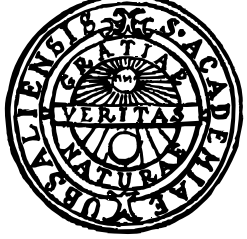
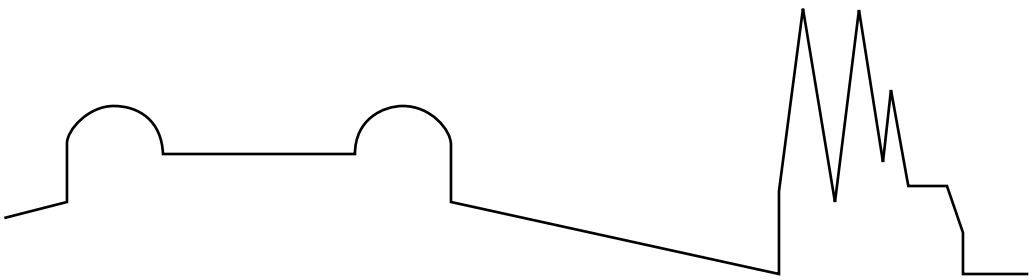


SLD-Resolution with Reflection

Pierangelo Dell'Acqua



Thesis for the Degree of
Licentiate of Philosophy

ISSN 0283-359X

UPMAIL
Computing Science Department
Uppsala University
Box 311
S-751 05 UPPSALA
Sweden

UPPSALA THESES IN COMPUTING SCIENCE
20

SLD-Resolution with Reflection

by
Pierangelo Dell'Acqua

UPMAIL
Computing Science Department
Uppsala University

Thesis for the Degree of
Licentiate of Philosophy

UPPSALA 1995

Abstract

The aim of this thesis is to define a procedural and declarative semantics of SLD-resolution with reflection, in the general framework defined by Jaffar et al. This scheme is in fact general enough to constitute a ground work suitable for (*i*) the possible integration of metaprogramming and constraint logic programming, so as to make more flexible and powerful logic languages possible; (*ii*) a discussion and comparison of various forms of encodings with respect to their impact on the semantic behaviour of extended resolution. To achieve our aim, we employ to a language containing names of expressions and an extension to SLD-resolution which allows metalevel computation and interlevel communication through reflection. We present a rewrite system for extended unification for the language, with certain similarities to a constraint solving system over names. Then, we discuss the declarative semantics of the language. In relation to the chosen naming policy and rewrite system for the unification, the extended SLD-resolution is shown to be sound and complete, provided that some requirements are fulfilled. Finally, comparisons are made with related languages and systems.

Keywords: Logic Programming, Metaprogramming, Reflection Principles, Naming Relations, Rewrite Systems

Acknowledgements

Many people have helped me during my work with this thesis. In particular I wish to thank Dr. Jonas Barklund, Dr. Henning Christiansen, Dr. Stefania Costantini, Dr. Patricia M. Hill and Professor Gaetano Aurelio Lanzarone for insightful comments on this work. Moreover, I would like to express my gratitude to Professor Alberto Pettorossi for his advice and for posing crucial questions on naming relations. Many thanks to my colleagues at the Computing Science Department in Uppsala for useful commenting on earlier versions of this thesis. In particular, I am indebted to Henrik Arro, Katrin Boberg, Dr. Anatoli Degtyarev, Dr. Andreas Hamfelt, Kent Saxin Hammarström, Margus Veanes and Dr. Andrei Voronkov for numerous helpful and pertinent comments. I also want to thank Marianne Ahrne for checking the English language of the thesis and for helping me with the Swedish language.

This research was supported financially first by Università degli Studi di Milano and later by the Swedish National Board for Technical and Industrial Development (NUTEK) under contract No. 92-10452 (ESPRIT BRP 6810: *Computational Logic 2*).

Finally, to my family and friends: grazie.

Contents

1 Introduction **1**
- 1.1 Metaprogramming in Logic . . . 1
- 1.2 Metalevel Architectures . . . 2
- 1.3 Naming Relations . . . 3
- 1.4 Reflection Principles . . . 4
- 1.5 An Overview and Outline of the Thesis . . . 5

2 An Abstract Metalanguage **8**
- 2.1 Alphabet . . . 8
- 2.2 The Language L . . . 9
- 2.3 An Equality Theory . . . 12

3 Unification **14**
- 3.1 Assignments, Herbrand Assignments and Substitutions . . . 14
- 3.2 A Unification Rewrite System . . . 15
- 3.3 Properties of the Unification Rewrite System . . . 17

4 Name Theories **19**
- 4.1 A Name Theory . . . 19
- 4.2 A Naming Rewrite System . . . 20
- 4.3 Properties of the Naming Rewrite System . . . 22
- 4.4 Formalizing Encodings in Equality Theories . . . 22

5 Combining Rewrite Systems for Unification and Naming **24**
- 5.1 The Rewrite System $R_{\mathbf{UN}}$ . . . 24
- 5.2 Properties of $R_{\mathbf{UN}}$ . . . 25

6 SLD*-Resolution **26**
- 6.1 The Language L* . . . 26
- 6.2 SLD*-Resolution . . . 27

7 Properties of SLD*-Resolution **30**
- 7.1 Declarative Semantics . . . 30
  - 7.1.1 Preliminaries . . . 30
  - 7.1.2 Extended E-Interpretations and Reflective E-Models . . . 32
  - 7.1.3 Fixpoint Semantics . . . 33
- 7.2 Soundness and Completeness of SLD*-Resolution . . . 34
  - 7.2.1 Soundness . . . 35
  - 7.2.2 Completeness . . . 35

8 Related Work & Concluding Remarks **38**
8.1 Related Work 38
8.2 Discussion and Concluding Remarks 40

A Proofs **47**
A.1 Proofs of Chapter 3 47
A.2 Proofs of Chapter 4 49
A.3 Proofs of Chapter 5 52
A.4 Proofs of Chapter 7 53

Chapter 1

Introduction

We introduce the basic concepts and terminology needed to read the thesis. We motivate the choices that have been made and outline the overall structure of the thesis.

1.1 Metaprogramming in Logic

Logic Programming is a kind of programming based on formal logic. It is born from the idea that logic can be used not only for expressing problems, but also for solving them. In fact, logic has traditionally been focussed upon the problem of determining whether a given conclusion (or theorem) C is logically implied by a set of assumptions (or theory) T. Vice versa, in the context of logic programming we are rather interested in computing solutions. Given a logic program (corresponding in logical terms to a theory) P and a goal statement (corresponding to a theorem) G, we are interested in determining a solution during the proof of G from P, where a solution is a set of bindings for the variables appearing in G. This solution is such to let G be logically implied by P.

Logic programming is traditionally based on a subset of first order predicate calculus, i.e., Horn clause logic. This restriction, which leads to a loss of expressive power, has two advantages. It allows the use of efficient inference rules, for example, SLD-resolution. Furthermore, from the theoretical point of view, logical consequences can be determined by examining a single model, called the least Herbrand model. On the other hand, Horn clause logic has many limitations, and to overcome them a number of both 'non-logical' and 'meta-logical' features have been added to logic programming languages. The first ones are called 'non-logical' because they are dependent on the procedural behaviour of the program and, although important from a practical point of view, they violate the declarative semantics of logic. The second ones allow us to access and manipulate linguistic constructs, such as programs, goals and proofs, viewed as data structures. Meta-logical features are important in many types of programs. In fact, many applications of logic, such as applications that formalize proof procedures (metainterpreters) and knowledge assimilators, are metalogic. This has led to the study and development of powerful metaprogramming techniques that allow us to extend and modify the semantics of an existing object language.

Metaprogramming has been so far employed in many applications including debuggers, compilers and program transformers [34, 59, 61]. Furthermore, also applications which can be formalized using other logics, such as modal logic of knowledge and belief [22, 45], or applications for theory construction [14] are objects of metaprogramming. A kind of architecture particularly suitable for metaprogramming is presented in the next section.

1.2 Metalevel Architectures

The advantages derived from the ability to explicitly express and use metalevel knowledge have been widely recognized, especially in the AI literature. Aiello et al. [1], for example, argues that metaknowledge and metalevel reasoning is suitable for devising proof strategies in automated deduction systems, for controlling the inference in problem solving, and for increasing the expressive power of knowledge representation languages.

The need to formally represent knowledge and metaknowledge has led to the study of metalevel architectures [2], where these two kinds of knowledge are explicitly represented. The basic features of a metalevel architecture can be summarized in:

1. the *naming relation*, providing names at the metalevel for the expressions of the object level theory;
2. the metalevel formalization of the object level properties;
3. the *linking rules*, that establish the connection between object level and metalevel in the inference process.

As the metalanguage is used to formalize properties of the object level theory, predicates in the metalanguage must take some representations of expressions of the object level theory as arguments. This is achieved by establishing a relationship, called the naming relation, between metalevel symbols and object level expressions.

At the metalevel, predicates can be used to define notions such as those of truth and provability. The notion of provability, for example, is typically formalized through a predicate $Demo(T', \alpha')$, that expresses the derivability of the object level formula α in the object level theory T, named α' and T' at the metalevel.

The distinguished metalevel predicates, expressing properties of the object level, are related to the object level itself by linking rules. The most widely proposed ones take the form of inference rules:

$$\frac{T \vdash_O \alpha}{Pr \vdash_M Demo(T', \alpha')} \quad \text{and} \quad \frac{Pr \vdash_M Demo(T', \alpha')}{T \vdash_O \alpha}$$

where Pr is a metalevel description of provability at the object level. The first one, *object-to-meta reflection*[1], allows one to assert the provability of $Demo(T', \alpha')$ in the metatheory when α is provable in the object theory T. The second linking rule, *meta-to-object reflection*, allows one to assert the provability of α in the object level theory T whenever it is possible to prove $Demo(T', \alpha')$ in the metatheory.

The application of reflection in the inference process can be *explicit* or *implicit*. In explicit reflection, the application of the linking rules is specified in the theory, i.e., it is necessary to specify in advance where to change the level during deduction. Whereas, in implicit reflection, the linking rules are integrated in the procedural semantics of the metalevel architecture. Different architectures are then characterized by different conditions for reflection.

Metalevel architectures can be classified into two main categories: *amalgamated* and *separated*. We consider an architecture to be amalgamated when sentences of the object language and sentences of the metalevel are defined within a single theory. Therefore, mixed sentences combining object level and metalevel expressions are allowed. The main amalgamated approaches in the AI literature have been established by Bowen & Kowalski [13], Attardi & Simi [5] and Costantini & Lanzarone [26].

[1] Some authors call these reflection principles upward and downward reflection, respectively; other authors, instead, use upward reflection (resp., downward reflection) to indicate the procedural shift from the object level to the metalevel (resp., from the metalevel to the object level). Therefore, we introduce new names to avoid such ambiguities.

In separated architectures, the object theory and the metatheory are distinct theories. Therefore, it is not possible to write sentences mixing object level and metalevel expressions. The connection between the metatheory and the object theory is provided by the naming relation and the linking rules. Among the separated approaches we mention the FOL system by Weyhrauch [67], 'Log by Cervesato & Rossi [15] and the Gödel programming language by Lloyd et al. [37] (for a survey of these approaches see reference [3]).

On the one hand, amalgamated approaches may turn out to be inconsistent. Tarski, for example, showed that if the object level and the metalevel are combined in the same language, then it is possible to express paradoxes such as the liar paradox. (See reference [65] for a brief discussion of the most important results achieved by logicians in this area.) On the other hand, separated approaches have less expressive power than amalgamated ones. Consider, for example, the sentence "A person is innocent if he or she cannot be proved guilty" (this example is taken from reference [13]). This can be formulated as:

$$Innocent(x) \leftarrow Person(x), not\, Demo(Facts, Guilty(x)'), Relevant(Facts)$$

Its formalization combines a metalanguage condition with an object language condition and conclusion. Here *Facts* names the relevant facts and assumptions which can be used in the attempt to establish guilt. Notice that a person might be guilty but not provable guilty. In such a case the single level sentence

$$Innocent(x) \leftarrow Person(x), not\, Guilty(x)$$

would not lead to the conclusion that he or she is innocent, whereas the mixed object language and metalanguage sentence would.

Thus, separated architectures are limited to express reasoning with one level of nesting, while many AI applications (e.g., applications requiring the formalization of epistemic notions such as knowledge and belief) often require deeper levels of nesting (e.g., $Demo(Facts, Belief(John, Belief(\ldots)'))$). See, for example, the issues raised by Moore [54]).

1.3 Naming Relations

The notion of naming relation originates in works of logic. Tarski [62], for example, defined a naming relation as a mapping from expressions in the object language to variable free terms in the metalanguage. In this way, it is possible to quantify in the metatheory over object level expressions, while staying within first-order logic. He discussed two kinds of naming relations: *quotation-mark* names and *structural descriptive* names. The former category associates with a formula a "monolithic" term as its name (Gödel encoding is an example of this kind of naming). The latter category associates with a formula a structured ground term that reflects the structure of the sentence it names. For example, the metalogical programming language *Reflective Prolog* [24, 26] implements a structural descriptive naming relation. The *Gödel* programming language [37] has names as an abstract data type, but provides enough predicates for testing and constructing names, and for extracting their constituents, to also qualify as having a structural descriptive naming relation.

Although many AI systems with a metalevel architecture have implemented naming relations, the formal properties that they must satisfy have usually been assumed without motivations. Bowen & Kowalski [13], for instance, allow the naming relation to be non-functional and require it to be injective, while Hill & Lloyd [36] define a naming relation that is total, injective and functional.

So far, the only investigation of formal properties of naming relations is that of van Harmelen [64]. He argues that naming relations should be definable and meaningful. If the naming relation is a fixed part of a metalevel architecture, then it is not possible, in general, to find one that is optimal for all metalevel theories. In fact, the richness[2] of the naming relation determines not only the expressivity of the metatheory, but also its computational complexity.[3] Therefore, the naming relation should be adapted to the particular requirements of a given metatheory, and/or the application at hand. This motivates the choice, in our framework, to provide the encoding as a separately definable component. Furthermore, a naming relation should not only encode syntactic information, but it should also be used to encode pragmatic and semantic information, i.e., it should be meaningful (for some examples of meaningful naming relations see Section 3.4 in reference [64]).

This is in contrast with the approach discussed by Eshghi [31]. In fact, he argues that the purpose of the naming relation is to allow us to refer, in the metalanguage, to the relevant constructs of the object language. Thus, the naming relation must have the following two properties: (*i*) it must be unambiguous, i.e., if the names of two objects are the same, and they appear in the same context, those objects must be identical up to renaming of variables; (*ii*) it must be transparent, i.e., the name of an object must be as similar to the object as possible in such a way to make mixed sentences more readable, and to aid implementation.

If a naming relation is a definable instead of being a predefined component of a metalevel architecture, then it is necessary to know which formal properties it must satisfy, because these properties cannot be enforced by the system implementor, but must be fulfilled at definition time.

Van Harmelen [64] only requires injectivity. He argues that the proper semantics, as pointed out by [62], is that the object theory should be regarded as a partial model of the metatheory, in the sense that the denotations of names at the metalevel are the corresponding expressions at the object level, i.e., $[\![y]\!] = x$ whenever y is the name of x. This makes the naming relation the inverse relation of semantic denotation. Since semantic denotation is required to be functional, the naming relation must therefore be injective.

Furthermore, van Harmelen argues that certain other properties might even be undesirable. Non-total naming relations allow information hiding between object level and metalevel. Because not all expressions can be described at the metalevel, they are hidden from the metatheory. Non-functional naming relations are useful for defining meaningful naming relations that may employ more than one name of an expression of the object level, according to the extra information they want to encode.

1.4 Reflection Principles

The idea of reflection dates back to a work by Feferman [33]. He introduced the concept of reflection principles defined as "a description of a procedure for adding to any set of axioms A certain new axioms whose validity follow from the validity of the axioms A and which formally express within the language of A evident consequences of the assumption that all the theorems of A are valid".

In the context of logic programming, logical reflection is a principled way to further empower the representation and reasoning capabilities of logic programming systems [23]. Reflection principles take the role of axiom schemata of a particular form

[2] The term richness is here used with the meaning of quantity of syntactic information encoded.

[3] In order to have an effective and efficient implementation of *'Log*, Cervesato & Rossi introduce in this language [15] two different but related metalevel representations (i.e., *quotation-mark* and *structural descriptive* names) for each syntactic object of the language (for a further discussion see Section 8.1.)

that, once added to a given logic program (the basic theory, or the initial axioms), enlarge the set of consequences sanctioned by those initial axioms.

One advantage of this procedure is that it is generally easier to write a basic theory and then augment it with axiom schemata, than it is to write a corresponding large (or even infinite) set of axioms in the first place. A more natural representation may thus be attained and the resulting logic program is more compact and manageable.

A complete formalization of a concept of reflection that constitutes a simple way of understanding reflective programs, and a description of how reflection allows one to treat uniformly different application areas has been presented by Costantini et al. [23].

1.5 An Overview and Outline of the Thesis

This thesis presents an extension of SLD-resolution with naming of expressions and two-way reflection. The extension is actually more like a schema than a specific theory, because it admits many choices of naming strategies and also a variety of techniques for rewriting equalities involving names. In order to be general we have therefore made few assumptions about naming relations and rewriting. With a particular choice of naming relation and a rewrite system for names that is a straightforward extension of the usual unification we obtain a slightly modified variant of Reflective Prolog.

There is an on-going debate in the metalogic programming community on whether names of expressions should necessarily be ground or not [10, 36, 44, 47, 52]. In this proposal we sidestep the question, as the proposed inference system restricts reflection so only ground expressions are ever communicated between levels. Therefore only naming of ground expressions is involved. The absence of variable names, however, does not restrict the effective expressive power of a language whose metalevel has the main purpose of extending the provability of the object level. In fact, in this context, metalevel variables can be used instead of names of variables (how this is done will be shown by some examples in the following chapters). In contrast, the metalevel features of languages, for example Gödel [37], developed with the primary aim of making program analysis or transformation, strictly need some sort of variable name facility.

A further question is then whether it is sufficient to let ground expressions be their own names or not. We believe that it is desirable to distinguish between ground expressions and the ground expressions that name them. One argument for this standpoint is that we may want to apply operations to names of expressions that it would be (semantically) improper to apply to the expressions themselves. For example, we may want to extract a name of the principal symbol from a name of an expression. A variable ranging over the functions symbols as such would take us into second order logic, but a variable ranging over the names of function symbols could still be a first-order entity. (For a discussion and comparison of various forms of encodings with respect to their semantic impact, see Section 8.2.) An argument against the distinction could be efficiency in implementation, but it is still an open question what the performance cost is for distinguishing between expressions and their names, if there is one at all.

The decision to only pass representations of ground expressions between levels might seem like a severe restriction. In general it is not, if we assume that names are compositional. In fact, interlevel communication of a partially instantiated expression allows the representations of the instantiated parts to be communicated while postponing communication of the uninstantiated parts. The only limitation is that it is impossible to pass information about the instantiation of an object level goal to the metalevel, which might be useful in some cases. This last possibility, however, could

destroy the independence of the selection rule enjoyed by SLD-resolution.[4]

The inference system that we propose is amalgamated in the sense that statements and metastatements about them are all in a single theory. However, the inference mechanism could be used without significant changes also for a system with separated theories (such as *Alloy* [6, 12]), if they all share the same language. (However, we have not established the semantics and properties of such a system to the same extent as we have for the amalgamated system.) It would be interesting to see if the present inference system could be generalized to be independent of this design decision as well. Moreover, for the sake of simplicity, we present an inference system that is structured into two levels: the object level and the metalevel. However, a possible generalization of this framework to a metalevel architecture with more than two layers is straightforward. The framework that we propose can also be easily extended in order to employ different kinds of naming. This allows us to obtain a multidimensional naming system where it is possible to select a naming that is the most appropriate, for example, with respect to the functionalities of the metalevel, or to efficiency considerations. (This is along the ideas of the metalogic language 'Log, where two different forms of encoding are employed.)

The extension to the inference system is similar to a constraint solving system that avoids search by delaying the computation of which values satisfy some constraint until enough variables have been instantiated to make the computation efficient. Because of the decision to compute names of ground expressions only, the computation of a name of a nonground expression is preferably delayed until some other part of the computation grounds these variables. That is the responsibility of the extended unification rewrite system, which is to be given as a parameter of the inference system.

The thesis is organized as follows. In Chapter 2 we first introduce a language L containing names of ground expressions that allows significant freedom in the choice of names: we only require that names of compound expressions are compositional. We mainly intend a language for multilevel knowledge representation and reasoning rather than for program manipulation. Therefore, for the sake of simplicity, we do not explicitly consider facilities for manipulating program fragments. Essentially then, what is syntactically needed is the possibility of representing terms and atomic sentences of the language in the language itself. We end the chapter by showing an equality theory for L, called ET.

In Chapter 3 we present a unification algorithm for L expressed in terms of a rewrite system R_U on equations. R_U is shown to be correct and complete with respect to ET.

A further extension concerning the introduction of naming consists in the possibility of relating names to what is named. Thus, in Chapter 4, we consider some examples of naming policies, in particular, a name theory NT with the corresponding rewrite system R_N. A name theory allows us to define an encoding whose interpretation defines a naming relation. We discuss some properties of R_N and show how Reflective Prolog can be seen as an instance of L.

In Chapter 5 we show how it is possible to combine the rewrite systems R_U and R_N in order to model unification extended to handle compositional names. We call the obtained rewrite system R_{UN}.

In Chapter 6 we define an inference system, SLD*-resolution, parameterized with a rewrite system R_E, that extends the usual SLD-resolution to allow metalevel computation and interlevel communication through reflection.

After having defined the declarative and fixpoint semantics, in Chapter 7 we establish results of soundness and completeness of SLD*-resolution. The declarative semantics is an extension of that proposed by Jaffar et al. [42] covering also the

[4]This is the difference with respect to Reflective Prolog mentioned above. In this regard, see Example 7.33.

metalevel part of the language. The extension is based on the work of Costantini & Lanzarone [26]. As a first immediate application, we obtain a generalization of Reflective Prolog, as the interaction between the levels does not require all the information to be available during the reflective step. We also discuss the requirements that a rewrite system R_E must fulfil for the given results to hold. As R_{UN} satisfies these requirements, SLD*-resolution is therefore sound and complete when parametrized with respect to R_{UN}.

Finally, comparisons with related work and some concluding remarks are presented in Chapter 8, while the proofs of theorems and lemmas of the thesis are reported in the Appendix.

Two parts of the thesis, in particular the naming rewrite system and SLD*-resolution, are joint works with Barklund, Costantini and Lanzarone [8, 9, 11]. The main contribution of the thesis concerns the introduction of name theories, the integration of the rewrite systems for unification and naming, and the formal properties of SLD*-resolution.

In the rest of the thesis, for logic programming we use the terminology of Lloyd [49] and for rewrite systems we follow that of Dershowitz & Jouannaud [29].

Chapter 2

An Abstract Metalanguage

In this section we introduce the basic syntactic features of a metalogic definite clause language, called in the following L. The language is that of definite programs, as defined by Lloyd [49], except that terms are defined differently in order to include *names* that are intended to represent the expressions of the language itself. In order to allow the results presented below to be as general as possible we use an abstract syntax for terms, which could be concretized in various ways.

2.1 Alphabet

The difference between the alphabet of L and the usual alphabet of definite clause programs lies in a distinction between various types of variables and in the presence of various metaconstants and operators.

The reason for the introduction of sorts of variables is the presence of metaconstants in the alphabet (and in general of expression names in the language) over which these variables range. However, the language we are going to introduce is not fully typed. Since our interest here is to be as general as possible, we only introduce the types strictly needed. (Note that extensions of the language to include a more precise set of types are possible, for example, along the lines of the programming language Gödel [37].) The reason for introducing types will be clear in Chapter 6 where some distinguished predicates (that for semantical reasons are required to be typed) are introduced.

Definition 2.1 The *alphabet* of the language L is the union of the following (disjoint) sets.

- A (non-empty) set of *predicate symbols* (Ps).
- A (possibly empty) set of *function symbols* (Fs).
- A (non-empty) set of *constant symbols* (C).
- A (countably infinite) set of *metaconstants*, which is the union of the following (disjoint) sets:
  - the (countably infinite) set of *constant names* (Cn) corresponding to the m-times named constants occurring in C (i.e., for each constant $c \in C$, the metaconstant $c^m \in Cn$ for every $m > 0$);
  - the (finite) set of *predicate names* (Pn) corresponding to the names of the predicate symbols occurring in Ps (i.e., for each predicate symbol $p \in Ps$, the metaconstant $p^1 \in Pn$);

  - the (finite) set of *function names* (Fn) corresponding to the names of the function symbols occurring in *Fs* (i.e., for each function symbol $f \in Fs$, the metaconstant $f^1 \in Fn$);
  - the (countably infinite) set of *functor names* (Fun) corresponding to m-times named predicate names and function names (i.e., for each function name $f^1 \in Fn$, the metaconstant $f^m \in Fun$ for every $m > 1$, and for each predicate name $p^1 \in Pn$, the metaconstant $p^m \in Fun$ for every $m > 1$).

- A (countably infinite) set of *variables*, the union of the following (disjoint) sets: V_{Fn}, V_{Pn}, V_{Fun}, V_{Tn} and V_T.
- The set containing the operators $\uparrow$ and $\downarrow$.
- The set of the usual logical connectives and the set of punctuation symbols: ',', '(', ')','[' and ']'.

We assume that we are given a mapping that associates with every constant, function and predicate symbol a metaconstant that is intended to name it. Thus, if α is a non-variable symbol in L, then α^1 represents the metaconstant in L intended as the name of α. Also metaconstants have names: if α^1 is a metaconstant, then α^2 represents the metaconstant in L intended as name of α^1. In general, we write α^k, $k \geq 0$, to mean α if $k = 0$ and some symbol in L that is the name of α^{k-1} if $k > 0$. Similarly, if β is the name of some constant or metaconstant then we will write β^{-1} for that symbol.

The alphabet of L contains two operators, $\uparrow$ and $\downarrow$. The intuition is that $\uparrow\alpha$ means the name of α (i.e., α^1), and $\downarrow\beta$ what β names (i.e., β^{-1}). We will also use the notation $\uparrow^i\beta$ (respectively, $\downarrow^i\beta$) to indicate i-times applications of the operator $\uparrow$ (respectively, $\downarrow$). If β is the constant name a^n, for example, then $\uparrow^i\beta$ is intended to be the constant name a^{n+i}. Note that $\uparrow\alpha$ is meaningful also when α is a variable although it cannot be computed because the alphabet of L does not contain names of variables. The computation of $\uparrow\alpha$ must be therefore suspended until α becomes sufficiently instantiated.

It is worth noting that L is a simply typed language, as there are different kinds of variables ranging over distinct domains. The use of typed variables allows us to define well-typed programs (w.r.t. the definition given below of compound term name and expression name). In order to define the language semantics, however, it is sufficient to deal with types with respect to variable assignment and unification only.

2.2 The Language L

The language L consists of all well-formed sentences in definite clausal form obtained from the alphabet of L. We start by presenting the definition of terms. It extends the usual one [49] to include term names and expression names. We define term names, expression names and terms by simultaneous induction.

Definition 2.2 The set of *term names* (Tn) is defined inductively as follows.

- A constant name (Cn) is a term name.
- A functor name (Fun) is a term name.
- For every term (defined below) α, $\uparrow\alpha$ is a term name.
- Any variable in one of the sets V_{Fun} and V_{Tn} is a term name.
- *Compound term names* (Ctn) are term names. A compound term name is a term name of the form $[\alpha_0, \alpha_1, \ldots, \alpha_n]$ $(n > 0)$, where $\alpha_0 \in (Fn \cup Fun \cup V_{Fn} \cup V_{Fun})$ and $\alpha_1, \ldots, \alpha_n$ are term names.

Term names allow the representation of ground terms of the language in the language itself. Constant names, functor names and compound term names are intended as names for constants, predicate names and function names, and compound terms, respectively.

Definition 2.3 The set of *expression names* (En) is defined inductively as follows.

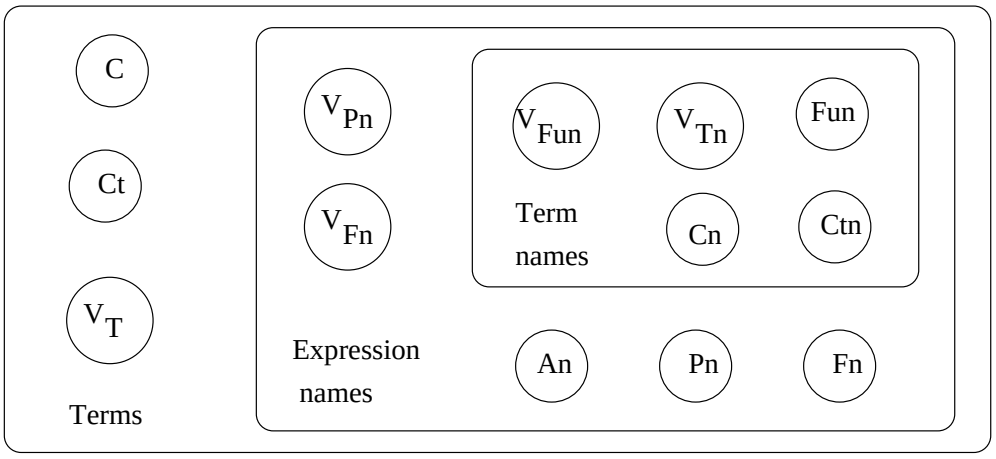
- A predicate name (Pn) is an expression name.
- A function name (Fn) is an expression name.
- Any variable in one of the sets V_{Pn} and V_{Fn} is an expression name.
- A term name (Tn) is an expression name.
- *Atom names* (An) are expression names. An atom name is an expression name of the form $[\alpha_0, \alpha_1, \ldots, \alpha_n]$ $(n > 0)$, where $\alpha_0 \in (Pn \cup V_{Pn})$ and $\alpha_1, \ldots, \alpha_n$ are term names.

Expression names allow the representation of ground expressions of the language in the language itself. Function names, predicate names and atom names are intended as names for function symbols, predicate symbols and atoms, respectively. If α is a ground compound term name $[\alpha_0, \alpha_1, \ldots, \alpha_n]$ $(n > 0)$ in L, then the name of α is the compound term name $[\beta_0, \beta_1, \ldots, \beta_n]$, where each β_i is the name of α_i, $0 \leq i \leq n$. Furthermore, the name of the name of α is $[\gamma_0, \gamma_1, \ldots, \gamma_n]$, where each γ_i is the name of β_i, $0 \leq i \leq n$, etc.

Definition 2.4 The set of *terms* (T) is defined inductively as follows.

- A constant is a term.
- A variable in V_T is a term.
- An expression name is a term.
- *Compound terms* (Ct) are terms. A compound term is a term of the form $[\alpha_0, \alpha_1, \ldots, \alpha_n]$ where α_0 is a function symbol and $\alpha_1, \ldots, \alpha_n$ are terms.
- For every term name α, $\downarrow\alpha$ is a term.

The next figure depicts the terms of the language L.



For example, the name of the term $[f, a]$ (whose concrete syntax might be $f(a)$) is the compound term name $[f^1, a^1]$, and the name of the atom $p([f, a], b^1)$ is the atom name $[p^1, [f^1, a^1], b^2]$. In general, the k-th name of the term $[\alpha_1^{m_1}, \ldots, \alpha_n^{m_n}]$ is the term name $[\alpha_1^{m_1+k}, \ldots, \alpha_n^{m_n+k}]$.

Requiring that names of compound expressions to be compositional in this way allows us to use a unification procedure for constructing expression names and accessing parts of them. This is in contrast with the approach discussed by Hill & Lloyd [36]. They introduce two basic representation schemes for the logic language of the definite clauses (object language): a non-ground typed representation and a ground representation. With respect to these representations, function and predicate symbols of the object language are represented in the metalanguage by function symbols. For example, if $f(a)$ is a term in the object language they name $f(a)$ by the term $f'(a')$ in the metalanguage, where the constant a' is the representation of the constant a and the function symbol f' is the representation of f. This choice, however, does not allow f' to be accessed through unification.

'Log [15] provides two metarepresentations for every expression of the language: atomic names and structural descriptive names. The alphabet of 'Log therefore contains a metaconstant for every expression of the language. If $f(a)$ is a term of the language, for example, then the alphabet contains a metaconstant (represented in the concrete syntax as $'f(a)'$) that is a name of $f(a)$. Since 'Log employs also structural descriptive names, term names also include compound term names.

The Gödel programming language [37] has names as an abstract datatype. Names of expressions will thus always appear in metaprograms as variables of that type. The metalevel part of a program can use the predicates associated with the datatype to manipulate names.

We now present the definitions of definite program and Horn clause equality theory, and finally the definition of logic program.

Definition 2.5 If t and t' are terms, then $t = t'$ is an *equation*.

Definition 2.6 If p is an n-ary predicate symbol distinct from '=' and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is an *atom*.

Definition 2.7 Let A and $B_1, \ldots, B_r$ be atoms and $e_1, \ldots, e_q$ be equations. If $\uparrow$ and $\downarrow$ operators only occur in the equations $e_1, \ldots, e_q$, then

$$A \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r \qquad (q \geq 0, r \geq 0).$$

is a *definite clause*. If both $q = 0$ and $r = 0$, then the clause is called a *unit clause*.

Requiring that $\uparrow$ and $\downarrow$ operators only occur in equations is not a restriction. The idea behind this requirement is to treat those operators differently from the other ones.

Definition 2.8 A *definite program* is a finite set of definite clauses.

Definition 2.9 A *definite goal* is a clause of the form

$$\leftarrow B_1, \ldots, B_r \qquad (r \geq 0)$$

where $B_1, \ldots, B_r$ are atoms. Each B_i, $1 \leq i \leq r$, is called a *subgoal* of the goal.

Differently from definite programs, equality theories can contain a possibly infinite number of clauses.

Definition 2.10 A *Horn clause equality theory* is a (possibly infinite) set of clauses of the form:

$$e \leftarrow e_1, \ldots, e_q \quad \text{or} \quad \leftarrow e_1, \ldots, e_q \qquad (q \geq 0)$$

where e and $e_1, \ldots, e_q$ are equations.

Definition 2.11 Let L be an abstract metalanguage. If P is a definite program and E is a Horn clause equality theory, then (P, E) is an L *logic program.*

Intuitively, a logic program consists of a Horn clause equality theory E that contains equalities for expressions of L, and of a definite program P that contains the clauses representing the problem at hand.

Definition 2.12 Given an L logic program (P, E), the *alphabet* of (P, E) is the set of all the symbols occurring in (P, E) extended with the symbols α^k for every k, $k \geq 0$, whenever a symbol α^l occurs in (P, E), for some $l \geq 0$.

Definition 2.13 Given an L logic program (P, E), the *language of* (P, E) is the subset of L that is given from the alphabet of (P, E).

2.3 An Equality Theory

One possible Horn clause equality theory for the language L is an extension of that proposed by Clark [17]. We call this extended theory ET. (In the following we use the notation $\forall(\alpha)$ to universally quantify all free variables appearing in α.)

(1) $\forall x(x = x)$

(2) $\forall x \forall y(x = y \leftarrow y = x)$

(3) $\forall x \forall y \forall z(x = z \leftarrow x = y \wedge y = z)$

(4) for each function symbol f in Fs (having some arity k)
$\forall([f, x_1, \ldots, x_k] = [f, y_1, \ldots, y_k] \leftarrow x_1 = y_1 \wedge \ldots \wedge x_k = y_k)$

(5) for each x_0 and y_0 in $(Fn \cup Pn \cup Fun)$
$\forall([x_0, x_1, \ldots, x_k] = [y_0, y_1, \ldots, y_k] \leftarrow x_0 = y_0 \wedge x_1 = y_1 \wedge \ldots \wedge x_k = y_k)$

Together with the axioms, *freeness* axioms, that characterise the Herbrand interpretation of function symbols [17]

(6) for every function symbol f in Fs (having some arity k)
$\forall([f, x_1, \ldots, x_k] = [f, y_1, \ldots, y_k] \rightarrow x_1 = y_1 \wedge \ldots \wedge x_k = y_k)$

(7) for every compound term name
$\forall([x_0, x_1, \ldots, x_k] = [y_0, y_1, \ldots, y_k] \rightarrow x_0 = y_0 \wedge x_1 = y_1 \wedge \ldots \wedge x_k = y_k)$

(8) for every pair of distinct function symbols f and g in Fs
$\forall([f, x_1, \ldots, x_k] \neq [g, y_1, \ldots, y_k])$

(9) for every term $[x_0, x_1, \ldots, x_k]$ and $[y_0, y_1, \ldots, y_l]$ with $k \neq l$
$\forall([x_0, x_1, \ldots, x_k] \neq [y_0, y_1, \ldots, y_l])$

(10) for every constant, metaconstant, compound term or compound term name t and t' that are not of the same kind
$\forall(t \neq t')$

(11) for every non-variable term t strictly containing some variable x
$t[x] \neq x$.

(12) for every distinct constant c and d
$c \neq d$.

Observe that ET does not contain the axiom (corresponding to the substitution schema in Clark's equality theory)

for each predicate symbol p in Ps (having some arity k)
$\forall(p(x_1, \ldots, x_k) \leftarrow p(y_1, \ldots, y_k), x_1 = y_1 \wedge \ldots \wedge x_k = y_k)$.

In fact, by the work of Elcock [30], this axiom is not needed if we consider the homogeneous form of definite clauses; where the homogeneous form of

$$p(t_1, \ldots, t_n) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$$

is the definite clause

$$p(x_1, \ldots, x_n) \leftarrow x_1 = t_1, \ldots, x_n = t_n, e_1, \ldots, e_q, B_1, \ldots, B_r$$

where every x_i, $1 \leq i \leq n$, is a new variable.

Chapter 3

Unification

The main computational step of SLD-resolution is the unification [56] of one or more pair of terms $\{(\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)\}$. This computation returns a substitution (most general unifier) that, when applied to a pair (α_i, β_i), makes the terms α_i and β_i syntactically identical. The problem of finding a most general unifier can be recast as the problem to find a solution of the set of equations $\{\alpha_1 = \beta_1, \ldots, \alpha_k = \beta_k\}$ that can be expressed as an equivalent (under the Herbrand equality theory) Herbrand assignment. Unification algorithms expressed as rewrite systems on sets of equations have been defined, e.g, by Huet [38][1], Robinson [57] and Martelli & Montanari [51].

3.1 Assignments, Herbrand Assignments and Substitutions

In order to take into consideration the different sorts of variables of the alphabet and expression names of L, the usual definitions [18] of assignment, Herbrand assignment and substitution have to be properly modified.

Definition 3.1 A *binding* is an equation of the form $x = t$ in which the term t does not contain any occurrence of the symbols $\uparrow$ and $\downarrow$, the variable x is different from t (i.e., t cannot be the variable x itself), and the following hold:

- if $x \in V_{Fn}$, then $t \in (Fn \cup V_{Fn})$;
- If $x \in V_{Pn}$, then $t \in (Pn \cup V_{Pn})$;
- if $x \in V_{Fun}$, then $t \in (Fun \cup V_{Fun})$;
- if $x \in V_{Tn}$, then $t \in Tn$;
- if $x \in V_T$, then $t \in T$.

To take into consideration the symbols $\uparrow$ and $\downarrow$, we introduce the definitions of *name equation* and *name binding*.

Definition 3.2 A *name equation* is an equation that contains at least one occurrence of $\uparrow$ or $\downarrow$.

Definition 3.3 A *name binding* is a name equation of the form $x = t$ such that the following hold:

[1] In Chapter 5 (Unification dans les langages de premier ordre) of Huet's Ph.D. thesis there is a very elegant unification algorithm for first-order expressions, which always terminates.

- if t is $\uparrow t'$ and $t' \in (Fn \cup Pn \cup Fun \cup V_{Fn} \cup V_{Pn} \cup V_{Fun})$, then $x \in V_{Fun}$;
- if t is $\uparrow t'$ and $t' \in (Tn \cup T \cup V_{Tn} \cup V_T)$, then $x \in V_{Tn}$;
- if t is $\downarrow t'$ and $t' \in (Fun \cup V_{Fun})$, then $x \in (V_{Fun} \cup V_{Fn} \cup V_{Pn})$;
- if t is $\downarrow t'$ and $t' \in (Tn \cup V_{Tn})$, then $x \in (V_{Tn} \cup V_T)$;
- if t is a compound term name, then $x \in V_{Tn}$;
- if t is a compound term, then $x \in V_T$.

Note that since a name binding $x = t$ is a name equation, the term t must contain at least one occurrence of the symbols $\uparrow$ or $\downarrow$.

Definition 3.4 If a variable x is bound by an equation $x = t$, then x *immediately depends* on the variables of its binding term t. The *depends* relation is the transitive closure of the immediately depends relation.

Definition 3.5 An *assignment* is a (finite) set of bindings and name bindings $\{x_1 = t_1, \ldots, x_k = t_k\}$ in which $x_1, \ldots, x_k$ are distinct variables not depending on themselves. If the assignment contains only bindings, then it is called a *Herbrand assignment.*

Definition 3.6 A *substitution* is a Herbrand assignment $\{x_1 = t_1, \ldots, x_k = t_k\}$ in which no bound variable x_i appears in any of the binding terms $t_1, \ldots, t_k$.

The following result relates Herbrand assignments to substitutions.

Theorem 3.7 *For every Herbrand assignment H there is a substitution S such that H and S are equivalent for every interpretation.*

The proofs of this and the next theorem are slight modifications of the proofs given by Clark [18] in order to take into consideration the different sorts of variables and expression names of L. Given a Herbrand assignment H, we hereafter write $\widehat{H}$ to indicate the corresponding substitution.

The following theorem states a semantic property of substitutions.

Theorem 3.8 *The existential closure of a substitution is true for every interpretation.*

Definition 3.9 Given a substitution $\widehat{H}$, the *application* of $\widehat{H}$ to a term or formula α, written as $\alpha\widehat{H}$, is the replacement of each free occurrence of a variable x bound in $\widehat{H}$, by the term to which it is bound.

3.2 A Unification Rewrite System

The unification process of L is expressed in terms of a rewrite system, called R_U. R_U transforms sets of equations until a "solved form" is obtained, from which the most general unifier can be extracted.

R_U contains rewrite rules operating on triples $\langle H, F, E\rangle$, where $H \cup \{false\}$ is a Herbrand assignment, F a set of name bindings and E a set of equations. Intuitively, the set E contains equations yet to be solved, while H and F are the partial solutions. We use the constant *false* to indicate the absence of a solution. It is an implicit condition that with $E \cup \{t = t'\}$ we mean $E \cup \{t = t'\}$ or $E \cup \{t' = t\}$. The rewrite rules of R_U are the following. (Hereafter, to indicate n (respectively, as many as

possible) applications of rewrite rules of a rewrite system R_E, we use the notation $\overset{n}{\underset{E}{\Leftrightarrow}}$ (respectively, $\overset{*}{\underset{E}{\Leftrightarrow}}$).)

Identity. A trivial equality is superfluous.

$$(U_1) \quad \langle H, F, E \cup \{t = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F, E\rangle$$

Transitivity. If t does not contain x, then

$$(U_2) \quad \langle H \cup \{x = t'\}, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H \cup \{x = t'\}, F, E \cup \{t' = t\}\rangle$$

$$(U_3) \quad \langle H, F \cup \{x = t'\}, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F \cup \{x = t'\}, E \cup \{t' = t\}\rangle.$$

Variable. Typically an equation between a variable and something else can be moved to the solved part, but some equations cause failure (sort conflict or occur check).

If H and F do not contain any equation of the form $x = t'$ and $H \cup F \cup \{x = t\}$ is an assignment, then

if $x = t$ is a binding

$$(U_4) \quad \langle H, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H \cup \{x = t\}, F, E\rangle$$

if $x = t$ is a name binding

$$(U_5) \quad \langle H, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F \cup \{x = t\}, E\rangle$$

If $x = t$ is neither a binding nor a name binding (sort conflict), or in $H \cup F \cup \{x = t\}$ x depends on itself (occur check), then

$$(U_6) \quad \langle H, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle \{\mathit{false}\}, \{\}, \{\}\rangle.$$

Inhomogeneity. Constants, metaconstants, compound terms and compound term names are incompatible with each other.

If each of t and t' is a constant, a metaconstant, a compound term or a compound term name and they are not of the same kind, then

$$(U_7) \quad \langle H, F, E \cup \{t = t'\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle \{\mathit{false}\}, \{\}, \{\}\rangle.$$

Clash. If t and t' are distinct constants or metaconstants, compound terms with distinct principal symbols or compound term names of different length, then

$$(U_8) \quad \langle H, F, E \cup \{t = t'\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle \{\mathit{false}\}, \{\}, \{\}\rangle.$$

Compound terms. Equality between compound terms is propagated to the corresponding subexpressions.

$$(U_9) \quad \langle H, F, E \cup \{[f, t_1, \ldots, t_k] = [f, t'_1, \ldots, t'_k]\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F, E \cup \{t_1 = t'_1, \ldots, t_k = t'_k\}\rangle$$

Compound term names. Equality between compound term names is also propagated.

$$(U_{10}) \quad \langle H, F, E \cup \{[t_0, \ldots, t_k] = [t'_0, \ldots, t'_k]\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F, E \cup \{t_0 = t'_0, \ldots, t_k = t'_k\}\rangle$$

Given the sets H, F and E as previously defined, the unification rewrite system starts with the triple $\langle H, F, E\rangle$ and repeatedly performs one of the previous rewrite rules until none is applicable. The following examples show the answers returned by R_U.

Example 3.10 Let E be $\{[f^1, a^1] = [x, y]\}$, where $f^1 \in Fn$, $a^1 \in Cn$, $x \in V_{Fn}$ and $y \in V_{Tn}$. Then, R_U rewrites $\langle\{\}, \{\}, E\rangle$ as $\langle\{x = f^1, y = a^1\}, \{\}, \{\}\rangle$.

Example 3.11 Let E be $\{[p^1, [f^2, x]] = [y, [z, w]], w = a^1\}$, where $p^1 \in Pn$, $f^2 \in Fun$, $x \in V_T$, $y \in V_{Pn}$, $z \in V_{Fun}$, $w \in V_{Tn}$ and $a^1 \in Cn$. Then, R_U rewrites $\langle\{\}, \{\}, E\rangle$ as $\langle\{y = p^1, z = f^2, x = w, w = a^1\}, \{\}, \{\}\rangle$.

Notice that in the previous example, the first argument of the returned triple is not a substitution but a Herbrand assignment.

Example 3.12 Let E be $\{[x, a^1] = [f^1, \uparrow y]\}$, where $x \in V_{Fn}$, $a^1 \in Cn$, $f^1 \in Fn$ and $y \in V_T$. Then, R_U rewrites $\langle\{\}, \{\}, E\rangle$ as $\langle\{x = f^1\}, \{\}, \{a^1 = \uparrow y\}\rangle$.

The equation $a^1 = \uparrow y$ in the previous example cannot be rewritten, as it is not a name binding.

3.3 Properties of the Unification Rewrite System

We require that the unification rewrite system R_U has some properties that will play a central role in the proofs of the soundness and completeness of SLD*-resolution defined in Chapter 6.

The next result proves termination of the unification rewrite system. (The structure of the proof is similar to the proof given by Martelli & Montanari [51] for the unification algorithm (called *algorithm1* in [51]).) The proof of this and thse following theorems is reported in the Appendix.

Lemma 3.13 *The unification rewrite system R_U: (i) always terminates, no matter which choices are made; (ii) returns a triple $\langle\{false\}, \{\}, \{\}\rangle$ (a fail termination) or the triple $\langle H, F, E\rangle$ (a success termination), where H is a Herbrand assignment, F is a set of name bindings and $H \cup F$ is an assignment. Furthermore, E contains only equations of the form $t = \uparrow t'$ and $t = \downarrow t'$, where t is not a variable and t' is any term.*

Definition 3.14 A rewrite system R_E is *left-linear* if every rule $t \underset{E}{\Leftrightarrow\!\!\rightarrow} t_1$ in R_E is such that no variable in t occurs more than once.

Definition 3.15 A rewrite system R_E is *overlapping* if R_E contains two rules $l \underset{E}{\Leftrightarrow\!\!\rightarrow} r$ and $s \underset{E}{\Leftrightarrow\!\!\rightarrow} t$ such that there exists a nonvariable subterm s' of s that unifies with l.

Note that if a rule $l \underset{E}{\Leftrightarrow\!\!\rightarrow} r$ overlaps a rule $s \underset{E}{\Leftrightarrow\!\!\rightarrow} t$, then there exists a unifying substitution σ such that $l\sigma = s'\sigma$. Therefore, the overlapped term $s\sigma$ can be rewritten to either $t\sigma$ or $s''\sigma$, where $s''\sigma$ is the term obtained by substituting $r\sigma$ to $l\sigma$ in $s\sigma$.

Definition 3.16 A rewrite system is *orthogonal* if it is both left-linear and nonoverlapping.

Definition 3.17 A rewrite system R_E is *confluent* if for every term t, t_1 and t_2 such that $t \overset{n}{\underset{E}{\Leftrightarrow\!\!\rightarrow}} t_1$ and $t \overset{m}{\underset{E}{\Leftrightarrow\!\!\rightarrow}} t_2$ $(n \geq 0,\ m \geq 0)$, there exists a term t' such that $t_1 \overset{*}{\underset{E}{\Leftrightarrow\!\!\rightarrow}} t'$ and $t_2 \overset{*}{\underset{E}{\Leftrightarrow\!\!\rightarrow}} t'$.

Definition 3.18 A rewrite system is *canonical* if it is confluent and terminating.

When a rewrite system R_E is terminating and confluent, it follows that every term t has a unique normal form t', obtained as $t \overset{*}{\underset{E}{\Leftrightarrow\!\!\rightarrow}} t'$.

Theorem 3.19 *The unification rewrite system R_U is canonical.*

The correctness of R_U is stated by the following theorem. Recall that ET is the equality theory for the language L as defined in Section 2.3.

Theorem 3.20 *If $\langle H, F, E\rangle \overset{*}{\underset{U}{\Leftrightarrow}} \langle H', F', E'\rangle$, then $ET \models (H \cup F \cup E) \equiv (H' \cup F' \cup E')$.*

So far, we have first introduced the metalanguage L, and then we have shown an equality theory for L with the corresponding rewrite system. Now, what we need to do is to characterize the correspondence of names of L and what is named. This is the topic of the next chapter.

Chapter 4

Name Theories

A metalevel architecture is characterized by a naming relation, that specifies the correspondence between names and what is named. We formalize this concept by introducing a *name theory*, that allows us to explicitly express this correspondence and whose interpretations define the specific naming relation.

4.1 A Name Theory

A name theory for a metalanguage contains the axioms that relate terms to their intended names (i.e., term names) and names to what is named.

In this section we define a name theory, called NT, for the compositional names of L. We consider the case that the naming scheme is a bijection[1]. NT contains the following axioms.

(1) $\forall x, y(\uparrow x = \uparrow y \rightarrow x = y)$

(2) $\forall x, y \in (V_{Fun} \cup V_{Tn})(\downarrow x = \downarrow y \rightarrow x = y)$

(3) for every constant, predicate name, function name, atom name or compound term t,
$\forall x(t \neq \uparrow x)$

(4) $\forall x(\downarrow\uparrow x = x)$

(5) $\forall x \in (V_{Fun} \cup V_{Tn})(\uparrow\downarrow x = x)$

(6) for every constant, metaconstant, predicate name, function name or functor name c^n, $n \geq 0$,
$\uparrow c^n = c^{n+1}$

(7) for every compound term whose function symbol $f \in Fs$ (having some arity k),
$\forall(\uparrow[f, x_1, \ldots, x_k] = [f^1, \uparrow x_1, \ldots, \uparrow x_k])$

(8) for every compound term name or atom name, $[x_0, x_1, \ldots, x_k]$
$\forall(\uparrow[x_0, x_1, \ldots, x_k] = [\uparrow x_0, \uparrow x_1, \ldots, \uparrow x_k])$

Proposition 4.1 *The next two formulas are theorems of NT.*

(i) $\forall x \forall y \in (V_{Fun} \cup V_{Tn})(\uparrow x = y \rightarrow x = \downarrow y)$;

[1] Requiring the naming relation to be functional and injective is important for obtaining a powerful extension of unification; without this restriction, many rewrites would be disallowed. Moreover, it does not seem to restrict significantly the usefulness of the naming relation.

(ii) $\forall x \in (V_{Fun} \cup V_{Tn})\forall y(\downarrow x = y \rightarrow x = \uparrow y)$.

By case (i) of Proposition 4.1, the following formulas can be derived by the name theory NT.

(9) for every metaconstant or functor name c^n, $n > 0$
$\downarrow c^n = c^{n-1}$

(10) for every compound term name $[f^1, x_1, \ldots, x_k]$, where $f^1 \in Fn$
$\forall(\downarrow[f^1, x_1, \ldots, x_k] = [f, \downarrow x_1, \ldots, \downarrow x_k])$

(11) for every compound term name $[x_0, x_1, \ldots, x_k]$, where $x_0 \in (Fun \cup V_{Fun})$
$\forall(\downarrow[x_0, x_1, \ldots, x_k] = [\downarrow x_0, \downarrow x_1, \ldots, \downarrow x_k])$

The choice of a naming scheme above is one parameter of this inference scheme. Together with this naming scheme we must be given a naming rewrite system for the particular naming scheme.

4.2 A Naming Rewrite System

Given a Herbrand assignment H and a set of equations E, a naming rewrite system starts with the pair $\langle H, E\rangle$ and repeatedly performs one of the rewrite rules until none is applicable. It should terminate and rewrite the pair $\langle H, E\rangle$ to a unique pair $\langle H, E'\rangle$, under the same restrictions as for the unification rewrite system but where $H \cup E$ and $H \cup E'$ are equivalent under the name theory.

In this section we define a rewrite system, called R_N, suitable for computing names of expressions of L and whose underlying name theory is NT. The following rewrite rules characterize R_N. It is an implicit condition that E is not $\{false\}$ and that with $E \cup \{t = t'\}$ we mean $E \cup \{t = t'\}$ or $E \cup \{t' = t\}$.

Evaluate up arrows. If any expression $\uparrow t$ has become instantiated, then replace it with another expression.

If $t\widehat{H}$ is a constant, metaconstant, predicate name, function name or a functor name c^n, $n \geq 0$, then

(N_1) $\quad \langle H, E[\uparrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[c^{n+1}]\rangle$.

If $t\widehat{H}$ is a compound term $[f, t_1, \ldots, t_k]$, where f is a function symbol, then

(N_2) $\quad \langle H, E[\uparrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[[f^1, \uparrow t_1, \ldots, \uparrow t_k]]\rangle$.

If $t\widehat{H}$ is a compound term name or an atom name $[t_0, t_1, \ldots, t_k]$, then

(N_3) $\quad \langle H, E[\uparrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[[\uparrow t_0, \uparrow t_1, \ldots, \uparrow t_k]]\rangle$.

Evaluate down arrows. If any expression $\downarrow t$ has become instantiated, then replace it with another expression.

If $t\widehat{H}$ is a metaconstant or a functor name c^n, $n > 0$, then

(N_4) $\quad \langle H, E[\downarrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[c^{n-1}]\rangle$.

If $t\widehat{H}$ is a compound term name $[f^1, t_1, \ldots, t_k]$, where $f^1 \in Fn$, then

(N_5) $\quad \langle H, E[\downarrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[[f, \downarrow t_1, \ldots, \downarrow t_k]]\rangle$.

If $t\widehat{H}$ is a compound term name $[t_0, t_1, \ldots, t_k]$, where $t_0 \in (Fun \cup V_{Fun})$, then

(N_6) $\langle H, E[\downarrow t]\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E[[\downarrow t_0, \downarrow t_1, \ldots, \downarrow t_k]]\rangle$.

If $t\widehat{H}$ is a constant, a predicate name, a function name, an atom name or a compound term, then

(N_7) $\langle H, E[\downarrow t]\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, \{false\}\rangle$.

Simplify arrows. We take advantage of the fact that an arrow is the other's inverse.

For every expression t either of the form $\downarrow\uparrow t'$, or of the form $\uparrow\downarrow t'$ and $t' \in (Fun \cup Tn)$

(N_8) $\langle H, E[t]\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E[t']\rangle$.

Reverse arrow. There may be name equations where the arrow is on the "wrong" term so it cannot be rewritten.

If $i > 0$, $x\widehat{H}$ is a variable and $t\widehat{H}$ is not a variable, then

(N_9) $\langle H, E \cup \{\uparrow^i x = t\}\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E \cup \{x = \downarrow^i t\}\rangle$.

If $i > 0$ and $x\widehat{H}$ is a variable, then

(N_{10}) $\langle H, E \cup \{\downarrow^i x = t\}\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E \cup \{x = \uparrow^i t\}\rangle$.

If $i > 0$, $x\widehat{H}$ is $\downarrow^i[x_0, t_1, \ldots, t_k]$, $x_0 \in (V_{Fn} \cup V_{Tn})$ and $t\widehat{H}$ is not a variable, then

(N_{11}) $\langle H, E \cup \{x = t\}\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E \cup \{[x_0, t_1, \ldots, t_k] = \uparrow^i t\}\rangle$.

The following examples show the answers returned by the naming rewrite system R_N.

Example 4.2 Let H be the set $\{y = a\}$ and E be $\{x = \uparrow y, z = \downarrow[f^1, x]\}$, where $y \in V_T$, $a \in C$, $x \in V_{Tn}$, $z \in V_{Tn}$ and $f^1 \in Fn$. Then, R_N rewrites $\langle H, E\rangle$ as $\langle H, \{x = a^1,\ z = [f, \downarrow x]\}\rangle$.

Note that in the previous example the subexpression $\downarrow x$ cannot be rewritten since the binding for the variable x (i.e., $x = a^1$) is in E and not in H.

Example 4.3 Let H be the set $\{x = b^1,\ y = a\}$ and E be $\{x = \uparrow y\}$, where $x \in V_{Tn}$, $b^1 \in Cn$, $y \in V_T$ and $a \in C$. Then, R_N rewrites $\langle H, E\rangle$ as $\langle H, \{x = a^1\}\rangle$.

Observe that H extended with the equation $x = a^1$ is not satisfiable with respect to the equality theory ET. Next example shows that the naming rewrite system R_N is not able to decompose a compound name term into its components (it only takes into consideration $\uparrow$ or $\downarrow$ expressions).

Example 4.4 Let H be the set $\{y = b\}$ and E be $\{[f^2, a^1] = [\downarrow x, \uparrow y]\}$, where $y \in V_T$, $b \in C$, $f^2 \in Fun$, $a^1 \in Cn$ and $x \in V_{Fun}$. Then, R_N rewrites $\langle H, E\rangle$ as $\langle H, \{[f^2, a^1] = [\downarrow x, b^1]\}\rangle$.

The equation $[f^2, a^1] = [\downarrow x, b^1]$ of the previous example is not satisfiable with respect to the equality theory ET.

4.3 Properties of the Naming Rewrite System

Some properties enjoyed by the naming rewrite system R_N are shown. The first result proves termination.

Lemma 4.5 *The naming rewrite system R_N: (i) always terminates, no matter which choices are made; (ii) returns a pair $\langle H, \{false\}\rangle$ (a fail termination) or the pair $\langle H, E\rangle$ (a success termination) such that $E\widehat{H}$ contains $\uparrow$ or $\downarrow$ subexpressions only of the form $\uparrow x$, $\downarrow x$ or $\downarrow[x_0, t_1, \ldots, t_k]$, where $x_0 \in (V_{Fn} \cup V_{Tn})$. Furthermore, every name equation in $E\widehat{H}$ has the form $x = \uparrow y$, $x = \downarrow[x_0, t_1, \ldots, t_k]$, $x = t$ or $t = t'$, where x and y are variables, $x_0 \in (V_{Fn} \cup V_{Tn})$, and t and t' are compound terms.*

Theorem 4.6 *The naming rewrite system R_N is canonical.*

The correctness of R_N is stated by the following theorem.

Theorem 4.7 *Let H be a Herbrand assignment and E a set of equations. If $\langle H, E\rangle \overset{*}{\underset{N}{\Leftrightarrow}} \langle H, E'\rangle$, then $NT \models H \cup E \equiv H \cup E'$.*

4.4 Formalizing Encodings in Equality Theories

The following examples show the formalization of three encodings appearing in literature. (For an overview on encodings, cf. van Harmelen [64].)

Example 4.8 Various encodings can be axiomatized in an equality theory: for example, a simple one where no information is included into names. The encoding axiomatized by the first two axioms corresponds to the "encoding" (identity function) typically used in Prolog metainterpreters [59].

(1) For every constant c
$\uparrow c = c$

(2) For every function symbol f (having some arity k)
$\forall x_1, \ldots, x_k \in T\,(\uparrow([f, x_1, \ldots, x_k]) = [f, x_1, \ldots, x_k])$

This encoding, however, strongly reduces the expressive power of the metatheory. It is not possible, for example, to use a unification procedure for constructing names of expressions and accessing parts of them, as the name of the function symbol of a term is again a function symbol. A possible solution to this problem could be that of substituting axiom 2 with the following.

(3) For every function symbol f (having some arity k)
$\forall x_1, \ldots, x_k \in T\,(\uparrow([f, x_1, \ldots, x_k]) = [f, \uparrow x_1, \ldots, \uparrow x_k])$

where the symbol f appearing as first element in the name term is a metaconstant. One advantage of using overloading on names is that a rewrite system for such axioms can be very simple and efficient, but, on the other hand, ambiguous cases arise. Suppose, for example, that we want to obtain what the name term $[f, y_1, \ldots, y_k]$ names. Then we have an ambiguity because it could be either a name term of the form $[f, x_1, \ldots, x_k]$ or the term of the form $[f, x_1, \ldots, x_k]$. (Jiang introduces an ambivalent logic [44] where he tackles this problem by making no distinction between sentences and terms.)

Example 4.9 A more expressive encoding (this is reminiscent of the encoding proposed by Hill & Lloyd [36]) could be axiomatized as follows.

(1) For every constant c
$\uparrow c = constant\_name(c^1)$

(2) For every function symbol f (having some arity k)
$\forall x_1, \ldots, x_k \in T\,(\uparrow([f, x_1, \ldots, x_k]) = function(functor(f^1), arity(k),$
$arguments(y_1, \ldots, y_k)) \leftarrow \uparrow x_1 = y_1, \ldots, \uparrow x_k = y_k)$

Example 4.10 In Reflective Prolog [26], we are given sets of symbols together with a naming relation on them (actually the system of sorts is somewhat more elaborate than that of L so the equality theory and the rewrite system are slightly more elaborate). The compatibility with Prolog demands a certain ambiguity between constants, function symbols and predicate symbols, but each occurrence of a symbol is uniquely determined.

For example, the symbols `a`, `foo` and `42` are examples of object constants, while the symbols `"a"`, `"foo"`, `"42"`, `""a""` and `"""a"""` are examples of constant names. Moreover, it is specified that $\texttt{a}^1 = \texttt{"a"}$, $\texttt{42}^1 = \texttt{"42"}$, $\texttt{a}^2 = \texttt{"a"}^1 = \texttt{""a""}$, etc.

Similarly, the symbols `f` and `bar` are examples of function symbols, the symbols `{f}` and `{bar}` are examples of function names and the symbols `"{f}"`, `"{bar}"` and `""{f}""` are further examples of constant names. Moreover, it is specified that $\texttt{f}^1 = \texttt{\{f\}}$, $\texttt{f}^2 = \texttt{"\{f\}"}$, etc.

In the same way, the symbols `p` and `append` are examples of predicate symbols, the symbols `<p>` and `<append>` are examples of predicate names and the symbols `"<p>"`, `"<append>"` and `""<append>""` are also examples of constant names. Moreover, it is specified that $\texttt{append}^1 = \texttt{<append>}$, $\texttt{append}^2 = \texttt{"<append>"}$, etc.

(Except for the elaboration of the system of sorts mentioned above, SLD*-resolution together with the rewrite system for L can be used as presented for running Reflective Prolog programs.)

Chapter 5

Combining Rewrite Systems for Unification and Naming

A possible integration of the unification and the naming rewrite system is presented. Such an integration allows us to solve the problems arisen in examples 3.12, 4.2, 4.3 and 4.4.

5.1 The Rewrite System R_{UN}

The rewrite system R_{UN} is obtained by composing the unification rewrite system R_U and the naming rewrite system R_N. Given a Herbrand assignment H, a set of name bindings F and a set of equations E, R_{UN} rewrites the triple $\langle H, F, E\rangle$ until no rule is applicable. Its rewrite rules are the following.

Move name equations. If $t\widehat{H}$ contains either a ground occurrence of an $\uparrow$ or $\downarrow$ subexpression, or an occurrence of the form $\downarrow[t_0, \ldots, t_k]$, where t_0 is ground or a variable in V_{Fun}, then

$$(UN_1) \quad \langle H, F \cup \{x = t\}, E\rangle \overset{1}{\underset{UN}{\Leftrightarrow}} \langle H, F, E \cup \{x = t\}\rangle.$$

Consider rules of R_N. If $\langle H, E\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, \{false\}\rangle$, then

$$(UN_2) \quad \langle H, F, E\rangle \overset{1}{\underset{UN}{\Leftrightarrow}} \langle \{false\}, \{\}, \{\}\rangle.$$

Otherwise, if $\langle H, E\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E'\rangle$ and E' is not $\{false\}$, then

$$(UN_3) \quad \langle H, F, E\rangle \overset{1}{\underset{UN}{\Leftrightarrow}} \langle H, F, E'\rangle.$$

Consider rules of R_U. If $\langle H, F, E\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H', F', E'\rangle$, and UN_2 and UN_3 do not apply, then

$$(UN_4) \quad \langle H, F, E\rangle \overset{1}{\underset{UN}{\Leftrightarrow}} \langle H', F', E'\rangle.$$

The intuition is that first we compute names of ground expressions by means of the rules of R_N, and then we try the unification process by the rules of R_U. In fact, the rewrite rule UN_1 simply moves name equations from F to E in such a way that the remaining rewrite rules can be applied.

5.2 Properties of R_{UN}

It is not necessarily the case that the union of terminating and confluent rewrite systems is terminating and confluent (consider, e.g., the system consisting of the single rule $a \Leftrightarrow\!\!\!\rightarrow b$ and that consisting of the two rules $b \Leftrightarrow\!\!\!\rightarrow a$ and $a \Leftrightarrow\!\!\!\rightarrow c$). However, the next lemma shows that this is the case for R_{UN}.

Lemma 5.1 *The naming rewrite system R_{UN}: (i) always terminates, no matter which choices are made; (ii) returns a triple $\langle\{false\},\{\},\{\}\rangle$ (a fail termination) or the triple $\langle H,F,\{\}\rangle$ (a success termination), where H is a Herbrand assignment, F is a set of name equations and $H \cup F$ is an assignment. Furthermore, every name equation in $F\widehat{H}$ has the form $x = \uparrow y$, $x = \downarrow[x_0,t_1,\ldots,t_k]$ or $x = t$, where x and y are variables, $x_0 \in (V_{Fn} \cup V_{Tn})$, $t_1,\ldots,t_k$ are term names and t is a compound term. Every $\uparrow$ and $\downarrow$ subexpression of $x = t$ takes the form $\uparrow y$, $\downarrow y$ or $\downarrow[x_0,t_1,\ldots,t_k]$.*

Theorem 5.2 *The rewrite system R_{UN} is canonical.*

The correctness of R_{UN} is stated by the following theorem. Let NET be the theory consisting of all the axioms of the theories ET and NT.

Theorem 5.3 *If $\langle H,F,E\rangle \underset{UN}{\overset{*}{\Leftrightarrow\!\!\!\rightarrow}} \langle H',F',E'\rangle$, then $NET \models (H \cup F \cup E) \equiv (H' \cup F' \cup E')$.*

Theorem 5.4 *If $\langle H,F,E\rangle \underset{UN}{\overset{*}{\Leftrightarrow\!\!\!\rightarrow}} \langle H',F',\{\}\rangle$, then the existential closure of $H' \cup F'$ is true for every interpretation.*

We reconsider the examples presented in Chapters 3 and 4 concerning the rewrite systems R_U and R_N. We show how the arisen problems are solved by using R_{UN}.

Example 5.5 Let E be $\{[x,a^1] = [f^1,\uparrow y]\}$, where $x \in V_{Fn}$, $a^1 \in Cn$, $f^1 \in Fn$ and $y \in V_T$. Then, R_{UN} rewrites $\langle\{\},\{\},E\rangle$ as $\langle\{x = f^1, y = a\},\{\},\{\}\rangle$.

Example 5.6 Let H be the set $\{y = a\}$ and E be $\{x = \uparrow y, z = \downarrow[f^1,x]\}$, where $y \in V_T$, $a \in C$, $x \in V_{Tn}$, $z \in V_{Tn}$ and $f^1 \in Fn$. Then, R_{UN} rewrites $\langle H,\{\},E\rangle$ as $\langle\{y = a, x = a^1, z = [f,a]\},\{\},\{\}\rangle$.

Example 5.7 Let H be the set $\{x = b^1,\ y = a\}$ and E be $\{x = \uparrow y\}$, where $x \in V_{Tn}$, $b^1 \in Cn$, $y \in V_T$ and $a \in C$. Then, R_{UN} rewrites $\langle H,\{\},E\rangle$ as $\langle\{false\},\{\},\{\}\rangle$.

Example 5.8 Let H be the set $\{y = b\}$ and E be $\{[f^2,a^1] = [\downarrow x,\uparrow y]\}$, where $y \in V_T$, $b \in C$, $f^2 \in Fun$, $a^1 \in Cn$ and $x \in V_{Fun}$. Then, R_{UN} rewrites $\langle H,\{\},E\rangle$ as $\langle\{false\},\{\},\{\}\rangle$.

The equality theory of next example axiomatizes a kind of encoding that is similar to the one proposed for Reflective Prolog. We show that the corresponding rewrite system is not canonical.

Example 5.9 Suppose that the alphabet of L is extended to contain a (infinite) set of metaconstants that are intended as names for variables, and that the equality theory UN' contains the axioms of UN plus axioms of the form: for every variable x, $\uparrow x = x^1$. Furthermore, assume that $R_{UN'}$ contains the rewrite rules of R_{UN} and the rule

(1) $\langle H,F,S \cup \{e\}\rangle \underset{UN}{\overset{1}{\Leftrightarrow\!\!\!\rightarrow}} \langle H \cup \{x = y^1\},F,S\rangle$
if $e\widehat{H}$ is $x = \uparrow y$ for any variable x and y.

Then the obtained rewrite system is not canonical. In fact, it is easy to see that the triple $\langle\{\},\{\},\{x = a, y = \uparrow x\}\rangle$ can be rewritten to $\langle\{x = a, y = a^1\},\{\},\{\}\rangle$ or to $\langle\{x = a, y = x^1\},\{\},\{\}\rangle$ according to which equation is selected first.

Chapter 6

SLD*-Resolution

We show in this chapter how it is possible to model reflection. We take as a basis the implicit reflection of Reflective Prolog, which specifies what are probably the weakest conditions for level-shifting: each subgoal can be resolved at any level where there are applicable clauses. We introduce forms of reflection into SLD-resolution to shift from the object level to the metalevel and vice versa. The basic idea is that metalevel clauses with conclusion $solve(\alpha)$ play the role of additional clauses for every predicate whose instances match α.

6.1 The Language L*

In order to characterize reflection, we first extend the alphabet of the language L to contain the distinguished predicate symbol *solve*. We call the extended language L^*. The set of clauses defining the distinguished predicate symbol *solve* is called *metalevel*, and the set of the remaining clauses is called *object level*.

The definition of L^* definite clause includes, however, some syntactic restrictions. First, the argument of *solve* must be an atom name representing a goal. As we characterize reflection in a bilevel architecture, the role of the metalevel is to extend the deducibility of object level predicates only. Thus, metalevel clauses cannot express any property on other metalevel clauses, and, consequently, the argument of *solve* cannot be the name of any solve atom. Although we characterize reflection only between two levels, a generalization of the approach to architectures with n levels is straightforward.

Definition 6.1 An L^* definite clause is a definite clause where, for each contained atom $solve(\alpha)$, α is an atom name that does not contain $solve^m$ as subterm, for any integer $m > 0$. An L^* *definite program* is a finite set of L^* definite clauses.

The following example (taken from Costantini & Lanzarone [26]) shows an L^* definite program.

Example 6.2 Consider the following program

$friend(giorgio, mary) \leftarrow$ (Object level)
$amico(lucy, albert) \leftarrow$
$happy(x) \leftarrow friend(x, lucy)$
$symmetric(friend^1) \leftarrow$
$symmetric(equivalent^1) \leftarrow$
$equivalent(amico^1, friend^1) \leftarrow$

$solve([y, z_1, z_2]) \leftarrow symmetric(y), solve([y, z_2, z_1])$ (Metalevel)
$solve([y_1, z_1, z_2]) \leftarrow equivalent(y_1, y_2), solve([y_2, z_1, z_2])$

where $\{friend, amico, happy, symmetric, equivalent\} \subseteq Ps$, $\{giorgio, mary, lucy, albert\} \subseteq C$, $x \in V_T$, $\{y, y_1, y_2\} \subseteq V_{Pn}$ and $\{z_1, z_2\} \subseteq V_{Tn}$. The first three clauses of the object level define the relations *friend*, *amico* and *happy*. The two facts following them state that the relations *friend* and *equivalent* are both symmetric, and the last one states that the relations *amico* and *friend* are equivalent.

The metalevel consists of two solve clauses. The first clause declaratively defines the concept of symmetry in the theory: the objects whose names are z_1 and z_2 are in the relation whose name is y, provided that the relation denoted by y is asserted to be symmetric and that the object denoted by z_2 and z_1 are in the relation denoted by y. The second clause states that equivalent relations have the same extensions.

6.2 SLD*-Resolution

To begin with, note that it is well known how to reformulate SLD-resolution over definite clause programs in terms of sets of equations rather than substitutions (see, e.g., Clark [18]). A computation state is a pair $\langle M, H\rangle$, where M is a set of atoms that have to be proved and H is a Herbrand assignment. A set of equations may also contain the atom *false*; such a set is inconsistent. Unification can in this process be seen as a rewrite system that takes a set of equations to an equivalent (under the Herbrand equality theory) Herbrand assignment.

The assumption that unification rewrites the whole set of equations to a Herbrand assignment can be relaxed. Let a *state* instead consist of a triple $\langle M, H, F\rangle$, where M is a set of atoms, say $\{A_1, \ldots, A_n\}$, H is a Herbrand assignment (possibly containing *false*), and F is a set of name equations, say $\{e_1, \ldots, e_m\}$. Such a state represents the definite goal $(\leftarrow e_1, \ldots, e_m, A_1, \ldots, A_n)\widehat{H}$, where $\widehat{H}$ is the substitution corresponding to the Herbrand assignment H. We can see H and F as the solved and unsolved part of a single equation system. Given n equations $e_1, \ldots, e_n$ and a Horn clause equality theory E, a unification rewrite system whose underlying theory is E, indicated as R_E, now takes a triple $\langle H, F, \{e_1, \ldots, e_n\}\rangle$ to a new triple $\langle H', F', \{\}\rangle$ such that $H \subseteq H'$ and $H \cup F \cup \{e_1, \ldots, e_n\}$ is equivalent to $H' \cup F'$ under E.

Let (P, E) be a logic program and G a definite goal $\leftarrow B_1, \ldots, B_r$ $(r > 0)$. In the following to enhance readability, we write $(P, E) \cup \{G\}$ instead of $(P \cup \{G\}, E)$. An *initial state* when refuting $(P, E) \cup \{G\}$ is a triple $\langle\{B_1, \ldots, B_r\}, \{\}, \{\}\rangle$, a *failure state* is a triple $\langle M, \{false\}, \{\}\rangle$ and a *success state* is a triple $\langle\{\}, H, F\rangle$, where H does not contain *false* and F is a solvable set of name equations, i.e., there exists a Herbrand assignment H' such that $E \models F\widehat{H'}$.

Now we can extend SLD-resolution to consider the interaction between the different levels of the language L^*. In particular, to shift from the object level to the metalevel and vice versa, we introduce two forms of reflection into SLD-resolution.

Definition 6.3 *(SLD*-resolution)* Let E be a Horn clause equality theory and R_E a rewrite system for E. Let $\langle M, H, F\rangle$ be a state that is neither a failure state, nor a success state, A be the selected atom in M, and S be a set of equations. Assume that the variables of the selected clause C in P are standardised apart from $\langle M, H, F\rangle$. Then the state $\langle(M \setminus \{A\}) \cup \{B_1, \ldots, B_r\}, H', F'\rangle$ is derived from $\langle M, H, F\rangle$ and C by using R_E if and only if one of the following holds:

1. A is $p(t_1, \ldots, t_k)$
   C is $p(t'_1, \ldots, t'_k) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$
   S is $\{t_1 = t'_1, \ldots, t_k = t'_k, e_1, \ldots, e_q\}$
2. A is $solve([p^1, t_1, \ldots, t_k])$
   C is $p(t'_1, \ldots, t'_k) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$
   S is $\{t_1 = \uparrow t'_1, \ldots, t_k = \uparrow t'_k, e_1, \ldots, e_q\}$

3. A is $p(t_1, \ldots, t_k)$
   C is $solve([p^1, t'_1, \ldots, t'_k]) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$
   S is $\{t_1 = \downarrow t'_1, \ldots, t_k = \downarrow t'_k, e_1, \ldots, e_q\}$

and $\langle H, F, S\rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H', F', \{\}\rangle$

The first case corresponds to the operations of the modified SLD-resolution discussed above. The second case is an *object-to-meta* reflection, where an object level clause is used in the course of proving a metalevel atom. The third case, finally, is a *meta-to-object* reflection, where a metalevel clause is used to prove an object level atom.

Note that SLD*-resolution is a quite small extension that adds a naming scheme to the language and adds interlevel inference and access to the names of terms to the inference system. The two additional inference rules could also be added to other inference systems for definite clause programs that have provisions for delaying computations.

An SLD*-derivation is a (finite or infinite) path in the tree of states above. A SLD*-refutation is a finite path in the tree, ending with a success state.

Definition 6.4 Let E be a Horn clause equality theory and R_E be a rewrite system for E. Let (P, E) be a logic program and G be a definite goal. An *SLD*-derivation* of $(P, E) \cup \{G\}$ consists of a (finite or infinite) sequence of states $\langle M, \{\}, \{\}\rangle$, $\langle M_1, H_1, F_1\rangle, \ldots$ and a sequence $C_1, C_2, \ldots$ of variants of program clauses of P, such that each $\langle M_{i+1}, H_{i+1}, F_{i+1}\rangle$ is derived from $\langle M_i, H_i, F_i\rangle$ and C_{i+1} using R_E.

Definition 6.5 An *SLD*-refutation* of $(P, E) \cup \{G\}$ is a finite SLD*-derivation of $(P, E) \cup \{G\}$ which has a success state as last state in the derivation, i.e., a state of the form $\langle\{\}, H_n, F_n\rangle$. In that case the refutation has length n.

Example 6.6 Let UN and R_{UN} be the Horn clause equality theory and the rewrite system for UN defined in Chapter 5. Let (P, E) be the following program

$$\left(\left\{\begin{array}{l} friend(giorgio, mary) \leftarrow \\ symmetric(friend^1) \leftarrow \\ solve([friend^1, z_1, z_2]) \leftarrow symmetric(friend^1), \\ \qquad\qquad\qquad\qquad solve([friend^1, z_2, z_1]) \end{array}\right\}, UN\right)$$

Assume that R_E is the rewrite system R_{UN}. Then an SLD*-refutation for $(P, E) \cup \{\leftarrow friend(mary, x)\}$ is the following. The initial state is

$$\langle\{friend(mary, x)\}, \{\}, \{\}\rangle.$$

Consider the third clause in P. By meta-to-object reflection (case 3) and rewriting $\langle\{\}, \{\}, \{mary = \downarrow z_1, x = \downarrow z_2\}\rangle \underset{UN}{\overset{*}{\Leftrightarrow}} \langle\{z_1 = mary^1\}, \{z_2 = \uparrow x\}, \{\}\rangle$, the next state is

$$\langle\{symmetric(friend^1), solve([friend^1, z_2, z_1])\}, \{z_1 = mary^1\}, \{z_2 = \uparrow x\}\rangle.$$

Suppose that the selected atom is $symmetric(friend^1)$, and consider the second clause in P. By case 1 of SLD*-resolution and rewriting $\langle\{z_1 = mary^1\}, \{z_2 = \uparrow x\}, \{friend^1 = friend^1\}\rangle \underset{UN}{\overset{*}{\Leftrightarrow}} \langle\{z_1 = mary^1\}, \{z_2 = \uparrow x\}, \{\}\rangle$, we obtain

$$\langle\{solve([friend^1, z_2, z_1])\}, \{z_1 = mary^1\}, \{z_2 = \uparrow x\}\rangle.$$

Finally, by using the first clause in P, by object-to-meta reflection (case 2) and rewriting $\langle\{z_1 = mary^1\}, \{z_2 = \uparrow x\}, \{z_2 = \uparrow giorgio, z_1 = \uparrow mary\}\rangle \underset{UN}{\overset{*}{\Leftrightarrow}} \langle\{z_1 = mary^1, z_2 = giorgio^1, x = giorgio\}, \{\}, \{\}\rangle$, the next state is the success state

$$\langle\{\}, \{z_1 = mary^1, z_2 = giorgio^1, x = giorgio\}, \{\}\rangle.$$

Example 6.7 Let (P, E) be the following logic program

$$\left(\left\{ \begin{array}{l} p(x) \leftarrow y = \uparrow x, q(y) \\ q(a^1) \leftarrow \end{array} \right\}, UN \right)$$

where UN and R_{UN} are as above. Then an SLD*-refutation for $(P, E) \cup \{\leftarrow p(z)\}$ is the following. The initial state is

$$\langle \{p(z)\}, \{\}, \{\} \rangle.$$

Consider the first clause in P. By case 1 of SLD*-resolution and rewriting $\langle \{\}, \{\}, \{z = x, y = \uparrow x\} \rangle \underset{UN}{\overset{*}{\Leftrightarrow}} \langle \{z = x\}, \{y = \uparrow x\}, \{\} \rangle$ we obtain

$$\langle \{q(y)\}, \{z = x\}, \{y = \uparrow x\} \rangle.$$

Finally, by using the second clause in P and $\langle \{z = x\}, \{y = \uparrow x\}, \{y = a^1\} \rangle \underset{UN}{\overset{*}{\Leftrightarrow}} \langle \{z = x, x = a, y = a^1\}, \{\}, \{\} \rangle$, the next state is the success state

$$\langle \{\}, \{z = x, x = a, y = a^1\}, \{\} \rangle.$$

Chapter 7

Properties of SLD*-Resolution

After defining the declarative semantics of L^*, we present the soundness and completeness results of SLD*-resolution. The declarative semantics is an extension of that proposed by Jaffar et al. [42] covering also the metalevel part of the language. The extension is based on the work of Costantini & Lanzarone [26].

7.1 Declarative Semantics

7.1.1 Preliminaries

One of the advantages of working within the clausal form is that we can consider only Herbrand interpretations when we want to prove that a set of clauses is unsatisfiable. In fact, we first associate with a set P of clauses a canonical syntactic domain and function assignment. Then, to prove that a ground atom is a logical consequence of P, we can restrict ourselves to proving that it is true in the models associated with this syntactic domain. This domain is called the Herbrand universe and the associated models are called Herbrand models. Thus, we have

$$P \models A \text{ iff } P \models_U A$$

where A is a ground atom, U is the Herbrand universe of P and $\models_U$ denotes logical implication in the context of a fixed domain and function assignment (in this case the Herbrand domain and function assignment).

Nevertheless, by using Herbrand universe, the notion of equality can be accommodated via syntactic identity only. To overcome this problem, Jaffar et al. [41] proposed the use of quotient universes. Given an equality theory E and a congruence relation R consistent with E, the *quotient universe* of U with respect to R, indicated as U/R, is the set of the equivalence classes of U under R (i.e., the partition given by R in U). In general, there is an infinite number of such models R of E

$$(P, E) \models A \text{ iff } P \models_{U/R} A \text{ for all } R.$$

In order to have a canonical model of the equality theory there must exist a congruence relation R such that

$$E \models s = t \text{ iff }, (s) = , (t)$$

where , (s) and , (t) denote the R-congruence classes of the ground terms s and t. This can be achieved only if the equality theory has a finest congruence relation.

Jaffar et al. showed that each consistent Horn clause equality theory generates a finest congruence. As a consequence, it holds that

$$(P, E) \models A \text{ iff } P \models_{U/E} A$$

where U/E is the finest congruence generated by E.

In order to take equations into consideration, the definitions of base, interpretation and model of a logic program (P, E) have to be defined properly. In the following we call them E-base, E-interpretation and E-model, respectively.

Definition 7.1 The *E-base* $B_{(P,E)}$ of a logic program (P, E) is the set of all ground atoms which can be formed by using predicate symbols from the language of (P, E) with ground terms from the quotient universe U/E as arguments.

In the following sections, we write , (s) to indicate the element in U/E assigned to the ground term s. Similarly, if p is a predicate symbol of the language of (P, E), then we write , $(p(t_1, \ldots, t_n))$ as a shorthand for $p(, (t_1), \ldots, , (t_n))$.

Example 7.2 Let (P, E) be the following logic program

$$\left(\left\{ \begin{array}{l} solve([q^1, x]) \leftarrow x = \uparrow a \\ p(a) \leftarrow \end{array} \right\}, \left\{ \begin{array}{l} a^1 = \uparrow a \\ a^2 = \uparrow a^1 \\ a = \downarrow a^1 \\ a^1 = \downarrow a^2 \end{array} \right\} \right).$$

Let , be the mapping

$$\begin{array}{ll} , (\uparrow a) = a^1 & , (a) = a \\ , (\uparrow a^1) = a^2 & , (a^1) = a^1 \\ , (\downarrow a^1) = a & , (a^2) = a^2. \end{array}$$

Then U/E is $\{a, a^1, a^2\}$ and $B_{(P,E)}$ is

$$\left\{ \begin{array}{l} p(a), p(a^1), p(a^2) \\ q(a), q(a^1), q(a^2) \end{array} \right\} \cup \left\{ \begin{array}{l} solve([p^1, a^1]) \\ solve([p^1, a^2]) \end{array} \right\} \cup \left\{ \begin{array}{l} solve([q^1, a^1]) \\ solve([q^1, a^2]) \end{array} \right\}.$$

Definition 7.3 An *E-interpretation* of a logic program (P, E) is any subset of the E-base $B_{(P,E)}$.

Definition 7.4 Let I be an E-interpretation of a logic program (P, E). Then I *E-satisfies* a ground definite clause $A \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_m$ $(q \geq 0, m \geq 0)$ in P iff at least one of the following holds.

- , $(A) \in I$;
- there exists an i, $1 \leq i \leq q$, such that $E \not\models e_i$;
- there exists a j, $1 \leq j \leq m$, such that , $(B_j) \notin I$.

Definition 7.5 Let I be an E-interpretation of a logic program (P, E). Then I E-satisfies (P, E) iff I E-satisfies each ground instance of a clause in P. If there exists an I which E-satisfies (P, E), then (P, E) is said to be *E-satisfiable*. (P, E) is *E-unsatisfiable* iff there is no E-interpretation that E-satisfies (P, E).

Definition 7.6 Let I be an E-interpretation of a logic program (P, E). Then I is an *E-model* of (P, E) iff I E-satisfies (P, E).

Example 7.7 Consider the logic program (P, E) of Example 7.2. Then the following E-interpretation is an E-model of (P, E)

$$\left\{ \begin{array}{l} p(a) \\ \\ \end{array} \right\} \cup \left\{ \begin{array}{l} \\ solve([q^1, a^1]) \\ \\ \end{array} \right\}.$$

Definition 7.8 An atom A is a *logical E-consequence* of a logic program (P, E) if, for every E-interpretation I, I is an E-model for (P, E) implies that , $(A) \in I$.

7.1.2 Extended E-Interpretations and Reflective E-Models

To cover the metalevel part of L^*, the notions of E-interpretation, E-model and logical consequence are properly extended.

Definition 7.9 An *extended E-interpretation* is an E-interpretation that contains $solve([p^1, s_1, \ldots, s_n])$ whenever it contains $p(t_1, \ldots, t_n)$ such that , $(\uparrow t_i) = s_i$ for all i, $1 \leq i \leq n$.

The idea underlying the introduction of the extended E-interpretations is that of obtaining an architecture of the system where the object level is fully transparent for the metalevel. In contrast, what the metalevel communicates to the object level can be varied, as we show below.

The set of extended E-interpretations of (P, E) is a complete lattice under the partial order of set inclusion.

Example 7.10 Consider the logic program (P, E) of Example 7.2. Then the following is an extended E-interpretation of (P, E)

$$\left\{ \begin{array}{l} p(a) \\ \\ \end{array} \right\} \cup \left\{ \begin{array}{l} solve([p^1, a^1]) \\ solve([q^1, a^1]) \\ \end{array} \right\}.$$

Note that the metalevel contains the representation of every atom of the object level.

In order to characterize the communication from the metalevel to the object level, we extend a logic program (P, E) with new axioms. In this way it is possible to model different forms of communication by simply considering different sets of axioms.

Definition 7.11 Let I be an extended E-interpretation of a logic program (P, E). Then I *reflective E-satisfy* (P, E) iff I E-satisfies

$$(P \cup \{p(t_1, \ldots, t_n) \leftarrow t_1 = \downarrow s_1, \ldots, t_n = \downarrow s_n, solve([p^1, s_1, \ldots, s_n])\}, E)$$

for every predicate symbol p in (P, E) distinct from *solve*.

Such additional axioms of the form $\alpha \leftarrow e_1, \ldots, e_q, solve(\beta)$ are called *reflection axioms*. In the field of AI, a number of forms of reasoning can be modeled by means of different sets of reflection axioms [23].

In this approach, we define the reflection axioms in such a way that whenever $solve(\alpha)$ is provable at the metalevel, α is provable at the object level. Observe that it is not possible to extend a program with reflection axioms of the form $\alpha \leftrightarrow solve(\alpha^1)$ because they may lead to inconsistency (by the results of Tarski [62]) when negation is present in programs. Vice versa, by considering extended E-interpretations and reflection axioms it is possible to accommodate various kinds of negation without giving rise to any problem of inconsistency [25].

Definition 7.12 Let I be an extended E-interpretation of a logic program (P, E). If I reflectively E-satisfies (P, E), then (P, E) is said to be *reflective E-satisfiable*. (P, E) is *reflective E-unsatisfiable* iff there exists no extended E-interpretation that reflective E-satisfies (P, E).

Definition 7.13 Let I be an extended E-interpretation of a logic program (P, E). Then I is a *reflective E-model* of (P, E) iff I reflectively E-satisfies (P, E).

Our solution is similar to the approaches chosen for autoepistemic logics. Konolige [46], for instance, given a theory T, introduces a set of interpretations for T that satisfy some properties, e.g., the principle of groundedness, and then he adds modal schemata to T in order to ensure that a unique model always exists.

Reflective E-models are clearly models in the usual sense [42], as they are obtained by extending a given logic program with a set of definite clauses. Therefore the model intersection property still holds, and there exists a least reflective E-model of (P, E), indicated in the following as $LRM_{(P,E)}$.

Example 7.14 Consider the logic program (P, E) of Example 7.2. Then $LRM_{(P,E)}$ is

$$\left\{ \begin{array}{l} p(a) \\ q(a) \end{array} \right\} \cup \left\{ \begin{array}{l} solve([p^1, a^1]) \\ solve([q^1, a^1]) \end{array} \right\}.$$

Augmenting (P, E) with the reflection axiom $q(x) \leftarrow x = {\downarrow}y, solve([q^1, y])$ adds $q(a)$ in every reflective E-model.

Definition 7.15 An atom A is a *reflective logical E-consequence* of a logic program (P, E) if, for every extended E-interpretation I, I is a reflective E-model for (P, E) implies that , $(A) \in I$.

Proposition 7.16 *Let (P, E) be a logic program and $\leftarrow A_1, \ldots, A_k$ be a definite goal. Then $(P, E) \cup \{\leftarrow A_1, \ldots, A_k\}$ is reflectively E-unsatisfiable iff $A_1 \wedge \ldots \wedge A_k$ is a reflective logical E-consequence of (P, E).*

7.1.3 Fixpoint Semantics

A least reflective E-model can be characterized as the least fixed point of a mapping $T_{(P,E)}$ similar to the one proposed by Jaffar et al. [41], except that it produces both $p(t_1, \ldots, t_n)$ and $solve([p^1, s_1, \ldots, s_n])$, for any predicate symbol p distinct from *solve*, whenever one of them can be derived. $T_{(P,E)}$ is a function over extended E-interpretations.

Definition 7.17 Let (P, E) be a logic program and I be an extended E-interpretation of (P, E). Let $ground(P)$ be the set of all ground instances of clauses in P. Then $T_{(P,E)}$ is defined as follows.

$$\begin{array}{l} T_{(P,E)}(I) = \\ \quad \{p(t_1, \ldots, t_n), solve([p^1, s_1, \ldots, s_n]) \in B_{(P,E)}: \\ \qquad p(t'_1, \ldots, t'_n) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m \in ground(P) \\ \qquad p \text{ is not } solve, \\ \qquad E \models e_i \text{ for } 1 \leq i \leq q, \\ \qquad \{, (A_1), \ldots, , (A_m)\} \subseteq I, \\ \qquad , (t'_j) = t_j \text{ and } , (\uparrow t'_j) = s_j \text{ for } 1 \leq j \leq n\} \\ \qquad\qquad\qquad \cup \\ \quad \{solve([p^1, s_1, \ldots, s_n]), p(t_1, \ldots, t_n) \in B_{(P,E)}: \\ \qquad solve([p^1, s'_1, \ldots, s'_n]) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m \in ground(P) \end{array}$$

$E \models e_i$ for $1 \leq i \leq q$,
$\{, (A_1), \ldots, , (A_m)\} \subseteq I$,
$, (s'_j) = s_j$ and $, (\downarrow s'_j) = t_j$ for $1 \leq j \leq n\}$

Proposition 7.18 *Let (P, E) be a logic program. The mapping $T_{(P,E)}$ is continuous.*

Proposition 7.19 *Let (P, E) be a logic program and I an extended E-interpretation. Then I is a reflective E-model for (P, E) if and only if $T_{(P,E)}(I) \subseteq I$.*

As the class of extended E-interpretations forms a complete lattice (under the inclusion order), $T_{(P,E)}$ is continuous over this class, and the class of reflective E-models is given by $\{I \mid T_{(P,E)}(I) \subseteq I\}$, the result of van Emden & Kowalski [63] is applicable, and provides a fixed point characterization of the least reflective E-model of a logic program (P, E).

Theorem 7.20 $LRM_{(P,E)} = lfp(T_{(P,E)}) = T_{(P,E)} \uparrow \omega$.

Example 7.21 Consider the logic program (P, E) of Example 7.14. Then

$$\begin{array}{lcl} I_0 & = & \{\} \\ I_1 & = & T_{(P,E)}(I_0) = I_0 \cup \{p(a), solve([p^1, a^1]), q(a), solve([q^1, a^1])\} \\ I_2 & = & T_{(P,E)}(I_1) = I_1 = LRM_{(P,E)}. \end{array}$$

7.2 Soundness and Completeness of SLD*-Resolution

Results of soundness and completeness of SLD*-resolution with respect to the least reflective E-model are presented.

Definition 7.22 Let E be a Horn clause equality theory and F a set of name equations. An *E-solution* of F is a Herbrand assignment H such that $E \models \forall(F\widehat{H})$.

Definition 7.23 Let (P, E) be a logic program and G a definite goal. An *answer* for $(P, E) \cup \{G\}$ is a pair $\langle H, F\rangle$ consisting of a Herbrand assignment H and a set of name equations F.

Given a definite goal G, an answer for $(P, E) \cup \{G\}$ consists of a Herbrand assignment, containing all the bindings computed through the computation of G, and of a set containing all the name equations that have been accumulated.

Definition 7.24 Let (P, E) be a logic program, G a definite goal $\leftarrow A_1, \ldots, A_k$ and $\langle H, F\rangle$ be an answer for $(P, E) \cup \{G\}$. $\langle H, F\rangle$ is a *correct answer* for $(P, E) \cup \{G\}$ if, for every E-solution H' of F, $\forall((A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'})$ is a reflective logical E-consequence of (P, E).

Note that in order to take into consideration the set of name equations, we require that $\forall((A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'})$ is a reflective logical E-consequence of (P, E), for every E-solution H' of F.

Theorem 7.25 *Let (P, E) be a logic program and G be a definite goal of the form $\leftarrow A_1, \ldots, A_k$ $(k \geq 0)$. Let H be a Herbrand assignment, F a set of name equations and H' an E-solution of F. Suppose $\langle H, F\rangle$ is an answer for $(P, E) \cup \{G\}$ such that $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is ground. Then the following are equivalent:*

(a) $\langle H, F\rangle$ is correct.

(b) $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is true w.r.t. every reflective E-model of (P, E).

(c) $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is true w.r.t. the least reflective E-model of (P, E).

7.2.1 Soundness

Definition 7.26 The *success set* of a logic program (P, E) is the set of all ground atoms A such that $(P, E) \cup \{\leftarrow A\}$ has an SLD*-refutation.

Note that ground atoms can contain occurrences of $\uparrow$ and $\downarrow$ operators. Thus, these operators also occur in atoms of the success set.

Definition 7.27 Let (P, E) be a logic program and G a definite goal. Suppose that $\langle\{\}, H, F\rangle$ is the success state of an SLD*-refutation of $(P, E) \cup \{G\}$. Then $\langle H, F\rangle$ is a *computed answer* for $(P, E) \cup \{G\}$.

Next theorem states the main soundness result, i.e., computed answers are correct.

Theorem 7.28 (Soundness of SLD*-resolution) *Let E be a Horn clause equality theory and R_E a rewrite system for E. Let (P, E) be a logic program and G a definite goal. If R_E is correct with respect to E, then every computed answer for $(P, E) \cup \{G\}$ is a correct answer for $(P, E) \cup \{G\}$.*

Corollary 7.29 *Let E be a Horn clause equality theory and R_E a correct rewrite system for E. Let (P, E) be a logic program and G a definite goal. Suppose that there exists an SLD*-refutation of $(P, E) \cup \{G\}$. Then $(P, E) \cup \{G\}$ is reflectively E-unsatisfiable.*

As ground atoms can contain occurrences of $\uparrow$ and $\downarrow$, while reflective E-models only contain representative forms of such atoms, the success set of a logic program is in general not contained in its least reflective E-model. However, this property holds if we consider the representative forms of ground atoms. (Recall that the representative form of a ground atom A is written as , (A).)

Corollary 7.30 *If a ground atom A belongs to the success set of a logic program (P, E), then , (A) is contained in the least reflective E-model of (P, E).*

Now we strengthen Corollary 7.30 by showing that, if a ground atom A has an SLD*-refutation of length n, then , $(A) \in T_{(P,E)}{\uparrow}n$. This is an extension of the result due to Apt & van Emden [4].

Definition 7.31 The *closure* of an atom A, indicated as $\Psi(A)$, is the set of representative elements of all ground instances of A

$$\Psi(A) = \{A' \mid A' \text{ is } , (A'') \text{ for every ground instance } A'' \text{ of } A\}.$$

Theorem 7.32 *Let E be a Horn clause equality theory and R_E a correct rewrite system for E. Let (P, E) be a logic program and G a definite goal $\leftarrow A_1, \ldots, A_k$. Suppose $(P, E) \cup \{G\}$ has an SLD*-refutation of length n with computed answer $\langle H_n, F_n\rangle$. Then $\bigcup_{j=1}^{k} \Psi(A_j\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}{\uparrow}n$ for every E-solution H' of F_n.*

7.2.2 Completeness

The main result of this section is the completeness of SLD*-resolution. This result holds if the rewrite system which is the parameter of SLD*-resolution is canonical. As an example of an incompleteness case we consider an equality theory E whose rewrite system R_E is not canonical and show that SLD*-resolution is not complete when parameterized with respect to R_E.

Example 7.33 Let P be the program of Example 6.7 and E be the equality theory UN' of Example 5.9.

$$\left(\left\{ \begin{array}{l} p(x) \leftarrow y = \uparrow x, q(y) \\ q(a^1) \leftarrow \end{array} \right\}, UN' \right)$$

Suppose that $R_{UN'}$ contains the rewrite rules of R_{UN} and the rule 1 of Example 5.9. Then $(P, E) \cup \{\leftarrow p(a)\}$ has an SLD*-refutation, while $(P, E) \cup \{\leftarrow p(z)\}$ has not. In fact, its initial state is

$$\langle \{p(z)\}, \{\}, \{\} \rangle.$$

Consider the first clause in P. By case 1 of SLD*-resolution and as $\langle \{\}, \{\}, \{z = x, y = \uparrow x\} \rangle \underset{UN'}{\Leftrightarrow\!\!\!\rightarrow} \langle \{z = x, y = x^1\}, \{\}, \{\} \rangle$, we obtain

$$\langle \{q(y)\}, \{z = x, y = x^1\}, \{\} \rangle.$$

Now, the only clause in P defining the predicate symbol q is the second one. As the metaconstants x^1 and a^1 are distinct, they do not unify, thus $\langle \{z = x, y = x^1\}, \{\}, \{y = a^1\} \rangle \underset{UN'}{\Leftrightarrow\!\!\!\rightarrow} \langle \{false\}, \{\}, \{\} \rangle$. Hence, this SLD*-derivation ends with a failure state.

We begin by appropriately rephrasing the Lifting lemma [49].

Lemma 7.34 (Lifting lemma) *Let E be a Horn clause equality theory and R_E a canonical rewrite system for E. Let (P, E) be a logic program, H a Herbrand assignment and G a definite goal. Suppose there exists an SLD*-refutation of $(P, E) \cup \{G\widehat{H}\}$ with success state $\langle \{\}, H_n, F_n \rangle$. Then there exists an SLD*-refutation of $(P, E) \cup \{G\}$ of the same length with success state $\langle \{\}, H'_n, F'_n \rangle$ such that $\langle H'_n, F'_n, H \rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\!\rightarrow}} \langle H_n, F_n, \{\} \rangle$.*

This lemma essentially states that, if we can prove a goal $G\widehat{H}$ from a logic program, then we can also prove the less instantiated goal G. The two proofs have the same length and are such that the computed answer of $G\widehat{H}$ can be obtained from the one of G by taking into consideration the bindings contained in H.

The first completeness result gives the converse to Corollary 7.30.

Theorem 7.35 *A ground atom A belongs to the success set of a logic program (P, E) if and only if , (A) is contained in the least reflective E-model of (P, E).*

Theorem 7.36 *Suppose that $(P, E) \cup \{G\}$ is reflectively E-unsatisfiable. Then there exists an SLD*-refutation of $(P, E) \cup \{G\}$.*

Next we turn our attention to correct answers. It is not possible to prove the exact converse of Theorem 7.28 because computed answers are always more "general" than correct answers with respect to the variables $x_1, \ldots, x_n$ contained in the initial goal. However, we can prove that every correct answer is an instance of a computed answer with respect to $x_1, \ldots, x_n$.

Lemma 7.37 *Let (P, E) be a logic program and A an atom. Suppose that A contains the variables $x_1, \ldots, x_n$ and $\forall x_1, \ldots, x_n(A)$ is a reflective logical E-consequence of (P, E). Then there exists an SLD*-refutation of $(P, E) \cup \{\leftarrow A\}$ with computed answer $\langle H, F \rangle$ such that $E \models \forall x_1, \ldots, x_n \exists (H \cup F)$.*

Example 7.38 Let UN and R_{UN} be the Horn clause equality theory and the rewrite system defined in Chapter 5. Let (P, E) be the following logic program

$$\left(\left\{solve([p^1, y]) \leftarrow y = \uparrow x\right\}, UN\right)$$

Then $\forall z(p(z))$ is a reflective logical E-consequence of (P, E). In fact, the equation $y = \uparrow x$ is satisfied for every value associated with x. The computed answer for the goal $\leftarrow p(z)$ is $\langle\{z = x\}, \{y = \uparrow x\}\rangle$. It holds that $UN \models \forall z \exists x \exists y (z = x \wedge y = \uparrow x)$.

Now we are in the position to prove the main completeness result.

Theorem 7.39 (Completeness of SLD*-resolution) *Let E be a Horn clause equality theory and R_E a canonical rewrite system for E. Let (P, E) be a logic program and G a definite goal. Then for every correct answer $\langle H, F\rangle$ for $(P, E) \cup \{G\}$, there exists a computed answer $\langle H', F'\rangle$ for $(P, E) \cup \{G\}$. Furthermore, there exists a Herbrand assignment H'' such that, for every E-solution H_F and $H_{F'}$ of F and F', respectively, it holds that $G\widehat{H'}\widehat{H_{F'}}\widehat{H''} = G\widehat{H}\widehat{H_F}$.*

Chapter 8

Related Work & Concluding Remarks

8.1 Related Work

In this thesis reflection has been characterized both theoretically and procedurally. From the procedural point of view we have proposed a proof-theoretic extension of SLD-resolution based on the concept of state. Such an extension is reminiscent of the procedural behaviour of constraint programming languages, where the computation of values (constraints) is delayed until we have enough information to compute them. This corresponds in our approach to the idea of delaying the computation of names of expressions if they are not ground. We believe that our approach can be easily extended to allow different kinds of constraints as well as the "naming costraints".

We are aware of a few other attempts to combine metalogic programming and constraint logic programming. Here we present a brief overview, together with some other relevant work, in chronological order.

Heintze et al. [35] propose an approach that integrates metaprogramming facilities and the constraint paradigm. They extend the language $CLP(\mathcal{R})$ [43] with the aim of allowing manipulations of $CLP(\mathcal{R})$ programs. The resulting language, called $CLP(\mathcal{R}+\mathcal{M})$, provides a basis for coding terms in a form which allows arbitrary structural manipulation and for converting between terms and their coded forms. Furthermore, they introduce some facilities to access the coded form of the current set of constraints, and some predicates, such as *var*, *nonvar*, etc., to develop metaprograms. The addition of these facilities enables the development of new applications such as meta-circular interpreters incorporating constraint solvers and simplification algorithms, as well as constraint partial evaluators. The authors claim that the underlying language extension is consistent with the constraint paradigm and is defined in a manner which is not operational. However, they do not mention how the basic results of *CLP* [40] still hold. Consider their predefined binary predicate *coded_ccs* which takes a list of variables and bounds the second argument to a coding of the solved form of the current constraint with respect to those variables. The goal $\leftarrow X+Y=6, Y\leq 0, \mathit{coded\_ccs}([X],Z), \Leftrightarrow 2\leq Y$, for example, will bind Z to the coded form of $6\leq X$, and $\leftarrow X+Y=6, Y\leq 0, \Leftrightarrow 2\leq Y, \mathit{coded\_ccs}([X],Z)$ will bind Z to the coded form of $6\leq X, X\leq 8$. Thus the independence of the computation rule, as the previous example shows, does not hold.

Lim & Stuckey [48] propose a meta-level structure of computation in order to give a logical meaning to metaprograms. In particular, they extend the theory of *CLP* with a metalevel structure that incorporates a method for representing object level terms

and interpreted relations (constraints) for manipulating object level terms. Thus each metaprogram can be seen as a constraint logic program over this structure. The constraint *FrozenUniv*, for example, allows us to simulate the Prolog meta-logical predicate *univ*. $FrozenUniv(FrozenObj, [Functor, Arg_1, \ldots, Arg_n])$ succeeds if *FrozenObj* is a frozen object (i.e., a special metalevel term representing an object level term) and the second argument is a list whose first element is the principal functor of *FrozenObj* and the remainder is a list of frozen objects representing the arguments. Similarly to Cervesato & Rossi [15], they use delay techniques in order to solve these meta-level constraints. In the previous example, *FrozenUniv* is delayed until either the first or second argument is ground. In this case delayed constraints do not lead to floundering conditions because all constraints are defined in such a way as to manipulate frozen objects or to check the solvability of object level constraints. (Constraints, for example, cannot be used to check the satisfiability of predicates.)

Sato [58] has presented a style of metaprogramming based on a truth predicate in three-valued logic. It is interesting to note that the naming relation he proposes is compositional, although he considers naming also of nonground (but closed) expressions. His emphasis is not on presenting a proof system, but rather a metainterpreter for the truth predicate.

Cervesato & Rossi [15] present a meta-level extension of a logic programming language, called *'Log*. The extension consists of a *naming scheme* that associates two different meta-representations to every syntactic object of the language, and of a *destructuring* operator that relates these meta-representations. In particular, each syntactic object of the language can be represented at the metalevel by means both of an atomic name, which describes the entity as a whole, and of a structured ground term, called *destructuring*, which describes the structure of the entity in terms of the names of its components. The informal semantics [15, p.149] of the destructuring operator, represented as $\Leftrightarrow$, is: a goal $N \Leftrightarrow S$ is true if N is the name of an object and S is the structural representation of the same object. Thus, $\Leftrightarrow$ simply defines a binary relation, called *destructuring relation* (DR), between names and structural representations. Procedurally, a goal $N \Leftrightarrow S$ succeeds from a program P if either N is a name and there exists a ground instance S' of S such that N and S' are in DR, or N is a variable, S is ground and there exists an instance N' of N such that N' and S are in DR. If, on the contrary, N is a variable and S is not ground then the goal is unsolvable and its proof is delayed until either one of the cases above occurs. Given a program P and a goal G, a computed answer for a refutation of $P \cup \{G\}$ is defined as a pair $\langle \sigma, C \rangle$ where σ is a substitution for the variables in G and C is the (possibly empty) set of destructuring goals on unsolvable forms. By using delay conditions a fast and simple constraint solver relying on a local propagation scheme can be developed [66]. Not always, however, can the constraint solver determine when a system of constraints is unsatisfiable. For example, the constraint $\{N \Leftrightarrow f^1(X), N \Leftrightarrow g^1(Y)\}$ is unsatisfiable but this will not be detected by a delaying constraint solver, unless more information arrives. In contrast, in our approach we cannot use delay techniques because this may lead to floundering conditions. In fact, we use techniques similar to constraint satisfaction techniques in order to model reflective steps of SLD*-resolution (i.e., whenever a goal fails at the object level, the proof of the goal is attempted at metalevel through reflection), and to prove the goal over which we apply reflection further derivations steps are necessary.

Jiang [44] proposes a basically different approach from ours, where he takes the syntactic ambiguity of Prolog even further. His logic intentionally tries to remove the distinction between predicate and function symbols, etc. For ground expressions his naming scheme can be captured in our framework, by self-naming. However, he has a quite different inference system in mind.

We are aware of a study by Christiansen [16], whose idea is to use constraint tech-

niques to implement a binary *demo* predicate which is logically complete. The query *demo*(*prog-rep*, *query-rep*), for example, is true whenever its arguments represent a program and a query, respectively, such that the query is provable from the program. In order to represent programs and queries, he uses a ground representation, including variable names. Basically, his approach is based on the use of *instance* constraints that allow him to replace variable names in the representation of an expression with metavariables. Such kind of constraints resembles $\uparrow$ and $\downarrow$ operators in our framework. However, they differ mainly in two respects. While he allows names also for non-ground expressions, we have to postpone the computation of names of expressions involving variables until enough information becames available to ground those variables. Moreover, in our framework it is possible to have many levels of naming, i.e., names of expressions, names of names, etc., while, in his approach, only one level of naming is allowed.

We have already discussed the Gödel programming language [37], let us just summarize that its naming relation (though being an abstract datatype) is compositional and that it seems to fit into our framework.

Some of these approaches are only partly related to the present one, because they are mainly aimed at *syntactic* metaprogramming: program manipulation and transformation via metaprograms. Our approach instead, is mainly intended for multilevel knowledge representation and reasoning. In fact, it allows reflective steps (and thus multilevel reasoning) to be modeled by extending SLD-resolution so as to dynamically relate language expressions and their names. We have seen that such an approach is easily applicable to many multilevel formalisms. In fact, it is usable for both explicit and implicit reflection: for explicit reflection, the extended resolution will be applied only on demand, while using usual SLD-resolution for all the other steps. The conditions for the applicability of the approach are the following. The reflective inference mechanism must be formally defined. It must be parameterized with respect to unification (which can thus be extended to include the equation rewrite algorithm). The declarative semantics must not depend on the specific features of the naming relation.

8.2 Discussion and Concluding Remarks

In a metalevel system, the choice of encoding is important for at least two reasons. First, as we have shown with some simple examples in Section 4.4, an encoding directly determines the expressivity of the metatheory. If we consider an encoding, for example, that conveys little information to the metalevel, then we can design efficient rewrite systems for that encoding; but, on the other hand, the expressivity of the metatheory is low (for example, this is the case for a non-ground encoding along the lines of Example 4.8). Once we have decided which encoding is suitable for our purposes, then we have to investigate whether it satisfies properties necessary for sound and complete inference. For example, we have pointed out that in amalgamated metalevel systems, encodings employing variable names in certain ways (cf. Example 7.33) result in a loss of completeness. (In general, we have shown in Section 7.2.2 that, if we employ an encoding whose corresponding rewrite system is not confluent, we may lose completeness.) However, such encodings allows us to investigate the state of the computation (i.e., we can analyse the instantiation of variables in queries, etc.). Thus, one may choose this last kind of encoding if these capabilities are important (this is for instance the solution adopted in the language Reflective Prolog), provided that one is aware that certain properties are lost.

The second reason concerns the fact that encodings influence the semantics of metalogic languages. In fact, metalanguages that are based on formally defined encodings have clear and well-defined declarative semantics [26, 58, 60].

In contrast, in order to give a semantic account of a metalogic programming language that employs a trivial encoding (such that $\uparrow t = t$ for any term t), two main possibilities have been considered up to now. De Schreye & Martens [27] characterize a class of programs for which the least Herbrand model semantics still holds for the vanilla metainterpreter and a limited form of amalgamation. This is the class of language-independent programs where language independence extends both domain independence and range restrictedness. As already noted by Levi & Ramundo [47], however, the class of language independent programs is too small: it includes deductive database programs, but rules out any logic program computing partially determined data structures. Instead they point out another possibility for giving a declarative semantics to the vanilla metainterpreter and some enhanced metainterpreters by abandoning the least Herbrand semantics in favour of the S-semantics [32]. Besides resorting to a different semantics, that approach can be applied (as the authors recognize) to enhanced metainterpreters only when there exists a corresponding pure object-level solution. Again, this restricts the power of the language, undermining, in our view, the motivation for using metalevel systems altogether. In fact, a fully corresponding object-level solution does not exist for every metalogic program, as pointed out, for instance, by Yalcinap [68]. Therefore, in order to exploit the full potential of metalogic programming, this kind of "no naming", although simple and efficient, must give place to more expressive forms of encoding.

In this thesis we have introduced a framework, where (i) we have shown, through a number of examples, how encodings can be formalized as equational theories, and (ii) we have investigated which properties the encodings must satisfy. The methodology that we propose can be used by a language designer for evaluating possible choices of encodings and their impact on semantic properties, perhaps also their computational complexity. Our final suggestion is that of designing formalisms parametric with respect to encodings, so as to be able to choose the most appropriate one for the application domain at hand, while departing as little as possible from classical semantics.

We believe that the proposed extensions may be integrated easily both in the interpreter [28] and the compiler (at present under development) of the language without generating much more complexity in time or space.

There are different ways to go from here. One could be that of studying techniques for developing efficient constraint solvers, starting from the ones developed for systems such as *NuProlog* [55] and *Sepia* [53]. Furthermore, a closer connection between constraint logic programming and metaprogramming has been opened up by this essay. In the field of *CLP*, Jaffar & Lassez [40] have established conditions under which the basic theorems of logic programming remain valid (Cohen [19] presents a summary of the theoretical foundations of *CLP*), provided that the set of axioms specifying constraints satisfies some properties. In particular, the constraint theory must be *satisfaction-complete*, i.e., every constraint is provable either satisfiable or unsatisfiable, and its model must be *solution-compact*, i.e., each element of the domain can be specified by a (potentially infinite) number of constraints, and the complement of any constraint can be specified by a (possibly infinite) number of constraints. For those name theories that satisfy the properties above, we can consider extending the *CLP* theory in order to obtain a more general framework on which to model reflection and constraint satisfaction over different domains D, i.e., a class of languages *R(eflective)CLP(D)*.

Furthermore, we are considering lifting some of the definitions and results of our approach into the more general *CLP* setting (the lift of results from the theory of logic programming to the theory of *CLP* has been systematized by Maher [50]) in order to obtain stronger results and simpler proofs.

Bibliography

[1] Aiello, L. C., Cecchi, C. and Sartini, D., Representation and Use of Metaknowledge, *Proc. of the IEEE*, 74:1304–1321 (1986).

[2] Aiello, L. C., Nardi, D. and Schaerf, M., Reasoning about Knowledge and Reasoning in a Meta-Level Architecture, *Intl. J. of Applied Intelligence*, 1 (1991).

[3] Aiello, L. C., Nardi, D. and Schaerf, M., Reasoning about Knowledge: the Meta-Level Approach, in: *Proc. Scandinavian Conf. on Artificial Intelligence*, IOS Press, Copenhagen, 1991.

[4] Apt, K. R. and van Emden, M. H., Contributions to the Theory of Logic Programming, *J. ACM*, 29(3):841–862 (1982).

[5] Attardi, G. and M, S., Meta-Level Reasoning across Viewpoints, in: T. O'Shea (ed.), *Proc. European Conf. on Artificial Intelligence*, North-Holland, Amsterdam, 1984.

[6] Barklund, J., Boberg, K. and Dell'Acqua, P., A Basis for a Multilevel Metalogic Programming Language, in: F. Turini (ed.), *Proc. 4th Intl. Workshop on Metaprogramming in Logic*, LNCS, Springer-Verlag, Berlin, 1994.

[7] Barklund, J., Boberg, K., Dell'Acqua, P. and Veanes, M., Meta-programmming with Theory Systems, in: K. Apt and F. Turini (eds.), *Meta-programming in Logic*, MIT PRESS, Cambridge, Mass., 1995.

[8] Barklund, J., Costantini, S., Dell'Acqua, P. and Lanzarone, G. A., SLD-Resolution with Reflection, in: M. Bruynooghe (ed.), *Logic Programming – Proc. 1994 Intl. Symp.*, MIT Press, Cambridge, Mass., 1994.

[9] Barklund, J., Costantini, S., Dell'Acqua, P. and Lanzarone, G., Semantical Properties of SLD-Resolution with Reflection, Submitted to ICLP '95, 1994.

[10] Barklund, J., What is a Meta-Variable in Prolog?, in: H. Abramson and M. H. Rogers (eds.), *Meta-Programming in Logic Programming*, MIT Press, Cambridge, Mass., 1989.

[11] Barklund, J., Costantini, S., Dell'Acqua, P. and Lanzarone, G. A., Integrating Reflection into SLD-Resolution, in: A. Momigliano and M. Ornaghi (eds.), *Proc. Post-Conf. W. on Proof-Theoretical Extensions of Logic Programming*, 1994.

[12] Barklund, J. and Hamfelt, A., Hierarchical Representation of Legal Knowledge with Metaprogramming in Logic, *J. Logic Programming*, (in print).

[13] Bowen, K. A. and Kowalski, R. A., Amalgamating Language and Metalanguage in Logic Programming, in: K. L. Clark and S.-Å. Tärnlund (eds.), *Logic Programming*, Academic Press, London, 1982.

[14] Brogi, A., Mancarella, P., Pedreschi, D. and Turini, F., Composition Operators for Logic Theories, in: J. W. Lloyd (ed.), *Computational Logic*, Springer-Verlag, Berlin, 1990.

[15] Cervesato, I. and Rossi, G., Logic Meta-Programming Facilities in 'Log, in: A. Pettorossi (ed.), *Meta-Programming in Logic*, LNCS 649, Springer-Verlag, Berlin, 1992.

[16] Christiansen, H., Efficient and Complete Demo Predicates for Definite Clause Languages, Datalogiske Skrifter, Technical Report 51, Dept. of Computer Science, Roskilde University, 1994.

[17] Clark, K. L., Negation as Failure, in: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, New York, 1978.

[18] Clark, K. L., Logic-Programming Schemes and Their Implementations, in: J.-L. Lassez and G. Plotkin (eds.), *Computational Logic: Essays in Honor of Alan Robinson*, MIT Press, Cambridge, Mass., 1991.

[19] Cohen, J., Constraint Logic Programming Languages, *Comm. ACM*, 33:52–68 (1990).

[20] Costantini, S., Dell'Acqua, P. and Lanzarone, G. A., Beyond Prolog, in: *Proc. AICA*, Trieste, 1989.

[21] Costantini, S., Dell'Acqua, P. and Lanzarone, G. A., Higher-order Extensions to Prolog are Needed, in: A. Bossi (ed.), *Proc. 5th Italian Conf. on Logic Programming*, Padova, 1990.

[22] Costantini, S., Dell'Acqua, P. and Lanzarone, G. A., Reflective Agents in Metalogic Programming, in: A. Pettorossi (ed.), *Meta-Programming in Logic*, LNCS 649, Springer-Verlag, Berlin, 1992.

[23] Costantini, S., Dell'Acqua, P. and Lanzarone, G. A., Extending Horn Clause Theories by Reflection Principles, in: C. MacNish, D. Pearce and L. M. Pereira (eds.), *Logics in Artificial Intelligence*, LNAI 838, Springer-Verlag, Berlin, 1994.

[24] Costantini, S. and Lanzarone, G. A., A Metalogic Programming Language, in: G. Levi and M. Martelli (eds.), *Proc. 6th Intl. Conf. on Logic Programming*, MIT Press, Cambridge, Mass., 1989.

[25] Costantini, S. and Lanzarone, G. A., Metalevel Negation in Non-Monotonic Reasoning, *Intl. J. of Methods of Logic in Computer Science*, 1 (1994).

[26] Costantini, S. and Lanzarone, G. A., A Metalogical Programming Approach: Language, Semantics and Applications, *J. Exper. Theor. Artificial Intelligence*, 6 (1994).

[27] De Schreye, D. and Martens, B., A Sensible Least Herbrand Semantics for Untyped Vanilla Meta-Programming and its Extension to a Limited Form of Amalgamation, in: A. Pettorossi (ed.), *Meta-Programming in Logic*, LNCS 649, Springer-Verlag, Berlin, 1992.

[28] Dell'Acqua, P., Development of the Interpreter for a Metalogic Programming Language, Degree thesis, Univ. degli Studi di Milano, Milano, 1989.

[29] Dershowitz, N. and Jouannaud, J.-P., Rewrite Systems, in: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, Elsevier, Amsterdam, 1990.

[30] Elcock, E. W., PROLOG: Subsumption of Equality Axioms by the Homogeneous Form, *J. Logic Programming*, 1:45–56 (1989).

[31] Eshghi, K., Meta-Language in Logic Programming, Ph.D. Thesis, Dept. of Computing, Imperial College, London, 1986.

[32] Falaschi, M., Levi, G., Martelli, M. and Palamidessi, C., A new Declarative Semantics for Logic Languages, in: R. A. Kowalski and K. A. Bowen (eds.), *Proc. 5th Intl. Conf. Symp. on Logic Programming*, MIT Press, Cambridge, Mass., 1988.

[33] Feferman, S., Transfinite Recursive Progressions of Axiomatic Theories, *J. Symbolic Logic*, 27:259–316 (1962).

[34] Gallaire, H. and Lasserre, C., Metalevel Control for Logic Programs, in: K. L. Clark and S.-Å. Tärnlund (eds.), *Logic Programming*, Academic Press, London, 1982.

[35] Heintze, N., Michaylov, S., Stuckey, P. J. and Yap, R., On Metaprogramming in CLP(R), in: *Proc. 1989 North-American Conf. on Logic Programming*, MIT Press, Cambridge, Mass., 1989.

[36] Hill, P. M. and Lloyd, J. W., Analysis of Metaprograms, in: H. Abramson and M. H. Rogers (eds.), *Meta-Programming in Logic Programming*, MIT Press, Cambridge, Mass., 1988.

[37] Hill, P. M. and Lloyd, J. W., *The Gödel Programming Language*, MIT Press, Cambridge, Mass., 1994.

[38] Huet, G., Résolution d'équations dans les langages d'ordre 1, 2, ..., ω, Ph.D. Thesis, Université Paris VII, Paris, 1976.

[39] Huet, G., Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, *J. ACM*, 27(4):797–821 (1982).

[40] Jaffar, J. and Lassez, J.-L., Constraint Logic Programming, in: *Proc. 14th ACM Symp. on Principles of Programming Languages*, 1987.

[41] Jaffar, J., Lassez, J.-L. and Maher, M. J., A Theory of Complete Logic Programs with Equality, *J. Logic Programming*, 3:211–223 (1984).

[42] Jaffar, J., Lassez, J.-L. and Maher, M. J., A Logic Programming Language Scheme, in: D. DeGroot and G. Lindstrom (eds.), *Logic Programming—Functions, Relations, and Equations*, Prentice-Hall, Englewood Cliffs, N.J., 1986.

[43] Jaffar, J. and Michaylov, S., Methodology and Implementation of a CLP System, in: J.-L. Lassez (ed.), *Proc. 4th Intl. Conf. on Logic Programming*, MIT Press, Cambridge, Mass., 1987.

[44] Jiang, Y. J., Ambivalent Logic as the Semantic Basis of Metalogic Programming: I, in: P. V. Hentenryck (ed.), *Proc. 11th Intl. Conf. on Logic Programming*, MIT Press, Cambridge, Mass., 1994.

[45] Kim, J. S. and Kowalski, R. A., A Metalogic Programming Approach to Multi-Agent Knowledge and Belief, in: V. Lifschitz (ed.), *Artificial Intelligence and Mathematical Theory of Computation*, Academic Press, 1991.

[46] Konolige, K., An Autoepistemic Analysis of Metalevel Reasoning in Logic Programming, in: A. Pettorossi (ed.), *Meta-Programming in Logic*, LNCS 649, Springer-Verlag, Berlin, 1992.

[47] Levi, G. and Ramundo, D., A Formalization of Metaprogramming for Real, in: D. S. Warren (ed.), *Logic Programming — Proc. 10th Intl. Conf. on Logic Programming*, MIT Press, Cambridge, 1993.

[48] Lim, P. and Stuckey, P. J., Meta Programming as Constraint Programming, in: S. Debray and M. Hermenegildo (eds.), *Proc. 1990 North-American Conf. on Logic Programming*, MIT Press, Cambridge, Mass., 1990.

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

[50] Maher, M. J., A Logic Programming View of CLP, *Proc. 10th Intl. Conf. on Logic Programming*, pp. 737–753 (1993).

[51] Martelli, A. and Montanari, U., An Efficient Unification Algorithm, *ACM TOPLAS*, 4:258–282 (1982).

[52] Martens, B. and De Schreye, D., Why Untyped Non-Ground Meta-Programming Is Not (Much of) a Problem, CW report 159, Computer Science Dept., Katholieke Univ. Leuven, 1992.

[53] Meier, M., Dufresne, P. and Villeneuve, D., SEPIA Manual, ECRC Technical Report TR-LP-36, ECRC, Munich, 1988.

[54] Moore, R., Reasoning about Knowledge and Action, in: *Proc. Fifth Intl. Joint Conf. on Artificial Intelligence*, Morgan Kaufmann, Los Altos, Calif., 1977.

[55] Naish, L., Negation and Quantifiers in Nu-Prolog, in: E. Shapiro (ed.), *Proc. 3rd Intl. Conf. on Logic Programming*, Springer-Verlag, Berlin, 1986.

[56] Robinson, J. A., A Machine-Oriented Logic Based on the Resolution Principle, *J. ACM*, 12:23–41 (1965).

[57] Robinson, J. A., Fast Unification (abstract), in: *Tagung über Automatisches Beweisen*, Mathematisches Forschungsinstitut Oberwolfach, 1976.

[58] Sato, T., Meta-Programming through a Truth Predicate, in: K. Apt (ed.), *Proc. Joint Intl. Conf. Symp. on Logic Programming 1992*, MIT Press, Cambridge, Mass., 1992.

[59] L. Sterling and E. Y. Shapiro (eds.), *The Art of Prolog*, MIT Press, Cambridge, Mass., 1986.

[60] Subrahmanian, V. S., Foundations of Metalogic Programming, in: H. Abramson and M. H. Rogers (eds.), *Meta-Programming in Logic Programming*, MIT Press, Cambridge, Mass., 1988.

[61] Takeuchi, A. and Furukawa, K., Partial Evaluation of Prolog Programs and its Applications to Metaprogramming, in: H. J. Kugler (ed.), *Information Processing 86*, North-Holland, 1986.

[62] Tarski, A., The Concept of Truth in Formalized Languages, in: *Logic, Semantics, Metamathematics*, Clarendon Press, Oxford, 1956.

[63] van Emden, M. H. and Kowalski, R. A., The Semantics of Predicate Logic as a Programming Language, *J. ACM*, 23(4):733–742 (1976).

[64] van Harmelen, F., Definable Naming Relations in Meta-level Systems, in: A. Pettorossi (ed.), *Meta-Programming in Logic*, LNCS 649, Springer-Verlag, Berlin, 1992.

[65] van Harmelen, F., Wielinga, B., Bredeweg, B., Schreiber, G., Karbach, W., Reinders, M., Voß, A., Akkermans, H., Bartsch-Spörl, B. and Vinkhuyzen, E., Knowledge-Level Reflection, in: *Enhancing the Knowledge Engineering Process – Contributions from ESPRIT*, Elsevier Science, Amsterdam, The Netherlands, 1992.

[66] P. van Hentenryck (ed.), *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, Mass., 1989.

[67] Weyhrauch, R. W., Prolegomena to a Theory of Mechanized Formal Reasoning, *Artificial Intelligence*, pp. 133–70 (1980).

[68] Yalcinap, L. U., Meta-Programming for Knowledge Based Systems in Prolog, Ph.D. Thesis, Dept. of Computer Engineering and Science, Case Western Reserve University, Cleveland, USA, 1991.

Appendix A

Proofs

A.1 Proofs of Chapter 3

Lemma 3.13 *The unification rewrite system R_U: (i) always terminates, no matter which choices are made; (ii) returns a triple $\langle\{false\},\{\},\{\}\rangle$ (a fail termination) or the triple $\langle H,F,E\rangle$ (a success termination), where H is a Herbrand assignment, F is a set of name bindings and $H \cup F$ is an assignment. Furthermore, E contains only equations of the form $t = \uparrow t'$ and $t = \downarrow t'$, where t is not a variable and t' is any term.*

Proof.
(i) Given a Herbrand assignment H and a set of equations E, we define a function Φ_U mapping the triple $\langle H,F,E\rangle$ into a triple of natural numbers (n_1,n_2,n_3). The first number, n_1, is the sum of occurrences in E of all the variables occurring at the left-hand side of the equations in $H \cup F$. The second number, n_2, is the number of occurrences in E of the pair of symbols '[' and ']'. The third number, n_3, is the number of equations in E. Let $\underset{U}{\succ}$ be the lexicographic ordering on N^3 (where N is the set of natural numbers). With the above ordering, N^3 becomes a well-founded set, that is, a set where no decreasing sequence exists. Thus, if we prove that any rewrite rule of R_U transforms a triple $\langle H,F,E\rangle$ into a triple $\langle H',F',E'\rangle$ such that $\Phi_U(\langle H,F,E\rangle) \underset{U}{\succ} \Phi_U(\langle H',F',E'\rangle)$, we have proved termination. In fact, rewrite rules U_1, U_4, U_5, U_6, U_7 and U_8 decrease n_3. Rewrite rules U_2 and U_3 decrease n_1. Rewrite rules U_9 and U_{10} increase n_3, but they decrease n_2.

(ii) Let $\langle H',F',E'\rangle$ be a triple containing a Herbrand assignment H', a set of name bindings F' and a set of equations E' such that $H' \cup F'$ is an assignment. We prove the case (ii) of the lemma by showing that every rule of R_U takes the triple $\langle H',F',E'\rangle$ into a new triple $\langle H'',F'',E''\rangle$ such that H'' is a Herbrand assignment, F'' is a set of name bindings and $H'' \cup F''$ is an assignment. In fact, the claim is trivially true for the rules U_1, U_2, U_3, U_9 and U_{10}, because they change only the set E'. Since the conditions for applying the rule U_4 on a selected equation $x = t$ require that H' and F' do not contain any equation of the form $x = t'$, that $H' \cup F' \cup \{x = t\}$ is an assignment and that $x = t$ is a binding, the set $H'' = H' \cup \{x = t\}$ is a Herbrand assignment. Since U_4 does not change F', F'' is still a set of name bindings and $H'' \cup F''$ is an assignment. Similar considerations hold for the rule U_5. Finally, if one of the rules U_6, U_7 or U_8 is applied, then the new triple is $\langle\{false\},\{\},\{\}\rangle$ and no rewrite rule is applicable (a fail termination).

The only equations in E that cannot be rewritten take the form $t = \uparrow t'$ and $t = \downarrow t'$, where t is not a variable. In fact, either E is empty (a fail termination) or the following hold. Because the rules U_2, U_3, U_4 and U_5 are not applicable, every

equation in E does not take the form $x = t$, where x is a variable and t a term. Furthermore, because the rules U_9 and U_{10} cannot be applied, the terms t and t' of any equation $t = t'$ in E are not compound. Thus, every equation in E has the form $t = \uparrow t'$ or $t = \downarrow t'$, where t is not a variable. ∎

Theorem 3.19 *The unification rewrite system R_U is canonical.*

Proof. By Lemma 3.13 R_U is terminating. Since R_U is left-linear and nonoverlapping, R_U is orthogonal. By the results of Huet [39], orthogonal systems are always confluent. Thus, R_U is confluent and therefore canonical. ∎

Theorem 3.20 *If $\langle H, F, E\rangle \overset{*}{\underset{U}{\Leftrightarrow}} \langle H', F', E'\rangle$, then $ET \models (H \cup F \cup E) \equiv (H' \cup F' \cup E')$.*

Proof. We prove the theorem by showing that every rewrite rule of R_U preserves equivalence under the equality theory ET.

(U_1) $\quad \langle H, F, E \cup \{t = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F, E\rangle$

By axiom 1 of ET, an equation $t = t$ is always satisfied. Thus, it holds that $ET \models (H \cup F \cup E \cup \{t = t\}) \equiv (H \cup F \cup E)$.

(U_2) $\quad \langle H \cup \{x = t'\}, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H \cup \{x = t'\}, F, E \cup \{t' = t\}\rangle$

By axioms 2 and 3 of ET, the set $\{x = t', x = t\}$ is equivalent to $\{x = t', t' = t\}$. Thus, it holds that $ET \models (H \cup \{x = t'\} \cup F \cup E \cup \{x = t\}) \equiv (H \cup \{x = t'\} \cup F \cup E \cup \{t' = t\})$.

(U_3) $\quad \langle H, F \cup \{x = t'\}, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F \cup \{x = t'\}, E \cup \{t' = t\}\rangle$

Similar to the previous proof.

(U_4) $\quad \langle H, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H \cup \{x = t\}, F, E\rangle$

(U_5) $\quad \langle H, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F \cup \{x = t\}, E\rangle$

Trivially provable.

(U_6) $\quad \langle H, F, E \cup \{x = t\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle \{\mathit{false}\}, \{\}, \{\}\rangle$

This rule can be applied if $x = t$ is neither a binding nor a name binding, or in $H \cup F \cup \{x = t\}$ x depends on itself. In the former case, the equation $x = t$ is never satisfied, since it is not well-defined. Therefore, $H \cup E \cup F \cup \{x = t\}$ is equivalent to $\{\mathit{false}\}$. In the latter case, the set $H \cup F \cup \{x = t\}$ is never satisfied because of axiom 11 of ET. The set $H \cup F \cup E \cup \{x = t\}$ is not satisfiable and therefore it is equivalent to $\{\mathit{false}\}$.

(U_7) $\quad \langle H, F, E \cup \{t = t'\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle \{\mathit{false}\}, \{\}, \{\}\rangle$

This rule can be applied if t and t' are constants, metaconstants, compound terms or compound term names of a different kind. The equation $t = t'$ is not satisfiable because of axiom 10 of ET. Thus, the set $H \cup F \cup E \cup \{t = t'\}$ is equivalent to $\{\mathit{false}\}$.

(U_8) $\quad \langle H, F, E \cup \{t = t'\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle \{\mathit{false}\}, \{\}, \{\}\rangle$

This rule can be applied if t and t' are distinct constants or metaconstants, compound terms with distinct principal symbols or compound term names of different length. The equation $t = t'$ is not satisfiable because of axioms 8 or 9 of ET. Thus, the set $H \cup F \cup E \cup \{t = t'\}$ is equivalent to $\{false\}$.

(U_9) $\quad \langle H, F, E \cup \{[f, t_1, \ldots, t_k] = [f, t'_1, \ldots, t'_k]\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}}$
$\langle H, F, E \cup \{t_1 = t'_1, \ldots, t_k = t'_k\}\rangle$

By axioms 4 and 6 of ET, the sets $\{[f, t_1, \ldots, t_k] = [f, t'_1, \ldots, t'_k]\}$ and $\{t_1 = t'_1, \ldots, t_k = t'_k\}$ are equivalent. Thus, it holds that $ET \models (H \cup F \cup E \cup \{[f, t_1, \ldots, t_k] = [f, t'_1, \ldots, t'_k]\}) \equiv (H \cup F \cup E \cup \{t_1 = t'_1, \ldots, t_k = t'_k\})$.

(U_{10}) $\quad \langle H, F, E \cup \{[t_0, \ldots, t_k] = [t'_0, \ldots, t'_k]\}\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H, F, E \cup \{t_0 = t'_0, \ldots, t_k = t'_k\}\rangle$

By axioms 5 and 7 of ET, the sets $\{[t_0, \ldots, t_k] = [t'_0, \ldots, t'_k]\}$ and $\{t_0 = t'_0, \ldots, t_k = t'_k\}$ are equivalent. Thus, it holds that
$ET \models (H \cup F \cup E \cup \{[t_0, \ldots, t_k] = [t'_0, \ldots, t'_k]\}) \equiv (H \cup F \cup E \cup \{t_0 = t'_0, \ldots, t_k = t'_k\})$. ∎

A.2 Proofs of Chapter 4

Proposition 4.1 *The following hold:*

(i) $\forall x \forall y \in (V_{Fun} \cup V_{Tn})(\uparrow x = y \rightarrow x = \downarrow y)$;

(ii) $\forall x \in (V_{Fun} \cup V_{Tn})\forall y(\downarrow x = y \rightarrow x = \uparrow y)$;

Proof.

(*i*) Suppose that $\forall x \quad \forall y \in (V_{Fun} \cup V_{Tn})(\uparrow x = y)$. Then $\forall x \forall y \in (V_{Fun} \cup V_{Tn})(\downarrow\uparrow x = \downarrow y)$ holds. Hence, by axiom 4 of NT, $\forall x \forall y \in (V_{Fun} \cup V_{Tn})(x = \downarrow y)$. The claim follows.

(*ii*) Assume that $\forall x \in (V_{Fun} \cup V_{Tn})\forall y(\downarrow x = y)$. Then $\forall x \in (V_{Fun} \cup V_{Tn})\forall y(\uparrow\downarrow x = \uparrow y)$ holds. Hence, by axiom 5 of NT, $\forall x \in (V_{Fun} \cup V_{Tn})\forall y(x = \uparrow y)$. The claim follows. ∎

Lemma 4.5 *The naming rewrite system R_N: (i) always terminates, no matter which choices are made; (ii) returns a pair $\langle H, \{false\}\rangle$ (a fail termination) or the pair $\langle H, E\rangle$ (a success termination) such that $E\widehat{H}$ contains $\uparrow$ or $\downarrow$ subexpressions only of the form $\uparrow x$, $\downarrow x$ or $\downarrow[x_0, t_1, \ldots, t_k]$, where $x_0 \in (V_{Fn} \cup V_{Tn})$. Furthermore, every name equation in $E\widehat{H}$ has the form $x = \uparrow y$, $x = \downarrow[x_0, t_1, \ldots, t_k]$, $x = t$ or $t = t'$, where x and y are variables, $x_0 \in (V_{Fn} \cup V_{Tn})$, and t and t' are compound terms.*

Proof.
(i) Given a Herbrand assignment H and a set of equations E, we define a function Φ_N mapping the pair $\langle H, E\rangle$ into a quadruple of natural numbers (n_1, n_2, n_3, n_4). The first number, n_1, is the sum of occurrences in E of terms of the form $\downarrow^i x = y$, $\uparrow^i x = t$ and $\downarrow^i [x, t_1, \ldots, t_k] = t$, where x is a variable and t is not a variable. The second number, n_2, is the number of occurrences of the pair of symbols '[' and ']' inside of $\uparrow$ and $\downarrow$ subexpressions of E. Finally, numbers n_3 and n_4 are the number of occurrences in E of the symbols $\downarrow$ and $\uparrow$, respectively. Let $\underset{N}{\succ}$ be the lexicographic ordering on N^4 (where N is the set of natural numbers). With this ordering, N^4 becomes a well-founded set, that is, a set where no decreasing sequence exists. Thus,

if we prove that any rewrite rule of R_N transforms a pair $\langle H, E\rangle$ into a pair $\langle H', E'\rangle$ such that $\Phi_N(\langle H, E\rangle) \underset{N}{\succ} \Phi_N(\langle H', E'\rangle)$, we have proved termination. In fact, rewrite rule N_1 decreases n_4, and rewrite rules N_2, N_3, N_5 and N_6 decrease n_2. Rewrite rule N_4 decreases n_3. Rewrite rules N_7 and N_8 decrease n_3, and possibly n_1 and n_2. Finally, rewrite rules N_9, N_{10} and N_{11} decrease n_1.

(ii) If the rule N_7 is applied, then R_N returns the pair $\langle H, \{\mathit{false}\}\rangle$ (a fail termination). Otherwise, every subexpression of the form $\uparrow t$ in $E\widehat{H}$, where t is not a variable, is rewritten by the rules N_1, N_2 or N_3; every subexpression $\downarrow t$ is rewritten by N_4, N_5 or N_6 if t is neither a variable nor of the form $[t_0, t_1, \ldots, t_k]$ with $t_0 \in (V_{Fn} \cup V_{Tn})$. Thus, the only remaining $\uparrow$ or $\downarrow$ subexpressions take the form $\uparrow x$, $\downarrow x$ or $\downarrow[x_0, t_1, \ldots, t_k]$, where x is a variable and $x_0 \in (V_{Fn} \cup V_{Tn})$.

Now, we want to show that every name equation in $E\widehat{H}$ takes the form $x = \uparrow y$, $x = \downarrow[x_0, t_1, \ldots, t_k]$, $x = t$ or $t = t'$, where x and y are variables, $x_0 \in (V_{Fn} \cup V_{Tn})$, and t and t' are compound terms. Note first that every name equation has the form $x = t$ or $t = t'$, where t (in the first case) and t or t' (in the second) must contain at least a $\uparrow$ or $\downarrow$ subexpression. Suppose that $E\widehat{H}$ contains an equation $x = t$. Then t must be a compound term or (as previously showed) a term of the form $\uparrow y$ or $\downarrow[x_0, t_1, \ldots, t_k]$. (Observe that t cannot be of the form $\downarrow y$, because by N_{10} a name equation $x = \downarrow y$ is rewritten into $y = \uparrow x$.)

If $E\widehat{H}$ contains an equation $t = t'$, then t and t' must be compound terms. In fact, $E\widehat{H}$ cannot contain any equation of the form $t = \uparrow t'$, otherwise $\uparrow t'$ is computed, or, if it is a variable, by N_9 $t = \uparrow t'$ is rewritten into $t' = \downarrow t$. Neither can $E\widehat{H}$ contain equations of the form $t = \downarrow t'$, because, if t' is a variable or of the form $[x_0, t_1, \ldots, t_k]$, then the equation is rewritten as $t' = \uparrow t$, and in the remaining cases $\downarrow t'$ is computed. ∎

Theorem 4.6 *The naming rewrite system R_N is canonical.*

Proof. By Lemma 4.5 R_N is terminating. Since R_N is left-linear and nonoverlapping, R_N is orthogonal. By the results of Huet [39], orthogonal systems are always confluent. Thus, R_N is confluent and therefore canonical. ∎

Theorem 4.7 *Let H be a Herbrand assignment and E a set of equations. If $\langle H, E\rangle \overset{*}{\underset{N}{\Leftrightarrow}} \langle H, E'\rangle$, then $NT \models H \cup E \equiv H \cup E'$.*

Proof. Observe first that by Theorem 3.7 for every Herbrand assignment H there exists an equivalent substitution $\widehat{H}$, and that $H \cup E$ is equivalent (under the equality theory ET) to $H \cup E\widehat{H}$. This allows one to apply the rewrite rules of R_N to subexpressions of $H \cup E\widehat{H}$ rather than $H \cup E$.

We prove the theorem by showing that every rewrite rule of R_N preserves equivalence under the equality theory NT.

(N_1) If $t\widehat{H}$ is a constant, metaconstant, predicate name, function name or a functor name c^n, $n \geq 0$, then
$\langle H, E[\uparrow t]\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E[c^{n+1}]\rangle$

By axiom 6 of NT, $\uparrow c^n = c^{n+1}$, it holds that $NT \models (H \cup E[\uparrow t]) \equiv (H \cup E[c^{n+1}])$.

(N_2) If $t\widehat{H}$ is a compound term $[f, t_1, \ldots, t_k]$, where $f \in Fs$, then
$\langle H, E[\uparrow t]\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E[[f^1, \uparrow t_1, \ldots, \uparrow t_k]]\rangle$

By axiom 7 of NT, $\forall(\uparrow[f, t_1, \ldots, t_k] = [f^1, \uparrow t_1, \ldots, \uparrow t_k])$, it holds that $NT \models (H \cup E[\uparrow t]) \equiv (H \cup E[[f^1, \uparrow t_1, \ldots, \uparrow t_k]])$.

(N_3) If $t\widehat{H}$ is a compound term name or an atom name $[t_0, t_1, \ldots, t_k]$, then
$\langle H, E[\uparrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[[\uparrow t_0, \uparrow t_1, \ldots, \uparrow t_k]]\rangle$

By axiom 8 of NT, $\forall(\uparrow[t_0, t_1, \ldots, t_k] = [\uparrow t_0, \uparrow t_1, \ldots, \uparrow t_k])$, it holds that $NT \models (H \cup E[\uparrow t]) \equiv (H \cup E[[\uparrow t_0, \uparrow t_1, \ldots, \uparrow t_k]])$.

(N_4) If $t\widehat{H}$ is a metaconstant or a functor name c^n, $n > 0$, then
$\langle H, E[\downarrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[c^{n-1}]\rangle$

By case (i) of Proposition 4.1 and axiom 6 of NT, $\downarrow c^n = c^{n-1}$ (this corresponds to axiom 9 in Section 4). Thus, $NT \models (H \cup E[\downarrow t]) \equiv (H \cup E[c^{n-1}])$ holds.

(N_5) If $t\widehat{H}$ is a compound term name $[f^1, t_1, \ldots, t_k]$, where $f^1 \in Fn$, then
$\langle H, E[\downarrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[[f, \downarrow t_1, \ldots, \downarrow t_k]]\rangle$

By case (i) of Proposition 4.1 and axiom 7 of NT, $\forall(\downarrow[f^1, t_1, \ldots, t_k] = [f, \downarrow t_1, \ldots, \downarrow t_k])$ (this corresponds to axiom 10 in Section 4). Thus, $NT \models (H \cup E[\downarrow t]) \equiv (H \cup E[[f, \downarrow t_1, \ldots, \downarrow t_k]])$ holds.

(N_6) If $t\widehat{H}$ is a compound term name $[t_0, t_1, \ldots, t_k]$, where $t_0 \in (Fun \cup V_{Fun})$, then
$\langle H, E[\downarrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[[\downarrow t_0, \downarrow t_1, \ldots, \downarrow t_k]]\rangle$

By case (i) of Proposition 4.1 and axiom 8 of NT, $\forall(\downarrow[t_0, t_1, \ldots, t_k] = [\downarrow t_0, \downarrow t_1, \ldots, \downarrow t_k])$ (this corresponds to axiom 11 in Section 4). Thus, $NT \models (H \cup E[\downarrow t]) \equiv (H \cup E[[\downarrow t_0, \downarrow t_1, \ldots, \downarrow t_k]])$ holds.

(N_7) If $t\widehat{H}$ is a constant, a predicate name, a function name, an atom name or compound term, then
$\langle H, E[\downarrow t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, \{false\}\rangle$

If t is not a expression name, then $\downarrow t$ is not a well-defined expression of the language.

(N_8) For every expression t either of the form $\downarrow\uparrow t'$, or of the form $\uparrow\downarrow t'$ and $t' \in (Fun \cup Tn)$
$\langle H, E[t]\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E[t']\rangle$

By axiom 4 of NT, $\forall x(\downarrow\uparrow x = x)$, and by axiom 5 of NT, $\forall x \in (V_{Fun} \cup V_{Tn})(\uparrow\downarrow x = x)$, it holds that $NT \models (H \cup E[t]) \equiv (H \cup E[t'])$.

(N_9) If $i > 0$, $x\widehat{H}$ is a variable and $t\widehat{H}$ is not a variable, then
$\langle H, E \cup \{\uparrow^i x = t\}\rangle \underset{N}{\overset{1}{\Leftrightarrow}} \langle H, E \cup \{x = \downarrow^i t\}\rangle$

By applying case (i) of Proposition 4.1 i times.

(N_{10}) If $i > 0$ and $x\widehat{H}$ is a variable, then
$\langle H, E \cup \{\downarrow^i x = t\}\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E \cup \{x = \uparrow^i t\}\rangle$

By applying case (ii) of Proposition 4.1 i times.

(N_{11}) If $i > 0$, $x\widehat{H}$ is $\downarrow^i[x_0, t_1, \ldots, t_k]$, $x_0 \in (V_{Fn} \cup V_{Tn})$ and $t\widehat{H}$ is not a variable
$\langle H, E \cup \{x = t\}\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E \cup \{[x_0, t_1, \ldots, t_k] = \uparrow^i t\}\rangle$

By applying case (ii) of Proposition 4.1 i times. ∎

A.3 Proofs of Chapter 5

Lemma 5.1 *The naming rewrite system R_{UN}: (i) always terminates, no matter which choices are made; (ii) returns a triple $\langle\{false\}, \{\}, \{\}\rangle$ (a fail termination) or the triple $\langle H, F, \{\}\rangle$ (a success termination), where H is a Herbrand assignment, F is a set of name equations and $H \cup F$ is an assignment. Furthermore, every name equation in $F\widehat{H}$ has the form $x = \uparrow y$, $x = \downarrow[x_0, t_1, \ldots, t_k]$ or $x = t$, where x and y are variables, $x_0 \in (V_{Fn} \cup V_{Tn})$, $t_1, \ldots, t_k$ are term names and t is a compound term. Every $\uparrow$ and $\downarrow$ subexpression of $x = t$ takes the form $\uparrow y$, $\downarrow y$ or $\downarrow[x_0, t_1, \ldots, t_k]$.*

Proof.
(i) Observe first that the rewrite rules of R_U and R_N do not interact with each other. Since by Lemma 3.13 and 4.5 case (i) the rewrite systems R_U and R_N terminate, UN_2, UN_3 and UN_4 can only be applied a finite number of times. Furthermore, UN_1 cannot be applied indefinitely. In fact, UN_1 moves name equations containing ground $\uparrow$ or $\downarrow$ subexpressions from F to E, and the only rule that moves back name equations, UN_4, cannot be applied if UN_2 and UN_3 can. Thus, all the ground $\uparrow$ or $\downarrow$ expressions are computed before the equation can be moved back. Then the rule UN_1 is not any longer applicable.

(ii) If either R_N or R_U end in a fail termination, then UN_2, respectively UN_4, return the triple $\langle\{false\}, \{\}, \{\}\rangle$. Otherwise, R_{UN} returns a triple $\langle H, F, \{\}\rangle$, where H and F are a Herbrand assignment and a set of name equations, respectively, and $H \cup F$ is an assignment. In fact, the claim is trivially true for the rule UN_1, UN_2 and UN_3, while for the rule UN_4 this holds by Theorem 3.13.

After having computed all the $\uparrow$ and $\downarrow$ subexpressions in $E\widehat{H}$, by Lemma 4.5 every name equation in $E\widehat{H}$ takes the form $x = \uparrow y$, $x = \downarrow[x_0, t_1, \ldots, t_k]$, $x = t$ or $t = t'$, where x and y are variables, $x_0 \in (V_{Fn} \cup V_{Tn})$, and t and t' are compound terms. By applying UN_4, the equations $x = \uparrow y$, $x = \downarrow[x_0, t_1, \ldots, t_k]$ and $x = t$ are put in F, while $t = t'$ is decomposed. Thus, the remaining set is empty. Finally, by Lemma 4.5, every $\uparrow$ or $\downarrow$ subexpression in name equations in F takes the form $\uparrow x$, $\downarrow x$ or $\downarrow[x_0, t_1, \ldots, t_k]$. ∎

Theorem 5.2 *The rewrite system R_{UN} is canonical.*

Proof. By Lemma 5.1 R_{UN} is terminating. Since R_{UN} is left-linear and nonoverlapping, R_{UN} is orthogonal. Thus, by the results of Huet [39], R_{UN} is confluent and therefore canonical. ∎

Theorem 5.3 *If* $\langle H, F, E\rangle \overset{*}{\underset{UN}{\Leftrightarrow}} \langle H', F', E'\rangle$, *then* $NET \models (H \cup F \cup E) \equiv (H' \cup F' \cup E')$.

Proof. We prove the theorem by showing that every rewrite rule of R_{UN} preserves equivalence. In fact, the claim is obviously true for UN_1. Since rule UN_2 and UN_3 require that $\langle H, E\rangle \overset{1}{\underset{N}{\Leftrightarrow}} \langle H, E'\rangle$, by Theorem 4.7 $NT \models H \cup E \equiv H \cup E'$ holds. As the theory NET subsumes NT, $NET \models (H \cup F \cup E) \equiv (H \cup F \cup E')$. Finally, as UN_4 requires that $\langle H, F, E\rangle \overset{1}{\underset{U}{\Leftrightarrow}} \langle H', F', E'\rangle$, by Theorem 3.20 $ET \models (H \cup F \cup E) \equiv (H' \cup F' \cup E')$ holds. Hence, $NET \models (H \cup F \cup E) \equiv (H' \cup F' \cup E')$. The claim follows. ∎

Theorem 5.4 *If* $\langle H, F, E\rangle \overset{*}{\underset{UN}{\Leftrightarrow}} \langle H', F', \{\}\rangle$, *then the existential closure of* $H' \cup F'$ *is true for every interpretation.*

Proof. We can substitute any ground element of the language to the variables $\{y_1, \ldots, y_n\}$ of $F = \{x_1 = t_1, \ldots, x_k = t_k\}$ that appear in the terms $t_1, \ldots, t_k$, and then compute the $\uparrow$ and $\downarrow$ expressions of F with respect to the naming scheme. By definition, no x_i is included in $\{y_1, \ldots, y_n\}$. Therefore, we can assign to $\{x_1, \ldots, x_k\}$ the denotations of the resulting $t_1, \ldots, t_k$. Finally, we can assign any element of the domain of the interpretation to the remaining variables $\{y_{n+1}, \ldots, y_m\}$ of $H = \{x_{k+1} = t_{k+1}, \ldots, x_l = t_l\}$ that appear in the terms $t_{k+1}, \ldots, t_l$. Because $H' \cup F'$ is an assignment, the variables $\{x_{k+1}, \ldots, x_l\}$ are distincts from $\{x_1, \ldots, x_k\}$. Thus, we can assign to $\{x_{k+1}, \ldots, x_l\}$ the resulting denotations of $t_{k+1}, \ldots, t_l$. ∎

A.4 Proofs of Chapter 7

Proposition 7.16 *Let* (P, E) *be a logic program and* $\leftarrow A_1, \ldots, A_k$ *be a definite goal. Then* $(P, E) \cup \{\leftarrow A_1, \ldots, A_k\}$ *is reflectively E-unsatisfiable iff* $A_1 \wedge \ldots \wedge A_k$ *is a reflective logical E-consequence of* (P, E).

Proof. Suppose that $(P, E) \cup \{\leftarrow A_1, \ldots, A_k\}$ is reflectively E-unsatisfiable. Let I be any extended E-interpretation of (P, E). Suppose that I is a reflective E-model of (P, E). As $(P, E) \cup \{\leftarrow A_1, \ldots, A_k\}$ is reflectively E-unsatisfiable, I cannot be a reflective E-model for $\sim (A_1 \wedge \ldots \wedge A_k)$. Hence, each atom A_i, $1 \leq i \leq k$, is true under I. Thus, I is a reflective E-model for A_i and, therefore, $A_1 \wedge \ldots \wedge A_k$ is a reflective logical E-consequence of (P, E).

Conversely, suppose that $A_1 \wedge \ldots \wedge A_k$ is a reflective logical E-consequence of (P, E). Let I be an extended E-interpretation of (P, E) and suppose that I is a reflective E-model for (P, E). Then I is also a reflective E-model for A_i, $1 \leq i \leq k$. Thus, each A_i is true under I. Hence, I is not a reflective E-model for $\sim (A_1 \wedge \ldots \wedge A_k)$ and, therefore, $(P, E) \cup \{\leftarrow A_1, \ldots, A_k\}$ is reflectively E-unsatisfiable. ∎

Proposition 7.18 *Let* (P, E) *be a logic program. The mapping* $T_{(P,E)}$ *is continuous.*

Proof. Let X be a subset of $2^{B_{(P,E)}}$. Note first that $\{, (A_1), \ldots, , (A_m)\} \subseteq lub(X)$ iff $\{, (A_1), \ldots, , (A_m)\} \subseteq I$, for some $I \in X$. In order to show that $T_{(P,E)}$ is continuous, we have to show $T_{(P,E)}(lub(X)) = lub(T_{(P,E)}(X))$, for each directed subset X. Now we have that , $(A) \in T_{(P,E)}(lub(X))$ iff one of the points (i)-(iii) holds.

(*i*) $A \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ is a ground instance of a clause in P, $E \models e_i$ for all i, $1 \leq i \leq q$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq lub(X)$

iff $A \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ is a ground instance of a clause in P, $E \models e_i$ for all i, $1 \leq i \leq q$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq I$, for some $I \in X$

iff , $(A) \in T_{(P,E)}(I)$, for some $I \in X$

iff , $(A) \in lub(T_{(P,E)}(X))$

(ii) A is $solve([p^1, t_1, \ldots, t_n])$, $p(t'_1, \ldots, t'_n) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ is a ground instance of a clause in P, $E \models e_i$ for all i, $1 \leq i \leq q$, $E \models t_l = \uparrow t'_l$ for all l, $1 \leq l \leq n$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq lub(X)$

iff A is $solve([p^1, t_1, \ldots, t_n])$, $p(t'_1, \ldots, t'_n) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ is a ground instance of a clause in P, $E \models e_i$ for all i, $1 \leq i \leq q$, $E \models t_l = \uparrow t'_l$ for all l, $1 \leq l \leq n$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq I$, for some $I \in X$

iff , $(solve([p^1, t_1, \ldots, t_n])) \in T_{(P,E)}(I)$, for some $I \in X$

iff , $(solve([p^1, t_1, \ldots, t_n])) \in lub(T_{(P,E)}(X))$

(iii) A is $p(t_1, \ldots, t_n)$, $solve([p^1, t'_1, \ldots, t'_n]) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ is a ground instance of a clause in P, $E \models e_i$ for all i, $1 \leq i \leq q$, $E \models t_l = \downarrow t'_l$ for all l, $1 \leq l \leq n$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq lub(X)$

iff A is $solve([p^1, t_1, \ldots, t_n])$, $p(t'_1, \ldots, t'_n) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ is a ground instance of a clause in P, $E \models e_i$ for all i, $1 \leq i \leq q$, $E \models t_l = \uparrow t'_l$ for all l, $1 \leq l \leq n$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq I$, for some $I \in X$

iff , $(p(t_1, \ldots, t_n)) \in T_{(P,E)}(I)$, for some $I \in X$

iff , $(p(t_1, \ldots, t_n)) \in lub(T_{(P,E)}(X))$ ∎

Proposition 7.19 *Let (P, E) be a logic program and I an extended E-interpretation. Then I is a reflective E-model for (P, E) if and only if $T_{(P,E)}(I) \subseteq I$.*

Proof. I is a reflective E-model for (P, E) iff

(i) for every ground instance $p(t_1, \ldots, t_n) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ of each clause in P, where p is a predicate symbol distinct from *solve*, we have that $E \models e_i$ for all i, $1 \leq i \leq q$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq I$ implies:

- , $(p(t_1, \ldots, t_n)) \in I$, because I is an E-model of each axiom of P;
- , $(solve([p^1, t'_1, \ldots, t'_n])) \in I$, for every ground term t'_l, $1 \leq l \leq n$, such that $E \models t'_l = \uparrow t_l$, by the definition of extended E-interpretation.

(ii) for every ground instance $solve([p^1, t_1, \ldots, t_n]) \leftarrow e_1, \ldots, e_q, A_1, \ldots, A_m$ of each clause in P, we have that $E \models e_i$ for all i, $1 \leq i \leq q$, and $\{, (A_1), \ldots, , (A_m)\} \subseteq I$ implies:

- , $(solve([p^1, t_1, \ldots, t_n])) \in I$, because I is an E-model of each axiom of P;
- , $(p(t'_1, \ldots, t'_n)) \in I$, for every ground term t'_l, $1 \leq l \leq n$, such that $E \models t'_l = \downarrow t_l$, because I is an E-model of a reflection axiom $p(t'_1, \ldots, t'_n) \leftarrow t'_1 = \downarrow t_1, \ldots, t'_n = \downarrow t_n, solve([p^1, t_1, \ldots, t_n])$

iff $T_{(P,E)}(I) \subseteq I$. ∎

Theorem 7.25 *Let (P, E) be a logic program and G be a definite goal of the form $\leftarrow A_1, \ldots, A_k$ $(k \geq 0)$. Let H be a Herbrand assignment, F a set of name equations and H' be an E-solution of F. Suppose $\langle H, F \rangle$ is an answer for $(P, E) \cup \{G\}$ such that $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is ground. Then the following are equivalent:*

(a) $\langle H, F\rangle$ *is correct.*

(b) $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ *is true w.r.t. every reflective* E*-model of* (P, E).

(c) $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ *is true w.r.t. the least reflective* E*-model of* (P, E).

Proof.
$(a) \Rightarrow (c)$
By the definition of correct answer.
$(c) \Rightarrow (b) \Rightarrow (a)$
$(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is true w.r.t. the least reflective E-model of (P, E)
implies $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is true w.r.t. all reflective E-models of (P, E)
implies $\sim (A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is false w.r.t. all reflective E-models of (P, E)
implies $(P, E) \cup \{\sim (A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}\}$ has no reflective E-models
implies $(P, E) \cup \{\sim (A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}\}$ is reflectively E-unsatisfiable
implies $(A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'}$ is a reflective logical E-consequence of (P, E), by Proposition 7.16
implies $\langle H, F\rangle$ is correct, because H' is an E-solution of F. ∎

Theorem 7.28 (Soundness of SLD*-resolution) *Let* E *be a Horn clause equality theory and* R_E *a rewrite system for* E. *Let* (P, E) *be a logic program and* G *a definite goal. If* R_E *is correct with respect to* E, *then every computed answer for* $(P, E) \cup \{G\}$ *is a correct answer for* $(P, E) \cup \{G\}$.

Proof. Let G be a goal of the form $\leftarrow A_1, \ldots, A_k$ $(k \geq 0)$, and let $\langle H_n, F_n\rangle$ be the pair containing the Herbrand assignment and the set of name equations of the n-th step of the SLD*-refutation of $(P, E) \cup \{G\}$. We have to show that, for every E-solution H' of F_n, $\forall(G\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E). The result is proved by induction on the length of the SLD*-refutation.

Suppose that $n = 1$. This means that G is a goal of the form $\leftarrow A_1$, the initial state is $\langle\{A_1\}, \{\}, \{\}\rangle$ and one of the following cases applies.

(i) A_1 is an atom of the form $p(t_1, \ldots, t_h)$. P has a unit clause of the form $p(t'_1, \ldots, t'_h) \leftarrow$ and $\langle\{\}, \{\}, \{t_1 = t'_1, \ldots, t_h = t'_h\}\rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_1, \{\}, \{\}\rangle$. Note that, as unit clauses do not contain $\uparrow$ and $\downarrow$ operators, the set F_1 of name equations is the empty set. As R_E is correct w.r.t. E, $p(t_1, \ldots, t_h)\widehat{H_1}$ is an instance of $p(t'_1, \ldots, t'_h)$. Thus, $\forall(p(t_1, \ldots, t_h)\widehat{H_1})$ is a reflective logical E-consequence of (P, E).

(ii) A_1 is an atom of the form $p(t_1, \ldots, t_h)$, where p is some predicate symbol distinct from *solve*. P has a unit clause of the form $solve([p^1, t'_1, \ldots, t'_h]) \leftarrow$ and $\langle\{\}, \{\}, \{t_1 = \downarrow t'_1, \ldots, t_h = \downarrow t'_h\}\rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_1, F_1, \{\}\rangle$. For every E-solution H' of F_1, $solve([p^1, t'_1, \ldots, t'_h])\widehat{H_1}\widehat{H'}$ is a logical E-consequence of (P, E), being an instance of a unit clause of (P, E). As R_E is correct w.r.t. E, by reflection axioms, $\forall(p(t_1, \ldots, t_h)\widehat{H_1}\widehat{H'})$ is a reflective logical E-consequence of (P, E).

(iii) A_1 is an atom of the form $solve([p^1, t_1, \ldots, t_h])$. P has a unit clause of the form $p(t'_1, \ldots, t'_h) \leftarrow$ and $\langle\{\}, \{\}, \{t_1 = \uparrow t'_1, \ldots, t_h = \uparrow t'_h\}\rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_1, F_1, \{\}\rangle$. As $p(t'_1, \ldots, t'_h) \leftarrow$ is a unit clause, $\forall(p(t'_1, \ldots, t'_h)\widehat{H_1}\widehat{H'})$ is a reflective logical E-consequence of (P, E) for every E-solution H' of F_1. As R_E is correct w.r.t. E, by the definition of reflective E-model, $\forall(solve([p^1, t_1, \ldots, t_h])\widehat{H_1}\widehat{H'})$ is also a reflective logical E-consequence of (P, E).

Next, suppose that the result holds for computed answers coming from SLD*-refutations of length $n \Leftrightarrow 1$. Let C be the first input clause and A_m the selected atom in G. Let H' be an E-solution of F_n. One of the following cases applies.

(*i*) A_m is $p(t_1, \ldots, t_h)$. C is $p(t'_1, \ldots, t'_h) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$ $(q \geq 0, r \geq 0)$, and $\langle H_{n-1}, F_{n-1}, \{t_1 = t'_1, \ldots, t_h = t'_h, e_1, \ldots, e_q\}\rangle \overset{*}{\underset{E}{\Leftrightarrow}} \langle H_n, F_n, \{\}\rangle$. By the correctness of R_E and by the induction hypothesis, $\forall((A_1 \wedge \ldots \wedge A_{m-1} \wedge B_1 \wedge \ldots \wedge B_r \wedge A_{m+1} \wedge \ldots \wedge A_k)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E). Thus, if $r > 0$, $\forall((B_1 \wedge \ldots \wedge B_r)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E), and, consequently, $\forall(p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'})$, is also a reflective logical E-consequences of (P, E). Hence, $\forall((A_1 \wedge \ldots \wedge A_k)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E). Otherwise, if $r = 0$, then $\forall(p(t'_1, \ldots, t'_h)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E) because $E \models (e_1 \wedge \ldots \wedge e_q)\widehat{H'}$. Thus, $p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'}$ is a reflective logical E-consequence of (P, E) being an instance of $p(t'_1, \ldots, t'_h)$.

(*ii*) A_m is $p(t_1, \ldots, t_h)$, where p is some predicate symbol distinct from *solve*. C is $solve([p^1, t'_1, \ldots, t'_h]) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$ $(q \geq 0, r \geq 0)$ and $\langle H_{n-1}, F_{n-1}, \{t_1 = \downarrow t'_1, \ldots, t_h = \downarrow t'_h, e_1, \ldots, e_q\}\rangle \overset{*}{\underset{E}{\Leftrightarrow}} \langle H_n, F_n, \{\}\rangle$. By the correctness of R_E and by the induction hypothesis, $\forall((A_1 \wedge \ldots \wedge A_{m-1} \wedge B_1 \wedge \ldots \wedge B_r \wedge A_{m+1} \wedge \ldots \wedge A_k)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E), and consequently, also $\forall(solve([p^1, t'_1, \ldots, t'_h])\widehat{H_n}\widehat{H'})$ does. By reflection axioms, $\forall(p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E).

(*iii*) A_m is $solve([p^1, t_1, \ldots, t_h])$, C is $p(t'_1, \ldots, t'_h) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$ $(q \geq 0, r \geq 0)$, and $\langle H_{n-1}, F_{n-1}, \{t_1 = \uparrow t'_1, \ldots, t_h = \uparrow t'_h, e_1, \ldots, e_q\}\rangle \overset{*}{\underset{E}{\Leftrightarrow}} \langle H_n, F_n, \{\}\rangle$. By the correctness of R_E and by the induction hypothesis, $\forall((A_1 \wedge \ldots \wedge A_{m-1} \wedge B_1 \wedge \ldots \wedge B_r \wedge A_{m+1} \wedge \ldots \wedge A_k)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E). Consequently, also $\forall(p(t'_1, \ldots, t'_h)\widehat{H_n}\widehat{H'})$ is reflective logical E-consequence of (P, E). By the definition of reflective E-model, $\forall(solve([p^1, t_1, \ldots, t_h])\widehat{H_n}\widehat{H'})$ is also a reflective logical E-consequence (P, E). Hence, $\forall((A_1 \wedge \ldots \wedge A_k)\widehat{H_n}\widehat{H'})$ is a reflective logical E-consequence of (P, E). ∎

Corollary 7.29 *Let E be a Horn clause equality theory and R_E a correct rewrite system for E. Let (P, E) be a logic program and G a definite goal. Suppose that there exists an SLD*-refutation of $(P, E) \cup \{G\}$. Then $(P, E) \cup \{G\}$ is reflectively E-unsatisfiable.*

Proof. Let G be the goal $\leftarrow A_1, \ldots, A_k$. As R_E is correct w.r.t. E, by Theorem 7.28, the computed answer $\langle H, F\rangle$ of $(P, E) \cup \{G\}$ is correct. Thus, for every E-solution H' of F, $\forall((A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H'})$ is a reflective logical E-consequence of (P, E). It follows that $(P, E) \cup \{G\}$ is reflectively E-unsatisfiable. ∎

Corollary 7.30 *If a ground atom A belongs to the success set of a logic program (P, E), then , (A) is contained in the least reflective E-model of (P, E).*

Proof. Suppose that $(P, E) \cup \{\leftarrow A\}$ has an SLD*-refutation with computed answer $\langle H, F\rangle$. As R_E is correct w.r.t. E and A is ground, by Theorem 7.28, , (A) is a reflective logical E-consequence of (P, E). Hence , (A) is in the least reflective E-model of (P, E). ∎

Theorem 7.32 *Let E be a Horn clause equality theory and R_E a correct rewrite system for E. Let (P, E) be a logic program and G a definite goal $\leftarrow A_1, \ldots, A_k$. Suppose $(P, E) \cup \{G\}$ has an SLD*-refutation of length n with computed answer $\langle H_n, F_n \rangle$. Then $\cup_{j=1}^{k} \Psi(A_j \widehat{H_n} \widehat{H'}) \subseteq T_{(P,E)} \uparrow n$ for every E-solution H' of F_n.*

Proof. The result is proved by induction on the length n of the SLD*-refutation.

Suppose that $n = 1$. Then G is a goal of the form $\leftarrow A_1$ and one of the following cases holds.

(*i*) A_1 is $p(t_1, \ldots, t_h)$. P has a unit clause of the form $p(t'_1, \ldots, t'_h) \leftarrow$ and $\langle \{\}, \{\}, \{t_1 = t'_1, \ldots, t_h = t'_h\} \rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_1, \{\}, \{\} \rangle$. As R_E is correct w.r.t. E, $p(t_1, \ldots, t_h)\widehat{H_1}$ is an instance of $p(t'_1, \ldots, t'_h)$. Note that, as unit clauses do not contain $\uparrow$ and $\downarrow$ operators, the set F_1 of name equations is the empty set. Clearly, $\Psi(p(t'_1, \ldots, t'_h)) \subseteq T_{(P,E)} \uparrow 1$ and so does $\Psi(p(t_1, \ldots, t_h)\widehat{H_1})$.

(*ii*) A_1 is an atom $p(t_1, \ldots, t_h)$, where p is any predicate symbol distinct from *solve*. P has a unit clause of the form $solve([p^1, t'_1, \ldots, t'_h]) \leftarrow$ and $\langle \{\}, \{\}, \{t_1 = \downarrow t'_1, \ldots, t_h = \downarrow t'_h\} \rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_1, F_1, \{\} \rangle$. Clearly, for every E-solution H' of F_1, $\Psi(solve([p^1, t'_1, \ldots, t'_h])\widehat{H_1}\widehat{H'}) \subseteq T_{(P,E)} \uparrow 1$, and so does $\Psi(p(t_1, \ldots, t_h)\widehat{H_1}\widehat{H'})$, by the definition of $T_{(P,E)}$ and by correctness of R_E.

(*iii*) A_1 is $solve([p^1, t_1, \ldots, t_h])$. P has a unit clause of the form $p(t'_1, \ldots, t'_h) \leftarrow$ and $\langle \{\}, \{\}, \{t_1 = \uparrow t'_1, \ldots, t_h = \uparrow t'_h\} \rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_1, F_1, \{\} \rangle$. For every E-solution H' of F_1, $\Psi(p(t'_1, \ldots, t'_h)\widehat{H_1}\widehat{H'}) \subseteq T_{(P,E)} \uparrow 1$ and so does $\Psi(solve([p^1, t_1, \ldots, t_h])\widehat{H_1}\widehat{H'})$ by the definition of $T_{(P,E)}$ and by correctness of R_E.

Next suppose that the result holds for SLD*-refutations of length $n \Leftrightarrow 1$ and consider a refutation of length n. Let A_j be an atom in G. Suppose first that A_j is not the selected atom in G. Then $A_j\widehat{H_1}$ is an atom of G_1, the second goal of the SLD*-refutation. The induction hypothesis implies that $\Psi(A_j\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)} \uparrow (n \Leftrightarrow 1)$ for every E-solution H' of F_n. By the monotonicity of $T_{(P,E)}$, this holds $T_{(P,E)} \uparrow (n \Leftrightarrow 1) \subseteq T_{(P,E)} \uparrow n$.

Now suppose that A_j is the selected atom in G. Let C be the first input clause. One of the following cases holds. (In the remaining of the proof, let H' be an E-solution of F_n.)

(*i*) A_j is $p(t_1, \ldots, t_h)$. C is $p(t'_1, \ldots, t'_h) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$ $(q \geq 0, r \geq 0)$. Then $\langle H_{n-1}, F_{n-1}, \{t_1 = t'_1, \ldots, t_h = t'_h, e_1, \ldots, e_q\} \rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_n, F_n, \{\} \rangle$, and, by correctness of R_E, $p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'}$ is an instance of $p(t'_1, \ldots, t'_h)$. If $r = 0$, we have $\Psi(p(t'_1, \ldots, t'_h)) \subseteq T_{(P,E)} \uparrow 1$. Thus $\Psi(p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'}) \subseteq \Psi(p(t'_1, \ldots, t'_h)) \subseteq T_{(P,E)} \uparrow 1 \subseteq T_{(P,E)} \uparrow n$. If $r > 0$, by the induction hypothesis, $\Psi(B_i\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)} \uparrow (n \Leftrightarrow 1)$ for all i, $1 \leq i \leq r$. By the definition of $T_{(P,E)}$, we have $\Psi(p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)} \uparrow n$.

(*ii*) A_j is an atom $p(t_1, \ldots, t_h)$, where p is any predicate symbol distinct from *solve*. C is $solve([p^1, t'_1, \ldots, t'_h]) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$ $(q \geq 0, r \geq 0)$. Then $\langle H_{n-1}, F_{n-1}, \{t_1 = \downarrow t'_1, \ldots, t_h = \downarrow t'_h, e_1, \ldots, e_q\} \rangle \underset{E}{\overset{*}{\Leftrightarrow}} \langle H_n, F_n, \{\} \rangle$. If $r = 0$, we have $\Psi(solve([p^1, t'_1, \ldots, t'_h])\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)} \uparrow 1$, by correctness of R_E. Thus, by the definition of $T_{(P,E)}$, $\Psi(p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)} \uparrow 1$. If $r > 0$, by the induction hypothesis, $\Psi(B_i\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)} \uparrow (n \Leftrightarrow 1)$ for all i, $1 \leq i \leq r$.

Thus $\Psi(solve([p^1, t'_1, \ldots, t'_h])\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}\uparrow n$ and, by the definition of $T_{(P,E)}$, $\Psi(p(t_1, \ldots, t_h)\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}\uparrow n$.

(iii) A_j is $solve([p^1, t_1, \ldots, t_h])$. C is a clause $p(t'_1, \ldots, t'_h) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_r$ $(q \geq 0, r \geq 0)$, where p is any predicate symbol distinct from *solve*. Then $\langle H_{n-1}, F_{n-1}, \{t_1 = \uparrow t'_1, \ldots, t_h = \uparrow t'_h, e_1, \ldots, e_q\}\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_n, F_n, \{\}\rangle$. If $r = 0$, then we have $\Psi(p(t'_1, \ldots, t'_h)\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}\uparrow 1$, by correctness of R_E. Thus, by the definition of $T_{(P,E)}$, we have $\Psi(solve([p^1, t_1, \ldots, t_h])\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}\uparrow 1 \subseteq T_{(P,E)}\uparrow n$. If $r > 0$, by the induction hypothesis, $\Psi(B_i\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}\uparrow(n \Leftrightarrow 1)$, for each i, $1 \leq i \leq r$. Thus $\Psi(p(t'_1, \ldots, t'_h)\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}\uparrow n$ and, by the definition of $T_{(P,E)}$, $\Psi(solve([p^1, t_1, \ldots, t_h])\widehat{H_n}\widehat{H'}) \subseteq T_{(P,E)}\uparrow n$. ∎

Lemma 7.34 *Let E be a Horn clause equality theory and R_E a canonical rewrite system for E. Let (P, E) be a logic program, H a Herbrand assignment and G a definite goal. Suppose there exists an SLD*-refutation of $(P, E) \cup \{G\widehat{H}\}$ with success state $\langle\{\}, H_n, F_n\rangle$. Then there exists an SLD*-refutation of $(P, E) \cup \{G\}$ of the same length with success state $\langle\{\}, H'_n, F'_n\rangle$ such that $\langle H'_n, F'_n, H\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_n, F_n, \{\}\rangle$.*

Proof. Note first that, as the rewrite system R_E is canonical, if

$$\langle H, F, A \cup B\rangle \underset{E}{\overset{n}{\Leftrightarrow\!\!\rightarrow}} \langle H_1, F_1, B\rangle \underset{E}{\overset{m}{\Leftrightarrow\!\!\rightarrow}} \langle H_2, F_2, \{\}\rangle \quad (n \geq 1, m \geq 1),$$

then

$$\langle H, F, A \cup B\rangle \underset{E}{\overset{n'}{\Leftrightarrow\!\!\rightarrow}} \langle H'_1, F'_1, A\rangle \underset{E}{\overset{m'}{\Leftrightarrow\!\!\rightarrow}} \langle H_2, F_2, \{\}\rangle \quad (n' \geq 1, m' \geq 1).$$

Equivalently,

$$\langle H, F, A\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_1, F_1, \{\}\rangle \text{ and } \langle H_1, F_1, B\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_2, F_2, \{\}\rangle.$$

Now consider an SLD*-refutation of $(P, E) \cup \{G\widehat{H}\}$. Instead of applying the substitution $\widehat{H}$ to G, we, equivalently, consider the initial state $\langle G, H, \{\}\rangle$. In the following, let C_i and S_i be, respectively, the input clause and the set of equations needed to perform the i-th refutation step as defined in Definition 6.3. Thus, at the $i+1$ step of the SLD*-refutation of $(P, E) \cup \{G\widehat{H}\}$ the state $\langle G_{i+1}, H_{i+1}, F_{i+1}\rangle$ is obtained from $\langle G_i, H_i, F_i\rangle$ and C_i if $\langle H_i, F_i, S_{i+1}\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_{i+1}, F_{i+1}, \{\}\rangle$. Hence the following must hold:

$$\begin{aligned}
&\langle H, \{\}, S_1\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_1, F_1, \{\}\rangle\\
&\langle H_1, F_1, S_2\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_2, F_2, \{\}\rangle\\
&\ldots\\
&\langle H_{n-1}, F_{n-1}, S_n\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_n, F_n, \{\}\rangle.
\end{aligned}$$

As H is a Herbrand assignment, i.e., a a set of equations in solved form, this holds

$$\langle\{\}, \{\}, H\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H, \{\}, \{\}\rangle.$$

Finally, the rewrite system R_E being canonical

$$\langle\{\}, \{\}, S_1 \cup \ldots \cup S_n \cup H\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_n, F_n, \{\}\rangle$$

or, equivalently,

$$\langle\{\}, \{\}, S_1 \cup \ldots \cup S_n\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H'_n, F'_n, \{\}\rangle \text{ and } \langle H'_n, F'_n, H\rangle \underset{E}{\overset{*}{\Leftrightarrow\!\!\rightarrow}} \langle H_n, F_n, \{\}\rangle.$$ ∎

Theorem 7.35 *A ground atom A belongs to the success set of a logic program (P, E) if and only if , (A) is contained in the least reflective E-model of (P, E).*

Proof. By Corollary 7.30, it suffices to show that, if , (A) belongs to the least reflective E-model of (P, E), then A is contained in the success set of (P, E). Suppose , (A) is in the least reflective E-model of (P, E). By Theorem 7.20, , $(A) \in T_{(P,E)}{\uparrow}n$, for some $n \in \omega$. We prove by induction on n that , $(A) \in T_{(P,E)}{\uparrow}n$ implies that $(P, E) \cup \{\leftarrow A\}$ has a refutation and hence A is in the success set.

Suppose first that $n = 1$. Then , $(A) \in T_{(P,E)}{\uparrow}1$ means one of the following.

(*i*) A is an atom of the form $p(t_1, \ldots, t_h)$ and there exists a unit clause in P, say $p(t'_1, \ldots, t'_h) \leftarrow$, such that $E \models \exists(t_1 = t'_1 \wedge \ldots \wedge t_h = t'_h)$. By the correctness of R_E w.r.t. E, $\langle\{\}, \{\}, \{t_1 = t'_1, \ldots, t_h = t'_h\}\rangle \overset{*}{\underset{E}{\Leftrightarrow\!\!\!\Rightarrow}} \langle H, \{\}, \{\}\rangle$, for some Herbrand assignment H. Then, by the definition of SLD*-resolution (case 1), $(P, E) \cup \{\leftarrow p(t_1, \ldots, t_h)\}$ has an SLD*-refutation.

(*ii*) A is any ground atom $p(t_1, \ldots, t_h)$ whose predicate symbol p is distinct from *solve*, and there exists a unit clause $solve([p^1, t'_1, \ldots, t'_h]) \leftarrow$ in P such that $E \models \exists(t_1 = {\downarrow}t'_1 \wedge \ldots \wedge t_h = {\downarrow}t'_h)$. As R_E is correct w.r.t. E, $\langle\{\}, \{\}, \{t_1 = {\downarrow}t'_1, \ldots, t_h = {\downarrow}t'_h\}\rangle \overset{*}{\underset{E}{\Leftrightarrow\!\!\!\Rightarrow}} \langle H, F, \{\}\rangle$, for some Herbrand assignment H and set of name equations F. Hence, by the definition of SLD*-resolution (case 2), $(P, E) \cup \{\leftarrow p(t_1, \ldots, t_h)\}$ has an SLD*-refutation.

(*iii*) A is $solve([p^1, t_1, \ldots, t_h])$ and there exists a unit clause $p(t'_1, \ldots, t'_h) \leftarrow$ in P such that $E \models \exists(t_1 = {\uparrow}t'_1 \wedge \ldots \wedge t_h = {\uparrow}t'_h)$. Then, as R_E is correct w.r.t. E, $\langle\{\}, \{\}, \{t_1 = {\uparrow}t'_1, \ldots, t_h = {\uparrow}t'_h\}\rangle \overset{*}{\underset{E}{\Leftrightarrow\!\!\!\Rightarrow}} \langle H, F, \{\}\rangle$, for some Herbrand assignment H and set of name equations F. Hence, by the definition of SLD*-resolution (case 3), $(P, E) \cup \{\leftarrow solve([p^1, t_1, \ldots, t_h])\}$ has an SLD*-refutation.

Now suppose that the result holds for $n \Leftrightarrow 1$. Let , $(A) \in T_{(P,E)}{\uparrow}n$. By the definition of $T_{(P,E)}$, one of the following holds.

(*i*) A is an atom of the form $p(t_1, \ldots, t_h)$ and there exists a ground instance of a clause $p(t'_1, \ldots, t'_h) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_m \quad (q \geq 0, m \geq 0)$ in P such that $E \models \forall((t_1 = t'_1 \wedge \ldots \wedge t_h = t'_h, e_1, \ldots, e_q)\widehat{H})$ and $\{, (B_1\widehat{H}), \ldots, , (B_m\widehat{H})\} \subseteq T_{(P,E)}{\uparrow}(n \Leftrightarrow 1)$, for some Herbrand assignment H.

(*ii*) A is any ground atom $p(t_1, \ldots, t_h)$ whose predicate symbol p is distinct from *solve*, and there exists a ground instance of a clause $solve([p^1, t'_1, \ldots, t'_h]) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_m \quad (q \geq 0, m \geq 0)$ in P such that $E \models \forall((t_1 = {\downarrow}t'_1 \wedge \ldots \wedge t_h = {\downarrow}t'_h, e_1, \ldots, e_q)\widehat{H})$, and $\{, (B_1\widehat{H}), \ldots, , (B_m\widehat{H})\} \subseteq T_{(P,E)}{\uparrow}(n \Leftrightarrow 1)$, for some Herbrand assignment H.

(*iii*) A is an atom of the form $solve([p^1, t_1, \ldots, t_h])$ and there exists a ground instance of a clause $p(t'_1, \ldots, t'_h) \leftarrow e_1, \ldots, e_q, B_1, \ldots, B_m \quad (q \geq 0, m \geq 0)$ such that $E \models \forall((t_1 = {\uparrow}t'_1 \wedge \ldots \wedge t_h = {\uparrow}t'_h, e_1, \ldots, e_q)\widehat{H})$, and $\{, (B_1\widehat{H}), \ldots, , (B_m\widehat{H})\} \subseteq T_{(P,E)}{\uparrow}(n \Leftrightarrow 1)$, for some Herbrand assignment H.

Thus, by the induction hypothesis, $(P, E) \cup \{\leftarrow B_i\widehat{H}\}$, $1 \leq i \leq m$, has an SLD*-refutation. Because each $B_i\widehat{H}$ is ground, these refutations can be combined into a refutation of $(P, E) \cup \{\leftarrow (B_1, \ldots, B_m)\widehat{H}\}$. Hence $(P, E) \cup \{\leftarrow A\widehat{H}\}$ has an SLD*-refutation and we can apply the Lifting lemma to obtain an SLD*-refutation of $(P, E) \cup \{\leftarrow A\}$. ∎

Theorem 7.36 *Suppose that $(P, E) \cup \{G\}$ is reflectively E-unsatisfiable. Then there exists an SLD*-refutation of $(P, E) \cup \{G\}$.*

Proof. Let G be the goal $\leftarrow A_1, \ldots, A_k$. As $(P, E) \cup \{G\}$ is reflectively E-unsatisfiable, G is false w.r.t. $LRM_{(P,E)}$. Hence there exists some ground instance $G\widehat{H}$ of G such that , $(G\widehat{H})$ is false w.r.t. $LRM_{(P,E)}$. Thus $\{, (A_1\widehat{H}), \ldots, , (A_k\widehat{H})\} \subseteq LRM_{(P,E)}$. By Theorem 7.35, there is an SLD*-refutation for $(P, E) \cup \{\leftarrow A_i\widehat{H}\}$ for each i, $1 \leq i \leq k$. As each $A_i\widehat{H}$ is ground, its computed answer $\langle H_i, F_i\rangle$ contains variables that are distinct from the variables of the remaining computed answers. Thus we can combine these refutations into a refutation for $(P, E) \cup \{G\widehat{H}\}$. Finally, we apply the Lifting lemma. ∎

Lemma 7.37 *Let (P, E) be a logic program and A an atom. Suppose that A contains the variables $x_1, \ldots, x_n$ and $\forall x_1, \ldots, x_n(A)$ is a reflective logical E-consequence of (P, E). Then there exists an SLD*-refutation of $(P, E) \cup \{\leftarrow A\}$ with computed answer $\langle H, F\rangle$ such that $E \models \forall x_1, \ldots, x_n \exists (H \cup F)$.*

Proof. Let $a_1, \ldots, a_n$ be distinct constants or metaconstants not appearing in (P, E) or A and let H be the Herbrand assignment $\{x_1 = a_1, \ldots, x_n = a_n\}$ (we implicitly assume that for all i, $1 \leq i \leq n$, $x_i = a_i$ is a binding). Then $A\widehat{H}$ is a reflective logical E-consequence of (P, E). As $A\widehat{H}$ is ground, Theorem 7.35 states that $(P, E) \cup \{\leftarrow A\widehat{H}\}$ has an SLD*-refutation. As the a_i do not appear in (P, E) or A, by replacing a_i by x_i for all i, $1 \leq i \leq n$, in this refutation, we obtain an SLD*-refutation of $(P, E) \cup \{\leftarrow A\}$ with computed answer $\langle H, F\rangle$ such that the bindings in H for $x_1, \ldots, x_n$ are variable-pure. Furthermore, the equations in F are always satisfied independently from the values of $x_1, \ldots, x_n$, as they are satisfied in the SLD*-refutation for $(P, E) \cup \{\leftarrow A\widehat{H}\}$ when substituted by arbitrary constants, e.g., $a_1, \ldots, a_n$. Hence, it holds that $E \models \forall x_1, \ldots, x_n \exists (H \cup F)$. ∎

Theorem 7.39 *Let E be a Horn clause equality theory and R_E a canonical rewrite system for E. Let (P, E) be a logic program and G a definite goal. Then for every correct answer $\langle H, F\rangle$ for $(P, E) \cup \{G\}$, there exists a computed answer $\langle H', F'\rangle$ for $(P, E) \cup \{G\}$. Furthermore, there exists a Herbrand assignment H'' such that, for every E-solution H_F and $H_{F'}$ of F and F', respectively, $G\widehat{H'}\widehat{H_{F'}}\widehat{H''} = G\widehat{H}\widehat{H_F}$ holds.*

Proof. Suppose that G is the goal $\leftarrow A_1, \ldots, A_k$. As $\langle H, F\rangle$ is a correct answer, $\forall((A_1 \wedge \ldots \wedge A_k)\widehat{H}\widehat{H_F})$ is a reflective logical E-consequence of (P, E) for every E-solution H_F of F. By Lemma 7.37, there exists an SLD*-refutation of $(P, E) \cup \{\leftarrow A_i\widehat{H}\widehat{H_F}\}$ with computed answer $\langle H_i, F_i\rangle$ for all i, $1 \leq i \leq k$. Let H_{F_i} be an E-solution of F_i, $1 \leq i \leq k$. As $\widehat{H_i}\widehat{H_{F_i}}$ does not instantiate any of the variables in A_i and the variables in every SLD*-refutation are standardized apart, we can combine these SLD*-refutations into an SLD*-refutation of $(P, E) \cup \{G\widehat{H}\widehat{H_F}\}$. Assume the computed answer for $(P, E) \cup \{G\widehat{H}\widehat{H_F}\}$ is $\langle H''', F'''\rangle$. As $\forall(G\widehat{H}\widehat{H_F})$ is a reflective logical E-consequence of (P, E), by Lemma 7.37, $\widehat{H'''}\widehat{H_{F'''}}$ does not instantiate any variable in $G\widehat{H}\widehat{H_F}$. Thus, $G\widehat{H}\widehat{H_F} = G\widehat{H'''}\widehat{H_{F'''}}$ holds.

By the Lifting lemma, there exists an SLD*-refutation of $(P, E) \cup \{G\}$ with success state $\langle\{\}, H', F'\rangle$ such that $\langle H', F', H \cup H_F\rangle \overset{*}{\underset{E}{\Leftrightarrow}} \langle H''', F''', \{\}\rangle$. Let H'' be $H \cup H_F$. Then $G\widehat{H'}\widehat{H_{F'}}\widehat{H''} = G\widehat{H'''}\widehat{H_{F'''}} = G\widehat{H}\widehat{H_F}$ holds. ∎

UPMAIL
Computing Science Department
Uppsala University
Box 311
S-751 05 UPPSALA
Sweden

Phone: Nat 018 – 18 25 00
Int +46 18 18 25 00

Fax: Nat 018 – 52 12 70
Int +46 18 52 12 70

ISSN 0283-359X