resolution is complete because every repetitive SLD-refutation can be shortened to a non-repetitive SLD-refutation with a more general substitution.

The completeness proof presented below is based on the MGU Lemma which we state as follows:

Definition 4.4: An *unrestricted SLD-refutation* is an SLD-refutation, except that the unifiers in the refutation need not be most general unifiers.

Δ

Lemma 4.1: (MGU)[2]
Let P be a program and G be a goal. Suppose that $P \cup \{G\}$ has an unrestricted SLD-refutation S. Then $P \cup \{G\}$ has an SLD-refutation S' of the same length. Furthermore, if θ is the substitution of S, and θ' is the substitution of S', then $\theta' \geq \theta$.

Δ

First we prove a lemma which enables us to delete subgoals from a goal without losing completeness.

Lemma 4.2: (Deletion)
Let P be a program. Let

G: $\quad \leftarrow A_1, \ldots, A_k, B_1, \ldots, B_q, A_{k+1}, \ldots, A_m$

and

G': $\quad \leftarrow A_1, \ldots, A_m$

be goals. If there is an SLD-refutation from $P \cup \{G\}$ of length l with substitution θ via the leftmost selection rule, then there is an SLD-refutation from $P \cup \{G'\}$ of length $l' < l$ and with substitution $\theta' \geq \theta$ via the leftmost selection rule.

Δ

Proof: Let S be an SLD-refutation from $P \cup \{G\}$ of length l with substitution $\theta_1 \cdots \theta_l$. S is of the following form:

[2] See [Lloyd 84].

$$<\leftarrow A_1, \ldots, A_k, B_1, \ldots, B_q, A_{k-1}, \ldots, A_m;\ \_;\ \_>,$$
$$<G_1, C_1, \theta_1>,$$
$$\ldots$$
$$<G_{i-1}, C_{i-1}, \theta_{i-1}>,$$
$$<\leftarrow (B_1, \ldots, B_q, A_{k-1}, \ldots, A_m)\theta_1 \cdots \theta_i, C_i, \theta_i>,$$
$$\ldots$$
$$<\leftarrow (A_{k+1}, \ldots, A_m)\theta_1 \cdots \theta_i \cdots \theta_j, C_j, \theta_j>, \qquad (j > i)$$
$$<G_{j+1}, C_{j+1}, \theta_{j+1}>,$$
$$\ldots$$
$$<\Box, C_l, \theta_l>$$

Let G'_n, $1 \leq n \leq i-1$, be the goal obtained from G_n by deleting $B_1, \ldots, B_q$. The sequence

$$S' \qquad <\leftarrow A_1, \ldots, A_m;\ \_;\ \_>,$$
$$<G'_1, C_1, \theta_1>,$$
$$\ldots$$
$$<G'_{i-1}, C_{i-1}, \theta_{i-1}>,$$
$$<\leftarrow (A_{k+1}, \ldots, A_m)\theta_1 \cdots \theta_i \cdots \theta_j, C_i, \theta_i \cdots \theta_j>,$$
$$<G_{j+1}, C_{j+1}, \theta_{j+1}>,$$
$$\ldots$$
$$<\Box, C_l, \theta_l>$$

is an unrestricted SLD-refutation from $P \cup \{G'\}$ of length $l' = i+(l-j) = l-(j-i) < l$. The substitution of S' is $\theta_1 \cdots \theta_l$. By the MGU Lemma, there is an SLD-refutation from $P \cup \{G'\}$ of length l' and with substitution $\theta' \geq \theta_1 \cdots \theta_l$.

Q.E.D.

Lemma 4.3: (Shortening)
Let P be a program and let G be a goal. If there is a repetitive SLD-refutation from $P \cup \{G\}$ of length l and with substitution θ via the leftmost computation rule, then there is an SLD-refutation from $P \cup \{G\}$ of length $l' < l$ and with substitution $\theta' \geq \theta$ via the leftmost computation rule.

Δ

Proof: A repetitive SLD-refutation from $P \cup \{G\}$ via the leftmost computation rule is of the following form:

$$
\begin{array}{lll}
S: & <G, \_, \_>, & \\
 & <G_1, C_1, \theta_1>, & \\
 & \cdots & \\
 & <\leftarrow A_0, A; C_i; \theta_i>, & \\
 & \cdots & \\
 & <\leftarrow (B_0, B, A)\theta_{i+1} \cdots \theta_j, C_j, \theta_j>, & (j > i) \\
 & <\leftarrow (D, B, A)\theta_{i+1} \cdots \theta_{j+1}; H \leftarrow D; \theta_{j+1}>, & \\
 & \cdots & \\
 & <\Box, C_l, \theta_l> &
\end{array}
$$

where A, B and D stand for (possibly empty) sequences of atoms and

$$B_0\theta_{i+1} \cdots \theta_j \equiv A_0\theta_{i+1} \cdots \theta_j.$$

By the Deletion Lemma, there is an SLD-refutation

$$
\begin{array}{ll}
S': & <\leftarrow (D, A)\theta_{i+1} \cdots \theta_{j+1}, \_, \_>, \\
 & <G'_1, C'_1, \theta'_1>, \\
 & \cdots \\
 & <\Box, C'_k, \theta'_k>
\end{array}
$$

of length $k < l-(j+1)$ and with substitution $\theta'_1 \cdots \theta'_k \geq \theta_{j+2} \cdots \theta_l$. If B is an empty sequence, then k is $l-(j+1)$. Since

$$H\theta_{i+1} \cdots \theta_{j+1} \equiv B_0\theta_{i+1} \cdots \theta_{j+1} \equiv A_0\theta_{i+1} \cdots \theta_{j+1},$$

the concatenation

$$
\begin{array}{ll}
S'': & <G, \_, \_>, \\
 & <G_1, C_1, \theta_1>, \\
 & \cdots \\
 & <\leftarrow A_0, A; C_i; \theta_i>, \\
 & <\leftarrow (D, A)\theta_{i+1} \cdots \theta_{j+1}; H \leftarrow D; \theta_{i+1} \cdots \theta_{j+1}>, \\
 & <G'_1, C'_1, \theta'_1>, \\
 & \cdots \\
 & <\Box, C'_k, \theta'_k>
\end{array}
$$

is an unrestricted SLD-refutation. The length of S'' is $(i+1)+k$. The substitution of S'' is

$$
\begin{aligned}
& \theta_1 \cdots \theta_i(\theta_{i+1} \cdots \theta_{j+1})\theta'_1 \cdots \theta'_k \\
\geq\ & \theta_1 \cdots \theta_{j+1}\theta_{j+2} \cdots \theta_l.
\end{aligned}
$$

By the MGU Lemma, there is an SLD-refutation from $P \cup \{G\}$ of length $i+1+k$ and with substitution $\theta'' \geq \theta_1 \cdots \theta_i(\theta_{i+1} \cdots \theta_{j+1})\theta'_1 \cdots \theta'_k \geq \theta_1 \cdots \theta_l$. Since $k \leq l-(j+1)$,

$$i+1+k \leq l-(j+1)+(i+1) = l-(j-i).$$

Hence $i+1+k < l$, because $j > i$.

Q.E.D.

Theorem 4.1: (Completeness, non-repetitive SLD-resolution)
Let $\mathcal{P}$ be a program and let G be a goal. If there is a repetitive SLD-refutation from $\mathcal{P} \cup \{G\}$ with substitution θ, then there is a non-repetitive SLD-refutation from $\mathcal{P} \cup \{G\}$ of length $m < l$ with substitution $\sigma \geq \theta$.

△

Proof: Let S be a repetitive SLD-refutation from $\mathcal{P} \cup \{G\}$ of length l with substitution θ. By Lemma 4.3, there is an SLD-refutation S' from $\mathcal{P} \cup \{G\}$ of length $l' < l$ and with substitution $\theta' \geq \theta$. If S' is non-repetitive, then we are done. Otherwise repeat the argument and obtain another SLD-refutation S''. After a finite number of repetitions, we get a non-repetitive SLD-refutation from $\mathcal{P} \cup \{G\}$ of length $m < l$ and substitution $\sigma \geq \theta$. This is because l is finite and in each repetition we get an SLD-refutation of shorter length.

Q.E.D.

4.4. Non-Repetitive Derivations

We study in this section several natural generalizations of Rule 4.1. We show that the resulting systems ~~obtained~~ are incomplete if we replace the notion of repetitive derivation in Rule 4.1 by some stronger notions.

Strictly Repetitive Subgoal

First, observe that the notion of a repetitive subgoal is distinct from the notion of a *strictly repetitive subgoal* where a strictly repetitive subgoal is a subgoal syntactically identical to one of its ancestors. A repetitive subgoal is syntactically identical to the current instance of one of its ancestors. Let B be a subgoal in goal G_j and let A be an ancestor of B in goal G_i. The *current instance* of A in goal G_j is $A\theta_{i+1} \cdots \theta_j$.

Example 4.3:

$\mathcal{P}$: $P(a) \leftarrow P(a)$

G: $\leftarrow P(x)$

G_1: $\leftarrow P(a)$

△

The ancestor list of the subgoal $P(a)$ in G_1 is $[P(x)]\{a/x\} \equiv [P(a)]$. $P(a)$ is a repetitive subgoal because it is syntactically identical to the current instance of its ancestor $P(x)\{a/x\}$. $P(a) \not\equiv P(x)$, hence $P(a)$ is not a strictly repetitive subgoal. We consider next generalizations of Rule 4.1.

U-Repetitive Subgoal

Let us call subgoals which are *unifiable* with members of their ancestor lists *u-repetitive* subgoals. Consider the following generalization of Rule 4.1:
Rule 4.2: Fail all u-repetitive subgoals.

Δ

The system obtained by restricting SLD-resolution by Rule 4.2 is incomplete.
Example 4.4:

$\mathcal{P}$: $P(a)$
$P(b) \leftarrow P(x)$
G $\leftarrow P(b)$
G_1: $\boxed{\leftarrow P(x)}$
G_2: $\square$

Δ

The ancestor list of the subgoal $P(x)$ in G_1 is $[P(b)]$ and $P(x)$ is unifiable with $P(b)$. However, if we fail the subgoal $P(x)$, we cannot construct any refutation from $\mathcal{P} \cup \{G\}$. The incompleteness is caused by the fact that $P(x)$ is more general than $P(b)$ and we need to instantiate $P(x)$ later to an atom which is not unifiable with $P(b)$.

I-Repetitive Subgoal

Let us call subgoals which are instances of members of their ancestor lists *i-repetitive* subgoals. Consider the following generalization of Rule 4.1:
Rule 4.3: Fail all i-repetitive subgoals.

Δ

Rule 4.3 is stronger than Rule 4.1 but weaker than Rule 4.2. Nevertheless, the system obtained by restricting SLD-resolution by Rule 4.3 is not CAS-complete.
Example 4.5:

$\mathcal{P}$: $P(a)$ (1)
$Q(b)$ (2)
$Q(x) \leftarrow Q(b)$ (3)
$S1$: $\leftarrow Q(x)$
$\boxed{\leftarrow Q(b)}$ $\{\}$ (by 3)
$\square$ $\{\}$ (by 2)

S2:	$\leftarrow Q(x)$		
	$\square$	$\{b/x\}$	(by 2)

$\triangle$

S1 is an SLD-refutation with computed answer substitution $\{\}$ establishing the implication $\mathcal{P} \models \forall Q(x)$. However, the ancestor list of the subgoal $Q(b)$ is $[Q(x)]$ and $Q(b)$ is an instance of $Q(x)$. Pruning *S1* leaves *S2* as the only refutation. The correct answer substitution $\{\}$ is lost.

For systems such as Prolog in which the selection rule is fixed, the loss of CAS-completeness leads to the loss of refutational completeness.

Example 4.6:

$\mathcal{P}$:	$P(a)$		(1)
	$Q(b)$		(2)
	$Q(x) \leftarrow Q(b)$		(3)
S1:	$\leftarrow Q(x), P(x)$		
	$\leftarrow \boxed{Q(b),}\ P(x)$		(by 3)
	$\leftarrow P(x)$		(by 3)
	$\square$	$\{a/x\}$	(by 1)
S2:	$\leftarrow Q(x), P(x)$		
	$\leftarrow P(b)$	$\{b/x\}$	(by 2)
	fails		

$\triangle$

As in Example 4.5, the subgoal $Q(b)$ is an instance of a member of its ancestor list. Pruning *S1* leaves *S2* which is a failure derivation. There is no refutation of $\leftarrow Q(x), P(x)$ if *S1* is pruned.

Notice that if $P(x)$ is the selected atom in G_0, we obtain a refutation:

S3:	$\leftarrow Q(x), \boxed{P(x)}$		
	$\leftarrow Q(a)$	$\{a/x\}$	(by 1)
	$\leftarrow Q(b)$		(by 3)
	$\square$		(by 2)

However, the search space would increase tremendously if we need to consider alternative selection rules.

Quasi-Repetitive Subgoal

Let us call subgoals which are variants of members of their ancestor lists *quasi-repetitive subgoals*. Can we generalize Rule 4.1 to quasi-repetitive subgoals?

Rule 4.4: Fail all quasi-repetitive subgoals.

Although Rule 4.4 is weaker than Rule 4.3, the system obtained by restricting SLD-resolution by Rule 4.4 is also not CAS-complete.

Example 4.7:

P:	$a > b$		(1)
	$b > c$		(2)
	$x > y \leftarrow x > z, z > y$		(3)
S1:	$\leftarrow x > y$		
	$\square$	$\{a/x, b/y\}$	(by 1)
S2:	$\leftarrow x > y$		
	$\square$	$\{b/x, c/y\}$	(by 2)
S3:	$\leftarrow x > y$		
	$\leftarrow x > z, z > y$	$(quasi-repetitive)$	(by 3)
	$\leftarrow b > y$	$\{a/x, b/z\}$	(by 1)
	$\square$	$\{c/y\}$	(by 2)

△

In order to get the correct answer substitution $\{a/x, c/y\}$, we need to generate the quasi-repetitive subgoal $x > z$, which is a variant of its ancestor $x > y$. Therefore pruning *S3* leads to incompleteness.

However, quasi-repetitive subgoals may lead to infinite derivations.

Example 4.8:

P: $x > y \leftarrow x > z, z > y$

S4: $\leftarrow x > y$

$\leftarrow x > z_1, z_1 > y$

$\leftarrow x > z_2, z_2 > z_1, z_1 > y$

. . .

△

Including a transitivity axiom in a program causes infinite loops which pass the test against repetitive derivations. This explains why in Part II the transitivity of equality is subsumed by an extension of SLD-resolution.

4.5. Conclusion

In this chapter we have introduced a general strategy for searching (infinite) SLD-trees. We have distinguished a type of SLD-derivations called *repetitive SLD-derivations* and we have shown that pruning repetitive derivations from SLD-trees does not sacrifice completeness. However, there are non-repetitive infinite branches. We have shown that generalizations of Rule 4.1 which prune non-repetitive infinite branches by failing u-repetitive, l-repetitive and quasi-repetitive subgoals, respectively, lead to incompleteness.

Part II
Extended Unification for Equality

Chapter Five
Extended Unification

5.1. Logic Programming with Equality

Many first order languages have a predicate symbol $=$ for equality, which is interpreted as follows:

> For any pair of terms s and t, $s=t$ iff s and t denote the same object in the problem domain.

We can distinguish two components of the use of the equality predicate: a logical component and a language specific component.

The logical characteristics of equality are captured by the following axioms:

ref: $x=x$

sym: $x=y \leftarrow y=x$

tran: $x=y \leftarrow x=z, z=y$

Pred: $\{P(x_1, \ldots, x_n) \leftarrow P(y_1, \ldots, y_n), y_1=x_1, \ldots, y_n=x_n \mid$
P is an n-place predicate symbol in $L\}$

Fun: $\{f(x_1, \ldots, x_n)=f(y_1, \ldots, y_n) \leftarrow x_1=y_1, \ldots, x_n=y_n \mid$
f is an n-place function symbol in $L\}$

ref, *sym*, *tran*, *Pred*, and *Fun* are the reflexivity, symmetry, transitivity, predicate substitutivity, and function substitutivity axioms, respectively. We use $\mathcal{E}$ to denote the set $\{ref, sym, tran\} \cup Pred \cup Fun$. Axioms in $\mathcal{E}$ define the logical characteristics of $=$ which are shared by the equality symbols in all languages — an equivalence relation with substitutivity (i.e., a congruence relation) [Loveland 78, §5.1].

In addition to the logical axioms for equality, there are language specific equality axioms. In a language L, different terms may denote the same object.

Language specific equality axioms determine the relation of *co-reference* among terms. For example, the equality between rational numbers may be defined by the following axiom:

Example 5.1:

$$rat(N_1,D_1) = rat(N_2,D_2) \leftarrow D_1 \times N_2 = D_2 \times N_1$$

$\triangle$

where $rat(N,D)$ represents the rational number with numerator N and denominator D.

Although $\mathcal{E}$ is a set of definite clauses, including $\mathcal{E}$ in a logic program $\mathcal{P}$ causes infinite loops. We have illustrated in Part I infinite loops caused by *sym* (Example 4.1) and *tran* (Example 4.8). Including $\mathcal{P}red$ in a logic program also introduces infinite loops. If we include *sym*, *tran* and $\mathcal{P}red$ in the same logic program, we lose finite failure as well. This is because *every* subgoal $P(t_1, \ldots, t_n)$ is matched by the head of a clause in $\mathcal{P} \cup \{sym, tran\} \cup \mathcal{P}red$. If P is $=$, $t_1 = t_2$ is unifiable with the head of *sym* and *tran*. If P is not $=$, there is a predicate substitutivity axiom

$$P(x_1, \ldots, x_n) \leftarrow P(y_1, \ldots, y_n), y_1 = x_1, \ldots, y_n = x_n$$

for P and $P(t_1, \ldots, t_n)$ and $P(x_1, \ldots, x_n)$ are unifiable. Hence every failure branch is infinite.

We have illustrated in Examples 4.7 and 4.8 that infinite derivations introduced by *tran* are quasi-repetitive derivations and we lose completeness if we prune all quasi-repetitive derivations. Infinite derivations introduced by Pred are also quasi-repetitive derivations. Consequently, pruning repetitive derivations cannot eliminate infinite loops in logic programming with equality. In this chapter we will present an extension of SLD-resolution that builds in $\{sym, tran\} \cup \mathcal{P}red$ by extended unification.

5.2. Extended Unification

Extended unification has been used in many logic programming systems for building in equality. In a system with extended unification, two terms are unifiable if and only if they are equal according to the equality theory in the program. The challenge is to design a practically feasible logic programming system with extended unification that is both sound and complete with respect to $\mathcal{P} \cup \mathcal{E}$.

Previous foundational work on extended unification, such as [Plotkin 72, van Emden 84, Jaffar 84], concentrates on a form of extended unification in which the equality theory that determines the unification algorithm can be isolated from the rest of the program. We say that this form of extended unification is *separable*. For example, in Prolog II the unification algorithm and the corresponding equality theory are fixed and independent of the programs supplied by the users [van Emden 84]. Another example is the framework described in [Jaffar 84]. A logic program is partitioned into two parts: a definite clause logic program D and a definite clause equality theory E. D contains no equations and all atoms in E are equations. Extended unification is defined with respect to E alone.

However, in general an equality theory cannot be isolated from the rest of the program. Let us call program clauses with equations as their heads *equality clauses*; the set of all equality clauses in a program is the *equality theory* of the program. In general, equality clauses have predications other than equations in their bodies, and non-equality clauses also have equations in their bodies. For example, there is an equation in the body of the following non-equality clause.

Example 5.2:

$$brother(X,Y) \leftarrow father(X)=father(Y),\ male(X),\ male(Y).$$

Δ

Another example is the definition of equality of finite sets listed below. Let $set([E_1, \ldots, E_n])$ represent a set with $E_1, \ldots, E_n$ as elements. The following program defines the relations $\in$, $\subseteq$, and $=$.

Example 5.3:

$$X \in set([H|T]) \leftarrow X = H$$
$$X \in set([H|T]) \leftarrow X \in set(T)$$

$$set([H|T]) \subseteq S \leftarrow H \in S, set(T) \subseteq S$$
$$set([\,]) \subseteq S$$

$$set(L_1) = set(L_2) \leftarrow set(L_1) \subseteq set(L_2), set(L_2) \subseteq set(L_1)$$

Δ

In this example, the equality between two sets is defined in terms of the relation $\in$, while $\in$ is defined in terms of $=$. Equality theories in which $=$ is defined in terms of non-equality predicates (while some non-equality predicates are in turn defined in terms of $=$) are called *recursive equality theories*. These examples illustrate that recursive equality theories are commonly encountered in application domains.

In general, theories on the foundations of separable extended unification do not cover extended unification with recursive equality theories. For example, the general procedure introduced in van Emden and Lloyd [84] is defined for programs in which $=$ does not occur; it is therefore not directly applicable to recursive equality theories. In the case of Jaffar *et al.* [84], the framework is based on the existence of a finest congruence over the set of ground terms determined by the equality theory alone. However, recursive equality theories typically do not have such a finest congruence, because the extension of $=$ depends on the definition of non-equality predicates as well. Hence these frameworks are not adequate for naturally occurring systems which permit the use of recursive equality theories in determining the unification of terms.

In this chapter we present an extension of SLD-resolution called *SLDEU-resolution* that has equality built-in by extended unification. It is a generalization of mechanisms which, like Kornfeld's [83], attempt to build equality into the system by modifying the unification algorithm of Prolog. Our target is a general system that allows a user supplied equality theory (possibly a recursive equality theory) to determine the unification of terms. The system is sound and complete with respect to the user supplied programs and $\mathcal{E}$.

The rest of Part II is organized as follows. In the next section we generalize Kornfeld's system to obtain SLDEU-resolution. In Section 4, we restrict SLDEU-resolution to *right recursive SLDEU-resolution* in order to eliminate a source of infinite loops in the mechanism of extended unification. The soundness and completeness of SLDEU-resolution and right recursive SLDEU-resolution are established in Chapter 6 by semantic reduction. In Chapter 7 we apply the method developed in Chapter 4 to right recursive SLDEU-resolution, restricting it further to *non-repetitive right recursive SLDEU-resolution*. In Chapter 8 we show that infinite derivations are still possible in non-repetitive right recursive SLDEU-resolution and explain why it is impossible to eliminate all infinite computations. We conclude the dissertation by a brief discussion on some interesting directions for further research.

5.3. SLDEU-Resolution

K-Resolution

In Kornfeld [83] the unification algorithm of Prolog is modified as follows:

1. Given two terms s and t to be unified, the system first attempts to compute an mgu of s and t and uses the mgu as a variable binding.

2. If (and only if) step 1 fails, the system attempts to establish

   $\leftarrow s=t$[1]

   from the program and uses the substitution as a variable binding.

[1] The predicate symbol *equals* is used in [Kornfeld 83] to denote equality.

3. If (and only if) steps 1 and 2 fail, the system attempts to establish

   $\leftarrow t=s$

   from the program and use the substitution as a variable binding.

4. The unification of s and t fails iff steps 1, 2 and 3 fail.

We call the unification specified by this algorithm *K-unification*. Extensions of SLD-resolution based on K-unification are called *K-resolution*.

Let us illustrate how K-unification works.

Example 5.4:

P: $f(a)=g(b)$
$P(f(a))$

G: $\leftarrow P(g(x)), A_2, \ldots, A_m$

Δ

Standard unification of $f(a)$ and $g(x)$ fails, because $f(a)$ and $g(x)$ have no unifier. So the system establishes the goal

$\leftarrow f(a)=g(x)$

from P and uses the binding $\{b/x\}$ to generate the new goal

$\leftarrow (A_2, \ldots, A_m)\{b/x\}$

This example illustrates how predicate substitutivity is subsumed.

Example 5.5:

P: $g(b)=f(a)$
$P(f(a))$

G: $\leftarrow P(g(x))$

Δ

As in the previous example standard unification of $f(a)$ and $g(x)$ fails, so the system attempts to establish

$\leftarrow f(a)=g(x)$

from P. However, since $\exists(f(a)=g(x))$ does not follow from P, the system generates another goal

G': $\leftarrow g(x)=f(a)$

in which the terms are switched around. G' is then established from P with the binding $\{b/x\}$. Symmetry is subsumed by generating a new equality goal in which the terms to be unified are switched around.

Transitivity is subsumed by generating new equality goals recursively

Example 5.6:

P: $a=b$
$b=c$
$c=d$

G. $\leftarrow a=d$

K-unification of $a=b$ and $a=d$ generates the new goal

$\leftarrow b=d$.

Then K-unification of $b=c$ with $b=d$ generates another new goal

$\leftarrow c=d$.

Finally standard unification of c=d with the program clause $c=d$ succeeds, establishing the goal $\leftarrow c=d$. Hence K-unification of $b=c$ with $b=d$ succeeds, establishing the goal $\leftarrow b=d$, and finally K-unification of $a=b$ and $a=d$ succeeds, establishing the original goal $\leftarrow a=d$.

Generalization to SLDEU-Resolution

Reflexivity and symmetry of equality are not treated uniformly by K-resolution. Kornfeld [83] has not considered the case where an equation occurs in a goal or the body of a program clause. Equality goals are generated only by K-unification.[2] In SLDEU-resolution, equality subgoals may occur in goals or in

[2] In Kornfeld [83] equations (expressed as *equals*(s,t) in his system) occur only in the heads of program clauses. It is equations expressing syntactic equality $s\equiv t$ (expressed as $s=t$ in his system) that occur in goals or bodies of program clauses. Although it is not explained in [Kornfeld 83], it could be the case that $s\equiv t$ and $s=t$ are both interpreted as equality and the reflexivity and symmetry of equality are subsumed as follows. $\equiv$ is defined as usual by $x\equiv x$. When a subgoal of the form $s\equiv t$ is resolved with $x\equiv x$, x is instantiated to s. Then the system attempts to K-unify $x\{s/x\}$ (which is s) with t. If s and t are unifiable, then the binding generated is an mgu of s and t. Otherwise, the system will attempt to establish the goal $\leftarrow s=t$. If this fails, then the system attempts to establish the goal $\leftarrow t=s$. We prefer to make this mechanism explicit in our system.

bodies of program clauses. A program $\mathcal{P}$ is supplemented by the reflexivity axiom $x=x$, and when an equality subgoal $s=t$ is resolved with $x=x$ an mgu of $x=x$ and $s=t$ is computed. Extended unification does not apply to resolution with $x=x$. Symmetry of equality is subsumed by the rule that extended unification of an equality subgoal $s=t$ with the head $u=v$ of an equality clause succeeds if either (i) $\leftarrow u=s$ and $\leftarrow v=t$ are established, or (ii) $\leftarrow v=s$ and $\leftarrow u=t$ are established.[3] Symmetry in the extended unification of terms is then subsumed by symmetry in the extended unification of equations.

K-unification does not subsume function substitutivity of equality. In SLDEU-resolution a program is supplemented by the function substitutivity axioms Fun. Also, there is no backtracking in K-unification and this causes incompleteness.

Example 5.7:

$$
\begin{array}{lll}
\mathcal{P}: & f(a)=g(c) & (1) \\
 & f(b)=g(d) & (2) \\
 & Q(b) & \\
 & P(f(x)) \leftarrow Q(x) &
\end{array}
$$

$$
G. \qquad \leftarrow P(g(y))
$$

△

Using clause (1), K-unification of $P(f(x))$ with $P(g(y))$ results in the binding $\{a/x,\ c/y\}$ and the new goal

$$\leftarrow Q(a)$$

which cannot be established from the program. If the system can backtrack and use clause (2), then G will be established with the binding $\{b/x,\ d/y\}$. However, since there is no backtracking in K-unification, the system fails to establish the goal G. Backtracking in unification is allowed in SLDEU-resolution.

[3] This description is not quite accurate. See Definition 5.2

Formal Definitions

SLDEU-resolution is SLD-resolution with the following modifications:

- A program P is supplemented automatically by the equality theory $Fun \cup \{ref\}$.

- Standard syntactic unification is replaced by extended unification.

Definitions 5.1 to 5.6 define SLDEU-resolution by simultaneous recursion.

We first define the notion of *extended unification* for terms.

Definition 5.1: (Extended unification of terms)
Let P be a program. Let s and t be terms. A substitution θ is an *E-unifier* of the pair $<s,t>$ iff θ is the computed answer substitution for an E-refutation of $P \cup \{\leftarrow s=t\}$.

Δ

Example 5.8:

P: $f(a)=b$
$f(c)=b$

$\{a/x\}$ and $\{b/x\}$ are E-unifiers of the pair $<f(x), b>$, since the goal $\leftarrow f(x)=b$ can be established with the computed answer substitutions $\{a/x\}$ and $\{b/x\}$.

Δ

In general, there may be many computed answer substitutions for a goal $\leftarrow s=t$, hence there may be many E-unifiers for a pair of terms.

Next, we extend the notion of *extended unification* to equations.

Definition 5.2: (Extended unification of equations)
Let $s=t$ be a subgoal and let $u=v \leftarrow B_1, \ldots, B_q$ be an equality clause. A substitution θ is an *E-unifier* of $u=v$ and $s=t$ iff θ is the composition $\theta_1\theta_2$ of θ_1 and θ_2 satisfying one of the following conditions:

1. θ_1 is an E-unifier of the pair $<u,s>$ and
   θ_2 is an E-unifier of the pair $<v\theta_1, t\theta_1>$;

or

2. θ_1 is an E-unifier of the pair $<v,s>$ and
   θ_2 is an E-unifier of the pair $<u\theta_1, t\theta_1>$.

Δ

Symmetry is built into the E-unification of equations.

Example 5.9:

P: $a=b$
$a=f(c)$
$b=g(c)$

G. $\leftarrow g(x)=f(x)$

$\{c/x\}$ is an E-unifier of $a=b$ and $g(x)=f(x)$ because $\{c/x\}$ is an E-unifier of $<b, g(x)>$ and $\{\}$ is an E-unifier of $<a, f(x)>\{c/x\} \equiv <a, f(c)>$. The composition $\{c/x\}\{\}$ is $\{c/x\}$.

△

Finally, we extend the notion of extended unification to atoms other than equations.

Definition 5.3: (Extended unification of atoms)
Let $P(t_1, \ldots, t_n)$ be a subgoal and let $P(s_1, \ldots, s_n) \leftarrow B_1, \ldots, B_q$ be an input clause. A substitution θ is an *E-unifier* of $P(s_1, \ldots, s_n)$ and $P(t_1, \ldots, t_n)$ iff θ is the composition $\theta_1\theta_2 \cdots \theta_n$ of the E-unifiers $\theta_1, \theta_2, \ldots, \theta_n$ where θ_1 is an E-unifier of the pair $<s_1, t_1>$, and for each i, $2 \leq i \leq n$, θ_i is an E-unifier of the pair

$$<s_i\theta_1\theta_2 \cdots \theta_{i-1}, t_i\theta_1\theta_2 \cdots \theta_{i-1}>.$$

△

Example 5.10:

P: $P(d, g(y), f(y))$
$a=d$
$f(a)=g(b)$
$f(b)=c$

G: $\leftarrow P(x, f(x), c)$

$\theta_1 = \{a/x\}$ is an E-unifier of the pair of terms $<d, x>$.
$\theta_2 = \{b/y\}$ is an E-unifier of $<g(y)\theta_1, f(x)\theta_1> \equiv <g(y), f(a)>$.
$\theta_3 = \{\}$ is an E-unifier of $<f(y)\theta_1\theta_2, c\theta_1\theta_2> \equiv <f(b), c>$, and
$\theta = \theta_1\theta_2\theta_3 = \{a/x, b/y\}$ is an E-unifier of $P(d, g(y), f(y))$ and $P(x, f(x), c)$.

△

The notion of *selection rule* is identical to that used in SLD-resolution.

The notion of *derivation step* is extended by supplementing the standard syntactic unification by extended unification.

Definition 5.4: (E-derivation step)
Let G_i be $\leftarrow A_1, \ldots, A_m, \ldots, A_k$; C_{i+1} be $A \leftarrow B_1, \ldots, B_q$; and let R be a selection rule. Then G_{i+1} is *E-derived* from G_i and C_{i+1} using the unifier θ_{i+1} via R iff the following conditions hold.

1. $R(G_i)$ is A_m.

2. a. If C_{i+1} is $x=x$, then θ_{i+1} is an *mgu* of $x=x$ and A_m;
   b. otherwise θ_{i+1} is an E-unifier of A and A_m.

3. G_{i+1} is the goal $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta_{i+1}$.

$\triangle$

Note that condition 2a in the above definition stops the recursive generation of equality goals (induced by extended unification) by enforcing the standard syntactic unification of an equality goal with the axiom $x=x$.

Example 5.11:

G_0: $\quad \leftarrow a=y, P(y)$
C_1: $\quad x=x$

$G_1 \equiv \leftarrow P(a)$ is derived from G_0 and C_1 using the mgu $\{a/x, a/y\}$ via the leftmost selection rule.

$\triangle$

Example 5.12:

G_0: $\quad \leftarrow P(x, f(x), c)$
C_1: $\quad P(d, g(y), f(y))$

If P is the program in Example 5.10, then $G_1 \equiv \square$ is derived from G_0 and C_1 using the substitution $\{a/x, b/y\}$.

$\triangle$

Next we extend the notions of derivation and refutation to *E-derivation* and *E-refutation*.

Definition 5.5: (E-derivation)
Let P be a program, G a goal and R a selection rule. An *E-derivation* from $P \cup \{G\}$ via R is a sequence of quadruples

$T_0, T_1, T_2, \ldots$

satisfying the following conditions:

1. Each T_i, $i \geq 0$, is a quadruple

$<G_i, C_i, U_i, \Delta_i>$

where G_i is a goal, C_i is an input clause from $\mathcal{P} \cup Fun \cup \{ref\}$, U_i is an n-tuple of substitutions $<\theta_1, \theta_2, \ldots, \theta_n>$, and Δ_i is an m-tuple of E-refutations $<\delta_1, \ldots, \delta_m>$.

2. For each $i \geq 1$, T_{i-1} and T_i satisfy one of the following conditions:

   a. $R(G_{i-1})$ is $s=t$.
   C_i is $x=x$.
   U_i is $<\theta>$, where θ is an mgu of ref and $s=t$.
   Δ_i is the null-tuple $< >$.

   b. $R(G_{i-1})$ is $s=t$.
   C_i is an equality axiom
   $$u=v \leftarrow B_1, \ldots, B_q$$
   other than $x=x$.
   U_i is $<\theta_1, \theta_2>$, and
   Δ_i is $<\delta_1, \delta_2>$.
   They are related as follows:

      i. δ_1 is an E-refutation from $\mathcal{P} \cup \{\leftarrow u=s\}$ with the substitution θ_1 and
      δ_2 is an E-refutation from $\mathcal{P} \cup \{\leftarrow (v=t)\theta_1\}$ with the substitution θ_2;

   or

      ii. δ_1 is an E-refutation from $\mathcal{P} \cup \{\leftarrow v=s\}$ with the substitution θ_1 and
      δ_2 is an E-refutation from $\mathcal{P} \cup \{\leftarrow (u=t)\theta_1\}$ with the substitution θ_2.

   c. $R(G_{i-1})$ is an atom $P(t_1, \ldots, t_n)$.
   C_i is $P(s_1, \ldots, s_n) \leftarrow B_1, \ldots, B_q$.
   U_i is $<\theta_1, \theta_2, \ldots, \theta_n>$, and
   Δ_i is $<\delta_1, \ldots, \delta_n>$, where δ_1 is an E-refutation from $\mathcal{P} \cup \{\leftarrow s_1=t_1\}$ with the substitution θ_1 and for all j, $2 \leq j \leq n$, δ_j is an E-refutation from $\mathcal{P} \cup \{\leftarrow (s_j=t_j)\theta_1\theta_2 \cdots \theta_{j-1}\}$ with the substitution θ_j.

3. G_i is E-derived from G_{i-1} and C_i using the composition $\theta_1\theta_2 \cdots \theta_n$ of the substitutions in U_i.

Δ

Definition 5.6: An *E-refutation* is a finite E-derivation

$T_0, T_1, T_2, \ldots, T_k$

where the goal G_k in T_k is $\Box$.

We say that the E-refutation has *length k*.

Let σ_i be the composition $\theta_1\theta_2 \cdots \theta_n$ of the substitutions in U_i. The composition $\sigma_1\sigma_2 \cdots \sigma_k$ is called the *substitution of the E-refutation*. The substitution obtained by restricting the substitution of the E-refutation to the variables in G_0 is called the *computed answer substitution* of the E-refutation.

Δ

An E-refutation from $\mathcal{P} \cup \{G\}$ is a *formal object* that demonstrates the unsatisfiability of $\mathcal{P} \cup \{G\}$. The property of being an E-refutation is decidable.

Example 5.13:

$\mathcal{P}$: $a=b.$
$P(a).$
$Q(c) \leftarrow P(b).$

$\mathcal{A}1$ to $\mathcal{A}6$ are E-refutations from $\mathcal{P}$:

1. $\mathcal{A}1$: $<\Box, \_, \_, \_>$
2. $\mathcal{A}2$: $<\leftarrow a=a, \_, \_, \_>, <\Box, x=x, <\{a/x\}>, <>>$
3. $\mathcal{A}3$: $<\leftarrow b=b, \_, \_, \_>, <\Box, y=y, <\{b/y\}>, <>>$
4. $\mathcal{A}4$: $<\leftarrow a=b, \_, \_, \_>, <\Box, a=b, <\{a/x\}, \{b/y\}>, <\mathcal{A}2, \mathcal{A}3>>$
5. $\mathcal{A}5$: $<\leftarrow P(b), \_, \_, \_>, <\Box, P(a), <\{a/x, b/y\}>, <\mathcal{A}4>>$
6. $\mathcal{A}6$: $<\leftarrow Q(c), \_, \_, \_>,$
$<\leftarrow P(b), Q(c) \leftarrow P(b), <\{c/z\}>, <\mathcal{A}2'>>,$
$<\Box, P(a), <\{a/x\}, \{b/y\}>, <\mathcal{A}4>>$
where $\mathcal{A}2'$ is an E-refutation from $\mathcal{P} \cup \{\leftarrow c=c\}$ similar to $\mathcal{A}2$.

Δ

Definition 5.7: (*$\mathcal{E}$*-Correct answer substitution)
Let $\mathcal{P}$ be a program and let

G: $\leftarrow A_1, \ldots, A_m$

be a goal. A substitution θ for the variables in G is called an *$\mathcal{E}$-correct answer substitution* for $\mathcal{P} \cup \{G\}$ iff $\mathcal{P} \cup \mathcal{E} \models \forall((A_1 \wedge \cdots \wedge A_m)\theta)$.

Δ

The soundness and completeness of SLDEU-resolution are proved in the next chapter.

Search Space

We call the search space of E-resolution an *E-tree*. E-trees are more complicated than SLD-trees. For each pair consisting of a goal

G: $\leftarrow A_1, \ldots, A_m$

and an input clause

C: $A \leftarrow B_1, \ldots, B_q$.

there may be many descendants, one for each E-unifier of A and the selected atom of G.

Definition 5.8: (E-tree)
Let P be a program, G a goal and R a selection rule. Then the *E-tree* for $P \cup \{G\}$ via R is defined as follows:

1. Each node of the tree is a goal (which may be □).

2. The root node is G.

3. The descendants of a node N are the goal clauses E-derivable from N in one step. Each arc from N to a descendant is labeled by a pair $<C,\theta>$, where C is the input clause and θ the unifier used in the corresponding derivation step.

4. Nodes which are the empty clause have no descendants.

Δ

If there are infinitely many E-unifiers for A and the selected atom of G, then the descendants of G and $A \leftarrow B_1, \ldots, B_q$ form an infinite fan. There is a fan for each input clause whose head is E-unifiable with the selected atom of G. See Figures 5-1 and 5-2.

As in the case of SLD-trees, each branch of an E-tree corresponds to an E-

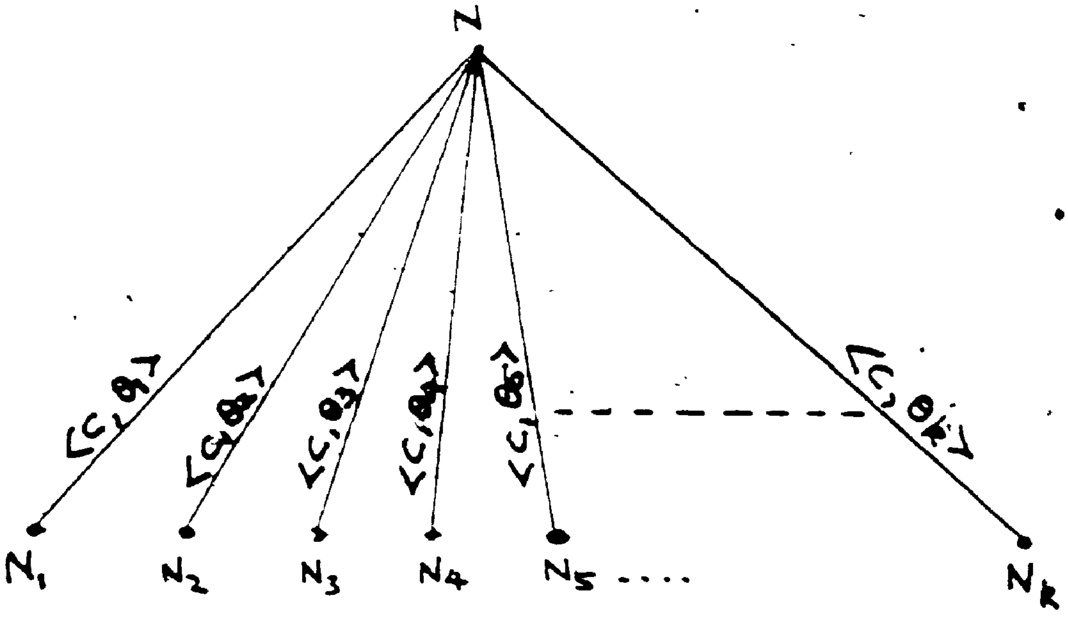

N
$\langle C,\theta_1\rangle$
$\langle C,\theta_2\rangle$
$\langle C,\theta_3\rangle$
$\langle C,\theta_4\rangle$
$\langle C,\theta_5\rangle$
$\langle C,\theta_k\rangle$
N_1
N_2
N_3
N_4
N_5 ....
N_k

Figure 5-1: A Fan of a node N and an input clause C

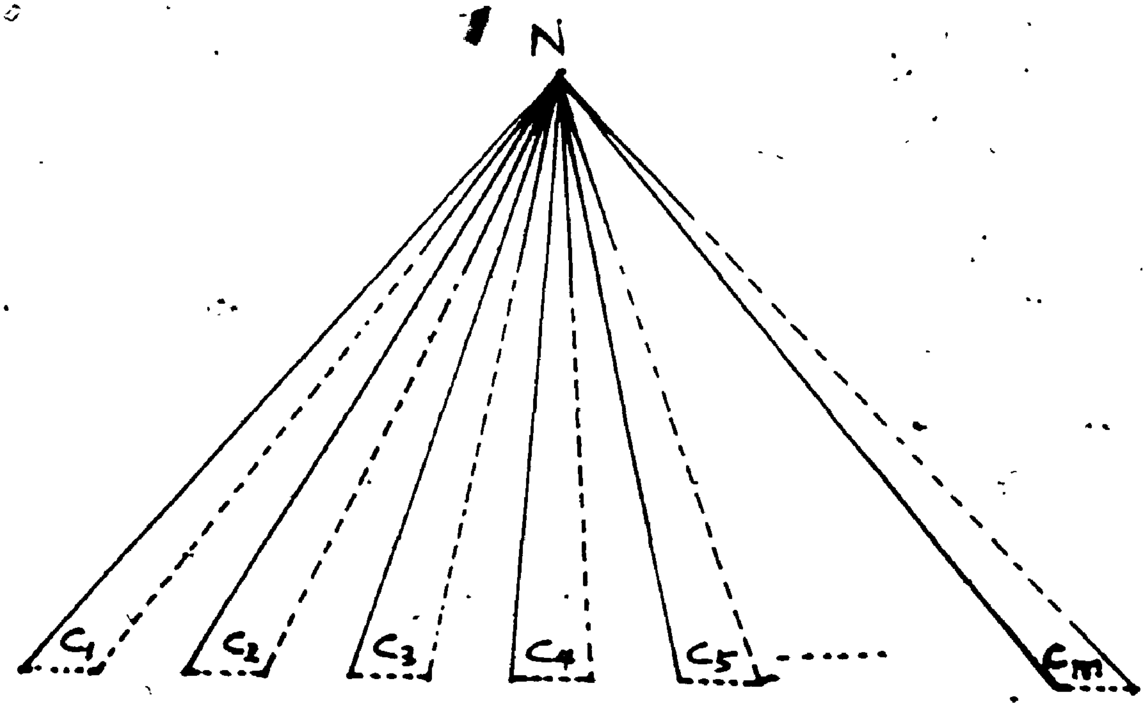


Figure 5-2: Fans of a Node

derivation from $\mathcal{P} \cup \{G\}$. Branches corresponding to successful, infinite and failed derivations are called *success*, *infinite* and *failure branches*, respectively.

An *E-refutation procedure* is a procedure for searching E-trees for E-refutations. An E-refutation procedure has two components: a selection rule, which determines the search trees and a search strategy which determines the way in which a tree is searched.

5.4. Right Recursive SLDEU-Resolution

In this section we consider a source of infinite computations that is built into SLDEU-resolution; we introduce a restriction on SLDEU-resolution that eliminates this kind of infinite computations.

In SLDEU-resolution, transitivity is subsumed by the recursive generation of new equality goals in the E-unification of equality subgoals.

Example 5.14:

$\mathcal{P}$: $a=b$
$b=c$
$c=d$

G: $\leftarrow a=d$

Δ

$a=b$ is E-unifiable with $a=d$ because $\leftarrow a=a$ can be established using $x=x$ and $\leftarrow b=d$ can be established as follows: after the goal

$$\leftarrow b=d$$

is generated, E-unification of $b=c$ and $b=d$ generates the goal

$$\leftarrow c=d$$

which can be established using the clause $c=d$.

However, the recursive generation of equality goals in E-unification is also a source of infinite computations.

Example 5.15:

$\mathcal{P}$: $a=b$

G: $\leftarrow c=d$

$c=d$ is not a theorem of P and all E-derivations from $P \cup \{G\}$ are infinite. Consider the terms a and c. A sequence of equality goals generated by the E-unification of a and c is listed as follows:

$\leftarrow a=c;$
$\leftarrow a=a$ (*succeeds*), $b=c;$
$\leftarrow a=b$ (*succeeds*), $b=c;$
$\leftarrow a=b$ (*succeeds*), $b=c;$
. . .

This is an infinite loop. The problem is completely general: equality subgoals never fail.

Example 5.14 suggests a way of avoiding such infinite loops. Note that in the example the first terms of the equality subgoals are always unifiable with a term in the head of an equality clause. Recursive generation of equality goals is needed only for the second terms of equality subgoals. On the other hand, if we rule out E-unification of the first terms in equality subgoals, the goal $\leftarrow c=d$ in Example 5.15 fails because c is unifiable with neither a nor b.

We propose the following restriction on the E-unification of equality subgoals.

Rule 5.1: (Right recursion)
Let $s=t$ be a subgoal and $u=v \leftarrow B_1, \ldots, B_q$ be an equality axiom. A substitution θ is an *R-unifier* of $u=v$ and $s=t$ iff θ is the composition of θ_1 and θ_2 satisfying one of the following conditions:

1. θ_1 is an mgu of the pair $<u,s>$ and
   θ_2 is an E-unifier of the pair $<v\theta_1, t\theta_1>$;

or

2. θ_1 is an mgu of the pair $<v,s>$ and

θ_2 is an E-unifier of the pair $<u\theta_1, t\theta_1>$.[4]

Δ

The version of extended unification obtained by modifying E-unification by Rule 5.1 is called *R-unification* and the system obtained by imposing Rule 5.1 on SLDEU-resolution is called *right recursive SLDEU-resolution*. A proof of the soundness and completeness of right recursive SLDEU-resolution is given in the next chapter. Here we will explain informally how the symmetry and transitivity of equality are subsumed in right recursive SLDEU-resolution.

The subsumption of the symmetry and transitivity axioms is based on the idea of an *E-chain*. Since equality is reflexive, symmetric and transitive, it is an equivalence relation. We have the following result, which holds for all equivalence relations:

Notation:
We use r, s, t, u, v and w, with or without subscripts, for terms.

Δ

Proposition 5.1: A ground equation $s=t$ is a theorem of $P \cup \{ref, sym, tran\}$ iff there is a sequence of ground terms

S: $r_0, r_1, \ldots, r_l$

satisfying the following conditions:

1. s is r_0, t is r_l.

2. for each i, $1 \le i \le l$, there is a clause

   C_i: $u=v \leftarrow A_1, \ldots, A_m$

   in P and a substitution θ such that $A_1\theta, \ldots, A_m\theta$ are ground atoms logically implied by $P \cup \{ref, sym, tran\}$ and either

   a. $r_{i-1} \equiv u\theta$ and $r_i \equiv v\theta$, or

[4] Rule 5.1 is not equivalent to Kornfeld's restriction that equality axioms can only be used one way [Kornfeld 83, §6]. Kornfeld's restriction corresponds to the requirement that the unifier for u must be an mgu. This causes incompleteness. Consider the following program:

P: $b=a.$
$b=c.$

Both $\leftarrow a=c$ and $\leftarrow c=a$ will fail because there is no mgu for $<a,b>$ nor for $<c,b>$.

b. $r_{i-1} \equiv v\theta$ and $r_i \equiv u\theta$.[5]

△

We call the sequence S an *E-chain* for s and t; each pair $<r_{i-1}, r_i>$ is called an *E-link*.

Consider the simple case where all program clauses are ground unit clauses. Suppose a ground equation $s=t$ is a theorem of $\mathcal{P} \cup \mathcal{E}$. Then there is an E-chain

$$r_0(\equiv s),\ r_1,\ \ldots,\ r_l(\equiv t)$$

linking s and t. $<s, r_1>$ is an E-link, so there is a clause

C: $u=v$

in $\mathcal{P}$[6] and either (i) $s \equiv u$ and $r_1 \equiv v$ or (ii) $s \equiv v$ and $r_1 \equiv u$. In either case s is *unifiable* with a term in the head of an equality clause in $\mathcal{P}$ and R-unification of s succeeds. The system then generates the new goal

$$\leftarrow r_1 = t.$$

The proof of $s=t$ is reduced to the proof of an equation $(r_1=t)$ with a shorter E-chain. The sequence of goals generated by R-unification of $u=v$ and $s=t$ is listed as follows:

$$\leftarrow r_1=t,\ \leftarrow r_2=t,\ \ldots,\ \leftarrow r_{l-1}=t,\ \leftarrow t=t.$$

The goal $\leftarrow t=t$ succeeds by $x=x$.

[5] This result may be easily proved by induction on $T_{\mathcal{P} \cup \{ref, sym, tran\}} \uparrow i$.

[6] The presence of Fun is ignored for the moment.

Chapter Six
Semantics of SLDEU-Resolution

In this chapter the semantics of SLDEU-resolution is reduced to the semantics of SLD-resolution. In Section 1 we define two program transformations * and r, * compiles a program $\mathcal{P}$ for SLDEU-resolution to an equivalent program $\mathcal{P}^*$ for SLD-resolution, while r compiles a program $\mathcal{P}$ for right recursive SLDEU-resolution to an equivalent program $\mathcal{P}^r$ for SLD-resolution. In Section 2 we prove some useful results concerning concatenations of SLD-refutations and compositions of unifiers. The Compilation Theorems for * and r are proved in Sections 3 and 4. The CAS-equivalence Theorems for * and r are proved in Section 5.

The Compilation Theorems are proved for SLDEU-resolution and right recursive SLDEU-resolution with the leftmost selection rule. What follows are the definitions of an E-derivation and an E-refutation for SLDEU-resolution using the leftmost selection rule.

Definition 6.1: (E-derivation, SLDEU-resolution)
Let $\mathcal{P}$ be a program and G a goal. An *E-derivation* from $\mathcal{P} \cup \{G\}$ via the leftmost selection rule is a sequence of quadruples

$T_0, T_1, T_2, \ldots$

satisfying the following conditions:

1. Each T_i, $i \geq 0$, is a quadruple

   $<G_i, C_i, U_i, \Delta_i>$

   where G_i is a goal, C_i is an input clause from $\mathcal{P} \cup Fun \cup \{ref\}$, U_i is an n-tuple of substitutions $<\theta_1, \theta_2, \ldots, \theta_n>$, and Δ_i is an m-tuple of E-refutations $<\delta_1, \ldots, \delta_m>$.

2. For each $i \geq 1$, T_{i-1} and T_i satisfy one of the following conditions:

   a. (Reflexivity)
   G_{i-1} is $\leftarrow s{=}t, A_2, \ldots, A_m$.
   C_i is *ref*.
   U_i is $<\theta>$, where θ is an mgu of $s{=}t$ and *ref*.

Δ_i is the null-tuple $< >$.
G_i is $\leftarrow (A_2, \ldots, A_m)\theta$.

b. (Symmetry and predicate substitutivity for $=$)
G_{i-1} is $\leftarrow s=t, A_2, \ldots, A_m$.
C_i is $u=v \leftarrow B_1, \ldots, B_q$, an equality axiom other than *ref*.
U_i is $<\theta_1, \theta_2>$, and
Δ_i is $<\delta_1, \delta_2>$.
G_i is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta_1\theta_2$.
They are related as follows:

i. δ_1 is an E-refutation from $\mathcal{P} \cup \{\leftarrow u=s\}$ with the substitution θ_1 and
δ_2 is an E-refutation from $\mathcal{P} \cup \{\leftarrow (v=t)\theta_1\}$ with the substitution θ_2;

or

ii. δ_1 is an E-refutation from $\mathcal{P} \cup \{\leftarrow v=s\}$ with the substitution θ_1 and
δ_2 is an E-refutation from $\mathcal{P} \cup \{\leftarrow (u=t)\theta_1\}$ with the substitution θ_2.

c. (Predicate substitutivity)
G_{i-1} is $\leftarrow P(t_1, \ldots, t_n), A_2, \ldots, A_m$.
C_i is $P(s_1, \ldots, s_n) \leftarrow B_1, \ldots, B_q$,
U_i is $<\theta_1, \theta_2, \ldots, \theta_n>$, and
Δ_i is $<\delta_1, \ldots, \delta_n>$, where δ_1 is an E-refutation from $\mathcal{P} \cup \{\leftarrow s_1=t_1\}$ with the substitution θ_1 and for all j, $2 \leq j \leq n$, δ_j is an E-refutation from $\mathcal{P} \cup \{\leftarrow (s_j=t_j)\theta_1\theta_2 \cdots \theta_{j-1}\}$ with the substitution θ_j.
G_i is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta_1\theta_2 \cdots \theta_n$.

Δ

Definition 6.2: An *E-refutation* is a finite E-derivation

Δ: $T_0, T_1, T_2, \ldots, T_k$

where the goal G_k in T_k is □. We say that k is the length of Δ. Let σ_i, $1 \leq i \leq k$, be the composition $\theta_1\theta_2 \cdots \theta_n$ of the substitutions in $U_i (\equiv <\theta_1, \theta_2, \ldots, \theta_n>)$. The composition $\sigma_1\sigma_2 \cdots \sigma_k$ is called the *substitution of the E-refutation*.

Δ

The definition of *R-derivation* for right recursive SLDEU-resolution is identical to that of E-derivation except for modifications in condition 2.b. These changes are highlighted by vertical bars at the left margin.

Definition 6.3: (R-derivation, right recursive SLDEU-resolution)
Let $\mathcal{P}$ be a program and G a goal. An *R-derivation* from $\mathcal{P} \cup \{G\}$ via the leftmost selection rule is a sequence of quadruples

$T_0, T_1, T_2, \ldots$

satisfying the following conditions:

1. Each T_i, $i \geq 0$, is a quadruple

   $<G_i, C_i, U_i, \Delta_i>$

   where G_i is a goal, C_i is (a variant of) an input clause from $\mathcal{P} \cup Fun \cup \{ref\}$, U_i is an n-tuple of substitutions $<\theta_1, \theta_2, \ldots, \theta_n>$, and Δ_i is an m-tuple of R-refutations $<\delta_1, \ldots, \delta_m>$.

2. For each $i \geq 1$, T_{i-1} and T_i satisfy one of the following conditions:

   a. (Reflexivity)
   G_{i-1} is $\leftarrow s{=}t, A_2, \ldots, A_m$.
   C_i is ref.
   U_i is $<\theta>$, where θ is an mgu of $s{=}t$ and ref.
   Δ_i is the null-tuple $<>$.
   G_i is $\leftarrow (A_2, \ldots, A_m)\theta$.

   b. (Symmetry and predicate substitutivity for $=$)
   G_{i-1} is $\leftarrow s{=}t, A_2, \ldots, A_m$.
   C_i is $u{=}v \leftarrow B_1, \ldots, B_q$, an equality axiom other than ref.
   U_i is $<\theta_1, \theta_2>$, and
   Δ_i is $<\delta_2>$.
   G_i is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta_1\theta_2$.
   They are related as follows:

      i. θ_1 is an mgu of u and s and
      δ_2 is an R-refutation from $\mathcal{P} \cup \{\leftarrow (v{=}t)\theta_1\}$ with the substitution θ_2;

   or

      ii. θ_1 is an mgu of v and s and
      δ_2 is an R-refutation from $\mathcal{P} \cup \{\leftarrow (u{=}t)\theta_1\}$ with the substitution θ_2.

c. (Predicate substitutivity)
G_{i-1} is $\leftarrow P(t_1, \ldots, t_n), A_2, \ldots, A_m$.
C_i is $P(s_1, \ldots, s_n) \leftarrow B_1, \ldots, B_q$.
U_i is $<\theta_1, \theta_2, \ldots, \theta_n>$, and
Δ_i is $<\delta_1, \ldots, \delta_n>$, where δ_1 is an R-refutation from $\mathcal{P} \cup \{\leftarrow s_1 = t_1\}$ with the substitution θ_1 and for all j, $2 \leq j \leq n$, δ_j is an R-refutation from $\mathcal{P} \cup \{\leftarrow (s_j = t_j)\theta_1\theta_2 \cdots \theta_{j-1}\}$ with the substitution θ_j.
G_i is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_m)\theta_1\theta_2 \cdots \theta_n$

Δ

The definition of an *R-refutation* is essentially the same as that for an E-refutation.

Definition 6.4: An *R-refutation* (for right recursive SLDEU-resolution) is a finite R-derivation

Λ: $T_0, T_1, T_2, \ldots, T_k$

where the goal G_k in T_k is □. We say that k is the length of Λ. Let σ_i, $1 \leq i \leq k$, be the composition $\theta_1\theta_2 \cdots \theta_n$ of the substitutions in $U_i (\equiv <\theta_1, \theta_2, \ldots, \theta_n>)$. The composition $\sigma_1\sigma_2 \cdots \sigma_k$ is called the *substitution of the R-refutation.*

Δ

6.1. The Program Transformations * and r

The * Transformation

Programs for SLDEU-resolution are *compiled* into equivalent programs for SLD-resolution by the following transformation *.

Notation: In this chapter we use α and β, with or without subscripts, for variables introduced by the program transformations * and r, and x, y, z, for other variables.

Δ

Definition 6.5: ($\mathcal{P}$*)
Let h be the transformation mapping a definite clause

C: $Q(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_q$

to its *homogeneous form*

$h(C)$: $Q(\alpha_1, \ldots, \alpha_n) \leftarrow t_1 = \alpha_1, \ldots, t_n = \alpha_n, B_1, \ldots, B_q$

where $\alpha_1, \ldots, \alpha_n$ are distinct variables not occurring in C.[1]

$$P^* = \{ref\} \cup \{h(C) \mid C \in S_=(P \cup Fun)\}$$

where $S_=(P \cup Fun)$ is the symmetric extension of $P \cup Fun$ with respect to $=$.

Δ

We will motivate the transformation * in two steps. First, we introduce some modifications on a standard interpreter for SLD-resolution using a leftmost computation rule. We can consider the new interpreter so obtained as an interpreter for SLDEU-resolution. Then we show that we may *implement* these modifications by the program transformation, *.

We need the following modifications on standard Prolog interpreters:

1. Supplementation of the program:

   a. Supplement a program P by Fun.

   b. Extend $P \cup Fun$ to $S_=(P \cup Fun)$.

   c. Supplement $S_=(P \cup Fun)$ by ref.

2. Modifications of the refutation procedure:

   Let G_{i-1} be $\leftarrow A_1, \ldots, A_m$. Then A_1 is the atom selected by the leftmost computation rule. Let C_i be an input clause from $S_=(P \cup Fun) \cup \{ref\}$.

   a. A_1 is $s=t$ and C_i is ref.

      i. if there is no mgu of $s=t$ and ref, then the derivation fails and the interpreter backtracks;

      ii. otherwise G_i is

      $$\leftarrow (A_2, \ldots, A_m)\theta_i$$

      where θ_i is an mgu computed for $s=t$ and ref.

[1] This is a variant of the homogeneous form transformation first introduced in [van Emden 84]. The homogeneous forms of logic programs are also used in similar contexts in [Elcock 86, Hoddinott 86a, Hoddinott 86b, Demopoulos 86, Chan 86a].

b. A_1 is $P(t_1, \ldots, t_n)$ (P may be $=$), and C_i is not *ref*, and C_i is $P(s_1, \ldots, s_n) \leftarrow B_1, \ldots, B_q$.

Then G_i is

$$\leftarrow s_1=t_1, \ldots, s_n=t_n, B_1, \ldots, B_q, A_2, \ldots, A_m.$$

Let us explain how SLDEU-resolution is *implemented* in this modified system:

- In an E-derivation, an auxiliary clause may come from the program P or from $Fun \cup \{ref\}$. This is accomplished by supplementing the program by Fun and $\{ref\}$ (step 1a and 1c).

- Symmetry is taken care of by extending $P \cup Fun$ to $S_{=}(P \cup Fun)$ (step 1b).

- In an E-derivation, if C_i is *ref*, then θ_i has to be an mgu of the selected equation and *ref*. This is accomplished in step 2a.

- If, in an E-derivation, C_i is not *ref*, then σ_i is an E-unifier of the selected atom and the head of C_i. The new goal is then $\leftarrow(B_1, \ldots, B_q, A_2, \ldots, A_m)\sigma_i$. This is enforced in step 2b. According to the definition of an E-unifier, σ_i is the composition $\theta_1 \cdots \theta_n$ of the E-unifiers $\theta_1, \ldots, \theta_n$ where θ_1 is an E-unifier of the pair of terms $<s_1, t_1>$, and θ_j, $2 \leq j \leq n$, is an E-unifier of $<s_j\theta_1 \cdots \theta_{j-1}, t_j\theta_1 \cdots \theta_{j-1}>$. Moreover, a substitution θ is an E-unifier of a pair of terms $<u, v>$ iff θ is an E-computed answer substitution for $\leftarrow u=v$. It is easy to see that a substitution σ is an E-computed answer substitution for
$$\leftarrow s_1=t_1, \ldots, s_n=t_n$$
iff σ is an E-unifier of $P(s_1, \ldots, s_n)$ and $P(t_1, \ldots, t_n)$. Now in the modified system, the new goal is
$$\leftarrow s_1=t_1, \ldots, s_n=t_n, B_1, \ldots, B_q, A_2, \ldots, A_m.$$
There is an E-unifier σ for $P(s_1, \ldots, s_n)$ and $P(t_1, \ldots, t_n)$ iff there is a derivation from
$$\leftarrow s_1=t_1, \ldots, s_n=t_n, B_1, \ldots, B_q, A_2, \ldots, A_m$$
to
$$\leftarrow(B_1, \ldots, B_q, A_2, \ldots, A_m)\sigma$$
in the modified system.

The above modifications may be *implemented* in SLD-resolution by transforming $S_{=}(\mathcal{P} \cup Fun)$ into its homogeneous form. The effect of adding $s_1=t_1, \ldots, s_n=t_n$ to the new goal is achieved by replacing each clause in $\mathcal{P} \cup Fun \cup S$ by its homogeneous form. However, *ref* is not replaced by its homogeneous form because the substitution used for the resolution of a subgoal with *ref* has to be an mgu.

The r Transformation

Programs for right recursive SLDEU-resolution may be *compiled* into equivalent programs for SLD-resolution by the following transformation r.

Definition 6.6: $(\mathcal{P}^r)$

Let a program $\mathcal{P}$ be partitioned into two sets D and E where D is a set of non-equality clauses and E is a set of equality clauses. Let h^r be the transformation mapping an equality clause

C: $\quad u=v \leftarrow B_1, \ldots, B_q$

to its *right homogeneous form*

$h^r(C)$: $\quad u=\alpha \leftarrow v=\alpha, B_1, \ldots, B_q$

where α is a variable not occurring in C.

$$\mathcal{P}^r = \{ref\} \cup \{h(C) \mid C \in D\} \cup \{h^r(C) \mid C \in S_{=}(E \cup Fun)\}$$

Δ

Note that the term u in C is not replaced by a new variable. This forces the unifier for the left term in an equality subgoal, $s=t$, to be an mgu of u and s. See Chapter 7 §7.4 for some examples that illustrate the operation of h^r.

6.2. Concatenations of SLD-refutations and Compositions of Unifiers

In this section we prove some useful results concerning concatenations of SLD-refutations and compositions of unifiers. The proofs of the Compilation Theorems are based on these results.

Concatenations of SLD-Refutations

Definition 6.7: (Concatenation of goals)
Let

G: $\leftarrow A_1, \ldots, A_m$
G': $\leftarrow A'_1, \ldots, A'_n$

be two goals. The concatenation of G and G', written as $G^\frown G'$, is the goal

$$\leftarrow A_1, \ldots, A_m, A'_1, \ldots, A'_n.$$

△

Definition 6.8: (Concatenation of SLD-refutations)
Let

S: $<G_0, C_0, \theta_0>, <G_1, C_1, \theta_1>, \ldots, <\Box, C_k, \theta_k>$
S': $<G'_0, C'_0, \theta'_0>, <G'_1, C'_1, \theta'_1>, \ldots, <\Box, C'_n, \theta'_n>$

be SLD-refutations. If G'_0 is $G\theta_1\theta_2 \cdots \theta_k$, then the *concatenation of S and S'*, written as $S^\frown S'$ is the sequence

$<G_0{}^\frown G, C_0, \theta_0>$,
$<G_1{}^\frown(G\theta_1), C_1, \theta_1>$,
$\ldots$,
$<G_{k-1}{}^\frown(G\theta_1 \cdots \theta_{k-1}), C_{k-1}, \theta_{k-1}>$,
$<G'_0, C_k, \theta_k>$,
$<G'_1, C'_1, \theta'_1>$,
$\ldots$,
$<\Box, C'_n, \theta'_n>$

△

Theorem 6.1: (Concatenation of SLD-refutations)
Let P be a program and let

S: $<G_0, C_0, \theta_0>, <G_1, C_1, \theta_1>, \ldots, <\Box, C_k, \theta_k>$
S': $<G'_0, C'_0, \theta'_0>, <G'_1, C'_1, \theta'_1>, \ldots, <\Box, C'_n, \theta'_n>$

be SLD-refutations from $P \cup \{G_0\}$ and $P \cup \{G'_0\}$, respectively. If G'_0 is $G\theta_1\theta_2 \cdots \theta_k$, then the concatenation $S^\frown S'$ is an SLD-refutation from $P \cup \{G_0{}^\frown G\}$ of length $k+n$ with $\theta_1\theta_2 \cdots \theta_k\theta'_1\theta'_2 \cdots \theta'_n$ as the substitution of the refutation.

△

Proof:
For all i, $1 \leq i \leq k-1$, $G_i{}^\frown G\theta_1 \cdots \theta_i$ is SLD-derived from $G_{i-1}{}^\frown G\theta_1 \cdots \theta_{i-1}$ and C_i using the mgu θ_i. Since G_k is $\Box$, $G'_0(\equiv G\theta_1 \cdots \theta_k)$ is SLD-derived from $G_{k-1}{}^\frown(G\theta_1 \cdots \theta_{k-1})$ and C_k using θ_k.

Q.E.D.

Compositions of Unifiers

Theorem 6.2: (Composition of unifiers)

$$\begin{aligned} \text{Let } \theta_1 &= \{r_1/x_1, \ldots, r_k/x_k, s_1/y_1, \ldots s_m/y_m\} \\ \theta_2 &= \{t_1/y_1, \ldots, t_m/y_m, u_1/z_1, \ldots, u_n/z_n\} \\ \eta_1 &= \{v_1/\alpha_1, \ldots, v_p/\alpha_p\} \\ \eta_2 &= \{w_1/\beta_1, \ldots, w_q/\beta_q\} \end{aligned}$$

where $x_1, \ldots, x_k, y_1, \ldots, y_m, z_1, \ldots, z_n, \alpha_1, \ldots, \alpha_p, \beta_1, \ldots, \beta_q$ are distinct variables. If the variables $\beta_1, \ldots, \beta_q$ do not occur in the terms of the substitutions θ_1 and η_1, then

$$(\theta_1 \cup \eta_1)(\theta_2 \cup \eta_2) = \theta_1\theta_2 \cup \{v_1\theta_2/\alpha_1, \ldots, v_p\theta_2/\alpha_p\} \cup \eta_2.$$

Δ

Proof:

We prove the result for the case where $k=m=n=p=q=1$. Generalization to the general case is straightforward. In this degenerate case

$$\begin{aligned} \theta_1 \cup \eta_1 &= \{r/x, s/y, v/\alpha\}, \\ \theta_2 \cup \eta_2 &= \{t/y, u/z, w/\beta\}. \end{aligned}$$

Since β does not occur in r, s and v,

$$\begin{aligned} r\,(\theta_2 \cup \eta_2) &= r\theta_2, \\ s\,(\theta_2 \cup \eta_2) &= s\theta_2, \\ v\,(\theta_2 \cup \eta_2) &= v\theta_2. \end{aligned}$$

Hence

$$\begin{aligned} (\theta_1 \cup \eta_1)(\theta_2 \cup \eta_2) &= \{r\theta_2/x, s\theta_2/y, v\theta_2/\alpha, u/z, w/\beta\} \\ &= \{r\theta_2/x, s\theta_2/y, u/z\} \cup \{v\theta_2/\alpha\} \cup \{w/\beta\} \\ &= \theta_1\theta_2 \cup \{v\theta_2/\alpha\} \cup \eta_2. \end{aligned}$$

Q.E.D.

6.3. Compilation Theorem for SLDEU-Resolution

In this section we prove the Compilation Lemma from which the Compilation Theorem for SLDEU-resolution follows. The proof of the Compilation Lemma for SLDEU-resolution is much more complicated than the proof of the Compilation Lemma for symmetric SLD-resolution. This is because E-refutations are fairly complex formal objects. Except for derivation steps in which the input clause is $x=x$, every step in an E-derivation incorporates a tuple of E-refutations which demonstrates that the substitution used in the step is in fact an E-unifier of the atoms resolved upon. In general, each of these embedded E-refutations contains other embedded E-refutations.[2] In what follows, we first define a function Γ from E-refutations to natural numbers; this function measures the *complexity* of E-refutations. We then prove that there is an E-refutation of complexity m from $P \cup \{G\}$ iff there is an SLD-refutation of length m from $P^* \cup \{G\}$.

Definition 6.9: (Complexity of E-refutations)
Let A be a E-refutation of the form

$$<G_0, \_, \_, \_>, <G_1, C_1, U_1, \Delta_1>, \ldots, <G_k, C_k, U_k, \Delta_k>$$

where Δ_i, $1 \leq i \leq k$, is $<\delta_i^1, \delta_i^2, \ldots, \delta_i^{n_i}>$. The *complexity* $\Gamma(A)$ of A is defined as follows:

$$\Gamma(A) = \begin{cases} 0, & \text{if } A \text{ is an E-refutation of length 0;} \\ \sum_{i=1}^{k} (1+\gamma(\Delta_i)), & \text{otherwise.} \end{cases}$$

where $\gamma(\Delta_i)$ is the sum of the complexities of the E-refutations in Δ_i and γ is defined as follows:

$$\gamma(\Delta_i) = \begin{cases} 0, & \text{if } \Delta_i \text{ is the null tuple } <>; \\ \sum_{j=1}^{n_i} \Gamma(\delta_i^j), & \text{if } \Delta_i \text{ is an } n_i\text{-tuple.} \end{cases}$$

△

Each step in A contributes one unit of complexity and $\Gamma(A)$ is the sum of the complexity of each constituent E-refutation in A plus the complexity of each step in A.

Example 6.1:

P: $a=b$.
$P(a)$.
$Q(c) \leftarrow P(b)$.

1. $A1$: $<\square, \_, \_, \_>$
$\Gamma(A1)=0$.
because the length of $A1$ is 0.

[2] Derivations of SLDNF-resolution are formal objects involving the same kind of complexity. A finitely failed SLDNF-tree is embedded in each step in which the selected literal of the previous goal is a negative literal [Clark 78, Lloyd 84, Chan 86b].

2. $A2$: $<\leftarrow a{=}a,\ \_,\ \_,\ \_>,\ <\Box, x{=}x, <\{a/x\}>, <>>$
$\Gamma(A2) = 1+\gamma(<>) = 1+0 = 1$

3. $A3$: $<\leftarrow b{=}b,\ \_,\ \_,\ \_>,\ <\Box, y{=}y, <\{b/y\}>, <>>$
$\Gamma(A3) = 1$

4. $A4$: $<\leftarrow a{=}b,\ \_,\ \_,\ \_>,\ <\Box, a{=}b, <\{a/x\},\{b/y\}>, <A2,A3>>$
$\Gamma(A4) = 1+\gamma(<A2,A3>) = 1+(1+1) = 3.$

5. $A5$: $<\leftarrow P(b),\ \_,\ \_,\ \_>,\ <\Box, P(a), <\{a/x, b/y\}>, <A4>>$
$\Gamma(A5) = 1+\gamma(<A4>) = 1+\Gamma(A4) = 1+3 = 4.$

6. $A6$: $<\leftarrow Q(c),\ \_,\ \_,\ \_>,$
$<\leftarrow P(b), Q(c) \leftarrow P(b), <\{c/z\}>, <A2'>>,$
$<\Box, P(a), <\{a/x\},\{b/y\}>, <A4>>$
where $A2'$ is an E-refutation from $P \cup \{\leftarrow c{=}c\}$ similar to $A2$.

$$\Gamma(A6) = \sum_1^2 (1+\gamma(\Delta_i)) = (1+\Gamma(A2')) + (1+\Gamma(A4)) = (1+1)+(1+3) = 6.$$

Δ

Notation: We use η, with or without subscripts and primes, for substitutions of variables introduced by *, and we use σ, θ, for substitutions of other variables.

Δ

Lemma 6.1: (For SLDEU-resolution via the leftmost computation rule)
Let P be a program and G a goal. There is an E-refutation from $P \cup \{G\}$ of complexity m with substitution σ iff there is an SLD-refutation from $P^* \cup \{G\}$ of length m with substitution $\sigma \cup \eta$ where η is a substitution for variables introduced by *.

Δ

Proof: (By induction on m)
($\Rightarrow$) Let

$$A:\quad \begin{array}{l} T_0(\equiv <G,\ \_,\ \_,\ \_>), \\ T_1(\equiv <G_1, C_1, U_1, \Delta_1>), \\ \quad \cdots \\ T_k(\equiv <\Box, C_k, U_k, \Delta_k>) \end{array}$$

be an E-refutation from $P \cup \{G\}$ of complexity m. Let σ_i, $1 \le i \le k$, be the composition $\theta_1\theta_2 \cdots \theta_n$ of the substitutions in $U_i(\equiv <\theta_1, \theta_2, \ldots, \theta_n>)$. $\sigma = \sigma_1\sigma_2 \cdots \sigma_k$ is the substitution of A.

Base case: $m=0$.
Then G is $\Box$ and $\{\}$ is the substitution of A. The sequence

$<\Box,\ \_,\ \_>$

is an SLD-refutation from $P^* \cup \{G\}$ of length 0 with the substitution $\{\}$.

Inductive step.

Assume that the result holds for all $i < m$. Consider the subsequence

$\mathcal{A}'$: $\quad T_1, T_2, \ldots, T_k$.

$\sigma_2\sigma_3 \cdots \sigma_k$ is the substitution of $\mathcal{A}'$. Let $\Gamma(\mathcal{A}')$ be m'.

$$m = 1 + \gamma(\Delta_1) + m'$$

Hence $m' < m$. By the inductive hypothesis, there is an SLD-refutation S' from $P^* \cup \{G_1\}$ of length m' with a substitution $\sigma_2\sigma_3 \cdots \sigma_k \cup \eta'$. There are three cases:

1. C_1 is *ref*.
2. C_1 is an equality clause other than *ref*.
3. C_1 is a non-equality clause.

Case 1.

G is $\leftarrow s=t, A_2, \ldots, A_p$;

C_1 is $x=x$;

Δ_1 is $<>$;

U_1 is $<\sigma_1>$, where σ_1 is an mgu of $s=t$ and $x=x$; and

G_1 is $\leftarrow (A_2, \ldots, A_p)\sigma_1$.

Since $\gamma(\Delta_1) = \gamma(<>) = 0$,

$$m' = \Gamma(\mathcal{A}) - 1 = m - 1.$$

Let S' be the sequence

$$\Psi_0, \Psi_1, \Psi_2, \ldots, \Psi_{m-1}.$$

Let S be the sequence

$$<G, \_, \_>, <G_1, C_1, \sigma_1>, \Psi_1, \Psi_2, \ldots, \Psi_{m-1}.$$

G_1 is SLD-derived from G and $C_1 (\equiv x=x)$ using the mgu σ_1. Hence S is an SLD-refutation from $P^* \cup \{G\}$ of length m. The substitution of S is

$$\sigma_1(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta') = \sigma_1\sigma_2 \cdots \sigma_k \cup \eta' \qquad \text{(by Theorem 6.2)}$$
$$= \sigma \cup \eta'.$$

Case 2.

G is $\leftarrow s=t, A_2, \ldots, A_p$;

C_1 is $u=v \leftarrow B_1, \ldots, B_q$;

Δ_1 is $<\delta_1, \delta_2>$,

U_1 is $<\theta_1, \theta_2>$,

and

G_1 is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\theta_2$.

There are two cases:

(I) δ_1 is an E-refutation from $\mathcal{P} \cup \{ \leftarrow u=s \}$ with substitution θ_1 and δ_2 is an E-refutation from $\mathcal{P} \cup \{ \leftarrow (v=t)\theta_1 \}$ with substitution θ_2;

or

(II) δ_1 is an E-refutation from $\mathcal{P} \cup \{ \leftarrow v=s \}$ with substitution θ_1 and δ_2 is an E-refutation from $\mathcal{P} \cup \{ \leftarrow (u=t)\theta_1 \}$ with substitution θ_2.

Consider case (II) where δ_1 is an E-refutation from $\mathcal{P} \cup \{ \leftarrow v=s \}$ with substitution $\theta_1 = \varsigma_1 \varsigma_2 \cdots \varsigma_d$ and δ_2 is an E-refutation from $\mathcal{P} \cup \{ \leftarrow (u=t)\theta_1 \}$ with substitution $\theta_2 = \pi_1 \pi_2 \cdots \pi_e$.

Let $\Gamma(\delta_1)$ be m_1 and $\Gamma(\delta_2)$ be m_2. Then

$m = (1+m_1+m_2)+m'$.

Hence $m_1, m_2 < m$. By the inductive hypothesis, there is an SLD-refutation $S1$ with the substitution $\theta_1 \cup \eta_1$ from $\mathcal{P}^* \cup \{ \leftarrow v=s \}$ of length m_1 and there is an SLD-refutation $S2$ with the substitution $\theta_2 \cup \eta_2$ from $\mathcal{P}^* \cup \{ \leftarrow (u=t)\theta_1 \}$ of length m_2.

Since η_2 is a substitution for variables introduced by the transformation *,

G_1: $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\theta_2$

is identical to

$\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1(\theta_2 \cup \eta_2)$.

Hence $S2^\frown S'$ is defined according to Definition 6.8. By the Concatenation Theorem $S2^\frown S'$ is an SLD-refutation from $\mathcal{P}^* \cup \{ \leftarrow (u=t, B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1 \}$ of length m_2+m' with the substitution $(\theta_2 \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta')$. Similarly, $S1^\frown(S2^\frown S')$ is defined and is an SLD-refutation from $\mathcal{P}^* \cup \{ \leftarrow v=s, u=t, B_1, \ldots, B_q, A_2, \ldots, A_p \}$ of length m_1+m_2+m' with the substitution $(\theta_1 \cup \eta_1)(\theta_2 \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta')$.

Because C_1 is $u=v \leftarrow B_1, \ldots, B_q$, there is a clause

$h(C_1)$: $\alpha=\beta \leftarrow v=\alpha, u=\beta, B_1, \ldots, B_q$

in $\mathcal{P}^*$. Let $S1^\frown(S2^\frown S')$ be the sequence

$\Psi_0, \Psi_1, \Psi_2, \ldots, \Psi_{m_1+m_2+m'}$.

Let S be the sequence

$<G, \_, \_>$,

$< \leftarrow v=s, u=t, B_1, \ldots, B_q, A_2, \ldots, A_p;\ h(C_1);\ \{s/\alpha, t/\beta\}>$,

$\Psi_1, \Psi_2, \ldots, \Psi_{m_1+m_2-m'}$.

$\leftarrow v{=}s, u{=}t, B_1, \ldots, B_q, A_2, \ldots, A_p$ is SLD-derived from G and $h(C_1)$ using the mgu $\{s/\alpha, t/\beta\}$. Hence S is an SLD-refutation from $\mathcal{P}^* \cup \{G\}$ of length $1+m_1+m_2+m'$ $(=m)$.

The substitution of S is

$$\begin{aligned} &\{s/x,\ t/y\}(\theta_1 \cup \eta_1)(\theta_2 \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta') \\ =&\{s/x,\ t/y\}(\theta_1\theta_2 \cup \eta_1' \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta') \\ =&\{s/x,\ t/y\}(\theta_1\theta_2\sigma_2\sigma_3 \cdots \sigma_k \cup \eta_1'' \cup \eta_2' \cup \eta') \\ =&\theta_1\theta_2\sigma_2\sigma_3 \cdots \sigma_k \cup \eta_3 \cup \eta_1'' \cup \eta_2' \cup \eta' \\ =&\theta_1\theta_2\sigma_2\sigma_3 \cdots \sigma_k \cup \eta \end{aligned}$$

where η is a substitution for variables introduced by $*$. Since $\theta_1\theta_2$ is σ_1, the substitution of S is $\sigma \cup \eta$.

Remaining cases may be proved in the same way.

($\Leftarrow$) Let

$$\begin{aligned} S: \quad &\Psi_0(\equiv <G,\ \_,\ \_>), \\ &\Psi_1(\equiv <G_1,\ C_1, \xi_1>), \\ &\ldots, \\ &\Psi_m(\equiv <\Box,\ C_m,\ \xi_m>) \end{aligned}$$

be an SLD-refutation from $\mathcal{P}^* \cup \{G\}$ of length m with substitution $\sigma \cup \eta$.

Base case: $m{=}0$.
$<\Box,\ \_,\ \_>$ is the only member of S and $\{\}$ is the substitution of S. The sequence

$$<\Box,\ \_,\ \_,\ \_>$$

is an E-refutation from $\mathcal{P} \cup \{G\}$ of complexity 0 and substitution $\{\}$.

Inductive step.
Assume that the result holds for all $i < m$. Consider Ψ_0 and Ψ_1. There are three cases:

1. C_1 is *ref*.
2. C_1 is the homogeneous form of a clause in $\mathcal{P} \cup \mathit{Fun}$.
3. C_1 is the homogeneous form of a clause in $S_{=}(\mathcal{P} \cup \mathit{Fun})-(\mathcal{P} \cup \mathit{Fun})$.

Case 1.
C_1 is $x{=}x$;
G is $\leftarrow s{=}t, A_2, \ldots, A_p$;
ξ_1 is an mgu of $x{=}x$ and $s{=}t$, and
G_1 is $\leftarrow(A_2, \ldots, A_p)\xi_1$.
The subsequence

$\Psi_1, \Psi_2, \ldots, \Psi_m$

is an SLD-refutation from $\mathcal{P}^* \cup \{G_1\}$ of length $m-1$ with substitution $\xi_2 \cdots \xi_m = \sigma' \cup \eta'$.

By the inductive hypothesis, there is an E-refutation

Λ': $T_0, T_1, \ldots, T_k$

from $\mathcal{P} \cup \{G_1\}$ of complexity $m-1$ with σ' as its substitution. The sequence

Λ: $<G, \_, \_, \_>$,
$<G_1, x=x, <\xi_1>, <>>$,
$T_1, \ldots, T_k$

is an E-refutation from $\mathcal{P} \cup \{G\}$ with substitution $\xi_1\sigma'$.

$$\Gamma(\Lambda) = (1+0)+(m-1) = m$$

$$\begin{aligned}\sigma \cup \eta &= \xi_1\xi_2 \cdots \xi_m \\ &= \xi_1(\sigma' \cup \eta') \\ &= \xi_1\sigma' \cup \eta' \qquad \text{(by Theorem 6.2)}\end{aligned}$$

Hence $\sigma(=\xi_1\sigma')$ is the answer substitution of Λ'.

We will prove case 3. Case 2 may be proved in a similar way.

C_1: $\alpha=\beta \leftarrow v=\alpha, u=\beta, B_1, \ldots, B_q$

is the homogeneous form of a clause C in $S(\mathcal{P} \cup Fun)-(\mathcal{P} \cup Fun)$. G is $\leftarrow s=t, A_2, \ldots, A_p$.

G_1 is $\leftarrow v=s, u=t, B_1, \ldots, B_q, A_2, \ldots, A_p$ and ξ_1 is $\{s/\alpha, t/\beta\}$. Let G' be $\leftarrow u=t, B_1, \ldots, B_q, A_2, \ldots, A_p$ and G'' be $\leftarrow B_1, \ldots, B_q, A_2, \ldots, A_p$. The sequence of goals in S is

G,

$G_1'{}^\sim G'$, $\qquad (\equiv \leftarrow v=s^\sim \leftarrow u=t, B_1, \ldots, B_q, A_2, \ldots, A_p)$

$G_2'{}^\sim G'\xi_2$,

$\ldots$

$G_k'{}^\sim G'\xi_2 \cdots \xi_k$, $\qquad (\equiv G_k'{}^\sim \leftarrow (u=t, B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_2 \cdots \xi_k)$

$G_{k+1}'{}^\sim G''\xi_2 \cdots \xi_{k+1}$, $\qquad (\equiv \leftarrow (u=t)\xi_2 \cdots \xi_{k+1}{}^\sim \leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_2 \cdots \xi_{k+1})$

$G_{k+2}'{}^\sim G''\xi_2 \cdots \xi_{k+2}$,

$\ldots$

$G_{k+l}'{}^\sim G''\xi_2 \cdots \xi_{k+l}$,

$G''\xi_2 \cdots \xi_{k+l+1}$, $\qquad (\equiv \leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_2 \cdots \xi_{k+l+1})$

$\ldots$

□

It is obvious that

$S1$: $\quad <G'_1, \_, \_>, <G'_2, C_2, \xi_2>, \ldots, <\Box, C_{k+1}, \xi_{k+1}>$

is an SLD-refutation from $\mathcal{P}^* \cup \{G'_1\}$.

$S2$: $\quad <G'_{k+1}, \_, \_>, <G'_{k+2}, C_{k+2}, \xi_{k+2}>, \ldots, <\Box, C_{k+l+1}, \xi_{k+l+1}>$

is an SLD-refutation from $\mathcal{P}^* \cup \{G'_{k+1}\}$ and

S' $\quad <G''\xi_2 \cdots \xi_{k+l+1}, \_, \_>,$

$<G_{k+l+2}, C_{k+l+2}, \xi_{k+l+2}>,$

$\ldots$

$<\Box, C_m, \xi_m>$

is an SLD-refutation from $\mathcal{P}^* \cup \{G''\xi_2 \cdots \xi_{k+l+1}\}$.

G'_1 is $\leftarrow v{=}s$ and $S1$ is an SLD-refutation of length k with substitution $\xi_2 \cdots \xi_{k+1} = \theta_1 \cup \eta_1$.
G'_{k+1} is $\leftarrow (u{=}t)\xi_2 \cdots \xi_{k+1}$ ($\equiv \leftarrow (u{=}t)(\theta_1 \cup \eta_1) \equiv \leftarrow (u{=}t)\theta_1$) and $S2$ is an SLD-refutation of length l with substitution $\xi_{k+2} \cdots \xi_{k+l+1} = \theta_2 \cup \eta_2$.
$G''\xi_2 \cdots \xi_{k+l+1}$ is $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_2 \cdots \xi_{k+l+1}$ and S' is an SLD-refutation of length $m-(k+l+1)$ with substitution $\xi_{k+l+2} \cdots \xi_m = \sigma' \cup \eta'$.

By the inductive hypothesis, there are E-refutations δ_1, δ_2, and Δ' such that (i) δ_1 is an E-refutation from $\mathcal{P} \cup \{\leftarrow v{=}s\}$ of complexity k and substitution θ_1, (ii) δ_2 is an E-refutation from $\mathcal{P} \cup \{\leftarrow (u{=}t)\theta_1\}$ of complexity l and substitution θ_2 and (iii)

Δ': $\quad T_0, T_1, \ldots, T_n$

is an E-refutation from $\mathcal{P} \cup \{\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_2 \cdots \xi_{k+l+1}\}$ of complexity $m-(k+l+1)$ and substitution σ'.

Let η_1 be $\{u_1/\alpha_1, \ldots, u_a/\alpha_a\}$ and η'_1 be $\{u_1\theta_2/\alpha_1, \ldots, u_a\theta_2/\alpha_a\}$.

$$\xi_2 \cdots \xi_{k+l+1} = (\theta_1 \cup \eta_1)(\theta_2 \cup \eta_2)$$
$$= \theta_1\theta_2 \cup \eta'_1 \cup \eta_2 \qquad \text{(by Theorem 6.2)}$$

Since η'_1 and η_2 are substitutions for variables not occurring in $\leftarrow B_1, \ldots, B_q, A_2, \ldots, A_p$,

$$\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_2 \cdots \xi_{k+l+1}$$

is

$$\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\theta_2$$

It is straightforward to check that the sequence

Δ: $\quad <G, \_, \_, \_>,$

$<\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\theta_2; u{=}v \leftarrow B_1, \ldots, B_q;$

$<\theta_1, \theta_2>; <\delta_1, \delta_2>>,$

$T_1, \ldots, T_n$

is an E-refutation with substitution $\theta_1\theta_2\sigma'$.

$$\begin{aligned}\Gamma(\mathcal{A}) &= (1+\gamma(<\delta_1 . \delta_2>)) + \Gamma(\mathcal{A}') \\ &= (1+(k+l)) + m-(k+l+1) \\ &= m\end{aligned}$$

$$\begin{aligned}\xi_1\xi_2 \cdots \xi_m &= \xi_1(\theta_1 \cup \eta_1)(\theta_2 \cup \eta_2)(\sigma' \cup \eta') \\ &= \xi_1(\theta_1\theta_2 \cup \eta_1' \cup \eta_2)(\sigma' \cup \eta') \\ &= \xi_1(\theta_1\theta_2\sigma' \cup \eta_1'' \cup \eta_2' \cup \eta') \\ &= \theta_1\theta_2\sigma' \cup \xi_1' \cup \eta_1'' \cup \eta_2' \cup \eta'\end{aligned}$$

for some substitutions ξ_1', η_1'', η_2' for variables introduced by the transformation * (by associativity and Theorem 6.2). Hence σ is $\theta_1\theta_2\sigma'$ and η is $\xi_1' \cup \eta_1'' \cup \eta_2' \cup \eta'$. $\mathcal{A}$ is an E-refutation from $\mathcal{P} \cup \{G\}$ of complexity m and substitution σ.

Q.E.D.

The Compilation Theorem follows from the Compilation Lemma.

Theorem 6.3: (Compilation. SLDEU-resolution)
Let $\mathcal{P}$ be a program and let G be a goal. There is an E-refutation from $\mathcal{P} \cup \{G\}$ with computed answer substitution θ iff there is an SLD-refutation from $\mathcal{P}^* \cup \{G\}$ with θ as the computed answer substitution.

Δ

6.4. Compilation Theorem for Right Recursive SLDEU-Resolution

The proofs of the Compilation Theorems for SLDEU-resolution and right recursive SLDEU-resolution are similar. The function Γ introduced in Definition 6.9 also applies to R-refutations of right recursive SLDEU-resolution. What follows is the Compilation Lemma for right recursive SLDEU-resolution with the leftmost selection rule.

Lemma 6.2: (For right recursive SLDEU-resolution via the leftmost computation rule) Let $\mathcal{P}$ be a program and G be a goal. There is an R-refutation from $\mathcal{P} \cup \{G\}$ of complexity m with substitution σ iff there is an SLD-refutation from $\mathcal{P}^r \cup \{G\}$ of length m with substitution $\sigma \cup \eta$.

Δ

Proof: (By induction on m.)
(The parts of the proof that are different from the proof of the Compilation Lemma for SLDEU-resolution are highlighted by vertical bars at the left margin.)
($\Rightarrow$) Let

A: $\quad T_0(\equiv <G, \_, \_, \_>)$,

$\quad T_1(\equiv <G_1, C_1, U_1, \Delta_1>)$,

$\quad \ldots$

$\quad T_k(\equiv <\square, C_k, U_k, \Delta_k>)$

be an R-refutation from $P \cup \{G\}$ of complexity m. Let σ_i, $1 \le i \le k$, be the composition $\theta_1\theta_2\cdots\theta_n$ of the substitutions in $U_i(\equiv <\theta_1, \theta_2, \ldots, \theta_n>)$. $\sigma = \sigma_1\sigma_2\cdots\sigma_k$ is the substitution of A.

Base case: $m=0$.
Then G is $\square$ and $\{\}$ is the substitution of A. The sequence

$\quad <\square, \_, \_>$

is an SLD-refutation from $P^r \cup \{G\}$ of length 0 with the substitution $\{\}$.

Inductive step.
Assume that the result holds for all $i < m$. Consider the subsequence

A': $\quad T_1, T_2, \ldots, T_k$

$\sigma_2\sigma_3\cdots\sigma_k$ is the substitution of A'. Let $\Gamma(A')$ be m'. Then

$$m = 1 + \gamma(\Delta_1) + m'$$

Hence $m' < m$.

By the inductive hypothesis, there is an SLD-refutation S' from $P^r \cup \{G_1\}$ of length m' with a substitution $\sigma_2\sigma_3\cdots\sigma_k \cup \eta'$. There are three cases:

1. C_1 is *ref*.
2. C_1 is an equality clause other than *ref*.
3. C_1 is a non-equality clause.

Case 1.
G is $\leftarrow s=t, A_2, \ldots, A_p$;
C_1 is $x=x$;
Δ_1 is $<>$;
U_1 is $<\sigma_1>$, where σ_1 is an mgu of $s=t$ and $x=x$; and
G_1 is $\leftarrow(A_2, \ldots, A_p)\sigma_1$.

Since $\gamma(\Delta_1) = \gamma(<>) = 0$,

$$m' = \Gamma(A) - 1 = m-1.$$

Let S' be the sequence

$\quad \Psi_0, \Psi_1, \Psi_2, \ldots, \Psi_{m-1}$.

Let S be the sequence

$\quad <G, \_, \_>, <G_1, C_1, \sigma_1>, \Psi_1, \Psi_2, \ldots, \Psi_{m-1}$.

G_1 is SLD-derived from G and $C_1(\equiv x=x)$ using the mgu σ_1. Hence S is an SLD-refutation from $\mathcal{P}^r \cup \{G\}$ of length m. The substitution of S is

$$\sigma_1(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta') = \sigma_1\sigma_2 \cdots \sigma_k \cup \eta' \qquad \text{(by Theorem 6.2)}$$
$$= \sigma \cup \eta'.$$

Case 2.
G is $\leftarrow s=t, A_2, \ldots, A_p$;
C_1 is $u=v \leftarrow B_1, \ldots, B_q$;
Δ_1 is $<\delta_2>$;
U_1 is $<\theta_1, \theta_2>$; and
G_1 is $\leftarrow(B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\theta_2$.

There are two cases:
(I) θ_1 is an mgu of u and s and
δ_2 is an R-refutation from $\mathcal{P} \cup \{\leftarrow(v=t)\theta_1\}$ with substitution θ_2;
or
(II) θ_1 is an mgu of v and s and
δ_2 is an R-refutation from $\mathcal{P} \cup \{\leftarrow(u=t)\theta_1\}$ with substitution θ_2.

Consider case (II) where δ_2 is an R-refutation from $\mathcal{P} \cup \{\leftarrow(u=t)\theta_1\}$ with substitution $\theta_2 = \pi_1\pi_2 \cdots \pi_e$.

Let $\Gamma(\delta_2)$ be m_2. $m = (1+m_2)+m'$. Hence $m_2 < m$. By the inductive hypothesis, there is an SLD-refutation $S2$ with the substitution $\theta_2 \cup \eta_2$ from $\mathcal{P}^r \cup \{\leftarrow(u=t)\theta_1\}$ of length m_2.

Since η_2 is a substitution for variables introduced by the transformation r,
G_1: $\quad \leftarrow(B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\theta_2$
is identical to

$$\leftarrow(B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1(\theta_2 \cup \eta_2).$$

Hence $S2^\frown S'$ is defined according to Definition 6.8. By the Concatenation Theorem $S2^\frown S'$ is an SLD-refutation from $\mathcal{P}^r \cup \{\leftarrow(u=t, B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\}$ of length m_2+m' with the substitution $(\theta_2 \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta')$.

Because C_1 is $u=v \leftarrow B_1, \ldots, B_q$, there is a clause
$h^r(C_1)$: $v=\alpha \leftarrow u=\alpha, B_1, \ldots, B_q$
in $\mathcal{P}^r$. Let $S2^\frown S'$ be the sequence

$$\Psi_0, \Psi_1, \Psi_2, \ldots, \Psi_{m_2-m'}.$$

Let S be the sequence

$<G, \_, \_>$,
$< \leftarrow (u{=}t, B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1;\ h^r(C_1);\ \theta_1\{t/\alpha\}>$,
$\Psi_1, \Psi_2, \ldots, \Psi_{m_2+m'}$

Since α does not occur in $\leftarrow (u{=}t, B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1$,

$$\leftarrow (u{=}t, B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1 \equiv$$
$$\leftarrow (u{=}t, B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\{t/\alpha\}$$

and $\leftarrow (u{=}t, B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1$ is SLD-derived from G and $h^r(C_1)$ using the mgu $\theta_1\{t/\alpha\}$. Hence S is an SLD-refutation from $\mathcal{P}^r \cup \{G\}$ of length $1 + m_2 + m'$ $(= m)$.

Let η_1 be $\{t/\alpha\}$. By Theorem 6.2, $\theta_1\eta_1 = \theta_1 \cup \eta_1$. The substitution of S is

$$\theta_1\eta_1(\theta_2 \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta')$$
$$= (\theta_1 \cup \eta_1)(\theta_2 \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta')$$
$$= (\theta_1\theta_2 \cup \eta_1' \cup \eta_2)(\sigma_2\sigma_3 \cdots \sigma_k \cup \eta')$$
$$= (\theta_1\theta_2\sigma_2\sigma_3 \cdots \sigma_k \cup \eta_1'' \cup \eta_2' \cup \eta')$$
$$= \theta_1\theta_2\sigma_2\sigma_3 \cdots \sigma_k \cup \eta$$

where η is a substitution for variables introduced by r. Since $\theta_1\theta_2$ is σ_1, the substitution of S is $\sigma \cup \eta$.

Remaining cases may be proved in the same way.

($\Leftarrow$) Let

S: $\quad \Psi_0 (\equiv <G, \_, \_>)$,
$\quad \Psi_1 (\equiv <G_1, C_1, \xi_1>)$,
$\quad \ldots$,
$\quad \Psi_m (\equiv <\Box, C_m, \xi_m>)$

be an SLD-refutation from $\mathcal{P}^r \cup \{G\}$ of length m with substitution $\sigma \cup \eta$.

Base case: $m{=}0$.
$<\Box, \_, \_>$ is the only member of S and $\{\}$ is the substitution of S. The sequence

$<\Box, \_, \_, \_>$

is an R-refutation from $\mathcal{P} \cup \{G\}$ of complexity 0 and substitution $\{\}$.

Inductive step.
Assume that the result holds for all $i < m$. Consider Ψ_0 and Ψ_1. There are three cases:

1. C_1 is *ref.*
2. C_1 is the homogeneous form of a clause in D.

3. C_1 is the right homogeneous form of a clause in $S_=(E \cup Fun)$.

Case 1.
C_1 is $x=x$.
G is $\leftarrow s=t, A_2, \ldots, A_p$.
ξ_1 is an mgu of $x=x$ and $s=t$.
and
G_1 is $\leftarrow (A_2, \ldots, A_p)\xi_1$.
The subsequence

$$\Psi_1, \Psi_2, \ldots, \Psi_m$$

is an SLD-refutation from $P^r \cup \{G_1\}$ of length $m-1$ with substitution $\xi_2 \cdots \xi_m = \sigma' \cup \eta'$.

By the inductive hypothesis, there is an R-refutation

Λ': $T_0, T_1, \ldots, T_k$

from $P \cup \{G_1\}$ of complexity $m-1$ with σ' as its substitution. The sequence

Λ: $<G, \_, \_, \_>$,
$<G_1, x=x, <\xi_1>, <>>$,
$T_1, \ldots, T_k$

is an R-refutation from $P \cup \{G\}$ with substitution $\xi_1\sigma'$.

$$\Gamma(\Lambda) = (1+0)+(m-1) = m$$
$$\begin{aligned}\sigma \cup \eta &= \xi_1\xi_2 \cdots \xi_m \\ &= \xi_1(\sigma' \cup \eta') \\ &= \xi_1\sigma' \cup \eta' \qquad \text{(by Theorem 6.2)}\end{aligned}$$

Hence $\sigma(=\xi_1\sigma')$ is the answer substitution of Λ'.

We will prove case 3. Case 2 may be proved in a similar way.

C_1: $u=\alpha \leftarrow v=\alpha, B_1, \ldots, B_q$
is the right homogeneous form of a clause C in $S(E \cup Fun)$. G is $\leftarrow s=t, A_2, \ldots, A_p$.
Let θ_1 be the mgu computed for $<u,s>$.
G_1 is $\leftarrow (v=t, B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_1$ where ξ_1 is $\theta_1\{t/\alpha\}$. Let G' be $\leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\xi_1$. The sequence of goals in S is

$G,$

$G_1'\sim G',\qquad (\equiv\ \leftarrow(v=t)\xi_1\sim\ \leftarrow(B_1,\ldots,B_q,A_2,\ldots,A_p)\xi_1)$

$G_2'\sim G'\xi_2,$

$\ldots,$

$G_k'\sim G'\xi_2\cdots\xi_k,\qquad (\equiv G_k'\sim$
$\qquad\leftarrow(B_1,\ldots,B_q,A_2,\ldots,A_p)\xi_1\xi_2\cdots\xi_k)$

$G'\xi_1\xi_2\cdots\xi_{k+1},\qquad (\equiv\ \leftarrow(B_1,\ldots,B_q,A_2,\ldots,A_p)\xi_1\xi_2\cdots\xi_{k+1})$

$\ldots,$

$\square$

It is obvious that

$S2$: $\quad <G_1',\_,\_>,\ <G_2',C_2,\xi_2>,\ldots,\ <\square,C_{k+1},\xi_{k+1}>$

is an SLD-refutation from $P^r\cup\{G_1'\}$, and

S': $\quad <G'\xi_1\xi_2\cdots\xi_{k+1},\_,\_>,\ <G_{k+2},C_{k+2},\xi_{k+2}>,\ldots,\ <\square,C_m,\xi_m>,$

is an SLD-refutation from $P^r\cup\{G'\xi_1\xi_2\cdots\xi_{k+1}\}$.

G_1' is $\leftarrow(v=t)\xi_1(\equiv\ \leftarrow(v=t)\theta_1)$ and $S2$ is an SLD-refutation of length k with substitution $\xi_2\cdots\xi_{k+1}=\theta_2\cup\eta_2$.
$G'\xi_1\xi_2\cdots\xi_{k+1}$ is $\leftarrow(B_1,\ldots,B_q,A_2,\ldots,A_p)\xi_1\xi_2\cdots\xi_{k+1}$ and S' is an SLD-refutation of length $m-(k+1)$ with substitution $\xi_{k+2}\cdots\xi_m=\sigma'\cup\eta'$.

By the inductive hypothesis, there are R-refutations δ_2 and Δ' such that δ_2 is an R-refutation from $P\cup\{\leftarrow(v=t)\theta_1\}$ of complexity k and substitution θ_2, and

Δ': $\quad T_0,T_1,\ldots,T_n$

is an R-refutation from $P\cup\{\leftarrow(B_1,\ldots,B_q,A_2,\ldots,A_p)\xi_1\xi_2\cdots\xi_{k+1}\}$ of complexity $m-(k+1)$and substitution σ'.

Let η_1 be $\{t/\alpha\}$. Then

$$\begin{aligned}\xi_1\xi_2\cdots\xi_{k+1}&=(\theta_1\{t/\alpha\})(\theta_2\cup\eta_2)\\&=(\theta_1\cup\eta_1)(\theta_2\cup\eta_2)\\&=\theta_1\theta_2\cup\eta^*\end{aligned}$$

where η^* is a substitution for variables not occurring in $\leftarrow B_1,\ldots,B_q,A_2,\ldots,A_p$. Hence

$\leftarrow(B_1,\ldots,B_q,A_2,\ldots,A_p)\xi_1\xi_2\cdots\xi_{k+1}$

is

$\leftarrow(B_1,\ldots,B_q,A_2,\ldots,A_p)\theta_1\theta_2$

Suppose

C: $\quad v=u\leftarrow B_1,\ldots,B_q$

is in $E\cup Fun$. It is straightforward to check that the sequence

$\mathcal{A}$: $\quad < G, \_, \_, \_ >,$

$< \leftarrow (B_1, \ldots, B_q, A_2, \ldots, A_p)\theta_1\theta_2; C; <\theta_1, \theta_2>; <\delta_2>>,$

$T_1, \ldots, T_n$

is an R-refutation with substitution $\theta_1\theta_2\sigma'$.

$$\begin{aligned} \Gamma(\mathcal{A}) &= (1+\gamma(<\delta_2>)) + \Gamma(\mathcal{A}') \\ &= (1+k) + m - (k+1) \\ &= m \end{aligned}$$

$$\begin{aligned} \xi_1\xi_2 \cdots \xi_m &= (\theta_1\theta_2 \cup \eta^*)(\sigma' \cup \eta') \\ &= \theta_1\theta_2\sigma' \cup \eta \end{aligned}$$

(by associativity of substitution composition and Theorem 6.2). Hence σ is $\theta_1\theta_2\sigma'$. $\mathcal{A}$ is an R-refutation from $\mathcal{P} \cup \{G\}$ of complexity m with substitution σ.

Q.E.D.

The Compilation Theorem follows from the Compilation Lemma.

Theorem 6.4: (Compilation, right recursive SLDEU-resolution)
Let $\mathcal{P}$ be a program and let G be a goal. There is an R-refutation from $\mathcal{P} \cup \{G\}$ with computed answer substitution θ iff there is an SLD-refutation from $\mathcal{P}^r \cup \{G\}$ with θ as the computed answer substitution.

Δ

6.5. Least Model Theorems for * and r

We prove, in this section, the Least Model Theorem for the program transformations * and r. Let $\mathcal{P}$ be a program and let L be an arbitrary finite extension of L_p. First, we will show that

$$M_L(\mathcal{P} \cup \mathcal{E}) \supseteq M_L(\mathcal{P}^*) \supseteq M_L(\mathcal{P}^r).$$

Then we complete the circle by showing that

$$M_L(\mathcal{P}^r) \supseteq M_L(\mathcal{P} \cup \mathcal{E}).$$

It follows that for all finite extensions L of L_p

$$M_L(\mathcal{P} \cup \mathcal{E}) = M_L(\mathcal{P}^*) = M_L(\mathcal{P}^r).$$

Lemma 6.3: Let $\mathcal{P}$ be a program. For all finite extensions L of L_p,

$$M_L(\mathcal{P} \cup \mathcal{E}) \supseteq M_L(\mathcal{P}^*) \supseteq M_L(\mathcal{P}^r).$$

Δ

Proof: It is straightforward to show that $\mathcal{P} \cup \mathcal{E} \models \mathcal{P}^* \models \mathcal{P}^r$. The result then follows from Proposition 1.1.

Q.E.D.

The crucial steps in the proof of $M(P^r) \supseteq M(P \cup \mathcal{E})$ consist in showing how the symmetry, transitivity, and predicate substitutivity axioms for equality are subsumed in P^r. The subsumption of the symmetry and transitivity axioms is based on the notions of *E-chain* and *E-link*.

Definition 6.10: (E-link)
A pair of ground terms $<s,t>$ is called an *E-link* iff there is a clause

$u = v \leftarrow A_1, \ldots, A_m$

in P and a substitution θ such that $A_1\theta, \ldots, A_m\theta \in T_{P^r} \uparrow \omega$ and either

1. $s \equiv u\theta$ and $t \equiv v\theta$,

or

2. $s \equiv v\theta$ and $t \equiv u\theta$.

△

Definition 6.11: (E-chain)
A sequence

S: $r_0, r_1, \ldots, r_l$

of ground terms is called an *E-chain* for the pair of terms $<r_0, r_l>$ iff $l \geq 0$ and for each i, $1 \leq i \leq l$, $<r_{i-1}, r_i>$ is an E-link. We say that the length of S is l. (Note that any ground term standing alone is an E-chain of length 0.)

△

The following propositions regarding properties of E-links and E-chains follow directly from Definitions 6.10 and 6.11 and do not depend on any property of P or P^r.

Proposition 6.1: (Symmetry, E-link)
Let s and t be ground terms. $<s,t>$ is an E-link iff $<t,s>$ is an E-link.

△

Proposition 6.2: (Concatenation, E-chain)
Let

$r_0, r_1, \ldots, r_l$

$u_0 (\equiv r_l), u_1, \ldots, u_m$

be E-chains. Then the concatenation

$r_0, r_1, \ldots, r_l, u_1, \ldots, u_m$

is also an E-chain.

△

Proposition 6.3: If the sequence

$r_0, r_1, \ldots, r_l$

is an E-chain, then the reverse sequence

$r_l, r_{l-1}, \ldots, r_0$
is also an E-chain.

Δ

Proof: The result follows from Proposition 6.1.

Q.E.D.

The following lemmas relate E-links, E-chains, and $\mathcal{M}(\mathcal{P}^r)$; recall that $\mathcal{M}(\mathcal{P}^r)$ is identical to $T_{\mathcal{P}^r}\uparrow\omega$.

Lemma 6.4: If a pair of ground terms $<s,t>$ is an E-link, then $s=t, t=s \in T_{\mathcal{P}^r}\uparrow\omega$.

Δ

Proof: By the definition of E-link, there is a clause

C: $\quad u=v \leftarrow A_1, \ldots, A_m$

in $\mathcal{P}$ and a substitution θ such that $A_1\theta, \ldots, A_m\theta \in T_{\mathcal{P}^r}\uparrow\omega$ and either

1. $s \equiv u\theta$ and $t \equiv v\theta$.

or

2. $s \equiv v\theta$ and $t \equiv u\theta$.

By the definition of $\mathcal{P}^r$,

C': $\quad u=x \leftarrow v=x, A_1, \ldots, A_m$.

and

C'': $\quad v=x \leftarrow u=x, A_1, \ldots, A_m$

are in $\mathcal{P}^r$.

Case 1: $s \equiv u\theta$ and $t \equiv v\theta$.

$s=t \leftarrow t=t, A_1\theta, \ldots A_m\theta$

is a ground instance of C'. Since $x=x$ is in $\mathcal{P}^r$, $t=t \in T_{\mathcal{P}^r}\uparrow 1 \subseteq T_{\mathcal{P}^r}\uparrow\omega$. Hence $s=t \in T_{\mathcal{P}^r}\uparrow\omega$.

The clause

$t=s \leftarrow s=s, A_1\theta, \ldots A_m\theta$

is a ground instance of C''. Hence $t=s \in T_{\mathcal{P}^r}\uparrow\omega$.

Case 2 may be proved in a similar way.

Q.E.D.

Lemma 6.5: (E-chain)
Let s and t be ground terms. If $s=t \in T_{\mathcal{P}^r}\uparrow\omega$, then there are E-chains for the pairs $<s,t>$ and $<t,s>$.

Δ

Proof: $s=t \in T_{\mathcal{P}^r}\uparrow\omega$ iff there is a k such that $s=t \in T_{\mathcal{P}^r}\uparrow k$. We prove the result by induction on k.

Base case: $k=0$.

$T_{P^r} \uparrow 0 = \phi$. The result is trivially true.

Inductive step.
Assume that the result holds for k. Consider $s{=}t \in T_{P^r} \uparrow k{+}1$. There are two cases:

1. $s{=}t$ is an instance of $x{=}x$. Then $s \equiv t$, and

$$s$$

is an E-chain of length 0 for $<s,t>$.

2. There is a clause

C: $u{=}x \leftarrow v{=}x, A_1, \ldots, A_m$

in P^r and there is a substitution θ such that $s \equiv u\theta$, $t \equiv x\theta$ and $v\theta{=}t, A_1\theta, \ldots, A_m\theta \in T_{P^r} \uparrow k$. By the inductive hypothesis, there is an E-chain

$$r_0(\equiv v\theta), r_1, \ldots, r_l(\equiv t)$$

for $<v\theta,t>$. By the definition of P^r either

C': $u{=}v \leftarrow A_1, \ldots, A_m$,

or

C": $v{=}u \leftarrow A_1, \ldots, A_m$

is in *P*. Hence $<u\theta,v\theta>$ is an E-link and

$$s(\equiv u\theta), r_0(\equiv v\theta), \ldots, r_l(\equiv t)$$

is an E-chain for $<s,t>$. By Proposition 6.3

$$r_l(\equiv t), \ldots, r_0(\equiv v\theta), s(\equiv u\theta)$$

is an E-chain for $<t,s>$.

Q.E.D.

The following lemma (which is the converse of the E-chain Lemma) is the crucial lemma on which our proofs of the Least Model Theorems are based. Symmetry and transitivity of equality follow immediately from this lemma and the E-chain Lemma.

Lemma 6.6: Let s and t be ground terms. If there is an E-chain for $<s,t>$, then $s{=}t$, $t{=}s \in T_{P^r} \uparrow \omega$.

△

Proof: (By induction on the length of the E-chains.)

Base case: $l{=}0$.
Then $s \equiv t$. $s{=}t$ and $t{=}s$ are instances of $x{=}x$.

Hence $s=t,\ t=s \in T_{\mathcal{P}^r}\uparrow 1 \subseteq T_{\mathcal{P}^r}\uparrow\omega$.

Inductive step.

Assume that the result holds for E-chains of length l. Let

$$r_0(\equiv s),\ r_1,\ \ldots,\ r_{l+1}(\equiv t)$$

be an E-chain for $<s,t>$. Then the subsequence

$$r_1,\ \ldots,\ r_{l+1}(\equiv t)$$

is an E-chain of length l. By the inductive hypothesis, $r_1=t \in T_{\mathcal{P}^r}\uparrow\omega$. Consider the E-link $<s,r_1>$. By Lemma 6.4 $s=r_1,\ r_1=s \in T_{\mathcal{P}^r}\uparrow\omega$. By the definition of E-link, there is a clause

C: $\quad u=v \leftarrow A_1,\ldots,A_m$

in $\mathcal{P}$ and a substitution θ such that $A_1\theta,\ldots,A_m\theta \in T_{\mathcal{P}^r}\uparrow\omega$ and either

1. $s\equiv u\theta$ and $r_1\equiv v\theta$,

or

2. $s\equiv v\theta$ and $r_1\equiv u\theta$.

By the definition of $\mathcal{P}^r$

C': $\quad u=x \leftarrow v=x, A_1,\ldots,A_m$

and

C'': $\quad v=x \leftarrow u=x, A_1,\ldots,A_m$

are in $\mathcal{P}^r$.

Case 1. $s\equiv u\theta$ and $r_1\equiv v\theta$.

$$s=t \leftarrow r_1=t, A_1\theta,\ldots,A_m\theta$$

is a ground instance of C'. Hence $s=t \in T_{\mathcal{P}^r}\uparrow\omega$.

Case 2. $s\equiv v\theta$ and $r_1\equiv u\theta$.

$$s=t \leftarrow r_1=t, A_1\theta,\ldots,A_m\theta$$

is a ground instance of C''. Hence $s=t \in T_{\mathcal{P}^r}\uparrow\omega$. By Proposition 6.3 there is an E-chain for $<t,s>$. By the same argument, $t=s \in T_{\mathcal{P}^r}\uparrow\omega$.

Q.E.D.

Lemma 6.7: (Symmetry)

Let s and t be ground terms. If $s=t \in T_{\mathcal{P}^r}\uparrow\omega$, then $t=s \in T_{\mathcal{P}^r}\uparrow\omega$.

Δ

Proof: By the E-chain Lemma, there is an E-chain

$$r_0(\equiv s),\ r_1,\ \ldots,\ r_l(\equiv t)$$

for $<s,t>$. Hence $t=s \in T_{\mathcal{P}^r}\uparrow\omega$ by Lemma 6.6.

Q.E.D.

Lemma 6.8: (Transitivity)

Let s, t and u be ground terms. If $s=t, t=u \in T_{\mathcal{P}^r}\uparrow\omega$, then $s=u \in T_{\mathcal{P}^r}\uparrow\omega$.

Δ

Proof: If $s{=}t, t{=}u \in T_{\mathcal{P}^r}\uparrow\omega$, then by the E-chain Lemma, there are E-chains

$$r_0(\equiv s), r_1, \ldots, r_l(\equiv t)$$

$$v_0(\equiv t), v_1, \ldots, v_m(\equiv u)$$

for the pairs $<s,t>$ and $<t,u>$, respectively. By Proposition 6.2, the sequence

$$r_0(\equiv s), r_1, \ldots, r_l(\equiv t), v_1, \ldots, v_m(\equiv u)$$

is an E-chain for $<s,u>$. By Lemma 6.6, $s{=}u \in T_{\mathcal{P}^r}\uparrow\omega$.

Q.E.D.

Now we are ready to prove the major theorem of this section.

Notation: $\mathcal{P} \cup \mathcal{E}$ is abbreviated as $\mathcal{PE}$.

Δ

Theorem 6.5: $T_{\mathcal{PE}}\uparrow\omega \subseteq T_{\mathcal{P}^r}\uparrow\omega$.

Δ

Proof: A ground atom $A \in T_{\mathcal{PE}}\uparrow\omega$ iff there is a k such that $A \in T_{\mathcal{PE}}\uparrow k$. We prove the result by induction on k.

Base case: $k{=}0$.

$$T_{\mathcal{PE}}\uparrow 0 = \phi \subseteq T_{\mathcal{P}^r}\uparrow\omega$$

Inductive step.

Assume that the result holds for k. Consider a ground atom $A \in T_{\mathcal{PE}}\uparrow k{+}1$. There is a ground instance

$$A \leftarrow A_1, \ldots, A_q$$

of a clause C in $\mathcal{P} \cup \mathcal{E}$ such that $A_1, \ldots, A_q \in T_{\mathcal{PE}}\uparrow k$. Since $\mathcal{E}$ is $\{ref, sym, tran\} \cup Pred \cup Fun$, there are five cases: C is (1) *ref*, (2) *sym*, (3) *tran*, (4) a clause in *Pred*, or (5) a clause in $\mathcal{P} \cup Fun$.

(1) A is a ground instance of $x{=}x$. Since $x{=}x \in \mathcal{P}^r$

$$A \in T_{\mathcal{P}^r}\uparrow 1 \subseteq T_{\mathcal{P}^r}\uparrow\omega.$$

(2) A is $s{=}t$,

$$s{=}t \leftarrow t{=}s$$

is an instance of *sym*, and $t{=}s \in T_{\mathcal{PE}}\uparrow k$. By the inductive hypothesis, $t{=}s \in T_{\mathcal{P}^r}\uparrow\omega$. Hence $s{=}t \in T_{\mathcal{P}^r}\uparrow\omega$ by the Symmetry Lemma.

(3) A is $s{=}t$.

$$s{=}t \leftarrow s{=}u, u{=}t$$

is a ground instance of *tran*, and $s{=}u, u{=}t \in T_{\mathcal{PE}}\uparrow k$.

By the inductive hypothesis, $s{=}u, u{=}t \in T_{\mathcal{P}^r}\uparrow\omega$. Hence $s{=}t \in T_{\mathcal{P}^r}\uparrow\omega$ by the Transitivity Lemma.

(4) A is $P(t_1, \ldots, t_n)$.

$$P(t_1, \ldots, t_n) \leftarrow P(s_1, \ldots, s_n), s_1 = t_1, \ldots, s_n = t_n$$

is a ground instance of an axiom in *Pred*, and

$$P(s_1, \ldots, s_n), s_1 = t_1, \ldots, s_n = t_n \in T_{\mathcal{P}\mathcal{E}} \uparrow k.$$

By the inductive hypothesis, there is an $m < \omega$ such that

$$P(s_1, \ldots, s_n), s_1 = t_1, \ldots, s_n = t_n \in T_{\mathcal{P}^r} \uparrow m \subseteq T_{\mathcal{P}^r} \uparrow \omega.$$

Hence there is a ground instance

$$P(s_1, \ldots, s_n) \leftarrow r_1 = s_1, \ldots, r_n = s_n, B_1, \ldots, B_q$$

of a clause C' in $\mathcal{P}^r$ such that

$$r_1 = s_1, \ldots, r_n = s_n, B_1, \ldots, B_q \in T_{\mathcal{P}^r} \uparrow m-1.$$

For each i, $1 \leq i \leq n$, $r_i = s_i, s_i = t_i \in T_{\mathcal{P}^r} \uparrow m$. By the Transitivity Lemma $r_i = t_i \in T_{\mathcal{P}^r} \uparrow m_i$, for some $m_i < \omega$. Let p be the largest number in $\{m, m_1, \ldots, m_n\}$. Because those variables introduced by the right homogeneous form transformation are distinct new variables,

$$P(t_1, \ldots, t_n) \leftarrow r_1 = t_1, \ldots, r_n = t_n, B_1, \ldots, B_q$$

is a ground instance of C'. Since

$$r_1 = t_1, \ldots, r_n = t_n, B_1, \ldots, B_q \in T_{\mathcal{P}^r} \uparrow p,$$

$P(t_1, \ldots, t_n) \in T_{\mathcal{P}^r} \uparrow p+1 \subseteq T_{\mathcal{P}^r} \uparrow \omega$.

(5) A is $P(t_1, \ldots, t_n)$.

C'': $\quad P(s_1, \ldots, s_n) \leftarrow B_1, \ldots, B_q$

is a clause in $\mathcal{P}$, and there is a substitution θ such that $B_1\theta, \ldots, B_q\theta \in T_{\mathcal{P}\mathcal{E}} \uparrow k$ and for all i, $1 \leq i \leq n$, $t_i \equiv s_i\theta$. Hence

$$P(t_1, \ldots, t_n) \leftarrow s_1\theta = t_1, \ldots, s_n\theta = t_n, B_1\theta, \ldots, B_q\theta$$

is a ground instance of $h(C'')$ in $\mathcal{P}^r$. By the inductive hypothesis, $B_1\theta, \ldots, B_q\theta \in T_{\mathcal{P}^r} \uparrow m$ for some $m < \omega$. For all i, $1 \leq i \leq n$, $s_i\theta = t_i \in T_{\mathcal{P}^r} \uparrow 1$. Hence

$$s_1\theta = t_1, \ldots, s_n\theta = t_n, B_1\theta, \ldots, B_q\theta \in T_{\mathcal{P}^r} \uparrow m+1$$

and $P(t_1, \ldots, t_n) \in T_{\mathcal{P}^r} \uparrow m+2 \subseteq T_{\mathcal{P}^r} \uparrow \omega$.

Q.E.D.

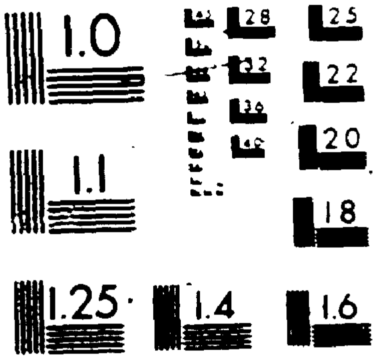
What follows is the Least Model Theorem for the transformation * and r.

Theorem 6.6: (Least Model)
Let $\mathcal{P}$ be a program. For all finite extensions L of $L_{\mathcal{P}}$

$$M_L(\mathcal{P} \cup \mathcal{E}) = M_L(\mathcal{P}^*) = M_L(\mathcal{P}^r).$$

Δ

2 OF/DE 2



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

Proof: It follows from Theorem 6.5 that $M_L(\mathcal{P} \cup \mathcal{E}) \subseteq M_L(\mathcal{P}^r)$. The result then follows from Lemma 6.3.

Q.E.D.

Theorem 6.7: (CAS-equivalence, $\mathcal{P}^*$ and $\mathcal{P}^r$)
Let $\mathcal{P}$ be a program. $\mathcal{P} \cup \mathcal{E} \cup \mathcal{P}^*$ and $\mathcal{P}^r$ are CAS-equivalent.

Δ

Proof: The theorem follows Theorem 6.6 and 2.2.

Q.E.D.

It follows from the Compilation Theorems, CAS-equivalence Theorems and Theorem 3.3 that both SLDEU-resolution and right recursive SLDEU-resolution are sound and complete.

Theorem 6.8: (Soundness and completeness)
For all programs $\mathcal{P}$ and all goals G,

1. every answer substitution computed by SLDEU-resolution or right recursive SLDEU-resolution is $\mathcal{E}$-correct (soundness);
2. every $\mathcal{E}$-correct answer substitution is subsumed by an answer substitution computed by SLDEU-resolution and right recursive SLDEU-resolution (completeness).

Δ

6.6. Conclusion

We have established in this chapter the soundness and completeness of SLDEU-resolution and right recursive SLDEU-resolution by reducing their semantics to the semantics of SLD-resolution.

Chapter Seven
Non-Repetitive Right Recursive SLDEU-Resolution

7.1. Repetitive Right Recursive SLDEU-Derivations

Rule 5.1 (p. 55) is not strong enough to eliminate all infinite loops. Consider the following example:

Example 7.1:

P: $a=b$
$c=d$

G. $\leftarrow a=d$

△

The sequence of goals generated by right recursive SLDEU-resolution are listed as follows:

$\leftarrow a=a$ (*succeeds*). $\leftarrow b=d$.
$\leftarrow b=b$ (*succeeds*). $\leftarrow a=d$.
$\leftarrow a=a$ (*succeeds*). $\leftarrow b=d$.
. . .

This is an infinite loop. The goal $\leftarrow b=d$ is generated recursively *ad infinitum*. This is an example of repetitive derivation.

In this chapter we restrict right recursive SLDEU-resolution further to non-repetitive right recursive SLDEU-resolution. This is accomplished by incorporating a mechanism that detects and fails repetitive derivations generated by E-unification.

7.2. Non-Repetitive Right Recursive SLDEU-Refutation

We record the history of an equality subgoal by a list of terms. These lists are called the *history* of the equality subgoals.

Definition 7.1: (History)
The history of equality subgoals in an R-derivation is defined as follows:

1. Equality subgoals without history are assigned the empty list $[\,]$ as their history.

2. Let $s=t$ be an equality subgoal with history L. Let

   $C \qquad u=v \leftarrow B_1, \ldots, B_q$

   be an equality clause other than $x=x$.

   a. If u is unifiable with s with the mgu θ, E-unification of $u=v$ and $s=t$ generates the new goal

      $\leftarrow (v=t)\theta$

      with the history $[s \mid L]\theta$.

   b. If v is unifiable with s with the mgu θ, E-unification of $u=v$ and $s=t$ generates the new goal

      $\leftarrow (u=t)\theta$

      with the history $[s \mid L]\theta$.

Δ

Definition 7.2: (Repetitive goal)
Let $\leftarrow s=t$ be an equality goal with history L. $\leftarrow s=t$ is called a *repetitive goal* iff s is syntactically identical to a member of L.

Δ

Definition 7.3: (Repetitive R-derivation)
A *repetitive R-derivation* is an R-derivation with repetitive equality goals.

Δ

Non-repetitive right recursive SLDEU-resolution is right recursive SLDEU-resolution restricted by the following rule:

Rule 7.1: (Non-repetitive right recursive SLDEU-resolution)
Fail all repetitive equality goals.[1]

Δ

Example 7.2:

P:	$a=b$	
	$b=c$	
	$c=d$	
G.	$\leftarrow a=d$	*history:* []

The E-chain for a and d is

[1] Rule 7.1 is motivated by similar rules used in [Elcock 86, Hoddinott 86b, Haridi 86]

$a, b, c, d.$

The sequence of new goals generated by right recursive extended unification is:

Goal	*History*
$\leftarrow b=d$	$[a]$
$\leftarrow c=d$	$[b, a]$
$\leftarrow d=d$	$[c, b, a]$

In example 7.1, the original goal is $\leftarrow a=d$ with $[\,]$ as its history, and the sequence of new goals generated by right recursive SLDEU-resolution is

	Goal	*History*
G1:	$\leftarrow b=d$	$[a]$
G2:	$\leftarrow a=d$	$[b, a]$

Since the left term a in goal $G2$ is a member of the history of $G2$, generation of $G2$ is forbidden according to non-repetitive right recursive SLDEU-resolution. $G1$ then finitely fails. Consequently the original goal finitely fails.

7.3. Completeness of Non-Repetitive Right Recursive SLDEU-Resolution

We establish the completeness of non-repetitive right recursive SLDEU-resolution via the completeness of non-repetitive SLD-resolution.

Lemma 7.1: Let P be a program and let G be a goal. If there is a repetitive R-refutation from $P \cup \{G\}$ of complexity m with substitution θ, then there is a repetitive SLD-refutation of length m from $P^r \cup \{G\}$ with $\theta \cup \eta$ as substitution. Δ

Proof: According to the definition of *history*, only equality goals generated by extended unification have non-empty lists as their histories. A repetitive R-refutation Λ has a sequence of recursively generated equality goals:

Goal	*Input Clause*	*History*
$\leftarrow s_0=t$		L
$\leftarrow (s_1=t)\theta_1$	$u_1=v_1 \leftarrow B_1$	$[s_0 \mid L]\theta_1$
$\leftarrow (s_2=t)\theta_1\theta_2$	$u_2=v_2 \leftarrow B_2$	$[s_1, s_0 \mid L]\theta_1\theta_2$
$\cdots$		
$\leftarrow (s_q=t)\theta_1 \cdots \theta_q$	$u_q=v_q \leftarrow B_q$	$[s_{q-1}, \ldots, s_0 \mid L]\theta_1 \cdots \theta_q$

where s_i, $i \geq 1$, is u_i (or v_i), θ_i is an mgu of s_{i-1} and v_i (or u_i), and $s_q\theta_1 \cdots \theta_q \equiv s_0\theta_1 \cdots \theta_q$. Let r_i be v_i if s_i is u_i and let r_i be u_i if s_i is v_i.

Consider the SLD-refutation S constructed in the proof of the Compilation Lemma for A. S has a corresponding sequence of goals:

Goal	*Ancestor List*
$\leftarrow s_0{=}t, A$	L'
$\leftarrow (s_1{=}t, B_1, A)\theta_1$	$[s_0{=}t \mid L']\theta_1$
$\leftarrow (s_2{=}t, B_2, B_1, A)\theta_1\theta_2$	$[s_1{=}t, s_0{=}t \mid L']\theta_1\theta_2$
. . .	
$\leftarrow (s_q{=}t, B_q, \ldots, B_1, A)\theta_1 \cdots \theta_q$	$[s_{q-1}{=}t, \ldots, s_0{=}t \mid L']\theta_1 \cdots \theta_q$

Since $s_q\theta_1 \cdots \theta_q \equiv s_0\theta_1 \cdots \theta_q$,

$$(s_q{=}t)\theta_1 \cdots \theta_q \equiv (s_0{=}t)\theta_1 \cdots \theta_q,$$

S is a repetitive SLD-refutation.

Q.E.D.

Theorem 7.1: Let P be a program and let G be a goal. If there is a repetitive right recursive SLDEU-refutation from $P \cup \{G\}$ of complexity m and substitution θ, then there is a non-repetitive right recursive SLDEU-refutation of complexity $l < m$ with substitution $\rho \geq \theta$.

Δ

Proof: Let A be a repetitive R-refutation from $P \cup \{G\}$ of complexity m with substitution θ. By Lemma 7.1 there is a repetitive SLD-refutation from $P^r \cup \{G\}$ of length m with substitution $\theta \cup \eta$. By Theorem 4.1 there is a non-repetitive SLD-refutation S' from $P^r \cup \{G\}$ of length $l < m$ with substitution $\theta' \cup \eta' \geq \theta \cup \eta$. By the Compilation Lemma, there is an R-refutation A' from $P \cup \{G\}$ of complexity $l' < m$ with substitution $\theta' \geq \theta$. A' is non-repetitive, otherwise S' is repetitive.

Q.E.D.

7.4. Implementation

Non-repetitive right recursive SLDEU-resolution may be implemented by the following variant of the right homogeneous form transformation:

Definition 7.4: (P^n)
Let $P = D \cup E$ be a program. Let *eq*, *mem* and *non−mem* be predicate symbols not occurring in P. Let h^n be a variant of the right homogeneous form transformation mapping a clause

C: $\quad u{=}v \leftarrow B_1, \ldots, B_q$

to

$$\begin{aligned} h^n(C):\ eq(u, x, L) \leftarrow\ & non{-}mem(v, [u \mid L]), \\ & eq(v, x, [u \mid L]), \\ & B_1, \ldots, B_q. \end{aligned}$$

The non-repetitive right homogeneous form of $\mathcal{P}$ is

$$\begin{aligned} \mathcal{P}^n = & \{eq(x, x, L)\} \\ & \cup \{h(C) \mid C \in D\} \\ & \cup \{h^n(C) \mid C \in S_{=}(E \cup Fun)\} \\ & \cup NONMEM \end{aligned}$$

where $NONMEM$ is the following program:

$$\begin{aligned} NONMEM:\quad & x{=}y \leftarrow eq(x, y, [\,]). \\ & \\ & non{-}mem(x, L) \leftarrow mem(x, L), !, fail.^{2} \\ & non{-}mem(x, L). \\ & \\ & mem(x, [H \mid T]) \leftarrow x \equiv H. \\ & mem(x, [H \mid T]) \leftarrow mem(x, T). \end{aligned}$$

The clause

$$x{=}y \leftarrow eq(x, y, [\,])$$

assigns the empty list $[\,]$ as the ancestor list of an equality subgoal without an ancestor list. The transformation h^n transforms an equality clause to its right homogeneous form adding the test *non−mem* which detects and fails repetitive derivations. Clauses in $NONMEM$ define the test *non−mem* in terms of *mem* which is identical to the standard definition of *member* except that syntactic identity $\equiv$ is used instead of unifiability.

Let us illustrate how non-repetitive right recursive SLDEU-resolution works by a few examples. Predicate substitutivity is subsumed by E-unification of non-equality subgoals, or in terms of program transformations, by the homogeneous form transformation of non-equality clauses:

Example 7.3:

$$\begin{aligned} \mathcal{P}:\quad & P(a) \\ & a{=}b \end{aligned}$$

[2] See [Clocksin 81] §4.3.2 for an explanation of the use of "!, *fail*" in Prolog.

$$\begin{array}{ll}
P^n. & P(x) \leftarrow a{=}x \\
 & eq(a,x,L) \leftarrow non{-}mem(b,[a\,|\,L]), eq(b,x,[a\,|\,L]) \\
 & eq(b,x,L) \leftarrow non{-}mem(a,[b\,|\,L]), eq(a,x,[a\,|\,L]) \\
 & x{=}y \leftarrow eq(x,y,[\,]) \\
 & eq(x,x,L) \\
 & \ldots \\
S\cdot & \leftarrow P(b). \\
 & \leftarrow a{=}b. \\
 & \leftarrow eq(a,b,[\,]). \\
 & \leftarrow non{-}mem(b,[a]), eq(b,b,[a]). \\
 & \ldots. \\
 & \leftarrow eq(b,b,[a]). \\
 & \Box\,.
\end{array}$$

Δ

Symmetry of equality is subsumed by the symmetry built into the R-unification of equality subgoals, or in terms of program transformations, by the symmetric extension transformation $S_{=}$.

Example 7.4:

$$\begin{array}{ll}
P: & a{=}b \\
P^n: & eq(a,x,L) \leftarrow non{-}mem(b,[a\,|\,L]), eq(b,x,[a\,|\,L]) \\
 & eq(b,x,L) \leftarrow non{-}mem(a,[b\,|\,L]), eq(a,x,[b\,|\,L]) \\
 & x{=}y \leftarrow eq(x,y,[\,]) \\
 & eq(x,x,L) \\
 & \ldots \\
S\cdot & \leftarrow b{=}a. \\
 & \leftarrow eq(b,a,[\,]). \\
 & \leftarrow non{-}mem(a,[b]), eq(a,a,[b]). \\
 & \ldots, \\
 & eq(a,a,[b]). \\
 & \Box\,.
\end{array}$$

Δ

Transitivity of equality is subsumed by the recursive generation of equality goals in the R-unification of equality subgoals, or in terms of program transformation, by the right homogeneous form transformations of equality clauses.

Example 7.5:

P: $a=b$
$b=c$
$c=d$

P^n: $eq(a, x, L) \leftarrow non-mem(b, [a \mid L]), eq(b, x, [a \mid L])$
$eq(b, x, L) \leftarrow non-mem(a, [b \mid L]), eq(a, x, [b \mid L])$
$eq(b, x, L) \leftarrow non-mem(c, [b \mid L]), eq(c, x, [b \mid L])$
$eq(c, x, L) \leftarrow non-mem(b, [c \mid L]), eq(b, x, [c \mid L])$
$eq(c, x, L) \leftarrow non-mem(d, [c \mid L]), eq(d, x, [c \mid L])$
$eq(d, x, L) \leftarrow non-mem(c, [d \mid L], eq(c, x, [d \mid L])$
. . .

S: $\leftarrow a=d.$
$\leftarrow eq(a, d, []).$
$\leftarrow non-mem(b, [a]), eq(b, d, [a]).$
. . . .
$\leftarrow eq(b, d, [a]).$
$\leftarrow non-mem(c, [b, a]), eq(c, d, [b, a]).$
. . . .
$\leftarrow eq(c, d, [b, a]).$
$\leftarrow non-mem(d, [c, b, a]), eq(d, d, [c, b, a]).$
. . . .
$\leftarrow eq(d, d, [c, b, a]).$
□

The test *non−mem* controls the search for refutations. Derivations with infinite loops generated by R-unification are detected and failed. There is no E-chain linking the terms a and d in the following example.

Example 7.6:

P: $a=b$
$c=d$

P^n: $eq(a, x, L) \leftarrow non-mem(b, [a \mid L]), eq(b, x, [a \mid L])$
$eq(b, x, L) \leftarrow non-mem(a, [b \mid L]), eq(a, x, [b \mid L])$
$eq(c, x, L) \leftarrow non-mem(d, [c \mid L]), eq(d, x, [c \mid L])$
$eq(d, x, L) \leftarrow non-mem(c, [d \mid L], eq(c, x, [d \mid L])$

S: $\leftarrow a = d.$

$\leftarrow eq(a, d, [\,]).$

$\leftarrow non{-}mem(b, [a]), eq(b, d, [a]).$

. . . .

$\leftarrow eq(b, d, [a]).$

$\leftarrow non{-}mem(a, [b, a]), eq(a, d, [b, a]).$

fails

7.5. Uncaught Repetitive Derivations

Non-repetitive right recursive SLDEU-resolution does not catch all repetitive derivations. This is because non-repetitive right recursive SLDEU-resolution has built-in tests only for repetitive equality subgoals generated by R-unification. Such tests are needed in order to stop infinite loops generated by R-unification. Without such tests, the system would run into infinite loops most of the time.

If the need arises, we may extend non-repetitive right recursive SLDEU-resolution by adding tests for other repetitive subgoals. However, since such tests are expensive to carry out, it is more desirable to minimize their use. On the other hand, unlike those infinite loops generated by R-unification, other repetitive derivations are caused by user supplied programs. They are thus under the control of the user.

Chapter Eight
Limitations and Future Directions

We conclude with a study of the limitations of non-repetitive right recursive SLDEU-resolution together with a brief discussion of interesting directions for further research.

8.1. Useful Infinite Derivations

Non-repetitive right recursive SLDEU-resolution does not have a built-in mechanism for stopping non-repetitive infinite derivations. And we have shown in Chapter 4 that generalizations of Rule 4.1 which prune non-repetitive infinite branches by failing u-repetitive, l-repetitive and quasi-repetitive subgoals, respectively, lead to incompleteness. There is a theoretical limit to the goal of pruning infinite search trees. Some infinite subtrees are useful or necessary: they are needed for the computation of all correct answer substitutions. Let us illustrate this point by the following example (for SLD-resolution).

Example 8.1:

P: $a=b$
$f(x)=f(y) \leftarrow x=y$
G: $\leftarrow x=y$

△

The SLD-tree for $P \cup \{G\}$ is shown in Figure 8-1 below. The peculiar thing about this SLD-tree is that the infinite branch is needed for the generation of all correct answer substitutions. Let us call a correct answer substitution θ a *maximal correct answer substitution* iff for all other correct answer substitutions ρ, $\rho \geq \theta$ only if $\theta \geq \rho$. There are infinitely many independent maximal correct answer substitutions for $P \cup \{G\}$:

$$\{a/x, b/x\}, \{f(a)/x, f(b)/y\}, \{f(f(a))/x, f(f(b))/y\}, \ldots$$

We need the infinite branch to generate all of them. We have the following general result.

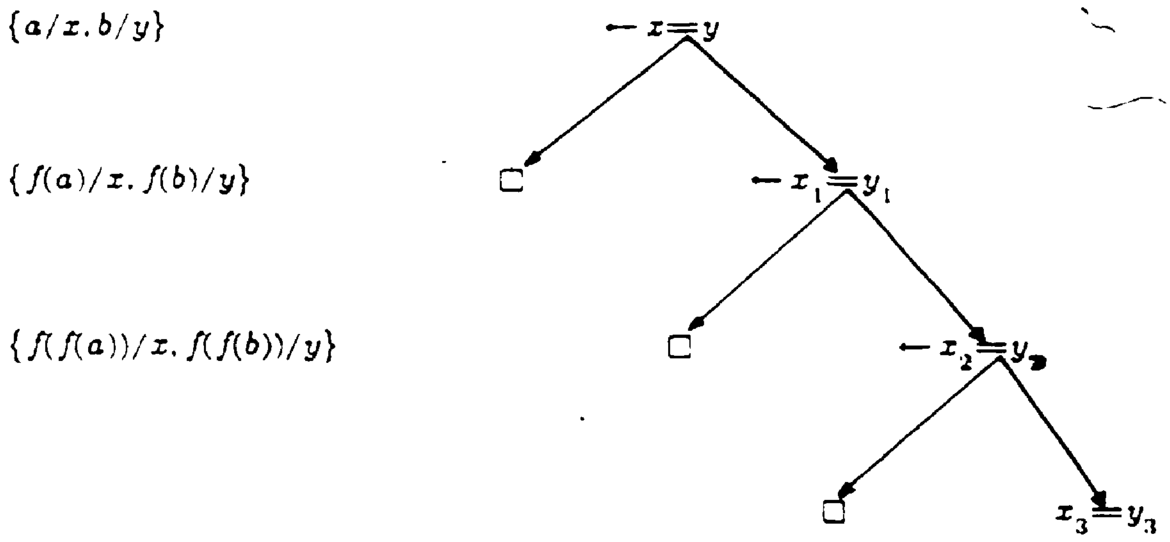


Figure 8-1: A Useful Infinite Branch

Definition 8.1: Let X be a logic programming system that computes one answer substitution per success branch. Let P be a program and let G be a goal. A subtree of a search tree for $P \cup \{G\}$ is said to be *complete* iff every correct answer substitution for $P \cup \{G\}$ is subsumed by the computed answer substitution of at least one success branch in the subtree.

△

Theorem 8.1:
Let X be a logic programming system that computes one answer substitution per success branch. Let P be a program and let G be a goal. If $P \cup \{G\}$ has infinitely many independent maximal correct answer substitutions, then all complete subtrees of the search trees of $P \cup \{G\}$ have at least one infinite fan or one infinite branch.

△

Proof: Each independent maximal correct answer substitution needs a separate success branch. Therefore a complete subtree has infinitely many nodes. If there is no infinite fan, then each node of a search tree has only finitely many arcs, the subtree has at least one infinite branch, otherwise the subtree is finite by König's Lemma [Loveland 78, p. 69].

Q.E.D.

Consequently, it is impossible to prune all infinite subtrees without sacrificing completeness. This result is not surprising in view of the undecidability of $\models$ in first order Horn clause languages.[1] Supplementing a program $\mathcal{P}$ by *Fun* causes non-repetitive infinite derivations. However, leaving out *Fun* causes incompleteness. In view of Theorem 8.1, it is impossible to eliminate such useful infinite branches without losing completeness.

8.2. Summary

We have presented (in Part I) three strategies for avoiding infinite computations in Horn clause logic programming systems. In *Chapter 4*, we introduced *non-repetitive SLD-resolution* as a general method for stopping repetitive computations in SLD-resolution. In a repetitive derivation, there are subgoals which are syntactically identical to the current instances of one of their ancestors. It was shown that pruning repetitive branches of SLD-trees is not strong enough to stop all infinite computations. Infinite derivations involving axioms such as the transitivity axiom are not repetitive. In *Chapter 2*, we introduced the notion of *CAS-equivalent programs* and presented a fixpoint criterion for CAS-equivalence. It was shown how transforming a program $\mathcal{P}$ into its symmetric extension $S(\mathcal{P})$ eliminates the need to supplement $\mathcal{P}$ by the symmetry axiom. $S(\mathcal{P})$ and $\mathcal{P}\cup\{sym\}$ are CAS-equivalent. In *Chapter 3* we studied the method of replacing axioms by inference rules, thereby eliminating the need to supplement programs by axioms which are a source of infinite loops. We also introduced *semantic reduction* as a method to establish the soundness and completeness of such extensions of SLD-resolution.

[1] In [Tarnlund 77] it is established that languages of first order Horn clauses are rich enough to describe the operations of Turing machines. It is shown in [Boolos 71] that $\models$ is an undecidable relation in languages rich enough to describe the operations of Turing machines.

In Part II we applied the strategy developed in Part I to the problem of avoiding infinite loops in logic programming with equality. We developed an extension of SLD-resolution called *non-repetitive right recursive SLDEU-resolution* which has symmetry, transitivity and predicate substitutivity of equality bulit-in by extended unification. Infinite loops caused by extended unification are detected by keeping ancestor lists for equality goals generated by extended unification. The soundness and completeness of non-repetitive right recursive SLDEU-resolution were established by semantic reduction. The program transformation n compiles a program $\mathcal{P}$ for non-repetitive right recursive SLDEU-resolution to an equivalent program $\mathcal{P}^{n}$ for SLD-resolution.

Nevertheless, infinite computations are still possible in non-repetitive right recursive SLDEU-resolution, and it is impossible to prune all infinite subtrees without losing completeness. This is due to the fact that some infinite (sub)trees contain infinitely many independent maximal correct answer substitutions, and we require an infinite fan or infinite branch to generate them all. Theorem 8.1 sets a theoretical limit to systems that aim at a complete implementation of equality: if any program and any query are permitted, no complete system can avoid infinite computations. One plausible direction is to limit the queries and/or the programs in some principled way. We need formal restrictions on queries and/or programs that rule out goals with infinitely many independent maximal correct answer substitutions. Another plausible direction is to allow conditional answer substitutions.

8.3. Future Directions

Restricting Correct Answer Substitutions

A cause of infinite computation is that there may be infinitely many terms all referring to the same individual. For example, the terms

$$1,\ 2-1,\ 0+1,\ 1\times 1,\ 3\times 4-11,\ 2-3+2,\ \ldots$$

all denote the number *one*. Consequently there are infinitely many correct answer substitutions for the goal $\leftarrow 1=x$. We may distinguish the term 1 as the canonical form of these co-referring terms, and require the system to compute only correct answers in canonical forms.

Confluent and Noetherian term rewriting systems such as van Emden and Yukawa's [86] contribute toward this end. However, what has been achieved is only a simulation of functional programming in a logic programming system: the system reduces ground terms to their normal forms. Such systems are too restricted. Very often intuitively appealing equality axioms are not canonical term rewriting systems. For example, the axiom

$$rat(N_1, D_1) = rat(N_2, D_2) \leftarrow D_1 \times N_2 = D_2 \times N_1$$

for the equality of rational numbers is not a canonical term rewriting system. It is desirable for the user to be allowed to specify intuitive axioms and to specify the canonical form of the terms involved. Secondly, such systems do not integrate logic programming and functional programming. Instantiating variables in goals is a distinctive feature of logic programming. However, such systems are incomplete for goals with uninstantiated variables.

Restricting Queries and/or Programs

A system integrating logic programming and functional programming should accept goals with uninstantiated variables because it is typical for logic programs to have variables in the body of a clause that are new to the head of the clause.

However, unrestricted queries and/or programs lead to incompleteness. What we need are formal restrictions on queries and programs which lead to complete and practically useful systems.

Conditional Answer Substitutions

Another possibility is to extend the notion of correct answer substitution to allow *conditional answer substitutions*.

Example 8.2:

P $\quad a=b$

$\quad f(x)=f(y)\leftarrow x=y$

G. $\quad \leftarrow x_1=y_1$

The set of $\mathcal{E}$-correct answer substitutions are finitely described by

$\{a/x_1, b/y_1\}, \{b/x_1, a/y_1\}, \{x_1/x_1, x_1/y_1\}$,

and

$\{f(x_2)/x_1, f(y_2)/y_1\}$ where $x_2=y_2$.

Kornfeld's work [83] on *partially specified objects* is a contribution along this direction.

References

[Apt 82] Apt, K. R. and van Emden, M. H.
"Contributions to the Theory of Logic programming".
Journal of the Association for Computing Machinery
29(3):841-862, July, 1982.

[Boolos 74] Boolos, G. S. and Jeffrey, R. C.
Computability and Logic
Cambridge University Press, Cambridge, 1974.

[Chan 86a] Chan, K. H.
"Equivalent Logic Programs and Symmetric Homogeneous Forms of Logic Programs with Equality"
1986.

[Chan 86b] Chan, K. H.
"Selective SLDNF-Resolution: A Logic Programming System for Representing Negative and Incomplete Information".
1986.

[Clark 78] Clark, Keith L.
"Negation as Failure".
Logic and DataBases
Plenum, 1978, pages 293-322.

[Clark 79] Clark, Keith L.
Predicate Logic as a Computational Formalism.
Technical Report, Department of Computing, Imperial College, London, 1979.

[Clocksin 81] Clocksin, W. F. and Mellish, C. S.
Programming in Prolog.
Springer-Verlag, New York, 1981.

[Cox 85] Cox, P. T. and Pietrzykowski, T.
"Incorporating Equality into Logic Programming via Surface Deduction".
Annals of Pure and Applied Logic , 1985.
To Appear.

[Demopoulos 86] Demopoulos, W.
"The Homogeneous Form of Logic Programs with Equality".
1986.

[Elcock 81] Elcock, E. W.
Logic and Programming Methodology.
Technical Report 80, Department of Computer Science, The University of Western Ontario, August, 1981.

[Elcock 86] Elcock, E. W. and P. Hoddinott.
"Classical Equality and Prolog".
In *Proceedings of the Canadian Artificial Intelligence Conference.* 1986.

[Haridi 86] Haridi, S., P B. Sheldan, and E. P. Stabler (Jr.)
"An Implementation of Equality Theorem Prover in Prolog"
1986.

[Hoddinott 86a] Hoddinott, P. and Elcock, E. W.
"Prolog: Subsumption of Equality Axioms by the Homogeneous Form".
In . IEEE, 1986.
Symposium on Logic Programming - 86.

[Hoddinott 86b] Hoddinott, P.
"Logic Programming and Equality".
Master's thesis, The University of Western Ontario, 1986.

[Hogger 84] Hogger, C. J.
Introduction to Logic Programming.
Academic Press, London, 1984.

[Jaffar 84] Jaffar, J., Lassez, J., and Maher, M. J.
"A Theory of Complete Logic Programs with Equality".
The Journal of Logic Programming 1(3):211-224, October, 1984.

[Kornfeld 83] Kornfeld, W. A.
"Equality for Prolog".
In *Proceedings, Seventh International Joint Conference on Artificial Intelligence*, pages 514-519. , 1983.

[Kowalski 71] Kowalski, R. A. and Kuehner, D.
"Linear Resolution with Selection Function".
Artificial Intelligence 2:227-260, 1971.

[Lloyd 84] Lloyd, J. W.
Foundations of Logic Programming.
Springer-Verlag, New York, 1984.

[Loveland 78] Loveland, D. W.
Automated Theorem Proving. A Logical Basis
North-Holland, New York, 1978.

[Plotkin 72] Plotkin, G.
"Building-in Equational Theories".
Machine Intelligence 7
John Wiley and Sons, Ltd., 1972, pages 73-90.

[Tarnlund 77] Tärnlund, S. A.
"Horn Clause Computability".
Bit 17:215-226, 1977.

[van Emden 76] van Emden, M. H. and Kowalski, R. A.
"The Semantics of Predicate Logic as a Programming Language".
Journal of the Association for Computing Machinery 23(4):733-742, October, 1976.

[van Emden 77] van Emden, M. H.
"Programming with Resolution Logic".
Machine Intelligence 8.
Halsted Press, 1977, pages 266-299.

[van Emden 84] van Emden, M. H. and Lloyd, J. W.
"A Logical Reconstruction of Prolog II".
The Journal of Logic Programming 1(2):143-150, August, 1984.

[van Emden 86] van Emden, M. H. and K. Yukawa.
Equational Logic Programming.
Technical Report, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1986.