NORTH-HOLLAND

WHY UNTYPED NONGROUND METAPROGRAMMING IS NOT (MUCH OF) A PROBLEM

BERN MARTENS* and DANNY DE SCHREYE†

▷ We study a semantics for untyped, vanilla metaprograms, using the nonground representation for object level variables. We introduce the notion of language independence, which generalizes range restriction. We show that the vanilla metaprogram associated with a stratified normal object program is weakly stratified. For language independent, stratified normal object programs, we prove that there is a natural one-to-one correspondence between atoms $p(t_1, \ldots, t_r)$ in the perfect Herbrand model of the object program and $solve(p(t_1, \ldots, t_r))$ atoms in the weakly perfect Herb and model of the associated vanilla metaprogram. Thus, for this class of programs, the weakly perfect Herbrand model provides a sensible semantics for the metaprogram. We show that this result generalizes to nonlanguage independent programs in the context of an extended Herbrand semantics, designed to closely mirror the operational behavior of logic programs. Moreover, we also consider a number of interesting extensions and/or variants of the basic vanilla metainterpreter. For instance, we demonstrate how our approach provides a sensible semantics for a limited form of amalgamation. ◁

*Partly supported by the Belgian National Fund for Scientific Research, partly by ESPRIT BRA COMPULOG II, Contract 6810, and partly by GOA "Non-Standard Applications of Abstract Interpretation," Belgium.

†Research Associate of the Belgian National Fund for Scientific Research.

Address correspondence to Bern Martens and Danny De Schreye, Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee Belgium. E-mail: {bern, dannyd}@cs.kuleuven.ac.be.

Received December 1992; accepted January 1994.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010

0743-1066/95/$9.50

SSDI 0743-1066(94)00015-X

1. INTRODUCTION

Since the appearance of [5] and [19], metaprogramming has become increasingly important in logic programming and deductive databases. Applications in knowledge representation and reasoning, program transformation, synthesis and analysis, debugging and expert systems, the modeling of evaluation strategies, the specification and implementation of sophisticated optimization techniques, the description of integrity constraint checking, etc., constitute a significantly large part of the recent work in the field (see, e.g., [4, 27, 42, 17, 40, 39, 7, 8]). A biennial, specialized workshop is dedicated to the subject, and its proceedings ([1, 6 and 32]) provide excellent reading material on foundations, implementational issues, and various applications.

[23] and [41] were the first to seriously investigate theoretical foundations for metaprogramming in logic programming. Particularly the ideas and results in [23] formed the starting point for the development of the novel logic programming language Gödel [21], which now constitutes a full-fledged declarative successor to Prolog, providing extensive support for the sound development of further metaprogramming applications.

It should be clear however, that it cannot have been the sound semantics for metaprogramming, nor the existence of Gödel, that attracted so much interest into metaprogramming in logic programming to start with. (Although they have clearly accelerated the activity in the area.) One attraction certainly is the desire to extend the expressiveness of Horn clause logic augmented with negation as failure. Metaprogramming adds extra knowledge representation and reasoning facilities [27]. A second attraction is related to practicality. In applicative languages (both pure functional and pure logical), data and programs are syntactically indistinguishable. This is an open invitation to writing programs that take other programs as input. We believe that the practical success of, in particular, untyped vanilla-type metaprogramming has resulted from this. However, in spite of this success, little or no effort was made to provide it with a sensible semantics in the usual framework of untyped Herbrand interpretations. Doing just this is the main motivation for the work reported in this paper.

In [23], the possibility of providing such a declarative semantics is rejected immediately on the basis that the intended interpretations of vanilla metaprograms can never be models in such a framework. Now, this statement seems somewhat inaccurate. The intended *meaning* of a vanilla-type metatheory (in which different variables range over different domains) can simply not be captured within the formal notion of an interpretation, as it is defined for untyped, first order logic. So, a more precise statement would be that *the intended meaning cannot be formalized as an untyped interpretation.* However, this problem is not typical for untyped vanilla programs; it generally appears in the semantics of most untyped logic programs. Indeed, any such program in which a functor is used to represent a partial function suffers from the same semantical problem and, in practice, total functions seldom occur in real applications. (See [13] for a thorough discussion of this issue.)

Whether this (and other) argument(s) in favor of typed logic programs should convince us to abandon the notational simplicity of untyped logic programs altogether is an issue we do not want to address in this paper.

From here on, we will assume that the semantics of an (untyped) program is captured by the alternative notion of its (least/perfect/well-founded/...) Herbrand

model, avoiding the problems with intended interpretations. Even in this more restricted context, there remain problems with the semantics of untyped vanilla metaprograms.

Consider the (definite) object program P:

$$p(x) \leftarrow$$
$$q(a) \leftarrow .$$

Let M denote the standard (definite) *solve* interpreter:

$$solve(empty) \leftarrow$$
$$solve(x \,\&\, y) \leftarrow solve(x), solve(y)$$
$$solve(x) \leftarrow clause(x, y), solve(y).$$

In addition, let M_P denote the program M augmented with the facts

$$clause(p(x), empty) \leftarrow$$
$$clause(q(a), empty) \leftarrow .$$

Although the least Herbrand model of our object program is $\{p(a), q(a)\}$, the least Herbrand model of the metaprogram M_P contains completely unrelated atoms, such as $solve(p(empty))$, $solve(p(q(a)))$, etc.

This is certainly undesirable, since we, in general, would like at least that the atoms of the form $solve(p(t))$ in the least Herbrand model of M_P correspond in a one-to-one way with the atoms of the form $p(t)$ in the least Herbrand model of P.

In this paper, we therefore introduce the notion of *language independence* and show that it generalizes range restriction. Next, we show that a stratified object program gives rise to a weakly stratified metaprogram. As one of our main results, we prove that the perfect Herbrand model of a stratified, language independent object program corresponds in a one-to-one way with a natural subset of its metatheory's weakly perfect Herbrand model. In addition, we show how this approach can be extended to provide a semantics for various related metaprograms, including a limited form of amalgamation. Finally, we demonstrate that the language independence condition can often be skipped in a semantics that reflects closer the program's operational behavior.

The paper is organized as follows. In the next two sections, we introduce and discuss the notion of language independence and weak stratification, respectively. Section 4 contains our basic results for stratified object programs and their straightforward untyped vanilla metaversion. Some related (more "useful") metaprograms are considered in Section 5. We justify overloading logical symbols and address various (limited) forms of amalgamation in Section 6. Section 7 contains our results in the context of S-semantics. We discuss and compare some related work in Section 8, and, of course, some concluding remarks and suggestions for further research round off the paper.

Throughout the rest of this paper, we assume the reader to be familiar with the basic concepts of predicate logic (see, e.g., [14]) and logic programming (see, e.g., [29]). We also suppose familiarity with the notions of (*local*) stratification and *perfect* model (see, e.g., [2, 36]).

Finally, we would like to point out that two earlier texts report on part of this work:

1. [10] addresses least Herbrand model semantics of definite object and metaprograms.
2. [30] is a preliminary and considerably abridged presentation of our results for (weakly) stratified object and metaprograms and their (weakly) perfect Herbrand models.

2. LANGUAGE INDEPENDENCE

The intuition behind *language independence* is simple. In the logic programming community, there seems to be broad agreement that the declarative semantics of most or all logic programs can be described by a particular Herbrand model of the program. Two well-known classes of programs whose semantics is the subject of little or no controversy[1] are *definite* programs with their *least* Herbrand model and stratified programs with their *perfect* model.

In general, however, these models depend on the language in which we are considering the Herbrand models. A simple example suffices to show this. Consider the program consisting of the single clause $p(x) \leftarrow$. Its least Herbrand model (in fact, its only Herbrand model) in a language with one constant symbol a and no function symbols is $\{p(a)\}$. If we, however, add the constant symbol b, we obtain $\{p(a), p(b)\}$. Basically, we will call a program *language independent* when its characteristic Herbrand model does not depend on the particular choice of language.

A formal introduction and characterization of the notion for stratified programs follows. Some comments and results on language independence for other classes of programs are included at the end of the section.

2.1. Language Independent Stratified Programs

Suppose P is a logic program. Let $\mathcal{R}_P$, $\mathcal{F}_P$, and $\mathcal{C}_P$ denote the sets of predicate, function, and constant symbols occurring in the program. We can then consider a first order language $\mathcal{L}_P$, which has exactly $\mathcal{R}_P$ and $\mathcal{F}_P$ as its sets of predicate and function symbols, respectively. As its set of constant symbols, we take $\mathcal{C}_P$ if it is *nonempty* and $\{*\}$, a set with a single arbitrary element $*$, if it is. $\mathcal{L}_P$ is called the *language underlying the program* P. Although this is not imposed as a limitation in, e.g., [29], Herbrand interpretations of the program are usually constructed with this underlying language in mind. For our purposes in this paper, however, we need more flexibility. We therefore introduce the following two definitions.

Definition 2.1. Let P be a normal program with underlying language $\mathcal{L}_P$. We call a language $\mathcal{L}'$, determined by $\mathcal{R}'$, $\mathcal{F}'$, and $\mathcal{C}'$, an *extension* of $\mathcal{L}_P$ iff $\mathcal{R}_P \subseteq \mathcal{R}'$, $\mathcal{F}_P \subseteq \mathcal{F}'$, and $\mathcal{C}_P \subseteq \mathcal{C}' \neq \emptyset$.

Notice that if $\mathcal{C}_P$ is empty, $\mathcal{C}'$ may be any nonempty set of constant symbols. In particular, it does not have to contain $*$. The following definition makes explicit the language in which Herbrand interpretations are constructed.

Definition 2.2. Let P be a normal program with underlying language $\mathcal{L}_P$. A

[1] See, however, Section 7 of this paper.

Herbrand interpretation of P in a language $\mathcal{L}'$, extension of $\mathcal{L}_P$, is called an *$\mathcal{L}'$-Herbrand interpretation* of P.

In the sequel, we will often refer to *Herbrand* interpretations and models of a program P with underlying language $\mathcal{L}_P$, when in fact we mean *$\mathcal{L}_P$-Herbrand* interpretations by models.

We are now in a position to introduce the notion of *language independence*.

Definition 2.3. A stratified program P with underlying language $\mathcal{L}_P$ is called *language independent* iff for any extension $\mathcal{L}'$ of $\mathcal{L}_P$, its perfect $\mathcal{L}'$-Herbrand model is equal to its perfect $\mathcal{L}'$-Herbrand model.

Notice that this definition immediately entails that no atom in any perfect Herbrand model can contain any predicate, function, and/or constant symbols not occurring in P. In particular, when $\mathcal{C}_P$ is empty, any perfect Herbrand model can only contain propositions. (This observation will be used implicitly in the sequel. It ensures that the infamous $*$ constant does not cause too much trouble. See also the remark following Proposition 4.2.)

We illustrate Definition 2.3 with some simple examples.

Example 2.1. Of the following programs, P_1, P_2, and P_6 are *not* language independent, while the others are:

$$
\begin{aligned}
P_1 &: p(x) \leftarrow, \\
P_2 &: p(x) \leftarrow \mathit{not}\ q(x) \\
&\quad\ q(a) \leftarrow, \\
P_3 &: p(x) \leftarrow r(x), \mathit{not}\ q(x) \\
&\quad\ r(a) \leftarrow, \\
P_4 &: p(x,y) \leftarrow r(x), \mathit{not}\ q(x) \\
&\quad\ q(x) \leftarrow h(x) \\
&\quad\ h(a) \leftarrow \\
&\quad\ r(a) \leftarrow, \\
P_5 &: p(x) \leftarrow q(x), \mathit{not}\ r(x,y) \\
&\quad\ q(a) \leftarrow, \\
P_6 &: p(x) \leftarrow q(x), \mathit{not}\ r(x,y) \\
&\quad\ q(a) \leftarrow \\
&\quad\ r(a,a) \leftarrow .
\end{aligned}
$$

It is clear that language independence, introduced here as a concept tuned toward the semantics of logic programs, is strongly related to *domain independence*. The latter concept was defined for any formula in the context of full first order logic (see, e.g., [11] and further references given there). The following example shows that the two notions do *not* coincide.

Example 2.2.

$$p(x) \leftarrow q(x), not\ r(x,y)$$
$$q(a) \leftarrow$$
$$r(a,a) \leftarrow$$
$$s(b) \leftarrow .$$

Indeed, this program is language independent. However, there are (non-Herbrand) models, having only one element in their domain of interpretation on which both a and b are mapped, in which $p(a)$ is *not* true. It therefore is *not* domain independent.

However, obviously, any stratified program which is domain independent is also language independent, and, indeed, language independent (stratified) programs generalize the concept of domain independent databases (*function-free*) as introduced in [43] (see Lemma 3 on page 229 in [43]). We feel, however, that the above introduced terminology better reflects the underlying intuition. We should point out that [43] contains an extensive discussion on the relation between *domain independence* and *allowedness*, a notion identical (at the clause level) to the notion of *range restriction*, introduced below. Finally, it is shown that for *function-free*, stratified, normal programs, domain independence (or, rather, language independence) is *decidable.*

In general, however, it is clear that, like domain independence for full first order logic (see, e.g., Theorem 2 on page 224 in [43] and further references given there), language independence is an undecidable property. To see this, observe that checking language independence boils down to checking refutability of goals.[2] It is therefore important to investigate the existence of syntactically recognizable, and thus decidable, subclasses of the class of language independent programs. It turns out that the well-known concept of *range restriction* determines such a class.

Let us first repeat its definition, specialized to the context of normal logic programs.

Definition 2.4. A clause in a program P is called *range restricted* iff every variable in the clause appears in a positive body literal. A program P is called *range restricted* iff all its clauses are range restricted.

It is obvious that *range restriction* is a syntactic property. It has been defined for more general formulas and/or programs and was used in other contexts. Two related notions are *safety*, used by Ullman [45], and *allowedness*, defined in [29] and important for avoiding floundering of negative goals. The following proposition states that this important class of logic programs is a subclass of the language independent ones.

Proposition 2.1. Let P be a stratified program. If P is range restricted, then P is language independent.

PROOF. The proof proceeds through induction on the strata of P. □

[2]This observation was first made by an anonymous referee of [30].

Example 2.3. It can be noted that of the programs in Example 2.1, only P_3 is range restricted.

2.2. Language Independence for Other Classes of Programs

Above, we have introduced the notion of language independence of *stratified* programs. It is obvious that this generalizes the concept of language independence for *definite* programs, based on the invariance of the *least* Herbrand model, as it was defined in [10]. It is also obvious that on the class of definite programs, both definitions coincide.

However, for definite programs, the difference between language independence and range restriction is more easily characterizable, as the following proposition shows.[3]

Proposition 2.2. Let P be a definite program. Then P is language independent iff all ground $(\mathcal{L}_P)$-instances of every non-range-restricted clause in P contain at least one body atom, not true in its least Herbrand model.

PROOF. First suppose there is a non-range-restricted clause that does not satisfy the stated condition. It immediately follows that P is not language independent. This proves the *only-if* part.

The proof of the *if* part is completely analogous to the reasoning on the bottom layer in the proof of Proposition 2.1. □

Notice that this result still leaves language independence as an undecidable property.

The notion of language independence can also be considered for classes broader than the one of satisfied programs. A generalization to *locally* stratified programs is straightforward. Also, *weakly* stratified programs with their *weakly* perfect Herbrand model (introduced in the next section) allow an obvious adaptation of Definition 2.3 and Proposition 2.1. A complete analysis of its usefulness in the context of different classes of logic programs and their semantics is outside the scope of the present paper. Notice, however, that in those semantics in which the "preferred" Herbrand model is three-valued (e.g., well-founded semantics [46]), one will demand only the positive information in the model to be invariant under language extensions. Indeed, it is obvious that the negative information will be affected for almost all reasonable programs and semantics. We return briefly to this issue in Section 9.

3. WEAK STRATIFICATION

In Section 1, we pointed out that, in general, there is a problem with the least Herbrand model of vanilla metaprograms. Much of the rest of this paper is devoted to showing that for language independent programs, this problem disappears. However, we do not wish to restrict our development to definite programs. In particular, we would like to consider stratified object and metaprograms, and compare their perfect Herbrand models.

[3]This observation is due to an anonymous referee of [10].

In this context, another difficulty has to be addressed first. We illustrate it through the following example.

Consider the stratified, language independent object program P.

$$p(x) \leftarrow r(x), not\, q(x)$$
$$r(a) \leftarrow .$$

Let M denote the standard (normal) *solve* interpreter:

$$solve(empty) \leftarrow$$
$$solve(x\&y) \leftarrow solve(x),\ solve(y)$$
$$solve(\neg x) \leftarrow notsolve(x) \quad \text{(i)}$$
$$solve(x) \leftarrow clause(x, y),\ solve(y). \quad \text{(ii)}$$

In addition, let M_P denote the program M augmented with the facts

$$clause(p(x), r(x)\&\neg q(x)) \leftarrow$$
$$clause(r(a), empty) \leftarrow .$$

Obviously, M_P is *not* stratified. Moreover, it is not even *locally* stratified. To see this, consider clause (ii) of M_P. For any two round atoms, $solve(t_1)$ and $solve(t_2)$ in the Herbrand base for the language underlying M_P, we have that both

$$solve(t_1) \leftarrow clause(t_1, t_2), solve(t_2)$$

and

$$solve(t_2) \leftarrow clause(t_2, t_1), solve(t_1)$$

are ground instances of (ii). Therefore, in any local stratification of M_P, all the ground atoms of the form $solve(t)$ must be in the same stratum. On the other hand, by clause (i), any ground atom $solve(\neg t)$ must be placed in a higher stratum than the corresponding atom $solve(t)$. So no local stratification can be possible.

However, there is a simple way to overcome this problem. Consider the new theory, M'_P, obtained from M_P by performing one unfolding step of the atom $clause(x, y)$ in clause (ii), using every available *clause* fact of M_P. Clause (ii) is replaced by the resultants:

$$solve(p(x)) \leftarrow solve(r(x)\&\neg q(x))$$
$$solve(r(a)) \leftarrow solve(empty).$$

This new theory M'_P is completely equivalent with M_P in the sense that they have identical models, but one can easily verify that M'_P *is* locally stratified. It can be shown that for any stratified object program P, the program obtained from its associated vanilla metaprogram through a similar unfolding transformation is locally stratified. In [33] and [35], Przymusinska and Przymusinski introduced *weakly* stratified logic programs and their unique *weakly* perfect Herbrand model. Now, from their definitions, it follows that programs which can be unfolded into a locally stratified one are weakly stratified. It can, therefore, be shown that a stratified object program gives rise to a weakly stratified vanilla metaprogram. This allows us to consider the weakly perfect Herbrand model of the latter as the description of its semantics.

So, before we actually embark on a study of metaprogram semantics, we devote the next subsection to a formal introduction of *weakly stratified programs* and *weakly perfect models*. Moreover, it turns out that we do not need the fully general concepts in our restricted context. We will, therefore, derive a more easily verifiable *sufficient* condition for weak stratification, to be used throughout the rest of this paper.

3.1. Weakly Stratified Programs and Weakly Perfect Models

Weakly stratified programs and weakly perfect models were first introduced in [33]. However, that paper is restricted to the case of *function-free* (so-called datalog) programs. Obviously, our needs surpass that limitation: the example metaprograms above clearly contain functors. Since the necessary generalizations are not completely straightforward, we turn to the fully general treatment in [35], and since both weak stratification and weakly perfect models are not (yet?) fully standard concepts in logic programming, we include the relevant parts of their definition. Finally, it can be pointed out that the overview paper [34] also has a section on weak stratification and weakly perfect models. The presentation there is essentially the same, in spite of the fact that some of its basic definitions are chosen differently. The following development is abridged (and slightly adapted) from [35]. For further details and examples, we refer to that paper.

Definition 3.1. Let P be a normal program. We denote by $Ground\ (P)$ the (Possibly infinite) set of ground instances of clauses in P and we call it the *ground program associated with* P.

In the sequel, we will often apply notions defined for finite programs in the context of infinite ground programs. The generalization of the "classical" definitions is straightforward. As usual, we use the notation B_P to denote the *Herbrand base* for the language underlying a program P.

Definition 3.2. The *ground dependency graph* $Depg(P)$ of a program P is defined as follows:

- Its vertices are the atoms in B_P.
- There is a positive directed edge from A to B if $Ground(P)$ contains a cluase $B \leftarrow \cdots, A, \cdots$.
- There is a negative directed edge from A to B if $Ground(P)$ contains a clause $B \leftarrow \cdots, not\ A, \cdots$.

Then we define the following relations between atoms in B_P:

Definition 3.3.

- $A \leq B$ iff there is a directed path from A to B in $Depg(P)$.
- $A < B$ iff there is a directed path from A to B in $Depg(P)$ passing through a negative edge.
- $A \sim B$ iff $(A = B) \vee (A < B \wedge B < A)$.

$\sim$ is an equivalence relation on B_P, the equivalence classes of which we will call

components of B_P. We can define a partial order between these components.

Definition 3.4. Suppose C_1 and C_2 are two components of B_P. We define

$$C_1 \prec C_2 \quad \text{iff } C_1 \neq C_2 \wedge \exists A_1 \in C_1, \exists A_2 \in C_2 (A_1 < A_2).$$

A component C_1 is called *minimal* iff there is no component C_2 such that $C_2 \prec C_1$.

Definition 3.5. By the *bottom stratum* $S(P)$ of a ground program P, we mean the union of all minimal components of P, i.e.,

$$S(P) = \cup\{C \mid C \text{ is a minimal component of } B_P\}.$$

By the *bottom layer* $L(P)$ of a ground program P, we mean the corresponding subprogram of P, i.e.,

$$L(P) = \text{the set of all clauses from } P, \text{whose heads belong to } S(P).$$

In the context of the ensuing construction, (possibly partial, three-valued) interpretations and models that explicitly register negative information will be used. They contain not only positive, but also negative ground literals. $Pos(I)$ will denote the positive subset of such an interpretation. It is, of course, itself a standard (two-valued) interpretation. In the next definition, as usual, we assume $not(not\, L) = L$ for any ground literal.

Definition 3.6. Let P be a ground program of which I is a partial interpretation. By a *reduction of* P *modulo* I, we mean a new (ground) program $\frac{P}{I}$ obtained from P by performing the following two reductions:

- Removing from P all clauses which contain a premise L such that $not\, L \in I$ or whose head $\in I$ (i.e., clauses true in I).
- Removing from the remaining clauses all premises $L \in I$.

Finally, remove all nonunit clauses whose heads appear as unit clauses.

We can now describe the (transfinite) iteration process that leads to the construction of a *weakly* perfect model. (The union of three-valued interpretations, used in Definition 3.7, is defined as straightforward set union. Since the S_α sets are disjunct, the corresponding H_α models cannot contain contradictory information and their union is well-defined.)

Definition 3.7. Suppose that P is a logic program and let $P_0 = Ground(P), H_0 = \emptyset$. Suppose that $\alpha > 0$ is a countable ordinal such that programs P_δ and partial interpretations H_δ have been already defined for all $\delta < \alpha$. Let

$$N_\alpha = \bigcup_{\delta<\alpha} H_\delta, \qquad P_\alpha = \frac{P_0}{N_\alpha}, \qquad S_\alpha = S(P_\alpha), \qquad L_\alpha = L(P_\alpha).$$

- If the program P_α is empty, then the construction stops and $H_P = Pos(N_\alpha)$ is the *weakly perfect Herbrand model* of P.

- Otherwise, if $S_\alpha = \emptyset$ or if L_α has no least Herbrand model, then the construction also stops. (P has no weakly perfect Herbrand model.)
- Otherwise, H_α is defined as the least Herbrand model of L_α and the construction continues.

Finally, we define *weakly stratified* programs.

Definition 3.8. We say that a program P is *weakly stratified* if it has a weakly perfect model and all L_α are definite.

The following result is immediate.

Proposition 3.1. Every (locally) stratified program is weakly stratified and its perfect and weakly perfect Herbrand models coincide.

PROOF. Immediate from Theorem 4.1 and Corollary 4.5 in [35]. □

Thus we see that weak stratification is a conservative extension of (local) stratification. It can even be argued that weak stratification is in many ways a more suitable extension of stratification than local stratification is. Indeed, the latter property is not invariant under some elementary program transformations, as was shown above.

Below, we show that metaprograms associated with stratified object programs are indeed weakly stratified. However, a number of details in the above general construction are rather inconvenient in that restricted context.

- We want to avoid the use of three-valued interpretations, even during construction of the (two-valued) weakly perfect model.
- We will not need transfinite induction.
- We are only interested in weakly prefect models for programs that are actually weakly stratified.
- Definition 3.7 takes S_α always equal to the entire union of minimal components of B_{P_α}. In other words, the weak stratification built is as "tight" as possible. Following this practice would considerably demage the elegance of the proofs in the subsequent sections. We need the ability to only consider "safe" subsets of that union.
- Finally, we will not remove nonunit clauses whose heads appear as unit clauses, as stipulated in Definition 3.6.

For all these reasons, we present a modified construction procedure in Proposition 3.2. Successful application of this procedure guarantees weak stratification and constructs the weakly perfect model. In other words, we show that Proposition 3.2 provides a *sufficient* condition for weak stratification—a condition which will then actually be used throughout the rest of this paper. Some simple programs, not satisfying it, but nevertheless weakly stratified according to Definitions 3.7 and 3.8 are included among the examples below. Before we can state Proposition 3.2, we need one more auxiliary concept.

Definition 3.9. If P is a ground program, then we denote by $CH(P)$ the set of all

heads of clauses in P.

Proposition 3.2. *Let P be a normal program. Then we define:*

$$P_1 = Ground(P)$$

If $P_i \neq \emptyset$ is defined and B_{P_i} has one or more minimal components, we define:

- $S_i = S(P_i)$.
- V_i: *a nonempty subset of S_i such that if $B \in V_i$ and $A \leq B$ (and hence $A \in S_i$), then $A \in V_i$.*
- L_i: *the set of clauses in P_i whose head is in V_i.*
- H_i: *the least Herbrand model of L_i if L_i is definite.*

If H_i is defined, we define the following:

- P'_{i+1} : *the set of all clauses in $P_i \backslash L_i$ for which the following hold:*
  - *—For every positive body literal $B, B \in V_i \Rightarrow B \in H_i$.*
  - *—For every negative body literal not $B, B \in V_i \Rightarrow B \notin H_i$.*
- P_{i+1} : P'_{i+1} *with the V_i body literals deleted from the clauses.*

If there is an i such that $P_i = \emptyset$, then take $H_P = \cup_{j<i} H_j$.
Else if H_j is defined for all $j < \omega$ and $\cap_{j<\omega} CH(P_j) = \emptyset$, then take $H_P = \cup_{j<\omega} H_j$. In both cases, P is weakly stratified and H_P is it weaklys perfect Herbrand model.

PROOF. First, it can be verified that our use of two-valued interpretations is a correct recasting of Definition 3.6 and its use in Definition 3.7. Not removing nonunit clauses in the presence of corresponding unit clauses might lead to failure (see also Example 3.8), but does not influence H_P upon success. A detailed formal proof of this point can proceed in a similar way to the proof for Lemma 3.1 below. Furthermore, the definition of the H_i now incorporates the condition that each L_i be definite, and P therefore weakly stratified. Next, note that $\cap_{j<\omega} CH(P_j) = \emptyset$ means that all clauses in $Ground(P)$ are consumed during the iteration process, and therefore halting successfully is correct. It remeans to be shown that constructing a nontight stratification is safe. In other words, that any choice of the V_i, leading to a successful halt, returns the same H_p. (The fact that some choices might lead to success, while others might not, is not a problem, since we are only claiming a sufficient condition.) This follows from Lemma 3.1 below. □

Lemma 3.1. *Let P be a normal program such that a series $V_1, \ldots$ leads to successful termination of the construction in Proposition 3.2 with resulting model H_P^V. Then the (maximal choice) series $S_1, \ldots$ also terminates successfully with $H_P^S = H_P^V$.*

PROOF. The proof of this lemma can be found in the Appendix. □

Let us now briefly turn to some examples. The first one is taken from [35].

Example 3.1. Let P be the following program:

$$p(1,2) \leftarrow$$

$$q(x) \leftarrow p(x,y), not\ q(y).$$

P is neither stratified, nor locally stratified. It is however weakly stratified:

$$\begin{aligned}
P_1: \quad & p(1,2) \leftarrow \\
& q(1) \leftarrow p(1,1), not\ q(1) \\
& q(1) \leftarrow p(1,2), not\ q(2) \\
& q(2) \leftarrow p(2,1), not\ q(1) \\
& q(2) \leftarrow p(2,2), not\ q(2), \\
S_1 &= \{p(1,1), p(1,2), p(2,1), p(2,2)\}, \\
V_1 &= S_1, \\
L_1 &= \{p(1,2) \leftarrow\}, \\
H_1 &= \{p(1,2)\}, \\
P_2' &= \{q(1) \leftarrow p(1,2), not\ q(2)\}, \\
P_2 &= \{q(1) \leftarrow not\ q(2)\}, \\
S_2 &= \{q(2)\}, \\
V_2 &= \{q(2)\}, \\
L_2 &= \emptyset, \\
H_2 &= \emptyset, \\
P_3' &= \{q(1) \leftarrow not\ q(2)\}, \\
P_3 &= \{q(1) \leftarrow\}, \\
S_3 &= \{q(1)\}, \\
V_3 &= \{q(1)\}, \\
L_3 &= \{q(1) \leftarrow\}, \\
H_3 &= \{q(1)\}, \\
P_4' &= \emptyset = P_4.
\end{aligned}$$

P's weakly perfect Herbrand model $H_P = \cup_{j<4} H_j = \{p(1,2), q(1)\}$.

The following well-known example is often presented as a motivation for extending the concept of stratification ot *local* stratification (see, e.g., [36]). We demonstrate that it is also *weakly* stratified.

Example 3.2. Let P name the following program:

$$\begin{aligned}
& even(0) \leftarrow \\
& even(s(x)) \leftarrow not\ even(x), \\
P_1 &= \{even(0) \leftarrow\} \cup \{even(s^{n+1}(0)) \leftarrow not\ even(s^n(0)) \mid n \geq 0\}, \\
S_1 &= \{even(0)\}, \\
V_1 &= S_1, \\
L_1 &= \{even(0) \leftarrow\}, \\
H_1 &= \{even(0)\}, \\
P_2' &= \{even(s^{n+1}(0)) \leftarrow not\ even(s^n(0)) \mid n \geq 1\},
\end{aligned}$$

$$
\begin{aligned}
P_2 &= P_2', \\
S_2 &= \{even(s(0))\}, \\
V_2 &= S_2, \\
L_2 &= \emptyset, \\
H_2 &= \emptyset, \\
P_3' &= P_2, \\
P_3 &= \{even(s(s(0))) \leftarrow\} \cup \{even(s^{n+1}(0)) \leftarrow not\ even(s^n(0)) \mid n \geq 2\}, \\
S_3 &= \{even(s(s(0)))\}, \\
V_3 &= S_3, \\
L_3 &= \{even(s(s(0))) \leftarrow\}, \\
H_3 &= \{even(s(s(0)))\}, \\
&\vdots \\
\bigcap_{j<\omega} CH(P_j) &= \emptyset.
\end{aligned}
$$

Example 3.3. The M_P example program at the beginning of Section 3 is also weakly stratified. Indeed, $Ground(M_p)$ contains a (countably) infinite amount of layers, each of which is composed of an infinite amount of ground clause instances. For more details, we refer to Theorem 4.1 and its proof.

The program in the next example has no weakly perfect Herbrand model.

Example 3.4.

$$
\begin{aligned}
p &\leftarrow not\ q \\
q &\leftarrow not\ p.
\end{aligned}
$$

Indeed, the complete program is contained in its single layer, which has two distinct minimal Herbrand models, and therefore no least.

Also, the following program P has no weakly perfect Herbrand model.

Example 3.5.

$$
\begin{aligned}
even(0) &\leftarrow \\
even(x) &\leftarrow not\ even(s(x)).
\end{aligned}
$$

The bottom stratum of $Ground(P)$ is empty.

A small change to Example 3.4 gives us a program that does have a weakly perfect Herbrand model, even though it is still not weakly stratified.

Example 3.6.

$$
\begin{aligned}
p &\leftarrow q \\
q &\leftarrow not\ p.
\end{aligned}
$$

The modified program still consists of a single (nondefinite) layer, which now, however, has a least Herbrand model: $\{p\}$.

Finally, we include two weakly stratified programs that do *not* satisfy the condition in Proposition 3.2.

Example 3.7.

$$\begin{aligned} P: \quad & p(a) \leftarrow \\ & p(f(x)) \leftarrow not\ p(x) \\ & q(x) \leftarrow not\ p(x) \\ & r(a) \leftarrow p(x), q(x) \\ & r(f(x)) \leftarrow not\ r(x). \end{aligned}$$

Clearly, $\cap_{j<\omega} CH(P_j) \neq \emptyset$ and yet P is weakly stratified. However, establishing the latter fact requires transfinite induction.

Example 3.8.

$$\begin{aligned} & p \leftarrow not\ q \\ & p \leftarrow r \\ & q \leftarrow not\ p \\ & r \leftarrow . \end{aligned}$$

Here, the removal of clauses (with a nonempty body) having a head for which also a fact is present (see the bottom line in Definition 3.6), makes a crucial difference. For our applications, this practice is not needed. We therefore did not incorporate it in the construction presented in Proposition 3.2.

Finally, we point out that in [38], the closely related notion of *modular stratification* has been defined for *datalog* programs. Essentially, it builds a componentwise dynamic local stratification. Since every modularly stratified (datalog) program is weakly stratified (Theorem 3.1 in [38]), the latter is the more general property.

4. VANILLA METAPROGRAMS

4.1. Definitions

We are now finally in a position where we can address the proper topic of this paper: the semantics of metaprograms. In this section, we present the two key results that lie at the heart of our work.

We set out with the following definitions, formally introducing the concept of a *vanilla metaprogram.*

Definition 4.1. Suppose $\mathcal{L}$ is a first order language and $\mathcal{R}$ its finite (or countable) set of predicate symbols. Then we define $\mathcal{F}_{\mathcal{R}}$ to be a *functorization* of $\mathcal{R}$ iff $\mathcal{F}_{\mathcal{R}}$ is a set of function symbols such that there is a one-to-one correspondence between elements of $\mathcal{R}$ and $\mathcal{F}_{\mathcal{R}}$ and the arity of corresponding elements is equal.

We introduce the following notation: Whenever A is an atom in a first order language $\mathcal{L}$ with predicate symbol set $\mathcal{R}$ and a functorization $\mathcal{F}_\mathcal{R}$ of $\mathcal{R}$ is given, A' denotes the term produced by replacing in A the predicate symbol by its corresponding element in $\mathcal{F}_\mathcal{R}$.

Definition 4.2. The following normal program M is called *vanilla metainterpreter.*

$$
\begin{aligned}
&solve(empty) \leftarrow \\
&solve(x\&y) \leftarrow solve(x), solve(y) \\
&solve(\neg x) \leftarrow not\ solve(x) \\
&solve(x) \leftarrow clause(x, y), solve(y).
\end{aligned}
$$

Notice M is neither language independent nor stratified (nor locally stratified), and these properties carry over to M_P-programs, defined as follows:

Definition 4.3. Let P be a normal program. Then M_P, the *vanilla metaprogram associated with* P, is the normal program consisting of M together with a fact of the form

$$clause(A', \cdots, \&B'\& \cdots \&\neg C'\& \cdots) \leftarrow$$

for every clause $A \leftarrow \cdots, B, \cdots, not\ C, \cdots$ in P and a fact of the form

$$clause(A', empty) \leftarrow$$

for every fact $A \leftarrow$ in P.

A number of remarks are in order:

- If $\mathcal{L}_P$, the language underlying P, is determined by $\mathcal{R}_P, \mathcal{F}_P$, and $\mathcal{C}_P$, then $\mathcal{L}_{M_P}$, the language underlying M_P, is determined by:
  - $\mathcal{R}_{M_P} = \{solve, clause\}$.
  - $\mathcal{F}_{M_P} = \mathcal{F}_P \cup \mathcal{F}_{\mathcal{R}_P} \cup \{\&, \neg\}$, where $\mathcal{F}_{\mathcal{R}_P}$ is a functorisation of $\mathcal{R}_P$, presupposed in Definition 4.3.
  - $\mathcal{C}_{M_P} = \mathcal{C}_P \cup \{empty\}$.
- For clarity, except when explicitly stated otherwise, we will demand:
  - $\mathcal{F}_P \cap \{solve, clause\} = \emptyset$,
  - $\mathcal{R}_P \cap \{solve, clause\} = \emptyset$,
  - $\mathcal{F}_P \cap \mathcal{F}_{\mathcal{R}_P} = \emptyset$,
  - $empty \notin \mathcal{C}_P$

  for all object programs P throughout the rest of this paper.
- In the sequel, when we refer to Herbrand interpretations and/or models (of a program P or M_P) and related concepts, this will be in the context of the languages $\mathcal{L}_P$ and $\mathcal{L}_{M_P}$, as defined above, unless stated explicitly otherwise.
- Finally, we introduce the following notation:
  - U_P: the Herbrand universe of a program P.
  - $U_P^n = U_P \times \cdots \times U_P$ (n copies).
  - p/r: a predicate symbol with arity r in $\mathcal{R}_P$.

— p'/r: its associated function symbol in $\mathcal{F}_{M_P}$.

The following proposition, which will implicitly be used in the sequel, is immediate.

Proposition 4.1. Let P be a normal program with $\mathcal{C}_P \neq \emptyset$ and M_P its vanilla metaprogram. Then $U_P \subset U_{M_P}$.

PROOF. Obvious from the definitions. □

Notice, however, that the property does *not* hold when $\mathcal{C}_P = \emptyset$. Indeed, in that case, U_P contains terms with $*$, while U_{M_P} does not.

4.2. Weak Stratification of M_P

We are now ready to formulate and prove the first of our two main results. It shows that the concept of *weak stratification has a very natural application in the realm of metaprogramming.*

Theorem 4.1. Let P be a stratified normal program. Then M_P, the vanilla metaprogram associated with P, is weakly stratified.

PROOF. We can choose:

$$V_1 = \{clause(t_1, t_2) \mid clause(t_1, t_2) \text{ is a ground instance of a } clause(t, s)\text{-fact in } M_P\}.$$

It immediately follows that L_1 is definite and $H_1 = V_1$. It also follows that

$$\begin{aligned} P_2 = \; & \{solve(empty)\} \\ & \cup \{C \mid C \text{ is a ground instance of } solve(x\&y) \leftarrow solve(x).\, solve(y)\} \\ & \cup \{C \mid C \text{ is a ground instance of } solve(\neg x) \leftarrow not\ solve(x)\} \\ & \cup \{solve(t_1) \leftarrow solve(t_2) \mid clause(t_1, t_2) \in V_1\}. \end{aligned}$$

Now suppose that $P^1, \ldots, P^k$ is a stratification of P. Then we can choose

$$V_2 = \{solve(t) \mid t \in \tau^1\}$$

with

$$\begin{aligned} \tau^1 = \; & \{p'(\bar{t}\,) \mid p/n \in P^1, \bar{t} \in U^n_{M_P}\} \\ & \cup \{t_1 \& t_2 \mid t_1, t_2 \in \tau^1\} \\ & \cup \{f(\bar{t}) \mid f/n \in \mathcal{F}_P, \bar{t} \in U^n_{M_P}\} \\ & \cup \mathcal{C}_{M_P}. \end{aligned}$$

It follows immediately that L_2 is a definite program. Moreover, for $3 \leq i \leq k+1$, we can choose

$$V_i = \{solve(t) \mid t \in \tau^{i-1}\}$$

with

$$\tau^{i-1} = \{p'(\bar{t}) \mid p/n \in P^{i-1}, \bar{t} \in U^n_{M_P}\} \cup \left\{ t_1 \& t_2 \mid t_1, t_2 \in \bigcup_{j \leq i-1} \tau^j \text{ and } t_1 \text{ or } t_2 \in \tau^{i-1} \right\} \cup \{\neg t \mid t \in \tau^{i-2}\}$$

and for $i > k+1$,

$$V_i = \{solve(t) \mid t \in \tau^{i-1}\}$$

with

$$\tau^{i-1} = \left\{ t_1 \& t_2 \mid t_1, t_2 \in \bigcup_{j \leq i-1} \tau^j \text{ and } t_1 \text{ or } t_2 \in \tau^{i-1} \right\} \cup \{\neg t \mid t \in \tau^{i-2}\}.$$

It follows that for all $3 \leq i$,

$$\{solve(\neg t) \leftarrow not\ solve(t) \mid t \in U_{M_P} \backslash \tau^{i-2}\}$$

is the set of clauses with a negative literal in P_i. From this, we can conclude that every $L_i (i \geq 3)$ is definite. Moreover, $\cap_{j<\omega} CH(P_j) = \emptyset$. □

This result shows that the metaprogram at the beginning of Section 3 is indeed weakly stratified, since its object program is stratified. In fact, we can infer from the proof above that our observation at the beginning of Section 3 about the local stratification of the program M'_P can also be generalized. Indeed, once P_2 is obtained, the rest of the stratification process is essentially static: P_2 is locally stratified. It can, therefore, be argued that the concept of *weak stratification* is perhaps a bit too strong for our purposes. However, we conjecture that all results in this paper formulated for *stratified* object programs, can be shown to hold for *weakly* stratified object programs. We will not explicitly address this topic in the rest of this paper, but it is clear that in the latter context, metaprograms will, in general, no longer be "almost" locally stratified.

4.3. A Sensible Semantics for M_P

Now follows our basic result:

Theorem 4.2. Let P be a stratified, language independent normal program and M_P its vanilla metaprogram. Let H_P denote the perfect Herbrand model of P and H_{M_P} the weakly perfect Herbrand model of M_P. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U^r_{M_P} : \quad solve(p'(\bar{t})) \in H_{M_P} \Leftrightarrow \bar{t} \in U^r_P \& p(\bar{t}) \in H_P.$$

PROOF. Suppose $P^1, \ldots, P^k$ is a stratification of P. The proof is through *induction the P-stratum* to which p belongs. Suppose first that $p \in P^1$. Let P_1 name the

collection of clauses in P corresponding to P^1. (P_1 is a definite, language independent program.) Let T_{P_1} name its immediate consequence operator. We learn from the proof of Theorem 4.1 that atoms of the form $solve(p'(\bar{t}))$ are in the V_2 layer of B_{M_P}. However, this means:

$$solve(p'(t)) \in H_{M_P} \Leftrightarrow solve(p'(\bar{t})) \in H_2. \qquad (*)$$

Let T_{L_2} name the immediate consequence operator corresponding to the (infinite) definite ground program L_2. (Throughout the rest of this proof, names such as V_2, H_2, and L_2 refer to the construction showing the weak stratification of M_P.)

We first prove

$$\forall \bar{t} \in U_P^r, \forall n \in \mathbb{N} : p(\bar{t}) \in T_{P_1} \uparrow n \quad \Rightarrow \quad \exists m \in \mathbb{N} : solve(p'(\bar{t})) \in T_{L_2} \uparrow m. \qquad (1)$$

The proof proceeds through induction on n. The base case ($n = 0; T_{P_1} \uparrow 0 = \emptyset$) is trivially satisfied. Now suppose that $p(\bar{t}) \in T_{P_1} \uparrow n, n > 0$. Then there must be at least one clause C in P_1 such that $p(\bar{t}) \leftarrow C_1, \ldots, C_k (k \geq 0)$ is a ground instance of C and $C_1, \ldots, C_k \in T_{P_1} \uparrow (n-1)$. Consider first the case that we have one with $k = 0$. In other words, $p(\bar{t}) \leftarrow$ is a ground instance of a fact in P. In that case, L_2 contains the clause $solve(p'(\bar{t})) \leftarrow solve(empty)$. It follows that $solve(p'(\bar{t})) \in T_{L_2} \uparrow 2$.

Suppose now $k \geq 1$. Then L_2 contains the clause

$$solve(p'(\bar{t})) \leftarrow solve(C_1' \& \cdots \& C_k')$$

as well as

$$solve(C_l' \& \cdots \& C_k') \leftarrow solve(C_l'), solve(C_{l+1}' \& \cdots \& C_k') \qquad \forall 1 \leq l < k.$$

The induction hypothesis guarantees for every C_i the existence of an $m_i \in \mathbb{N}$ such that $solve(C_i') \in T_{L_2} \uparrow m_i$. Let mm denote the maximum of these m_i. It takes only a straightforward proof by induction on l to show

$$\forall 1 \leq l \leq k : \quad solve(C_l' \& \cdots \& C_k') \in T_{L_2} \uparrow (mm + k - l).$$

From this, it follows that, in particular,

$$solve(C_1' \& \cdots \& C_k') \in T_{L_2} \uparrow (mm + k - 1)$$

and therefore

$$solve(p'(\bar{t})) \in T_{L_2} \uparrow (mm + k).$$

This completes the proof of (1). (Notice that in this part of the proof the language independence of P is only implicitly used to deal with the case that $\mathcal{C}_P = \emptyset$. See also the remark following Proposition 4.2.)

Next, we prove

$$\forall \bar{t} \in U_{M_P}^r, \forall n \in \mathbb{N} : \quad solve(p'(\bar{t})) \in T_{L_2} \uparrow n$$
$$\Rightarrow \quad \bar{t} \in U_P^r \,\&\, \exists m \in \mathbb{N} : p(\bar{t}) \in T_{P_1} \uparrow m. \qquad (2)$$

We first define $\mathcal{L}'$ to be the language determined by $\mathcal{R}_{P_1}, \mathcal{F}_{M_P}$, and $\mathcal{C}_{M_P}$. $\mathcal{L}'$ is an extension of $\mathcal{L}_{P_1}$.

The proof again proceeds through an induction on n. The base case ($n = 0; T_{L_2} \uparrow 0 = \emptyset$) is trivially satisfied. Suppose that $solve(p'(\bar{t})) \in T_{L_2} \uparrow n$, where $n > 0$. Then either there is a *clause*-fact in M_P of which $clause(p'(\bar{t}), empty) \leftarrow$ is a ground instance or this is not the case. Suppose first there is. Then P_1 must contain a fact of which $p(\bar{t}) \leftarrow$ is a ground instance in $\mathcal{L}'$. This means that $p(\bar{t}) \in T_{P_1} \uparrow 1$ in $\mathcal{L}'$, but, since P_1 is language independent, this implies that $\bar{t} \in U^r_{P_1}$ (and thus $\bar{t} \in U^r_P$) and $p(\bar{t}) \in T_{P_1} \uparrow 1$.

If there is no such *clause*-fact in M_P, then there must be one with a ground instance $clause\ (p'(\bar{t}), C'_1\ \&\ \cdots\ \&\ C'_k)$, where $k \geq 1$, such that $solve(C'_1\ \&\ \cdots\ \&\ C'_k) \in T_{L_2} \uparrow (n-1)$. A simple induction argument on k shows that we obtain

$$\forall 1 \leq i \leq k : \exists n_i < n \in \mathbb{N} : \quad solve(C'_i) \in T_{L_2} \uparrow n_i.$$

Through the induction hypothesis, we get

$$\forall 1 \leq i \leq k : \exists m_i \in \mathbb{N} : \quad C_i T_{P_1} \uparrow m_i \,\&\, \overline{t_i} \in U^{r_i}_P$$

(where $\bar{t}_i$ is the tuple of arguments appearing in the atom C_i and r_i is the arity of its predicate). From the above and the fact that P_1 is language independent, (2) follows.

The desired result for the bottom stratum is an immediate consequence of (*), (1), and (2).

Now, let $p \in P^i (i > 1)$ and assume that the result is obtained for all $q \in P^j, j < i$. We first prove the *if* part. From $p(\bar{t}) \in H_P$ and $p \in P^i$, we know there must be a clause in P such that

$$p(\bar{t}) \leftarrow C_1, \ldots, C_n \quad (n \geq 0)$$

is a ground instance of it and all positive $C_j \in H_P$ and for no negative $C_j = not\, B_j, B_j \in H_P$. Now:

- If there is no C_j containing a predicate symbol $\in P^i$, then we can prove (through induction on i)

  $$solve(p'(\bar{t})) \leftarrow \in L_{i+1}.$$

  From this, the desired result follows immediately.
- In the other case, we can, without loss of generality, suppose that $C_1, \cdots, C_l$ are the literals containing predicate symbols $\in P^i$. We then have in L_{i+1}:

  $$\begin{aligned} &solve(p(\bar{t})) \leftarrow solve(C'_1 \& \cdots \& C'_n) \\ &solve(C'_1 \& \cdots \& C'_n) \leftarrow solve(C'_1), solve(C'_2 \& \cdots \& C'_n) \\ &solve(C'_2 \& \cdots \& C'_n) \leftarrow solve(C'_2), solve(C'_3 \& \cdots \& C'_n) \\ &\quad \vdots \\ &solve(C'_l \& \cdots \& C'_n) \leftarrow solve(C'_l). \end{aligned}$$

  (Here, if C_n is a negative literal $not\, B, C'_n$ of course denotes $\neg B'$.) $solve(p'(\bar{t})) \in H_{M_P}$ now follows through an induction argument analogous to the one used in the proof of (1) above.

Finally, we turn to the *only-if* part. We have one of the following two cases:

- If $solve(p'(\bar{t}\,)) \leftarrow \in L_{i+1}$, then there is a clause in P such that

$$p(\bar{t}\,) \leftarrow C_1, \cdots, C_n \qquad (n \geq 0)$$

  is a ground istance of it, all C_j contain predicate symbols $\in \cup_{j<i} P^j$ and are true in H_P. The desired result now follows from the language independence of P.
- Otherwise, L_{i+1} must contain a clause

$$solve(p'(\bar{t}\,)) \leftarrow solve(C'_1 \& \cdots \& C'_n)$$

  such that $solve(C'_1 \& \cdots \& C'_n) \in H_{i+1}$. (Here again, C'_n possibly denotes $\neg B'$ for some atom B.) Let us (without loss of generality) suppose that $C_1, \ldots, C_l$ are the only literals containing predicate symbols $\in P^i$. Then it follows that L_{i+1} also contains

$$\begin{aligned} &solve(C'_1 \& \cdots \& C'_n) \leftarrow solve(C'_1), solve(C'_2 \& \cdots \& C'_n) \\ &solve(C'_2 \& \cdots \& C'_n) \leftarrow solve(C'_2), solve(C'_3 \& \cdots \& C'_n) \\ &\quad \vdots \\ &solve(C'_l \& \cdots \& C'_n) \leftarrow solve(C'_l). \end{aligned}$$

An induction argument similar to the one in the proof of (2) above now brings us the desired result. □

Strictly speaking, $\bar{t} \in U_P^r$ is of course implied by $p(\bar{t}\,) \in H_P$, but since we judge the former fact to be an important result in its own right, we have chosen to include the statement explicitly in the formulation of the theorem.

Theorem 4.2 shows that untyped nonground vanilla metaprograms have a very reasonable Herbrand model semantics for a large class of object programs. Notice, however, that it does not incorporate any results on "negative" information. It is obvious that a straightforward generalization to atoms of the form $solve(\neg p'(\bar{t}\,))$ is not possible:

Example 4.1. Indeed, consider a very simple stratified, language independent object program P:

$$p(a) \leftarrow .$$

We have $solve(\neg p'(empty)) \in H_{M_P}$, while of course $empty \notin U_P$.

However, we do have the result below:

Corollary 4.1. Let P be a stratified, language independent program with $\mathcal{C}_P \neq \emptyset$ and M_P its vanilla metaprogram. Let H_P denote the perfect Herbrand model P and let H_{M_P} be the weakly perfect Herbrand model of M_P. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U_P^r : \quad solve(\neg p'(\bar{t}\,)) \in H_{M_P} \quad \Leftrightarrow \quad p(\bar{t}\,) \notin H_P.$$

PROOF. From Theorem 4.2, we have

$$\forall \bar{t} \in U_P^r : \quad solve(p'(\bar{t}\,)) \in H_{M_P} \quad \Leftrightarrow \quad p(\bar{t}\,) \in H_P,$$

and, therefore,

$$\forall \bar{t} \in U_P^r : \quad not\ solve(p'(\bar{t})) \text{ is satisfied in } H_{M_{P'}} \quad \Leftrightarrow \quad p(\bar{t}) \notin H_P.$$

However, then Definition 4.2 implies

$$\forall \bar{t} \in U_P^r : \quad solve(\neg p'(\bar{t})) \in H_{M_{P'}} \quad \Leftrightarrow \quad p(\bar{t}) \notin H_P.$$

□

Obviously, this result can be extended to $\bar{t} \in U_{M_{P'}}^r$, if one so desires. Note that it is essential that $\mathcal{C}_P \neq \emptyset$.

To conclude this section, we briefly dwell upon the "strength" of Theorem 4.2. In other words, can a similar result be proven for classes of programs significantly larger than the class of language independent programs? We believe this not to be the case. One argument in favor of this is the fact that the proof of Theorem 4.2 relies on the language independence of the object program in a very natural way. However, one direction of the $\Leftrightarrow$ *can* be shown to hold for (almost) *all definite* object programs (in Proposition 4.2, M_P is supposed to be defined *without* the third clause of Definition 4.2; see Definition 7.7 and 7.8):

Proposition 4.2. Let P be a definite program with $\mathcal{C}_P \neq \emptyset$ and M_P its vanilla metaprogram. Let H_P denote the least Herbrand model of P and $H_{M_{P'}}$ the least Herbrand model of M_P. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U_P^r : \quad p(\bar{t}) \in H_P \quad \Rightarrow \quad solve(p'(\bar{t})) \in H_{M_{P'}}.$$

PROOF. The result follows immediately from Lemma 9 in [10]. □

(In fact, we can just as well use M_P as defined in Definitions 4.2 and 4.3, as the first part of the proof for Theorem 4.2 shows.) Again, $\mathcal{C}_P \neq \emptyset$ is an essential condition. Indeed, the least Herbrand model of a non-language-independent program contains nonpropositional elements. All terms in these atoms are of course ground and the only constant involved is $*$, which is not even present in the language of the metaprogram. It should also be mentioned that when the restriction to terms in the *object* universe is imposed as a precondition, the reverse of Proposition 4.2 does hold. This is proved in [25].

The following example shows that Proposition 4.2 *cannot* be extended to the class of *stratified* programs and their (weakly) perfect Herbrand models.

Example 4.2. The following program P is stratified, but *not* language independent:

$$\begin{aligned} r(x) &\leftarrow not\, p(x) \\ p(x) &\leftarrow not\, q(y) \\ q(a) &\leftarrow . \end{aligned}$$

We find that $r(a) \in H_P$, and yet $solve(r'(a)) \notin H_{M_{P'}}$.

It seems then that Theorem 4.2, Corollary 4.1, and Proposition 4.2 are about the best we can do in the context of "classical" ground Herbrand semantics. In Section

7, we show that it *is* often possible to drop the condition of language independence in the framework of an extended Herbrand semantics, designed to mirror more closely the operational behavior of logic programs. However, first, in the next few sections, we present some interesting extensions of the basic results obtained above.

5. EXTENSIONS

Theorem 4.2 is interesting because, for a large class of programs, it provides us with a reasonable semantics for nonground vanilla metaprogramming. However, it also shows that we do not seem to *gain* much by this kind of programming. Indeed, (the relevant part of) the metasemantics can be *identified* with the object semantics. So why go through the trouble of writing a metaprogram in the first place? The answer lies, of course, in useful *extensions* of the vanilla interpreter (see, e.g., [40] and further references given there). In this section, we study metaprograms that capture the essential characteristics of many such extensions. We will first consider definite object and metaprograms and turn to the normal case afterward.

5.1. Definite Programs and their Extended Metaprograms

Definition 5.1. A definite program of the following form will be called *extended* $(d-)$*metainterpreter.*

$$\begin{aligned} &solve(empty, t_{11}, \cdots, t_{1n}) \leftarrow C_{11}, \cdots, C_{1m_1} \\ &solve(x\&y, t_{21}, \cdots, t_{2n}) \leftarrow solve(x, t_{31}, \cdots, t_{3n}), solve(y, t_{41}, \cdots, t_{4n}), \\ &\qquad\qquad C_{21}, \cdots, C_{2m_2} \\ &solve(x, t_{51}, \cdots, t_{5n}) \leftarrow clause(x, y), solve(y, t_{61}, \cdots, t_{6n}), \\ &\qquad\qquad C_{31}, \cdots, C_{3m_3}, \end{aligned}$$

where the t_{ij} terms are extra arguments of the *solve* predicate and the C_{kl} atoms extra conditions in its body, together with defining clauses for any other predicates occurring in the C_{kl} (none of which contain *solve* or *clause*).

The prefix "d" serves to make a distinction with normal metainterpreters. However, when it is clear from the context whether a definite or a normal metaprogram is intended, we will often not write down that "d" explicitly.

Definition 5.2. Let P be a definite program and let E be an extended $(d$-$)$metainterpreter. Then E_P, the *E-extended d-metaprogram associated with P*, is the definite program consisting of E together with a fact of the form

$$clause(A', B_1' \& \cdots \& B_n') \leftarrow$$

for every clause $A \leftarrow B_1, \cdots, B_n$ in P and a fact of the form

$$clause(A', empty) \leftarrow$$

for every fact $A \leftarrow$ in P.

As an example of this kind of metaprogramming, we include the following program E, adapted from [40]. It builds proof trees for definite object level programs and queries.

Example 5.1.

$$solve(empty, empty) \leftarrow$$
$$solve(x\&y,\ proof\, x\&\, proof\, y) \leftarrow solve(x, proof\, x), solve(y, proof\, y)$$
$$solve(x, x\, if\ proof) \leftarrow clause(x, y), solve(y, proof).$$

As is illustrated in [40], the proof trees thus constructed can be used as a basis for explanation facilities in expert systems. Further examples can be found in, e.g., [39].

We have the following proposition:

Proposition 5.1. Let P be a definite, language independent program and E_P an E-extended d-metaprogram associated with P. Let H_P and H_{E_P} denote their respective least Herbrand models. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U^r_{E_P}: \quad \big(\exists \bar{s} \in U^n_{E_P}: \quad solve(p'(\bar{t}), \bar{s}) \in H_{E_P}\big)$$
$$\Rightarrow \quad \bar{t} \in U^r_P \,\&\, p(\bar{t}) \in H_P.$$

PROOF. It follows immediately from an obvious property of definite logic programs that $solve(p'(\bar{t}), \bar{s}) \in H_{E_P}$ implies $solve(p'(\bar{t})) \in \mathcal{L}_{E_P} - H_{M_P}$ (M_P's least $\mathcal{L}_{E_P}$-Herbrand model). Let $\mathcal{L}'$ be the language determined by $\mathcal{R}_P$, $\mathcal{F}_{E_P}$, and $\mathcal{C}_{E_P}$. $\mathcal{L}'$ is an extension of $\mathcal{L}_P$. The result now follows from the language independence of P. □

It can be noted that the right-hand side of the implication in Proposition 5.1 is equivalent with $\bar{t} \in U^r_{M_P} \& solve(p'(\bar{t})) \in H_{M_P}$ (where M_P denotes the vanilla d-metaprogram associated with P, defined as in Definition 7.8, and H_{M_P} is its least Herbrand model). This follows from Theorem 11 in [10], the proof of which is similar to the bottom stratum part of the proof for Theorem 4.2 above.

Proposition 5.1 essentially ensures us that working with extended metaprograms is "safe" for definite, language independent programs. Indeed, no unwanted solutions of the kind mentioned in Section 1 are produced. Further research can perhaps determine conditions on E that allow an equivalence in Proposition 5.1. We know such extended interpreters exist: The proof tree building program in Example 5.1 above presents an instance. In general, programs where the extra arguments and conditions neither cause failures nor additional bindings on the main arguments are obviously safe. (See Section 7.4 for a related comment.)

5.2. Normal Extensions

We will now address normal object programs and their normal extended metaprograms. The definitions we set out with, are of course very similar to those in the previous subsection.

Definition 5.3. A normal program of the following form will be called *extended program metainterpreter*:

$$\begin{aligned}
&solve(empty, t_{11}, \cdots, t_{1n}) \leftarrow C_{11}, \cdots, C_{1m_1}\\
&solve(x\&y, t_{21}, \cdots, t_{2n}) \leftarrow solve(x, t_{31}, \cdots, t_{3n}), solve(y, t_{41}, \cdots, t_{4n}),\\
&\qquad\qquad C_{21}, \cdots, C_{2m_2}\\
&solve(\neg x, t_{51}, \cdots, t_{5n}) \leftarrow not\ solve(x, t_{61}, \cdots, t_{6n}), C_{31}, \cdots, C_{3m_3}\\
&solve(x, t_{71}, \cdots, t_{7n}) \leftarrow clause(x, y), solve(y, t_{81}, \cdots, t_{8n}),\\
&\qquad\qquad C_{41}, \cdots, C_{4m_4},
\end{aligned}$$

where the t_{ij} terms are extra arguments of the *solve*-predicate and the C_{kl} literals extra conditions, defined through a stratified program included in the extended metainterpreter (but not containing *solve* or *clause*).

Definition 5.4. Let P be a normal program and E an extended metainterpreter. Then E_P, the *E-extended metaprogram associated with* P, is the normal program consisting of E together with a fact of the form

$$clause(A', \cdots \& B' \& \cdots \& \neg C' \& \cdots) \leftarrow$$

for every clause $A \leftarrow \cdots, B, \cdots, not\, C, \cdots$ in P and a fact of the form

$$clause(A', empty) \leftarrow$$

for every fact $A \leftarrow$ in P.

The first question now is: Are such programs weakly stratified? The following proposition shows they indeed are, when the object program is stratified.

Proposition 5.2. Let P be a stratified program and E an extended metainterpreter. Then E_P, the E-extended metaprogram associated with P, is weakly stratified.

PROOF. A construction completely analogous to the one in the proof of Theorem 4.1 can be used, since;

- The strata of the program that defines the C_{kl} literals can be considered first.
- We can still devide he *solve*-atoms in strata, based on the structure of their first argument. □

Having established this result, we would now like to generalize Proposition 5.1. However, the following simple examples demonstrate that this is not possible.

Example 5.2.

$$\begin{aligned}
P:\ &p \leftarrow not\ q\\
&q\ .
\end{aligned}$$

Notice P is language independent and (trivially) range restricted.

E : First 3 clauses as in M (Definition 4.2)

$$solve(x) \leftarrow clause(x, y), solve(y), good(y)$$
$$good(\neg q').$$

We have $p \notin H_P$ and yet $solve(p') \in H_{E_P}$.

Example 5.3.

$$P : p(x, y) \leftarrow r(x), not\ q(x)$$
$$q(a) \leftarrow h(a)$$
$$r(a)$$
$$h(a).$$

Notice P is language independent, but *not* range restricted.

$$E : \text{First 3 clauses as in } M \text{ (Definition 4.2)}$$
$$solve(x) \leftarrow clause(x, y), solve(y), not\ bad(y)$$
$$bad(h'(a)).$$

We have $solve(p'(a, empty)) \in H_{E_P}$ (and, of course, $(a, empty) \notin U_P^2$).

However, we *do* have the following result:

Proposition 5.3. Let P be a stratified, range restricted program and let E_P be an extended metaprogram associated with P. Let H_{E_P} be its weakly perfect Herbrand model. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U_{E_P}^r : \quad \left(\exists \bar{s} \in U_{E_P}^n : solve(p'(\bar{t}), \bar{s}) \in H_{E_P}\right) \Rightarrow \bar{t} \in U_P^r.$$

PROOF. First, observe that all fact in a range restricted program are ground. Second, all variables in the head of a clause appear in positive body literals. Therefore, basically, when performing deductions, all variables are instantiated with terms propagated upward from the ground facts. This property carries over to the metaprogram and ensures that the $\bar{t}$-arguments cannot be instantiated with terms outside U_P^r. A fully formal elaboration of this argument involves a completely straightforward induction proof. □

A few remarks are in order:

- Example 5.2 shows that this proposition cannot be strengthened to also include $p(\bar{t}) \in H_P$ in the right-hand side of the implication. Additionally, in view of what might be required in actual applications, this seems very natural. Consider, for example, a jury finding a guilty person innocent through lack of sufficient proof. When formalized by means of an extended metainterpreter, this would result in something like $solve(innocent')$ being true at the metalevel, while *innocent* would be false at the object level.
- Proposition 5.3 does certify that no nonsensical $solve(p'(\bar{t}, \bar{s}))$ atoms appear in the weakly perfect Herbrand model of the extended metaprogram, at least for *range restricted* object programs. Indeed, Example 5.3 shows that even this minimum result *cannot* be extended to the class of all stratified, *language*

independent programs. In other words, language independence proves to be too weak a concept in the context of extended normal metaprograms.

6. AMALGAMATION

6.1. A Justification for Overloading

In the this section, we extend some of the results of Section 4 to provide a semantics for a limited form of amalgamation. The simplest example of the kind of programs we will consider is the (textual) combination $P + M_P$ of the clauses of an object program P with the clauses for its associated vanilla program M_P. A more complex case is obtained by further (textually) extending $P + M_P$ with additional *clause*/2 facts and statements, covering the clauses in M_P itself. In the most general case, we also allow the occurrence of *solve*/1 calls in the bodies of clauses of P. Futhermore, we will impose the use of one particular functorization $\mathcal{F}_{R_P}$, namely, the one in which all functors in $\mathcal{F}_{R_P}$ are identical to their associated predicate symbols in $\mathcal{R}_P$. (In the more complex cases, we will proceed similarly for the predicates *solve* and *clause*.)

We first address the more basic problem with the semantics of such programs, caused by overloading the symbols in the language. Clearly, the predicate symbols of P occur both as predicate symbol and as functor in $P + M_P$ (and in any further extensions). Now, although this was not made explicit in, e.g., [29], an underlying assumption of first order logic is that the class of functors and the class of predicate symbols of a first order language $\mathcal{L}$, are disjoint (see, e.g., [14]). So, if we aim to extend our results to amalgamated programs—without introducing any kind of naming to avoid the overloading—we need to verify whether the constructions, definitions, and results on the foundations of logic programming are still valid if the functors and predicate symbols of the language overlap. Of course, in that case, there is, in general, no way to distinguish well-formed formulas from terms. They as well have a nonempty intersection. However, this causes no problem in the definition of preinterpretations, variable and term assignments, and interpretations. It is clear, however, that a same syntactical object can be both term and formula and can therefore be given two different meanings, one under the preinterpretation and variable assignment, the other under the corresponding interpretation and variable assignment. This causes no confusion on the level of truth assignment to well-formed formulas under an interpretation and a variable assignment. This definition performs a complete parsing of the well-formed formulas, making sure that the appropriate assignments are applied for each syntactic substructure. In particular, it should be noted that no paradoxes can be formulated in these languages, since each formula obtains a unique truth value under every interpretation and variable assignment.

On the level of declarative logic program semantics, the main results both for definite programs and for (weakly) stratified normal programs remain valid in the extended languages. Thus, the amalgamated programs we aim to study can be given a unique semantics. Below, we demonstrate that it is also a sensible semantics.

6.2. Amalgamated Vanilla Metaprograms

From here on, throughout the rest of the paper, functorizations will always contain *exactly the same symbols* as their corresponding sets of predicate symbols. This leads to an increased flexibility in considering metaprograms with several layers. In fact, as shown in Section 6.3, we can now deal with an unlimited amount of metalayers. However, we first briefly consider a completely straightforward extension: Including the object program in the resulting metaprogram.

Definition 6.1. Let P be a normal program and M_P its associated vanilla metaprogram (see Definition 4.3). Then we call the textual combination $P + M_P$ of P and M_P the *amalgamated vanilla metaprogram associated with* P.

Notice that $\mathcal{L}_{P+M_P}$ is determined by:

- $\mathcal{R}_{P+M_P} = \mathcal{R}_P \cup \{solve, clause\}$.
- $\mathcal{F}_{P+M_P} = \mathcal{F}_P \cup \mathcal{R}_P \cup \{\&, \neg\}$.
- $\mathcal{C}_{P+M_P} = \mathcal{C}_P \cup \{empty\}$.

We immediately have the following:

$$U_{P+M_P} = U_{M_P}.$$

The semantic properties of $P + M_P$ are, of course, straightforward variants of those obtained above for M_P. First, we have the following:

Proposition 6.1. Let P be a stratified program. Then $P + M_P$, its associated amalgamated vanilla metaprogram, is weakly stratified.

PROOF. For the construction in Proposition 3.2, we can first consider the strata of P and then continue as in the proof of Theorem 4.1. □

This enables us to formulate the next theorem:

Theorem 6.1. Let P be a stratified, language independent program, M_P its vanilla, and $P + M_P$ its amalgamated vanilla metaprogram. Let H_P, H_{M_P}, and H_{P+M_P} denote their (weakly) perfect Herbrand models. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\begin{aligned}
&\forall \bar{t} \in U^r_{P+M_P}: \quad solve(p(\ \bar{t}\ )) \in H_{P+M_P} \ \Leftrightarrow\ p(\ \bar{t}\ ) \in H_{P+M_P},\\
&\forall \bar{t} \in U^r_{P+M_P}: \quad solve(p(\ \bar{t}\ )) \in H_{P+M_P} \ \Leftrightarrow\ \bar{t} \in U^r_P \& p(\ \bar{t}\ ) \in H_P,\\
&\forall t \subset U^r_{P+M_P}: \quad solve(t) \in H_{P+M_P} \ \Leftrightarrow\ t \in U_{M_P} \& solve(t) \in H_{M_P}.
\end{aligned}$$

PROOF. Obvious from Definition 6.1 and Theorem 4.2. □

Naturally, adapted versions of Corollary 4.1 and Proposition 4.2 also hold.

Considering *extended amalgamated metaprograms* is likewise straightforward. We will not do this explicitly and only illustrate by an example the extra programming power one can gain in this context.

Example 6.1. In applications based on the proof tree recording program from Example 5.1, it may be the case that users are not interested in branches for particular predicates. To reflect this, clauses of the form

$$solve(p(x), some\_info) \leftarrow p(x)$$

can be added (combined with extra measures to avoid also using the standard clause for these cases).

6.3. Meta2-Programs

In this section, we consider metaprograms that include *clause*-information for the M_P-clauses themselves, thus allowing the use of an unlimited amount of metalayers. Programming of this kind is relevant in, e.g., the contexts of reasoning about reasoning (see, e.g., [26]) and proof-plan construction and manipulation (see, e.g., [20]).

We start with a formal definition.

Definition 6.2. Let P be a normal program. Then M_P^2, the *vanilla meta2 program associated with* P, is the program M (see Definition 4.2) together with the following clause:

$$clause(clause(x, y), empty) \leftarrow clause(x, y) \qquad (*)$$

and a fact of the form

$$clause(A, \cdots \& B \& \cdots \& \neg C \& \cdots) \leftarrow$$

for every clause $A \leftarrow \cdots, B, \cdots, not\, C, \cdots$ in P or M and a fact of the form

$$clause(A, empty) \leftarrow$$

for every fact $A \leftarrow$ in P or M.

Notice that this definition essentially adds to the vanilla metaprogram *clause* facts for the four *solve* clauses in M and for every *clause* fact. An actual textual execution of the latter intention would, however, demand the addition of an infinite amount of *clause* facts. Indeed, we do not only want *clause* facts for the clauses in P and M and the *clause* facts in M_P, but also for the *clause* facts about these *clause* facts, etc. Rule (*) in Definition 6.2 covers all the "facts about facts" cases. Definition 6.2 implies that $\mathcal{L}_{M_P^2}$ is determined by:

- $\mathcal{R}_{M_P^2} = \{solve, clause\}$.
- $\mathcal{F}_{M_P^2} = \mathcal{F}_P \cup \mathcal{R}_P \cup \{solve, clause, \&, \neg\}$.
- $\mathcal{C}_{M_P^2} = \mathcal{C}_P \cup \{empty\}$.

It follows that $\mathcal{C}_P \neq \emptyset \Longrightarrow U_P \subset U_{M_P^2}$.

The proof of the following theorem is a straightforward adaptation of the one for Theorem 4.1.

Theorem 6.2. Let P be a stratified program. Then M_P^2, the vanilla meta2 program associated with P, is weakly stratified.

PROOF. Suppose that $P^1, \ldots, P^k$ is a stratification of P. To see that M_P^2 is indeed weakly stratified, it suffices to take the sets V_i in the construction described in Proposition 3.2 as follows:

- $V_1 = \{clause(t_1, t_2) \mid t_1, t_2 \in U_{M_P^2}\}$.
- $V_2 = \{solve(t) \mid t \in \tau^1\}$ with

$$\begin{aligned}\tau^1 \;=\; & \{p(\,\bar{t}\,) \mid p/n \in P^1, \bar{t} \in U^n_{M_P}\} \\ & \cup \{t_1 \& t_2 \mid t_1, t_2 \in \tau^1\} \\ & \cup \{f(\,\bar{t}\,) \mid f/n \in \mathcal{F}_P, \bar{t} \in U^n_{M_P}\} \\ & \cup \mathcal{C}_{M_P} \\ & \cup \{clause(t_1, t_2) \mid t_1, t_2 \in U_{M_P^2}\} \\ & \cup \{solve(t) \mid t \in \tau^1\}.\end{aligned}$$

- For $3 \leq i \leq k+1$, $V_i = \{solve(t) \mid t \in \tau^{i-1}\}$ with

$$\begin{aligned}\tau^{i-1} \;=\; & \{p(\,\bar{t}\,) \mid p/n \in P^{i-1}, \bar{t} \in U^n_{M_P}\} \\ & \cup \left\{t_1 \& t_2 \mid t_1, t_2 \in \bigcup_{j \leq i-1} \tau^j \text{ and either } t_1 \text{ or } t_2 \in \tau^{i-1}\right\} \\ & \cup \{\neg t \mid t \in \tau^{i-2}\} \\ & \cup \{solve(t) \mid t \in \tau^{i-1}\}.\end{aligned}$$

- For $k+1 < i$, $V_i = \{solve(t) \mid t \in \tau^{i-1}\}$ with

$$\begin{aligned}\tau^{i-1} \;=\; & \left\{t_1 \& t_2 \mid t_1, t_2 \in \bigcup_{j \leq i-1} \tau^j \text{ and either } t_1 \text{ or } t_2 \in \tau^{i-1}\right\} \\ & \cup \{\neg t \mid t \in \tau^{i-2}\} \\ & \cup \{solve(t) \mid t \in \tau^{i-1}\}. \quad \square\end{aligned}$$

At least part of the following theorem will by now no longer come as a surprise.

Theorem 6.3. Let P be a stratified, language independent program and M_P^2 its vanilla meta2 program. Let H_P denote the perfect Herbrand model of P and let $H_{M_P^2}$ be the weakly perfect Herbrand model of M_P^2. Then the following holds:

$$\forall t \in U_{M_P^2}: \quad solve(solve(t)) \in H_{M_P^2} \quad \Leftrightarrow \quad solve(t) \in H_{M_P^2}.$$

Moreover, the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U^r_{M_P^2}: \quad solve(p(\,\bar{t}\,)) \in H_{M_P^2} \quad \Leftrightarrow \quad \bar{t} \in U^r_P \;\&\; p(\,\bar{t}\,) \in H_P.$$

PROOF. The proof of the second equivalence is analogous to the proof of Theorem 4.2. To prove the first, we discern between different possibilities for the structure of t.

- $t = empty$. From Definition 6.2, it clearly follows that both $solve\,(solve(empty))$ and $solve(empty)$ are in $H_{M_P^2}$.
- $t = t_1 \& t_2$ and suppose the equivalence holds for t_1 and t_2. Then we have:

$$\begin{aligned}
& solve(solve(t)) \in H_{M_P^2} \\
\Leftrightarrow \quad & \exists y\ clause(solve(t_1 \& t_2), y) \text{ and } solve(y) \in H_{M_P^2} \\
\Leftrightarrow \quad & solve(solve(t_1) \& solve(t_2)) \in H_{M_P^2} \\
\Leftrightarrow \quad & solve(solve(t_1)) \text{ and } solve(solve(t_2)) \in H_{M_P^2} \\
\Leftrightarrow \quad & solve(t_1) \text{ and } solve(t_2) \in H_{M_P^2} \\
\Leftrightarrow \quad & solve(t_1 \& t_2) \in H_{M_P^2}
\end{aligned}$$

- $t = \neg t'$ and suppose the equivalence holds for t'. Then we have:

$$\begin{aligned}
& solve(solve(t)) \in H_{M_P^2} \\
\Leftrightarrow \quad & \exists y\ clause(solve(\neg t'), y) \text{ and } solve(y) \in H_{M_P^2} \\
\Leftrightarrow \quad & solve(\neg solve(t')) \in H_{M_P^2} \\
\Leftrightarrow \quad & solve(solve(t')) \notin H_{M_P^2} \\
\Leftrightarrow \quad & solve(t') \notin H_{M_P^2} \\
\Leftrightarrow \quad & solve(\neg t') \in H_{M_P^2}.
\end{aligned}$$

- Finally, we deal with the remaining cases.

$$\begin{aligned}
& solve(solve(t)) \in H_{M_P^2} \\
\Leftrightarrow \quad & \exists y\ clause(solve(t), y) \text{ and } solve(y) \in H_{M_P^2} \\
\Leftrightarrow \quad & \exists y' solve(clause(t, y')) \& solve(y')) \in H_{M_P^2} \\
\Leftrightarrow \quad & \exists y' solve(clause(t, y')) \text{ and } solve(solve(y')) \in H_{M_P^2} \\
\Leftrightarrow \quad & \exists y', y'' clause(clause(t, y'), y''), solve(y'') \\
& \text{and } solve(solve(y')) \in H_{M_P^2} \\
\Leftrightarrow \quad & \exists y' clause(t, y') \text{ and } solve(solve(y')) \in H_{M_P^2}.
\end{aligned}$$

Now suppose $t \in \tau^i$. Then we can use induction on the level at which the $T_{L_{i+1}}$-operator derives $solve(solve(t))$. Indeed, $solve(solve(y'))$ will be derived at a lower level, and therefore we have:

$$\begin{aligned}
\Leftrightarrow \quad & \exists y' clause(t, y') \text{ and } solve(y') \in H_{M_P^2} \\
\Leftrightarrow \quad & solve(t) \in H_{M_P^2}. \quad \square
\end{aligned}$$

Theorem 6.3 shows that vanilla meta2 programs have a sensible Herbrand semantics, just like plain vanilla metaprograms. Notice that the language independence of P is not used in the proof of the first equivalence.

It is obvious that Corollary 4.1 can be rephrased for meta2 programs, and we also have the following:

Corollary 6.1. Let M_P^2 be the vanilla meta2 program associated with a stratified object program P. Let $H_{M_P^2}$ denote its weakly perfect Herbrand model. Then the following holds:

$$\forall t \in U_{M_P^2}: \quad solve(\neg solve(t)) \in H_{M_P^2} \Leftrightarrow solve(t) \notin H_{M_P^2}.$$

PROOF. Immediate from Definition 6.2 and the first equivalence in Theorem 6.3. □

Various amalgamated and/or extended meta2 programs can be treated. We will just point out one interesting further step it is possible to take. Indeed, we can consider meta2 programs in which the "object" clauses contain metacalls. It is clear that we can in such cases no longer discern between an object level and a metalevel. Results similar to what we obtained before make no sense, but we *can* state the following proposition:

Proposition 6.2. Let P be a stratified program and let $P + M_P^2$ be its amalgamated vanilla meta2 program. *Let PM be a program textually identical to $P + M_P^2$, except that an arbitrary number of atoms A in the bodies of clauses in the P part of it have been replaced by $solve(A)$. Then the following holds:*

$$H_{P+M_P^2} = H_{PM}$$

PROOF. (Sketch) First, both $P + M_P^2$ and PM can be shown to be weakly stratified. For $PM, p(t)$ and $solve(p(t))$ atoms of course have to be taken together in the same stratum. Moreover, the same can be done in the dynamic stratification of $P + M_P^2$. A proof through induction can then be produced to show that every layer in one program has the same least Herbrand model as its corresponding layer in the other program. □

Notice that language independence is not immediately relevant here, since $P + M_P^2$ and PM have identical underlying languages.

Example 6.2. In this framework, we can address interesting examples from [5]. Consider, e.g., the following clause, telling us that a person is innocent when he is not found guilty:

$$innocent(x) \leftarrow person(x), not\ solve(guilty(x)).$$

Of course, such possibilities only become really interesting when using extended metainterpreters, involving, e.g., an extra *solve* argument limiting the resources available for proving a person's guilt. Results similar to those in Section 5, but now pertaining to the relationship between PM-like programs and their extended versions, can be stated and proved.

Further extensions are possible, but we believe that the above sufficiently illustrates the flexibility, elegance, and power of our approach.

Finally, Examples 6.1 and 6.2 indicate how the amalgamated metaprograms considered in this section, provide a basis for incorporating *reflection*. More comments and references on this subject can be found in [31].

7. S-SEMANTICS FOR METAINTERPRETERS

7.1. Introduction

We believe that in many applications, the conditions we imposed on the object programs in the past few sections are very naturally satisfied. However, it is a fact that our basic results no longer hold for classes of object programs considerably surpassing the limitation of language independence. In spite of the above expressed belief, this can be regarded as a somewhat annoying limitation. Indeed, the actual *practice* of metaprogramming reaches beyond this boundary, without experiencing much trouble. The underlying reason for this phenomenon is the fact that least/(weakly) perfect Herbrand semantics does not really accurately reflect the operational behavior of many logic programs. Indeed, Herbrand models contain only ground atoms, while logic program execution often produces *nonground answer substitutions*.

In [15] and [16], nonground Herbrand models are introduced to bridge the gap between declarative semantics and operational behavior. In this section, we will show that many of our results can be generalized beyond language independence in a context of the so-called *S-semantics*. However, since currently S-semantics is only fully developed for definite programs, this whole section is restricted to *definite* object and metaprograms.

7.2. S-Semantics

We first recapitulate some relevant basic notions and results concerning the S-semantics for definite logic programs, as it was introduce in [15] and [16].

For atoms A and A', we define $A \leq A'$ (A is *more general* than A') iff there exists a substitution θ such that $A\theta = A'$. The relation $\leq$ is a preorder. Let $\approx$ be the associated equivalence relation (*renaming*). (Similarly for terms.) Then we can define the following.

Definition 7.1. Let P be a definite program with underlying language $\mathcal{L}_P$. Then its *S-Herbrand universe* U_P^S is the quotient set of all terms in $\mathcal{L}_P$ with respect to $\approx$.

So, U_P^S basically contains *all* possible terms, not only ground ones. Notice, however, that terms which are renamings of each other are considered to be one and the same element of U_P^S. The following definition similarly extends the concept of Herbrand *base*.

Definition 7.2. Let P be a definite program with underlying language $\mathcal{L}_P$. Then its *S-Herbrand base* B_P^S is the quotient set of all atoms in $\mathcal{L}_P$ with respect to $\approx$.

We can now extend the notions of *interpretation*, *truth*, and *model*.

Definition 7.3. Let P be a definite program. Any subset I_P^S of B_P^S is called an *S-Herbrand interpretation* of P.

Definition 7.4. Let P be a definite program and I_P^S an S-Herbrand interpretation of P.

- A (possibly nonground) *atom* A in $\mathcal{L}_P$ is *S-true* in I_P^S iff there exists an atom A', such that (the equivalence class of) A' belongs to I_P^S and $A' \leq A$.
- A definite *clause* $A \leftarrow B_1, \ldots, B_n$ in $\mathcal{L}_P$ is *S-true* in I_P^S iff for every $B_1', \ldots, B_n'$ belonging to I_P^S, if there exists a most general unifier $\theta = mgu((B_1', \ldots, B_n'), (B_1, \ldots, B_n))$, then $A\theta$ belongs to I_P^S.

Definition 7.5. Let I_P^S be an S-Herbrand interpretation of a definite program P. I_P^S is an *S-Herbrand model* of P iff every clause of P is S-true in I_P^S.

It is clear that S-Herbrand interpretations and models contain nonground atoms. Notice that the notion of S-truth is defined differently for atoms and for facts (i.e., clauses with no literals in the body). The reason for demanding $A\theta \in I_P^S$, instead of $A\theta$ S-true in I_P^S, is the wish to attach a different semantics to programs such as P_2 and P_3 in Example 7.1 below. Definitions 7.3 to 7.5 are taken from [15]. In [16], a more elegant, but also slightly more elaborate approach leads to the same results.

On the set of S-Herbrand interpretations of a given program, we impose an ordering through set inclusion, just as in the case of "ordinary" ground Herbrand interpretations. We can then include the following result from [15].

Theorem 7.1. For every definite logic program P, there is a unique least S-Herbrand model H_P^S.

We will consider this least S-Herbrand model of a definite program P as the description of its S-semantics.

A fixpoint characterization of the least S-Herbrand model is possible.

Definition 7.6. Let P be a definite program. The mapping T_P^S on the set of S-Herbrand interpretations, associated with P, is defined as

$$T_P^S(I_P^S) = \{A' \in B_P^S \mid \exists A \leftarrow B_1, \ldots, B_n \text{ in } P, \exists B_1', \ldots, B_n' \in I_P^S, \exists \theta = mgu((B_1', \ldots, B_n'), (B_1, \ldots, B_n)), \text{ and } A' = A\theta\}.$$

The following theorem can now be included from [15]. It provides the desired least fixpoint characterization of H_P^S.

Theorem 7.2. For every definite program P,

$$H_P^S = lfp(T_P^S) = \bigcup_{n \in \omega} T_P^{S^n}(\emptyset) \left(= T_P^S \uparrow \omega \right).$$

Finally, it can be pointed out that the least S-Herbrand model of a program exactly characterizes computed answer substitutions for completely uninstantiated

queries with respect to this program. We refer to [15] and/or [16] for a full formal development with soundness and completeness results.

We conclude this brief introduction to S-semantics with a few simple examples to illustrate the concept of least S-Herbrand model.

Example 7.1.

$$\begin{aligned}
P_1 &: \quad p(a) \leftarrow \\
H^S_{P_1} &= \{p(a)\}, \\
P_2 &: \quad p(x) \leftarrow \\
H^S_{P_2} &= \{p(x)\}, \\
P_3 &: \quad p(x) \leftarrow \\
&\qquad p(a) \leftarrow, \\
H^S_{P_3} &= \{p(x), p(a)\}, \\
P_4 &: \quad p(x) \leftarrow q(x) \\
&\qquad q(a) \leftarrow, \\
H^S_{P_4} &= \{p(a), q(a)\}, \\
P_5 &: \quad p(x, y) \leftarrow q(x) \\
&\qquad q(a) \leftarrow, \\
H^S_{P_5} &= \{p(a, x), q(a)\}.
\end{aligned}$$

Notice that * (see Section 2) does *not* show up in $H_{P_2^S}$. Indeed, Definition 7.6 and Theorem 7.2 show that atoms in the least S-Herbrand model of a program without contants do not contain any constants either. In particular, the special constant *, added to the underlying language, plays no role in least S-Herbrand semantics although it does of course occur in the S-Herbrand universe of such programs.

7.3. Vanilla Metainterpreters

As pointed out above, this section is about definite programs. For clarity and completeness, we include the definition of a definite program's vanilla metaprogram below. We leave out the "d", used in similar circumstances in Section 5.1, since in the context of the present section, no confusion with normal metaprograms is possible. Finally, we remind the reader of the fact that we use functorizations which are identical to their associated sets of predicate symbols.

Definition 7.7. The following definite program M is called *vanilla metainterpreter*:

$$\begin{aligned}
&solve(empty) \leftarrow \\
&solve(x\&y) \leftarrow solve(x), solve(y) \\
&solve(x) \leftarrow clause(x, y), solve(y).
\end{aligned}$$

Definition 7.8. Let P be a definite program. Then M_P, the *vanilla metaprogram associated with* P, is the definite program consisting of M together with a fact

of the form

$$clause(A, B_1 \& \cdots \& B_n) \leftarrow$$

for every clause $A \leftarrow B_1, \cdots, B_n$ in P and a fact of the form

$$clause(A, empty) \leftarrow$$

for every fact $A \leftarrow$ in P.

We have:

Proposition 7.1. Let P be a definite program with $C_P \neq \emptyset$ and M_P its vanilla metaprogram. Then $U_P^S \subset U_{M_P}^S$.

PROOF. Obvious from the definitions. □

Just like before, $*$ precludes a generalization of Proposition 7.1 to all definite object programs. However, our observation above guarantees the absence of problems with $*$ when considering the least S-Herbrand model of programs where $C_P = \emptyset$ and their metaprograms.

We are now in a position to start proving the main result of this section. We set out with the following two lemmas.

Lemma 7.1. Let P be a definite program and let M_P be its vanilla metaprogram. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\begin{aligned} \forall \bar{t} \in U_P^{S^r}, \forall n \in \mathbb{N}: \quad & p(\bar{t}) \in T_P^S \uparrow n \\ \Rightarrow \quad & \exists m \in \mathbb{N}: \quad \text{solve}(p(\bar{t})) \in T_{M_P}^S \uparrow m. \end{aligned}$$

PROOF. The proof is through induction on n. The base case ($n = 0; T_P^S \uparrow 0 = \emptyset$) is trivially satisfied. Now suppose that $p(\bar{t}) \in T_P^S \uparrow n, n > 0$. Then there must be at least one clause $A \leftarrow B_1, \ldots, B_k$ ($k \geq 0$) in P such that $\exists C_1, \ldots, C_k \in T_P^S \uparrow (n-1), \exists \theta = mgu((C_1, \ldots, C_k), (B_1, \ldots, B_k))$, and $p(\bar{t}) = A\theta$. Consider first the case that we have one with $k = 0$. In other words, $p(\bar{t}) \leftarrow$ is a fact in P. In that case, Definition 7.8 immediately implies that $solve(p(\bar{t})) \in T_{M_P}^S \uparrow 2$.

Suppose now $k \geq 1$. The induction hypothesis guarantees for every C_i the existence of an $m_i \in \mathbb{N}$ such that $solve(C_i) \in T_{M_P}^S \uparrow m_i$. Let mm denote the maximum of these m_i. This means that $\forall 1 \leq i \leq k : solve(C_i) \in T_{M_P}^S \uparrow mm$. In particular, $solve(C_k) \in T_{M_P}^S \uparrow mm$. Moreover, for any $1 \leq l < k$,

$$solve(C_{l+1} \& \cdots \& C_k) \in T_{M_P}^S \uparrow (mm + k - l - 1)$$

implies

$$solve(C_l < C_{l+1} \& \cdots \& C_k) \in T_{M_P}^S \uparrow (mm + k - l).$$

It follows (induction on l) that

$$\forall 1 \leq l \leq k: \quad solve(C_l \& \cdots \& C_k) \in T_{M_P}^S \uparrow (mm + k - l).$$

In particular,

$$\boldsymbol{solve(C_1 \& \cdots \& C_k) \in T_{M_P}^S \uparrow (mm + k - 1).}$$

Since we also know that

- $clause(A, B_1 \& \cdots \& B_k) \in T^S_{M_{P'}} \uparrow 1$,
- $\theta = mgu((C_1, \cdots, C_k), (B_1, \cdots, B_k))$,
- $p(\ \bar{t}\ ) = A\theta$,

it follows that $solve(p(\ \bar{t}\ )) \in T^S_{M_{P'}} \uparrow (mm + k)$. □

Lemma 7.2. Let P be a definite program and let M_P be its vanilla metaprogram. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U^{S''}_{M_{P'}}, \forall n \in \mathbb{N} : \ solve(p(\ \bar{t}\ )) \in T^S_{M_{P'}} \uparrow n$$
$$\Rightarrow \quad \bar{t} \in U^{S'}_P \& \exists m \in \mathbb{N} : \ p(\ \bar{t}\ ) \in T^S_P \uparrow m.$$

PROOF. The proof proceeds through an induction on n. The base case ($n = 0; T^S_{M_{P'}} \uparrow 0 = \emptyset$) is trivially satisfied. Suppose that $solve(p(\ \bar{t}\ )) \in T^S_{M_{P'}} \uparrow n$, $n > 0$. Then either there is a fact $clause(p(\ \bar{t}\ ), empty) \leftarrow$ in M_P or this is not the case. Suppose first there is. Then there must be a fact $p(\ \bar{t}\ ) \leftarrow$ in P and the result follows. If there is no such *clause* fact in M_P, then the following must be true:

- $\exists\ clause(A, B_1 \& \cdots \& B_k)(k > 0) \in T^S_{M_{P'}} \uparrow (n-1)$,
- $\exists\ solve(C_1 \& \cdots \& C_k) \in T^S_{M_{P'}} \uparrow (n-1)$,
- $\exists\ \theta = mgu\ ((clause(A, B_1 \& \cdots \&\ B_k),\ solve\ (C_1 \& \cdots \& C_k)),\ (clause\ (x, y),\ solve(y)))$,

  and $solve(p(\ \bar{t}\ )) = solve(x)\theta$.

From the last point, it follows that if $\sigma = mgu\ ((B_1 \& \cdots \& B_k),\ (C_1 \& \cdots \& C_k))$, then $p(\ \bar{t}\ ) = A\sigma$. Furthermore, the first point implies the presence of a clause $A \leftarrow B_1, \ldots, B_k$ in P and the second can only be true if

$$\forall 1 \leq i \leq k, \exists n_i < n \in \mathbb{N} : \quad solve(C_i) \in T^S_{M_{P'}} \uparrow n_i.$$

Through the induction hypothesis, we obtain

$$\forall 1 \leq i \leq k, \exists m_i \in \mathbb{N} : \quad C_i \in T^S_P \uparrow m_i \ \&\ \overline{t_i} \in U^{S'}_P,$$

where t_i is the tuple of arguments appearing in the atom C_i and r_i is the arity of its predicate. The desired result now follows. □

Our main result on the S-semantics of vanilla metainterpreters is expressed by the following theorem.

Theorem 7.3. Let P be a definite program and let M_P be its vanilla metaprogram. Let H^S_P and $H^S_{M_{P'}}$ denote the least S-Herbrand model of P and M_P, respectively. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U^{S''}_{M_{P'}} : \quad solve(p(\ \bar{t}\ )) \in H^S_{M_{P'}} \Leftrightarrow \bar{t} \in U^{S''}_P \ \&\ p(\ \bar{t}\ ) \in H^S_P.$$

PROOF. The theorem follows immediately from Lemmas 7.1 and 7.2. □

When we compare this theorem with Theorem 4.2, we first of all notice that it is restricted to *definite* object and metaprograms. However, we conjecture that this limitation follows from the current state of S-semantics and is not inherent to metaprogramming. We briefly return to this issue at the end of this section. More important is the *absence of the language independence condition.* Indeed, Theorem 7.3 shows that there is a sensible correspondence between the S-semantics of *any* definite object program and its vanilla metaprogram. Therefore, this theorem can be regarded as a formal confirmation that this kind of programming gives on *practical* problems, not even for programs that are *not* language independent. Notice, by the way, that Theorem 7.3 also generalizes Proposition 4.2. Indeed, as indicated above, S-semantics allows a more elegant treatment of the $C_P = \emptyset$ case than classical ground Herbrand semantics.

7.4. Extended Metainterpreters

Having established Theorem 7.3, the next question that comes to mind is: Can we similarly generalize Proposition 5.1? Contrary to our initial expectations, this question has to be answered negatively.

The definitions of definite extended metainterpreters and metaprograms were already given in Section 5.1. We will not repeat them here and simply refer to Definitions 5.1 and 5.2. Now, consider the following example.

Example 7.2.

P : $p(x) \leftarrow$,
$H_P^S = \{p(x)\}$,
E_P: $solve(empty) \leftarrow$
$solve(x\&y) \leftarrow solve(x), solve(y)$
$solve(x) \leftarrow clause(x, y), solve(y), inst(x)$
$clause(p(x), empty) \leftarrow$
$inst(p(a)) \leftarrow$.

It is easy to see that $solve(p(a)) \in H_{E_P}^S$.

The source of the problem is clearly the fact that answers might become further instantiated than is the case in the object program. Since the least S-Herbrand model represents the most general answer substitutions, this means that a straightforward generalization of Proposition 5.1 is impossible.

However, we *can* prove a more modest variant of Proposition 5.1.

We first prove the following:

Proposition 7.2. Let P be a definite program. Let H_P denote its least Herbrand and H_P^S its least S-Herbrand model, respectively. Then P is language independent iff $H_P = H_P^S$.

PROOF. The proposition follows from Definition 7.6, Theorem 7.2, and Proposition 2.2 via a straightforward induction proof. □

Notice that $H_P = H_P^S$ implies that H_P^S contains only ground atoms. It follows that:

Lemma 7.3. Let P be a definite, language independent program and M_P its associated vanilla metaprogram. Let $H^S_{M_{P'}}$ denote the least S-Herbrand model of M_P. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U^{S^r}_{M_{P'}} : \quad solve(p(\ \bar{t}\ )) \in H^S_{M_{P'}} \quad \Rightarrow \quad \bar{t} \text{ is ground.}$$

PROOF. The lemma follows from Theorem 7.3 and Proposition 7.2 □

We need a second lemma:

Lemma 7.4. Let P be a definite program, M_P the vanilla, and E_P an exended metaprogram associated with P. Then the following holds:

$$\forall t \in U^S_{E_{P'}} : \quad (\exists \bar{s} \in U^{S^n}_{E_{P'}} : \quad solve(t, \bar{s}) \in H^S_{E_{P'}})$$
$$\Rightarrow \quad \exists \theta, \exists solve(t') \in H^S_{M_{P'}} : \ t = t'\theta.$$

PROOF. Obvious from the definitions. □

This allows us to prove the following variant of Proposition 5.1:

Proposition 7.3. Let P be a definite, language independent program and E_P an E-extended metaprogram associated with P. Let H^S_P and $H^S_{E_{P'}}$ denote the least S-Herbrand model of P and E_P, respectively. Then the following holds for every $p/r \in \mathcal{R}_P$:

$$\forall \bar{t} \in U^{S^r}_{E_P} : \quad (\exists \bar{s} U^{S^n}_{E_{P'}} : \quad solve(p(\ \bar{t}\ ), \bar{s}) \in H^S_{E_{P'}})$$
$$\Rightarrow \quad \bar{t} \in U^{S^r}_P \ \&\ p(\ \bar{t}\ ) \in H^S_P.$$

PROOF. From Lemmas 7.3 and 7.4, we obtain that $solve(p(\ \bar{t}\ ), \bar{s}) \in H^S_{E_{P'}}$ implies $solve(p(\ \bar{t}\ )) \in H^S_{M_{P'}}$ (and therefore $\bar{t} \in U^{S^r}_{M_{P'}}$). The result now follows from theorem 7.3. □

Example 7.2 shows that the S-semantics results for vanilla metaprograms can not immediately be carried over to extended metaprograms, and, indeed, in practice, logic programming with extended metaprograms *can* generate unwanted answer substitutions. Proposition 7.3 shows that this is not the case for language independent object programs. It is of course possible to investigate conditions on the metaprogram which would ensure that Proposition 7.3 holds for any definite object program. It is not even very difficult to conjecture some such conditions. (See the comment concluding Section 5.1.) However, we do not wish to pursue this topic in the present paper.

7.5. Concluding Remarks

A treatment of amalgamated metaprograms and meta2 programs in the context of S-semantics is straightforward and not particularly enlightening. All the results

from Section 6 can be generalized in the expected way. We will neither state nor prove them explicitly.

Concerning the limitation to definite programs, we can remark that [44] extends [15] and [16], drawing from work on constructive negation, and in this way perhaps provides a setting for addressing normal object and metaprograms.

Finally, it should be noted that [28] independently extended the result for vanilla metaprograms of language independent programs in [10] to all definite object programs and their vanilla metaprograms in the context of S-semantics. Moreover, a sizeable metaprogramming application is sketched and its semantics discussed in the proposed framework.

8. DISCUSSION AND SOME CLOSELY RELATED WORK

This section contains some further comments on our approach to metasemantics and its results, mainly through a discussion of some related work. Since there is a vast literature on metalogic, its semantics, possible applications, advantages, and disadvantages, we do not strive for completeness. Instead, we only consider some closely related papers within logic programming.

Let us first consider [23]. In the first part of that paper, it is shown that through the use of appropriate *typing*, vanilla metaprogams can be given a suitable declarative (the well-known Clark's completion semantics is used) and procedural semantics. Moreover, in a second part, a ground representation for object level terms at the metalevel is considered, and it is shown how a number of problematic Prolog built-ins (static and of the type called "first order," i.e., not referring to clauses or goals, in [3]) can be given a declarative meaning in this setting.

Addressing the latter topic first, it should be noted that such Prolog metapredicates, of which *var*/1 and *nonvar*/1 are prototypical examples, are not included in our language. This certainly puts some limitation on the obtained expressiveness. Observe, however, that in the typed nonground representation proposed in [23], no alternative for *var*/1 was inroduced either and that the declarative *var*/1 predicate introduced in the ground representation approach provides no direct support for the sort of functionalities (e.g., control and coroutining facilities) that the *var*/1 predicate in Prolog is typically used for. Recently, [3] proposed a declarative semantics for such predicates. We conjecture that if one so desires, our basic methodology can be adapted to the semantics described there, thus enabling the inclusion of such built-ins in our language. A (perhaps superior) alternative for providing the latter kind of facilities are *delay* control annotations as in Gödel.

For the *assert*/1 and *retract*/1 Prolog built-in predicates, the solution of [22], to represent dynamic theories as terms in the metaprogram, can as well be applied in our approach. However, this requires special care with variable bindings, and leads to some inconveniences in the context of a ground Herbrand model approach to semantics. A thorough discussion of the problems related to these predicates is given in [22]. They are also treated in [41], which is discussed below. As a final remark about Prolog built-ins, we would like to mention that our use of overloading largely eliminates the need for a *call*/1 predicate, as Example 6.1 illustrates.

Next, observe that the condition of range restriction, which is the practical, verifiable, sufficient condition for language independence our approach was mostly designed for, is strongly related to typing. Indeed, typing can, in principle, be

converted into additional atoms that are added in the bodies of clauses, expressing the range of each variable. See, e.g., [14] and [29]. In this context, it is interesting to note the apparent duality between range restriction at the object level and typing at the metalevel. If one "hardwires" types into the code of the object program through range restriction, typing (or range restriction) at the metalevel is no longer required for a sensible declarative semantics. [18] can also be mentioned here. It presents a program transformation technique that enables us to minimize run-time type checking is systems which represent types as (unary) predicates, thus largely eliminating one of the main advantages types might offer in comparison with ranges.

Finally, [23] does not address amalgamation, and Gödel (see [21]), the programming language whose extensive metaprogramming facilities are largely based upon the foundations laid out in [23] and [22], does not allow it. While dealing with amalgamated programs of the kind addressed in Section 6.2 seems relatively straightforward, a generalization of the typed approach to meta2 programs is probably not immediate.

With respect to the extension to amalgamated programs, we should point out that our use of overloading is strongly related to the logic proposed in [37]. Indeed, in his analysis of the problems connected with reference and modality, Richards considers logical languages that contain their *well-formed formulas as terms*. He interprets these languages on specially devised models whose domains are a union of *the constants and the sentences* (i.e., closed formulas) in the language.

Kalsbeek [25] recently proposed a variant of Richards' logic as a suitable basis for studying the semantics of metaprograms. She allows *any formula to be a term* in the language and uses (Herbrand) interpretations with *arbitrary closed terms* in the domain. In the logic framework thus obtained, soundness and completeness of definite vanilla metaprograms with respect to their definite object program are proved, restricted to terms in the *object* level language.

Jiang [24] proposes an even more ambivalent language, also allowing *terms as formulas*. He shows that a number of interesting properties (Herbrand theorem, completeness), lost in Richards' logic, are recovered. Vanilla metaprograms are considered, leading to similar results as obtained by others. The framework is, however, more powerful and particularly suitable for addressing quantified object level statements and full amalgamation. To this end, a sophisticated treatment of variables, not syntactically distinguishing between variables and their names, is proposed. This allows to consider metatheories dealing with full first order object statements. However, at the time of writing, this work is not yet in a finalized state. Particularly the treatment of variables seems to require further study.

In summary, our technique of overloading function and predicate symbols is probably less powerful than approaches using more fully ambivalent syntax, but it requires no modification of the familiar notion of Herbrand interpretations and models.

A comparison with the work in [9] basically leads to the same conclusion. The semantic techniques proposed there provide a first order semantics for a class of programming constructs that includes the kind of amalgamation we consider, but significantly surpasses it. Clauses like $solve(x) \leftarrow x$ are allowed, as well as higher order functions, generic predicate definitions, etc. The basic characteristic of HiLog logic is the fact that "terms may represent individuals, functions, predicates, and atomic formulas in different contexts." However, the semantics required to support this is a less immediate extension of the common first order logic semantics.

Finally, for *definite* program, a powerful framework is proposed in [41]. It provides quantification of object level formulas in metalevel statements, as well as a fully developed naming mechanism. However, the price to be paid for this is a nontrivial modification of the standard Herbrand semantics for logic programs. How this approach generalizes to programs *with negation* is also not immediately clear.

We conclude that *our approach is a good compromise between complicating semantics and enhancing programming power.* If one considerably wants to extend the latter, e.g., by allowing full quantification in named object-level statements, a more complex semantics is probably inevitable.

9. CONCLUSION

In this paper, we have considered untyped nonground metaprograms. We have studied rather extensively the semantic properties of vanilla metainterpreters of this kind, and we have looked at interesting extensions and variants involving (a limited form of) amalgamation. It turned out that in most of these cases, least or weakly perfect Herbrand semantics is well-behaved for definite (respectively stratified) *language independent* object programs (and if not, then at least for *range restricted* ones; see Proposition 5.3). This is an interesting result since these semantics are widely accepted as good declarative semantics for logic programs. Moreover, we believe that our methodology can often also be applied when considering untyped, nonground metaprograms that do not immediately fall within one of the categories we explicitly considered (see [31] for further comments and references). We have also shown how, for nonextended metaprograms, the restriction of *language independence* can be lifted in the context of a declarative semantics that more closely reflects the procedural behavior of logic programs. These results explain why the language independence condition almost never surfaces in logic programming practice.

Along the way, we have, moreover, shown that vanilla metaprograms, as well as their variants, associated with stratified object programs are *weakly* stratified. We have conjectured that this result can be generalized to *weakly* stratified *object* programs. Our work, therefore, seems to provide evidence in favor of the view that *weak* stratification is a much more natural and useful generalization of stratification than *local* stratification appears to be. In that context, Lemma 3.1 and its proof in the Appendix are interesting. Indeed, we conjecture that they provide the essential ingredients for a completely general proof showing that a weakly stratified program possesses a unique weakly perfect model, independent of the particular (successful) "weak stratification" used to obtain it. In other words, Definition 3.7 can be reformulated to allow the use of nonmaximal layers as in Proposition 3.2. An important issue from a practical point of view, and one that was, to the best of our knowledge, not addressed before.

It seems to us that almost all realistic logic programs are at least weakly stratified. Nevertheless, it can be a topic of further research to investigate whether our results can be generalized in the context of a semantics that is able to deal with *any* logic program. Well-founded semantics (see, e.g., [46]) is one such approach which recently has gained popularity. It attaches to any logic program a (possibly partial, three-valued) unique *well-founded* Herbrand model. If we want to recast our results in this setting, the first step is of course a reformulation of the *lan-*

guage independence notion. This provides no fundamental difficulties, but some care has to be taken since for most sensible programs, the negative information implied (now explicitly incorporated in the well-founded model) *does* depend on the language. So, one should only require stability of the positive information. Having established this, a generalization of our results can be attempted. However, well-founded semantics remains just one of various formalisms proposed as a semantics for arbitrary normal logic programs. It might, therefore, be more worthwhile to study metaprogram semantics in the context of a generic unifying semantic framework such as provided in, e.g., [12]. By restricting ourselves to a class of programs the semantics of which is the subject of little or no controversy, we have, in the present work, taken a dual approach.

Other possible topics for further research include the following:

- A characterization of extended metaprograms that would allow more powerful variants of Propositions 5.1, 5.3, and/or 7.3. The classification of enhancements of the vanilla metainterpreter presented in [39] can perhaps be a starting point here.
- An extension of Section 7 to normal programs.
- A reformulation of our results in the context of a declarative semantics for Prolog built-ins, as proposed in [3].
- A study of object theories beyond the scope of normal logic programs, and their associated metaprograms.

APPENDIX: PROOF OF LEMMA 3.1

Throughout this Appendix, P will denote some normal program. Observe that, for any normal program P, $Ground(P)$ is a (possibly infinite, but countable) set of ground clauses. It is then possible and convenient to associate a particular natural number with each clause in $Ground(P)$. So, we will occasionally refer to a clause in $Ground(P)$ as being a couple (C, n_C), where no two clauses in $Ground(P)$ have the same associated n_C. In this way, we can in the construction in Proposition 3.2, always identify the unique clause in $Ground(P)$ from which a clause in a given P_i is derived (through the deletion of true body literals): they have identical labels. In the same way, correspondences between clauses in P_i sets in two different such constructions can be established. Finally, when we refer to some series $V_1, \ldots$ below, we always mean a series of sets chosen to serve as V_i sets in the context of the construction in Proposition 3.2. We will occasionally use the notation $\mathcal{V}$ (or other characters in a similar way) to denote such a series, and we will use superscripts to distinguish P_i, L_i, H_i, and H_P sets in different constructions. Observe that for any series $\mathcal{V}, P_1^{\mathcal{V}} = Ground(P)$, and also $B_{P_1^{\mathcal{V}}} = B_{Ground(P)} = B_P$.

We start with a fairly basic lemma:

Lemma A.1. *Let $\mathcal{V} = V_1, \ldots$ be a series such that $P_i^{\mathcal{V}}$ is defined. Let $N_i^{\mathcal{V}} = \{n_C \in \mathbb{N} \mid$ there exists a clause (C, n_C) in $P_i^{\mathcal{V}}\}$. Then:*

- *$N_1^{\mathcal{V}}, \ldots$ is monotonically decreasing.*
- *$B_{P_1^{\mathcal{V}}}, \ldots$ is monotonically decreasing.*

PROOF. In the construction in Proposition 3.2, each P^V_{i+1} is obtained from P^V_i by:

1. Deleting some clauses.
2. Removing some literals from the remaining clauses.

Both statements above trivially follow. □

Lemma A.1 enables the following definition:

Definition A.1. Let $\mathcal{V} = V_1, \ldots$ be a series such that P^V_i is defined. Then for each atom $A \in B_P \backslash B_{P^V_i}$, we define the $\mathcal{V}$-layer of $A, l_\mathcal{V}(A)$, to be the smallest $j\,(1 \leq j < i)$ such that $A \notin B_{P^V_{j+1}}$.

Notice that $l_\mathcal{V}(A)$ is well defined in the sense that it does not depend on i. In particular, if $\mathcal{V}$ terminates successfully, then each $A \in B_P$ has a unique associated layer $l_\mathcal{V}(A)$.

The layer concept is interesting, since we can easily show:

Lemma A.2. Let $A \in B_P$ such that $i = l_\mathcal{V}(A)$. Then for any $j \neq i$ such that L^V_j is defined, there are no clauses with head A in L^V_j.

PROOF. For $j < i, A \in B_{P^V_{j+1}}$. Therefore, $A \notin V_j$, from which the result follows. On the other hand, $A \notin B_{P^V_{i+1}}$, so that the desired result for $j > i$ follows from Lemma A.1. □

So, when a atom is "consumed" by a series, the unique definite program corresponding to its layer is the only one that possibly contains clauses defining A. It is, therefore, this program which decides whether A is in the resulting model (if any).

Another useful property of the layer concept is the following:

Lemma A.3. Let $A \in B_P$ such that $l_\mathcal{V}(A) = i$. For any $B \in B_{P^V_i}$, such that $B \leq A$, we have $l_\mathcal{V}(B) = i$.

PROOF. By the construction in Proposition 3.2, $B \in V_i$. Thus, $B \notin B_{P^V_{i+1}}$ (which exists since $l_\mathcal{V}(A) = i$ exists). Obviously, $B \in B_{P^V_i}$, so that the result follows. □

We can now prove a quite powerful result, basically establishing equality of truth according to two different series for the same P.

Lemma A.4. Let $\mathcal{V}$ and $\mathcal{W}$ be two series such that P^V_i and P^W_j are defined. Let A be an atom in $B_P \backslash (B_{P^V_i} \cup B_{P^W_j})$. Then:

$$A \in \bigcup_{m<i} H^V_m \;\Leftrightarrow\; A \in \bigcup_{m<j} H^W_m.$$

PROOF. Notice that the result trivially holds if either i or j equals 1, since in that case $B_P \backslash (B_{P^V_i} \cup B_{P^W_j}) = \emptyset$. Suppose, therefore, that they are both bigger than 1.

It follows from Lemma A.2 that for any $A \in B_P \backslash (B_{P_i^V} \cup B_{P_j^W})$,

$$A \in \bigcup_{m<i} H_m^V \quad \Leftrightarrow \quad A \in H_{l_\mathcal{V}(A)}^V$$

and

$$A \in \bigcup_{m<j} H_m^W \quad \Leftrightarrow \quad A \in H_{l_\mathcal{W}(A)}^W.$$

So, it is sufficient to show that:

$$A \in H_{l_\mathcal{V}(A)}^V \quad \Leftrightarrow \quad A \in H_{l_\mathcal{W}(A)}^W.$$

The proof proceeds through *induction on $l_\mathcal{V}(A)$ and $l_\mathcal{W}(A)$*.

We first comment on the *structure of the induction proof*, which is nonstandard. Let us denote by $Equi(k,l)$ the formula

$$\begin{aligned} &\forall A \in B_P \backslash \big(B_{P_i^V} \cup B_{P_j^W}\big) : \\ &\quad (l_\mathcal{V}(A) \leq k \wedge l_\mathcal{W}(A) \leq l) \quad \Rightarrow \quad \big(A \in H_{l_\mathcal{V}(A)}^V \quad \Leftrightarrow \quad A \in H_{l_\mathcal{W}(A)}^W\big). \end{aligned}$$

Notice that, since for any $A \in B_P \backslash (B_{P_i^V} \cup B_{P_j^W})$, $(l_\mathcal{V}(A) \leq i-1 \wedge l_\mathcal{W}(A) \leq j-1)$ trivially holds, we need to prove $Equi(i-1, j-1)$. In order to achieve this, we will show:

1. $Equi(1,l)$, for all $l \leq j-1$.
2. $Equi(k-1,l) \wedge Equi(k,l-1) \Rightarrow Equi(k,l)$, for all $k \leq i-1$ and $l \leq j-1$.

From 1, due to symmetry, $Equi(k,1)$, for all $k \leq i-1$, follows. Then, by repeatedly applying 2, $Equi(i-1, j-1)$ is implied by a finite conjunction of formulas of the type $Equi(1,l)$, $l \leq j-1$, and $Equi(k,1)$, $k \leq i-1$, which completes the proof.

The *proof of* 1 is by induction on l. For the base case, $l = 1, A \in H_1^V \Leftrightarrow A \in H_1^W$ follows immediately, because the subprograms of L_1^V and L_1^W on which A depends are identical. This is due to the construction in Proposition 3.2, which ensures that:

- L_1^V and L_1^W contain *all* clauses of $Ground(P)$ with head A.
- V_1 and W_1 contain all atoms B with $B \leq A$.
- L_1^V and L_1^W also contain *all* clauses of $Ground(P)$ with such B's as their head.

Next, assume that $Equi(1, l-1)$ holds, with $l \leq j-1$. We prove $Equi(1,l)$. The increment of $Equi(1,l)$ over $Equi(1,l-1)$ is

$$\begin{aligned} \forall A \in B_P \backslash (B_{P_i^V} \cup B_{P_j^W}) : \quad &(l_\mathcal{V}(A) = 1 \wedge l_\mathcal{W}(A) = l) \\ &\Rightarrow \quad (A \in H_1^V \Leftrightarrow A \in H_l^W). \end{aligned}$$

So, let A be an atom with $l_\mathcal{V}(A) = 1$ and $l_\mathcal{W}(A) = l$. From the way clauses are transformed and/or deleted in the construction procedure, we obtain the following correspondence between clauses with head A in L_1^V and L_l^W:

1. **Every such clause $(C,n) \in L_1^V$ such that there is no clause $(C^W, n) \in L_l^W$, has a body with at least one atom in $\cup_{m<1} W_m$, but not in $\cup_{m<l} H_m^W$.**

2. There is a one-to-one correspondence between other clauses $(A \leftarrow B^V, n) \in L_1^V$ and $(A \leftarrow B^W, n) \in L_l^W$, where B^V is identical to B^W except for the possible occurrence of atoms in B^V that are not in B^W but in $\cup_{m<l} W_m$ and in $\cup_{m<l} H_m^W$.

We show by induction that

$$\forall A \in B_P \backslash (B_{P_i^V} \cup B_{P_j^W}): \quad (l_{\mathcal{V}}(A) = 1 \wedge l_{\mathcal{W}}(A) = l)$$
$$\Rightarrow \quad (\exists m: A \in T_{L_1^V} \uparrow m \quad \Leftrightarrow \quad \exists m': A \in T_{L_l^W} \uparrow m').$$

First, if $A \in T_{L_1^V} \uparrow 1$, then L_1^V contains a fact $(A \leftarrow, n)$. Only 2 is applicable and, since the L_1^V clause can only have *more* body atoms than the corresponding L_l^W clause, $(A \leftarrow, n) \in L_l^W$, so that $A \in T_{L_1^W} \uparrow 1$. Conversely, if $A \in T_{L_1^W} \uparrow 1$, then $(A \leftarrow, n) \in L_l^W$. Again, only 2 is applicable. Thus, L_1^V contains a clause $(A \leftarrow B^V, n)$, where all atoms in B^V are in $\cup_{m<l} W_m$ and in $\cup_{m<l} H_m^W$. Let B be such an atom. Since $L_{\mathcal{V}}(B) = 1$ and $L_{\mathcal{W}}(B) < l$, we can apply the induction hypothesis $Equi(1, l-1)$. It states that $B \in H_1^V \Leftrightarrow B \in H_{l_{\mathcal{W}}(B)}^W$. However, since $B \in \cup_{m<l} H_m^W$, the right-hand side holds, so that $B \in H_1^V$. Therefore, there exists an m_B such that $B \in T_{L_1^V} \uparrow m_B$. Taking $m_A = \max(\{m_B \mid B \text{ in } B^V\}) + 1$, we get $A \in T_{L_1^V} \uparrow m_A$.

For the induction step and the left-to-right implication, let $A \in T_{L_1^V} \uparrow m$ and assume that this implication holds for all $B \in T_{L_1^V} \uparrow (m-1)$. So, there exists a clause $(A \leftarrow B^V, n) \in L_1^V$, such that $B \in T_{L_1^V} \uparrow (m-1)$, for each $B \in B^V$.

First, we prove that $(A \leftarrow B^V, n)$ must be of type 2. Assume that it is of type 1. Then there exists a B in B^V such that $B \in \cup_{m<l} W_m \backslash \cup_{m<l} H_m^W$. Clearly, $l_{\mathcal{V}}(B) = 1$ and $l_{\mathcal{W}}(B) < l$, so that the induction hypothesis $Equi(1, l-1)$ applies. So, $B \in H_1^V \Leftrightarrow B \in H_{l_{\mathcal{W}}(B)}^W$. However, since $B \in H_{l_{\mathcal{W}}(B)}^W \Leftrightarrow B \in \cup_{m<l} H_m^W$ and $B \notin \cup_{m<l} H_m^W$, we obtain $B \notin H_1^V$. This contradicts $B \in T_{L_l^V} \uparrow (m-1)$. So, $(A \leftarrow B^V, n)$ is of type 2 and there exists a clause $(A \leftarrow B^W, n)$ in L_l^W containing a subset of the body atoms in B^V. For each such atom, B in B^W, $B \in T_{L_1^V} \uparrow (m-1)$ holds. By the induction hypothesis, $B \in T_{L_l^W} \uparrow m_B$ for some m_B. Taking $m_A = \max(\{m_B \mid B \text{ in } B^W\}) + 1$, we get $A \in T_{L_l^W} \uparrow m_A$.

Finally, for the induction step on the right-to-left implication, let $A \in T_{L_l^W} \uparrow m$ and assume that the implication holds for all $B \in T_{L_l^W} \uparrow (m-1)$. Again, there exists a clause $(A \leftarrow B^W, n)$ in L_l^W, such that $B \in T_{L_l^W} \uparrow (m-1)$, for all B in B^W. Only case 2 can apply. Thus there exists a clause $(A \leftarrow B^V, n)$ in L_1^V, identical to $(A \leftarrow B^W, n)$, except that it may contain some additional body atoms B, where $B \in \cup_{m<l} W_m \cap \cup_{m<l} H_m^W$. For the body atoms B which are both in B^V and B^W, we can apply the right-to-left induction hypothesis: $B \in T_{L_1^V} \uparrow m_B$, for some m_B. For the atoms B not in B^W, we again have $l_{\mathcal{V}}(B) = 1, l_{\mathcal{W}}(B) < l$, and $B \in H_{l_{\mathcal{W}}(B)}^W$. So, we apply $Equi(1, l-1)$, obtaining $B \in H_1^V$, which means that $B \in T_{L_1^V} \uparrow m_B$, for some m_B. Again, $m_A = \max(\{m_B \mid B \text{ in } B^V\}) + 1$ allows us to conclude the proof.

Next, *we prove the implication*

$$\forall k < i, \forall l < j: \quad Equi(k-1, l) \wedge Equi(k, l-1) \quad \Rightarrow \quad Equi(k, l).$$

The proof is completely similar to the previous step, except that more different correspondence cases between clauses in L_k^V and L_l^W can be distinguished. Let A be an atom with $l_{\mathcal{V}}(A) = k$ and $l_{\mathcal{W}}(A) = l$. From the way clauses are transformed and/or deleted in the construction in Proposition 3.2, we can now distinguish the following possible cases for clauses in L_k^V or in L_l^W, having A as their head:

1. Every such clause $(C^V, n) \in L_k^V$ such that there is no clause $(C^W, n) \in L_l^W$, contains at least one body atom B, such that B is in $\cup_{m<l} W_m$, but not in $\cup_{m<l} H_m^W$.
   Notice that this is less trivial than the corresponding statement for L_1^V and L_l^W above. Although it is clear that the clause (C, n) in $Ground(P)$ contains at least one such body atom B, in (C^V, n) this body atom could have been removed. However, since $Equi(k, l-1)$ is given and since $B \notin \cup_{m<l} H_m^W$, we have $B \notin \cup_{m<k} H_m^V$. Thus B has not been removed from (C^V, n). Furthermore, $Equi(k, l-1)$ also implies that this atom B is not in H_k^V.
2. Every such clause $(C^W, n) \in L_l^W$ such that there is no clause $(C^V, n) \in L_k^V$, contains at least one atom B such that B is in $\cup_{m<k} V_m$, but not in $\cup_{m<k} H_m^V$. (Here, we have used $Equi(k-1, l)$.) Now, $Equi(k-1, l)$ also implies that $B \notin H_l^W$.
3. There is a one-to-one correspondence between other clauses $(A \leftarrow B^V, n) \in L_k^V$ and $(A \leftarrow B^W, n) \in L_l^W$, where B^V is identical to B^W except that:
   (a) B^V might contian atoms not occurring in B^W. All such atoms are in $\cup_{m<l} H_m^W$, but then also in H_k^V since $Equi(k, l-1)$ holds.
   (b) B^W might contain atoms not occurring in B^V. All such atoms are in $\cup_{m<k} H_m^V$, but then also in H_l^W since $Equi(k-1, l)$ holds.

The proof again proceeds through induction, proving the statement

$$\begin{aligned} &\forall A \in B_P \backslash (B_{P_i^V} \cup B_{P_j^W}) : \\ &\quad (l_{\mathcal{V}}(A) = k \wedge l_{\mathcal{W}}(A) = l) \quad \Rightarrow \quad (\exists m : A \in T_{L_k^V} \uparrow m \\ &\qquad\qquad\qquad\qquad\qquad\qquad \Leftrightarrow \quad \exists m' : A \in T_{L_l^W} \uparrow m'). \end{aligned}$$

It is very similar to the proof of the induction step for $Equi(1, l)$. The main differences are:

1. Due to symmetry, we only need to prove one of the implications.
2. We occasionally use Lemma A.3 to establish that any atom B occurring in L_k^V (resp. L_l^W), on which A depends, also has $l_{\mathcal{V}}(B) = k$(resp. $l_{\mathcal{W}}(B) = l$).

We omit the details. □

Let us now take a first look at the maximal choice series for some P and its relationship with another successful series.

Lemma A.5 *Let $V_1, \ldots$ be a series such that the construction in Proposition 3.2 terminates successfully for P with resulting model H_P^V. Let $S_1, \ldots$ be the maximal choice series for P. Then, for every i such that P_i^S is defined,*

$$\bigcup_{j<i} H_j^S = H_P^V \cap (B_P \backslash B_{P_i^S}).$$

PROOF. Let i be such that P_i^S is defined. First, if $A \in B_{P_i^S}$, then both $A \notin \cup_{j<i} H_j^S$ and $A \notin B_P \backslash B_{P_i^S}$ are trivially satisfied. So, suppose that $A \notin B_{P_i^S}$. Then $A \in H_P^V \Leftrightarrow A \in \cup_{j<l_V(A)+1} H_j^V$ and, of course, $A \notin B_{P_{l_V(A)+1}^V}$. The result now follows from Lemma A.4. □

We need one more lemma before we can actually complete the proof of Lemma 3.1.

Lemma A.6. Let $\mathcal{V} = V_1, \ldots$ be a series such that the construction in Proposition 3.2 terminates successfully for P with resulting model H_P^V. Let $\mathcal{S} = S_1, \ldots$ be the maximal choice series for P. Then, for every i such that P_i^V is defined, one of the following holds:

1. There is a $j \leq i$ such that $P_j^S = \emptyset$ and
$$\bigcup_{k<i} H_k^V = H_P^S \cap (B_P \backslash B_{P_i^V}).$$
2. Alternatively, P_i^S is defined and nonempty and the following both hold:
   (a)
   $$\bigcup_{k<i} H_k^V = \bigcup_{k<i} H_k^S \cap (B_P \backslash B_{P_i^V}).$$
   (b) $\forall (C^S, n_C) \in P_i^S : \exists (C^V, n_C) \in P_i^V$, where C^V is identical to C^S except for the possible presence of extra body literals, with an atom in $\cup_{k<i} S_k$, which are satisfied in $\cup_{k<i} H_k^S$.

PROOF. Not surprisingly, the proof proceeds through induction on i. Again, the base case ($i = 1$) is immediate (either $Ground(P) = \emptyset$ and 1 holds or $Ground(P) \neq \emptyset$ and 2 holds). So, let us prove the induction step. In other words, we assume that the property is satisfied for every $i \leq n$. If $P_n^V = \emptyset$, then P_i^V is not defined for $i > n$ and we are done. Suppose, therefore, that $P_n^V \neq \emptyset$. Then P_{n+1}^V is defined and we have to show that the property holds for $i = n + 1$.

Now, if there is a $j \leq n + 1$ such that $P_j^S = \emptyset$, then it follows from Lemma A.4 that
$$\forall A \in B_P \backslash B_{P_{n+1}^V} : A \in H_P^S \Leftrightarrow A \in \bigcup_{k<n+1} H_k^V.$$
Moreover, obviously
$$\forall A \in B_{P_{n+1}^V} : A \notin \bigcup_{k<n+1} H_k^V$$
and the result follows.

This leaves us with the case that P_n^S is defined and nonempty. We have to show that also P_{n+1}^S is defined, and that 2(a) and 2(b) hold for $i = n + 1$ in case it is nonempty.

- We first show that P_n^S has at least one minimal component. Suppose that this is not the case. Then every atom in $B_{P_n^S}$ is a member of some infinite

series of atoms $A_1, A_2, \ldots$ such that $A_1 > A_2 > \cdots$. Now, the clauses in *Ground*(P) (for the atoms in $B_{P_n^S}$) that correspond to the clauses in P_n^S might contain extra body literals whose atom is not in $B_{P_n^S}$ and which are satisfied according to $\cup_{k<n} H_k^S$. Lemma A.4 ensures that these literals can never be falsified in the $\mathcal{V}$-construction. So, the $\mathcal{V}$-construction can delete none of these clauses without first having an atom from $B_{P_n^S}$ in a bottom component of some $B_{P_l^V}$. This, however, is not possible and therefore the $\mathcal{V}$-construction cannot terminate successfully, which contradicts the assumption that it does.

- Our next task is to show that L_n^S is definite. Suppose that it is not. Then there is a bottom component B of $B_{P_n^S}$ containing at least one atom with a nondefinite defining clause C in P_n^S. However, then, if any atom in B is defined in terms of an atom not in B, B would not be a bottom component of $B_{P_n^S}$. Therefore, all clauses in P_n^S with a head in B only contain literals with an atom in B. Again, it follows that none of the corresponding clauses in *Ground*(P) can be deleted by the $\mathcal{V}$-construction without first having B as part of some V_l. However, then also L_i^V would be nondefinite, which again contradicts the assumption that $\mathcal{V}$ terminates successfully.
- So, P_{n+1}^S is indeed defined. Above, we have already dealt with the case that it is empty. Assume, therefore, that it is nonempty. We prove 2(a) and 2(b) for $i = n + 1$.

  We start with 2(b) and first show that $V_n \cap B_{P_n^S} \subseteq S_n$. Let $A \in V_n \cap B_{P_n^S}$. We show that the subprogram of P_n^S on which A depends is either empty or definite. In both cases, A does not depend negatively on any atom in $B_{P_n^S}$, so that by definition, $A \in S_n$. If P_n^S contains no clause with A as its head, we are done. Otherwise, let $(A \leftarrow B^S, n_C) \in P_n^S$. By the induction hypothesis 2(b), there exists a clause $(A \leftarrow B^V, n_C) \in P_n^V$, such that every literal in B^S is also in B^V. Because $A \in V_n$ and L_n^V is definite, $(A \leftarrow B^V, n_C)$ is definite and so is $(A \leftarrow B^S, n_C)$. It remains to be shown that for all other atoms $B \in B_{P_n^S}$, such that $B \leq A$, the same holds. Let B be such an atom. Using induction hypothesis 2(b) again, $B \leq A$ also holds in P_n^V: for any sequence of clauses $(C_k^S, {}^nC_k)_k$ in P_n^S, establishing the dependency of A and B in P_n^S: the corresponding sequence $(C_k^V, {}^nC_k)_k$ in P_n^V establishes the dependency in P_n^V. Therefore, $B \in V_n \cap B_{P_n^S}$, and, by the same argument as used for A above, the clauses in P_n^S with head B are definite. Thus, the subprogram of P_n^S on which A depends is definite, and $A \in S_n$.
  Next, we show that

$$\forall (C^S, n_C) \in P_{n+1}^S : \exists (C^V, n_C) \in P_{n+1}^V.$$

  Since it is given that

$$\forall (C^S, n_C) \in P_n^S : \exists (C^V, n_C) \in P_n^V,$$

  it suffices to prove that if a clause $(C^S, n_C) \in P_n^S$ is not pruned during the construction of P_{n+1}^S from P_n^S, then neither is the corresponding clause $(C^S, n_C) \in P_n^V$ pruned while constructing P_{n+1}^V from P_n^V.
  Suppose that such a $(C^V, n_C) \in P_n^V$ is pruned. One possible cause is that $(C^V, n_C) \in L_n^V$. Since (C^S, n_C) has the same head, say A, as (C^V, n_C) and

$A \in V_n \cap B_{P^S_n}$, by $V_n \cap B_{P^S_n} \subseteq S_n$, we have $A \in S_n$. So, $(C^S, n_C) \in L^S_n$ and is pruned too.
Alternatively, $(C^V, n_C) \in P^V_n \backslash L^V_n$ can be pruned because it contains a body literal B, with atom $B^a \in V_n$, such that B is falsified in H^V_n. By induction hypothesis 2(b), the only body literals of (C^V, n_C) that do not occur in the body of (C^S, n_C) are satisfied in $\cup_{k<n} H^S_k$. Furthermore, by Lemma A.4,

$$\forall A \in B_P \backslash \left(B_{P^S_{n+1}} \cup B_{P^V_{n+1}} \right) : \quad A \in \bigcup_{k<n+1} H^V_k \quad \Leftrightarrow \quad A \in \bigcup_{k<n+1} H^S_k . (*)$$

so that B is not satisfied in $\cup_{k<n} H^S_k$ and is a body literal of (C^S, n_C). Finally, $B^a \in S^n$, because $B^a \in V_n \cap B_{P^S_n}$, and B is falsified in H^S_n, because it is falsified in H^V_n and $(*)$ holds. Thus, (C^S, n_C) would also be pruned in the $\mathcal{S}$-construction.
It remains to be proved that C^S and C^V are related as stated in 2(b). To see this, let $(C^S, n_C) \in P^S_n$, $(C^V, n_C) \in P^V_n$, $(C'^S, n_C) \in P^S_{n+1}$, and $(C'^V, n_C) \in P^V_{n+1}$. We know that C^S and C^V are related in the proper way. Moreover, C'^S is identical to C^S except for the possible removal of body literals with an atom in S^n which are satisfied in H^S_n. Additionally, C'^V is identical to C^V except for the possible removal of body literals with an atom in V^n which are satisfied in H^V_n. The result now again follows from $(*)$ and $V_n \cap B_{P^S_n} \subseteq S_n$, since we obtain that no literal which occurs in the bodies of both C^S and C^V can be removed from the latter without likewise being removed from the former.

— Finally, we prove 2(a) for $i = n+1$. First, 2(b) for $i = n+1$ implies

$$B_{P^S_{n+1}} \subseteq B_{P^V_{n+1}} .$$

Therefore, $(*)$ can be rewritten as

$$\forall A \in B_P \backslash B_{P^V_{n+1}} : \quad A \in \bigcup_{k<n+1} H^V_k \quad \Leftrightarrow \quad A \in \bigcup_{k<n+1} H^S_k .$$

Moreover, from the definition of the construction in Proposition 3.2,

$$\bigcup_{k<n+1} H^V_k \cap \left(B_P \backslash B_{P^V_{n+1}} \right) = \bigcup_{k<n+1} H^V_k ,$$

so that 2(a) for $i = n+1$ follows.

□

We can now complete the proof for Lemma 3.1:

Proof of lemma 3.1. First, it is clear that $\mathcal{S}$ will also terminate successfully. Indeed, suppose there is some i such that $P^V_i = \emptyset$. Then case 2 in Lemma A.6 can not hold for i, due to 2(b). Therefore, case 1 holds and $\mathcal{S}$ terminates successfully. So, suppose $P^V_i \neq \emptyset$ for all i. If there is an i such that case 1 in Lemma A.6 holds, we again obtain successful termination of $\mathcal{S}$. When case 2 holds for every i, P^S_i is defined for every $i < \omega$. However, since $\cup_{i<\omega} CH(P^V_i) = \emptyset$, 2(b) implies that also $\cap_{i<\omega} CH(P^S_i) = \emptyset$.

Next, Lemma A.5 implies that for every j such that P_j^S is defined, $\cup_{k<j} H_k^S \subseteq H_P^V$. It follows that $H_P^S \subseteq H_P^V$. Similarly, Lemma A.6 guarantees that for every i such that P_i^V is defined, $\cup_{k<i} H_k^V \subseteq H_P^S$ and we obtain $H_P^V \subseteq H_P^S$. Therefore, the two models are equal. □

We thank Antonio Brogi, Maurice Bruynooghe, Francois Bry, Michael Codish, Marc Denecker, Wlodzimierz Drabent, Michael Gelfond, Pat Hill, Y.J. Jiang, Marianne Kalsbeek, Robert Kowalski, John Lloyd, Rodney Topor, and Frank van Harmelen for interesting discussions and/or comments. We also greatly appreciated several helpful observations from anonymous referees. Finally, we are obliged to Maurice Bruynooghe for suggesting the title of this paper.

REFERENCES

1. Abramson, H., and Rogers, M. H. (eds.), *Meta-Programming in Logic Programming, Proceedings of Meta'88*, MIT Press, Cambridge, MA, 1989.
2. Apt, K. R., Blair, H., and Walker, A., Towards a Theory of Declarative Knowledge, in: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, Los Altos, CA, 1988, pp. 89–148.
3. Apt, K R., Marchiori, E., and Palamidessi, C., A Theory of First-Order Built-In's of Prolog, in: H. Kirchner and G. Levi (eds.), *Proceedings of the 3rd International Conference on Algebraic and Logic Programming, Lecture Notes in Computer Science* 632, Springer, New York, 1992, pp. 69–83.
4. Bowen, K. A., Meta-Level Programming and Knowledge Representation, *New Generation Comput.* 3(4):359–383 (1985).
5. 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, New York, 1982, pp. 153–172.
6. Bruynooghe, M. (ed.), *Meta'90, Proceedings of the Second Workshop on Metaprogramming in Logic*, Katholieke Univ. Leuven, 1990.
7. Bry, F., Query Evaluation in Recursive Databases: Bottom-Up and Top-Down Reconciled, *Data Knowledge Eng.* 5(4):289–312 (1990).
8. Bry, F., Manthey, R., and Martens, B., Integrity Verification in Knowledge Bases, in: A Voronkov (ed.), *Proceedings 1st and 2nd Russian Conference on Logic Programming*, 592 Springer, New York, 1992, pp. 114–139.
9. Chen, W., Kifer, M., and Warren, D. S., HiLog: A Foundation for Higher-Order Logic Programming, *J. Logic Program.* 15(3):187–230 (1993).
10. 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.), *Proceedings of Meta'92*, Lecture Notes in Computer Science 649, Springer, New York, 1992, pp. 192–204.
11. Demolombe, R., Syntactical Characterization of a Subset of Domain Independent Formulas, *J. ACM* 39(1):71–94 (1992).
12. Denecker, M., and De Schreye, D., Justification Semantics: A Unifying Framework for the Semantics of Logic Programs, in: A. Nerode and L. Pereira (eds.), *Proceedings of LPNMR'93*, Lisbon, Portugal, MIT Press, Cambridge, MA, 1993, pp. 365–379.
13. Denecker, M., De Schreye, D., and Willems, Y. D., Terms in Logic Programs: A Problem with Their Semantics and Its Effect on the Programming Methodology, *CCAI, J. Integrated Study of Artificial Intelligence, Cognitive Sci. Appl. Epistemology* 7(3&4):363–383 (1990).

14. Enderton, H. B., *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.
15. Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C., Declarative Modeling of the Operational Behavior of Logic Programs, *Theoret. Comput. Sci.* 69:289–318 (1989).
16. Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C., A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs, *Inform. Computation* 103(1):86–113 (1993).
17. Gallagher, J., Transforming Logic Programs by Specialising Interpreters, in: *Proceedings ECAI'86*, 1986, pp. 109–122.
18. Gallagher, J., de Waal, D. A., Deletion of Redundant Unary Type Predicates from Logic Programs, in: K.-K. Lau and T. Clement (eds.), *Proceedings LOPSTR'92, Lecture Notes in Computer Science*, Springer, New York, 1993.
19. Gallaire, H., and Lasserre, C., Metalevel Control of Logic Programs, in: K. L. Clark and S.-A. Tärnlund (eds.), *Logic Programming*, Academic Press, New York, 1982, pp. 173–185.
20. Giunchiglia, F., and Traverso, P., Plan Formation and Execution in a Uniform Architecture of Declarative Metatheories, in: M. Bruynooghe (ed.), *Proceedings Meta'90*, Leuven, April 1990, pp. 306–322.
21. Hill, P., and Lloyd, J., *The Gödel Programming Language*, MIT Press, Cambridge, MA, 1994.
22. Hill, P. M., and Lloyd, J. W., Meta-Programming for Dynamic Knowledge Bases, Technical Report CS-88-18, Computer Science Department, University of Bristol, Great Britain, 1988.
23. Hill, P. M., and Lloyd, J. W., Analysis of Meta-Programs, in: H. D. Abramson and M. H. Rogers (eds.), *Proceedings Meta'88*, MIT Press, Cambridge, MA, 1989, pp. 23–51.
24. Jiang, Y. J., On the Semantics of Real Metalogic Programming—A Preliminary Report, Technical Report, Department of Computing, Imperial College, London, Great Britain, July 1993.
25. Kalsbeek, M., The Vanilla Meta-Interpreter for Definite Logic Programs and Ambivalent Syntax, Technical Report CT-93-01, Department of Mathematics and Computer Science, University of Amsterdam, The Netherlands, January 1993.
26. Kim, J.-S., and Kowalski, R. A., An Application of Amalgamated Logic to Multi-Agent Belief, in: M. Bruynooghe (ed.), *Proceedings of Meta'90, Leuven*, April 1990, pp. 272–283.
27. Kowalski, R. A., Problems and Promises of Computational Logic, in: J. W. Lloyd (ed.), *Proceedings of the Esprit Symposium on Compuational Logic*, Springer, New York, 1990, pp. 1–36.
28. Levi, G., and Ramundo, D., A Formalization of Metaprogramming for Real, in: D. S. Warrant (ed.), *Proceedings ICLP'93*, Budapest, MIT Press, Cambridge, MA, 1993, pp. 354–373.
29. Lloyd, J. W., *Foundations of Logic Programming*, Springer, New York, 1987.
30. Martens, B., and De Schreye, D., A Perfect Herbrand Semantics for Untyped Vanilla Meta-Programming, in: K. Apt (ed.), *Proceedings JICSLP'92*, Washington, MIT Press, Cambridge, MA, 1992, pp. 511–525.
31. Martens, B., and De Schreye, D., Why Untyped Non-Ground Meta-Programming is not (much of) a Problem, Technical Report CW 159, Department Computerweten-

schappen, Kathdieke Univ. Leuven, Belgium, December 1992; revised November 1993.

32. Pottorossi, A. (ed.), *Meta-Programming in Logic, Proceedings of Meta'92, Lecture Notes in Computer Science* Springer, 649, New York, 1992.
33. Przymusinska, H., and Przymusinski, T. C., Weakly Perfect Model Semantics for Logic Programs, in: R. A. Kowalski and K. A. Bowen (eds.), *Proceedings ICSLP'88*, 1988, pp. 1106–1120.
34. Przymusinska, H., and Przymusinski, T. C., Semantic Issues in Deductive Databases and Logic Programs, in: R. B. Banerji (ed.), *Formal Techniques in Artificial Intelligence*, Elsevier Science Publishers B.V., Amsterdam, 1990, pp. 321-367.
35. Przymusinska, H., and Przymusinski, T. C., Weakly Stratified Logic Programs, *Fundamenta Informaticae* XIII:51–65 (1990).
36. Przymusinski, T. C., On the Declarative Semantics of Deductive Databases and Logic Programs, in: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, Los Altos, CA, 1988, pp. 193–216.
37. Richards, B., A Point of Reference, *Synthese* 28:361–454 (1974).
38. Ross, K. A., Modular Stratification and Magic Sets for DATALOG Programs with Negation, in: *Proceedings PODS'90*, Nashville, Tenessee, ACM, New York, 1990, pp. 161-171.
39. Sterling, L., and Beer, R. D., Meta Interpreters for Expert System Construction, *J. Logic Program.* 6(1&2):163–178 (1989).
40. Sterling, L., and Shapiro, E., *The Art of Prolog*, MIT Press, Cambridge, MA, 1986.
41. Subrahmanian, V. S., A Simple Formulation of the Theory of Metalogic Programming, in: H. D. Abramson and M. H. Rogers (eds.), *Proceedings Meta'88*, MIT Press, Cambridge, MA, 1989, pp. 65-101.
42. Takeuchi, A., and Furukawa, K., Partial Evaluation of Prolog Programs and Its Application to Metaprogramming, in: H.-J. Kugler (ed.), *Information Processing. 86*, 1986, pp. 415–420.
43. Topor, R. W., and Sonenberg, E. A., On Domain Independent Databases, in: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, Los Altos, CA, 1988, pp. 217–240.
44. Turi, D., Extending S-Models to Logic Programs with Negation, in: K. Furukawa (ed.), *Proceedings of ICLP'91*, Paris, MIT Press, Cambridge, MA, 1991, pp. 397–411.
45. Ullman, J. D., *Database and Knowledge-Base Systems*, Computer Science Press, Rockville, MD, 1988, vol. I.
46. Van Gelder, A., Ross, K. A., and Schlipf, J. S., The Well-Founded Semantics for General Logic Programs, *J. ACM* 38(3):620–650 (1991).