

On termination of meta-programs

Alexander Serebrenik

Danny De Schreye

Report CW 306, February 2001



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

On termination of meta-programs

Alexander Serebrenik

Danny De Schreye

Report CW 306, February 2001

Department of Computer Science, K.U.Leuven

Abstract

The term *meta-programming* refers to the ability of writing programs that have other programs as data and exploit their semantics.

The aim of this paper is presenting a methodology allowing to perform a correct termination analysis for a broad class of practical meta-interpreters, including negation and performing different tasks during the execution, together with different classes of object programs. It is based on combining power of general term-orders, used in proving termination of term-rewrite systems and programs, and on the well-known acceptability condition, used in proving termination of logic programs.

The methodology establishes a relationship between the order needed to prove termination of the object program and the order needed to prove termination of the meta-interpreter together with this object program. If such a relationship is established, termination of one of those implies termination of the other one, i.e., that the meta-interpreter improves (preserves) termination.

Among the meta-interpreters that are analysed correctly are a proof trees constructing meta-interpreter, different kinds of tracers and reasoners.

Keywords : termination analysis, meta-programming

CR Subject Classification : D.1.6, D.2.4

1 Introduction

The term *meta-programming* refers to the ability of writing programs that have other programs as data and exploit their semantics [5]. The choice of logic programming as a basis for meta-programming offers a number of practical and theoretical advantages. One of them is the possibility of tackling critical foundation problems of meta-programming within a framework with a strong theoretical basis. Another is the surprising ease of programming. These reasons motivated an intensive research on meta-programming inside the logic programming community [5, 15, 18, 21, 22].

On the other hand, termination analysis is one of the most intensive research areas in logic programming as well. See [12] for the survey. More recent work on this topic can be found among others in [13, 14, 17, 19, 23, 25, 27, 29].

Traditionally, termination analysis of logic programs have been done either by the “transformational” approach or by the “direct” one. A transformational approach first transforms the logic program into an “equivalent” term-rewrite system (or, in some cases, into an equivalent functional program). Here, equivalence means that, at the very least, the termination of the term-rewrite system should imply the termination of the logic program, for some predefined collection of queries¹. Direct approaches do not include such a transformation, but prove the termination directly on the basis of the logic program. In [24] we have developed an approach that provides the best of both worlds: a means to incorporate into “direct” approaches the generality of general term-orderings.

The aim of this paper is presenting a methodology allowing to perform a correct termination analysis for a broad class of meta-interpreters together with different classes of object programs. This methodology is based on the “combined” approach to termination analysis mentioned above.

Example 1. Our research has been motivated by the famous “vanilla” meta-interpreter, undoubtedly belonging to logic programming classics.

$$\begin{aligned} & \text{solve}(\text{true}). \\ & \text{solve}((\text{Atom}, \text{Atoms})) \leftarrow \text{solve}(\text{Atom}), \text{solve}(\text{Atoms}). \\ & \text{solve}(\text{Head}) \leftarrow \text{clause}(\text{Head}, \text{Body}), \text{solve}(\text{Body}). \end{aligned}$$

¹ The approach of Arts [6] is exceptional in the sense that the termination of the logic program is concluded from a weaker property of *single-redex normalisation* of the term-rewrite system.

Termination of this meta-interpreter, presented in Example 1, has been studied by Pedreschi and Ruggieri. They proved, that it improves termination (Corollary 40, [22]). However, we can claim more—“vanilla” does not just improve termination, but preserves it. \square

In order for meta-interpreters to be useful in applications they should be able to cope with a richer language than the one of the “vanilla” meta-interpreter, including, for example, negation. Moreover, typical applications of meta-interpreters, such as debuggers, will also require producing some additional output or performing some additional tasks during the execution, such as constructing proof trees or cutting “unlikely” branches for an uncertainty reasoner with cutoff. These extensions can and usually will influence termination properties of the meta-interpreter.

By extending the suggested technique [24] to normal programs, we are able to perform the correct analysis of a number of (possibly extended) meta-interpreters, performing tasks as described above. We identify popular classes of meta-interpreters, such as *extended meta-interpreters* [21], and using this extended technique prove that termination is usually improved. We also state some more generic conditions implying preservation of termination.

The rest of this paper is organised as following. We start by some preliminary remarks and basic definitions. Then, we present the methodology developed applied to the “vanilla” meta-interpreter. Afterwards we show how the same methodology can be applied for more advanced meta-interpreters and conclude.

2 Preliminaries

2.1 Term ordering

A *quasi-order* over a set S is an reflexive, antisymmetric and transitive relation \geq defined on elements of S . We define the associated equivalence relation $=_{>}$ as $s =_{>} t$, if and only if $s \geq t$ and $t \geq s$, and the associated strict *partial ordering* $>$ if and only if $s \geq t$ but not $t \geq s$. If neither $s \geq t$, nor $t \geq s$ we write $s \parallel_{>} t$.

An ordered set S is said to be *well-founded* if there are no infinite descending sequences $s_1 > s_2 > \dots$ of elements of S . If the set S is clear from the context we will say that the order, defined on it, is well-founded.

2.2 Logic Programs

We follow the standard notation for terms and atoms. A *query* is a finite sequence of atoms. Given an atom A , $rel(A)$ denotes the predicate

occurring in A . $Term_P$ and $Atom_P$ denote, respectively, sets of all terms and atoms that can be constructed from the language underlying P . The extended Herbrand Universe U_P^E (the extended Herbrand base B_P^E) is a quotient set of $Term_P$ ($Atom_P$) modulo the variant relation.

We refer to an SLD-tree constructed using the left-to-right selection rule of Prolog, as an LD-tree. We will say that a goal G *LD-terminates* for a program P , if the LD-tree for (P, G) is finite.

The following definition is borrowed from [1].

Definition 1. *Let P be a program and p, q be predicates occurring in it.*

- *We say that p refers to q in P if there is a clause in P that uses p in its head and q in its body.*
- *We say that p depends on q in P and write $p \sqsupseteq q$, if (p, q) is in the transitive, reflexive closure of the relation refers to.*
- *We say that p and q are mutually recursive and write $p \simeq q$, if $p \sqsupseteq q$ and $q \sqsupseteq p$.*

We also abbreviate $p \sqsupseteq q, q \sqsupseteq p$ by $p \sqsupset p$.

Results for termination of meta-interpreters presented in this paper are based on notion of term-acceptability, introduced in [24]. This notion of term-acceptability generalises the notion of acceptability w.r.t. a set [13] in two ways: 1) it generalises it to general term orders, 2) it generalises it to mutual recursion, using the standard notion of mutual recursion [1]—the original definition of acceptability required decrease only for calls to the predicate that appears in the head of the clause. This restriction limited the approach to programs only with direct recursion.

Definition 2. *Let S be a set of atomic queries and P a definite program. P is term-acceptable w.r.t. S if there exists a well-founded order $>$, such that*

- *for any $A \in \text{Call}(P, S)$*
- *for any clause $A' \leftarrow B_1, \dots, B_n$ in P , such that $\text{mgu}(A, A') = \theta$ exists,*
- *for any atom B_i , such that $\text{rel}(B_i) \simeq \text{rel}(A)$*
- *for any computed answer substitution σ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:*

$$A > B_i\theta\sigma$$

In [24] we prove the following theorem.

Theorem 1. *Let P be a program. P is term-acceptable w.r.t. a set of atomic queries S if and only if P is LD-terminating for all queries in S .*

Additional notion we are going to use is a notion of *partition*. Intuitively, a sequence of sequences $(x_{1,1}, \dots, x_{1,n_1}), \dots, (x_{m,1}, \dots, x_{m,n_m})$ forms a partition of a sequence $(x_1, \dots, x_{\sum n_i})$ if they are exactly the same, except for additional division to subsequences. More formally,

Definition 3. Let S be a set, and let (x_1, \dots, x_k) be a sequence of elements of this set. Let $(x_{1,1}, \dots, x_{1,n_1}), \dots, (x_{m,1}, \dots, x_{m,n_m})$ be sequences of elements of S . We say that $(x_{1,1}, \dots, x_{1,n_1}), \dots, (x_{m,1}, \dots, x_{m,n_m})$ forms a partition of (x_1, \dots, x_k) if the following holds:

- $x_1 = x_{1,1}$
- If $x_l = x_{i,j}$ then

$$x_{l+1} = \begin{cases} x_{i,(j+1)} & \text{if } j < n_i \\ x_{(i+1),1} & \text{otherwise} \end{cases}$$

3 Basic definitions

In this section we present a number of basic definitions.

Definition 4. Let P be a program. The clause-encoding $\gamma_{ce}(P)$ is a collection of facts of a new predicate clause, such that $\text{clause}(H, B) \in \gamma_{ce}(P)$ if and only if $H \leftarrow B$ is a clause in P .

Example 2. Let P be the following program:

$$p(X) \leftarrow q(X, Y), p(Y). \quad q(f(Z), Z).$$

Then, the following program is $\gamma_{ce}(P)$:

$$\text{clause}(p(X), (q(X, Y), p(Y))). \quad \text{clause}(q(f(Z), Z), \text{true}).$$

A *meta-interpreter* for a language is an interpreter for the language written in the language itself. We follow [26] by using a predicate `solve` for the meta-interpreter.

More formally,

Definition 5. The program P is called a meta-program if it can be represented as $M \cup I$, such that:

- M defines a predicate `solve` that does not appear in I .
- I is a clause-encoding of some program P' .

M is called the meta-interpreter. P' is called the interpreted program.

We also assume that $,/2$ and $clause/2$ do not appear in the language underlying the interpreted program.

Now we are going to define the notions of *correctness* and *completeness* for meta-interpreters, that will relate computed answers of the interpreted program and of the meta-program. Intuitively, if a *correct* meta-interpreter together with a clause-encoding of the interpreted program compute an answer of the form $solve(t_0, t_1, \dots, t_n)$, t_0 is also an answer of the corresponding goal w.r.t. the interpreted program, even if some of those answers might be missed. Similarly, if the meta-interpreter is *complete*, we can be sure that all computed answers of the interpreted program and the corresponding goal are computed by the meta-program, even if some irrelevant answers might be computed as well. We formalise this intuition in the following definitions.

Definition 6. *The meta-interpreter M is called correct if for every program P and every goal $G \in B_P^E$ if $solve(t_0, t_1, \dots, t_n)$ is a computed answer for $\{solve(G)\} \cup M \cup \gamma_{ce}(P)$ then t_0 is a computed answer for $\{G\} \cup P$.*

Definition 7. *The meta-interpreter M is called complete if for every program P and every goal $G \in B_P^E$ if t_0 is a computed answer for $\{G\} \cup P$ then there exist t_1, \dots, t_n such that $solve(t_0, t_1, \dots, t_n)$ is a computed answer for $\{solve(G)\} \cup M \cup \gamma_{ce}(P)$.*

Example 3. This example demonstrates two small meta-interpreters.

(a) $solve(A) \leftarrow fail.$ (b) $solve(A) \leftarrow A.$

The meta-interpreter (a) is correct, since there are no computed answers for $\leftarrow solve(G)$. The meta-interpreter (b), exploiting the meta-variable facility of Prolog, is both correct and complete. \square

Our goal is to study the termination of goals of the form $solve(Goal)$, where $Goal$ is a goal with respect to the interpreted program.

The first issue is to define the *notion of termination*, and then to state conditions implying termination.

Observe, that the meta-interpreter may change the termination property of the interpreted program. For applications such as debuggers the desired behaviour is termination preservation.

Definition 8. *Let M be a meta-interpreter and P' be an interpreted program. M is called improving LD-termination if for every goal G finiteness of the LD-tree of $\{G\} \cup I$ implies finiteness of the LD-tree of $\{solve(G)\} \cup (M \cup \gamma_{ce}(P'))$.*

The meta-interpreter is called *preserving LD-termination*, if the second direction of the implication also holds. Meta-interpreter (a), shown above is improving termination, while meta-interpreter (b) is preserving it.

4 Termination of “vanilla” meta-interpreter

Termination of the “vanilla” meta-interpreter, presented in Example 1, has been studied by Pedreschi and Ruggieri. They proved, that “vanilla” improves termination (Corollary 40, [22]). However, we can claim more—“vanilla” does not just improve termination, but preserves it.

We base our proof on correctness and completeness of this meta-interpreter, proved in [18]. Observe that in general correctness and completeness are not sufficient for the calls set to be preserved. Indeed, consider the following example, motivated by the ideas of unfolding [7]. It eliminates calls to undefined predicates.

Example 4.

$$\begin{aligned} & \text{solve}(\text{true}). \\ & \text{solve}((A, B)) \leftarrow \text{solve}(A), \text{solve}(B). \\ & \text{solve}(A) \leftarrow \text{clause}(A, B), \text{check}(B), \text{solve}(B). \\ \\ & \text{check}((A, B)) \leftarrow \text{check}(A), \text{check}(B). \\ & \text{check}(A) \leftarrow \text{clause}(A, _). \end{aligned}$$

This meta-interpreter is correct and complete, i.e., preserves computed answers. However, it does not preserve termination. Indeed, let P' be the following program [7].

$$p \leftarrow q, r. \quad t \leftarrow r, q. \quad q \leftarrow q.$$

and let p be the goal. Then, p w.r.t. P' does not terminate, while $\leftarrow \text{solve}(p)$ w.r.t. $M_P \cup \gamma_{ce}(P')$ terminates (finitely fails). Thus, this meta-interpreter does not preserve LD-termination. Observe, that unfolding may only improve termination [7], thus, this meta-interpreter is improving LD-termination. \square

Thus, we need some additional result, claiming that the calls are preserved.

Lemma 1. *Let P' be an interpreted program, M_P be the vanilla meta-interpreter and $G \in B_{P'}^E$, then*

$$\{\text{solve}(A) \mid A \in \text{Call}(P', G)\} \equiv \text{Call}(M_P \cup \gamma_{ce}(P'), \text{solve}(G)) \cap \{\text{solve}(A) \mid A \in B_{P'}^E\}$$

where \equiv means equality up to variable renaming.

This lemma extends Theorem 9 [22], by claiming that not only every call of the meta-program and a meta-goal “mimics” the original execution, but also that every call of the original program and goal is “mimicked” by the meta-program.

Proof. See Appendix 9.

Theorem 2. *Let P' be a definite program, S a set of queries, and M_P the vanilla meta-interpreter, such that $M_P \cup \gamma_{ce}(P')$ is LD-terminating for all queries in $\{\text{solve}(G) \mid G \in S\}$. Then, P' is LD-terminating for all queries in S .*

Proof. (sketch) By Theorem 1 $M_P \cup \gamma_{ce}(P')$ is term-acceptable w.r.t. $\{\text{solve}(G) \mid G \in S\}$. We are going to prove term-acceptability of P' w.r.t. S . By Theorem 1 this will imply termination.

Since $M_P \cup \gamma_{ce}(P')$ is term-acceptable w.r.t. $\{\text{solve}(G) \mid G \in S\}$ there is a well-founded order $>$, satisfying requirements of Definition 2. Define a new order on atoms of P' as following: $A \succ B$ if $\text{solve}(A) > \text{solve}(B)$.

This order is defined on $\{G \mid \text{solve}(G) \in \text{Call}(M_P \cup \gamma_{ce}(P'), \text{solve}(S))\}$. By Lemma 1 this set coincides with $\text{Call}(P', S)$. From the corresponding properties of $>$ follows that \succ is well-defined and well-founded.

The only thing remains to be proved is that P' is term-acceptable w.r.t. S via \succ . Let $A \in \text{Call}(P', S)$ and let $A' \leftarrow B_1, \dots, B_n$ be a clause in P' , such that $\text{mgu}(A, A') = \theta$ exists. Then, θ is also mgu of $\text{solve}(A)$ and $\text{solve}(A')$.

Let σ map $B\theta$ to $(B_1, \dots, B_n)\theta$. It is one of possible computed answer substitutions for $\leftarrow \text{clause}(A\theta, B\theta)$. Thus, by term-acceptability of $M_P \cup \gamma_{ce}(P')$ w.r.t. $\text{solve}(S)$ holds that $\text{solve}(A) > \text{solve}((B_1, \dots, B_n)\theta)$.

Term-acceptability also implies $\text{solve}((B_1, \dots, B_n)\theta) > \text{solve}(B_1\theta)$ and $\text{solve}((B_1, \dots, B_n)\theta) > \text{solve}((B_2, \dots, B_n)\theta\sigma_1)$, where σ_1 is a computed answer substitution for $\text{solve}(B_1\theta)$. By proceeding this way we conclude, that for any atom B_i , $\text{solve}(A) > \text{solve}(B_i\theta\sigma_1 \dots \sigma_{i-1})$, where σ_j is a computed answer substitution for $\leftarrow \text{solve}(B_j\theta\sigma_1 \dots \sigma_{j-1})$. By definition of \succ , this means that $A \succ B_i\theta\sigma_1 \dots \sigma_{i-1}$.

Observe, that if A was not in $\{G \mid \text{solve}(G) \in \text{Call}(P, \text{solve}(S))\}$ “stripping off” solve was not possible.

Correctness and completeness imply the term-acceptability and complete the proof. ■

The second direction of the theorem has been proved by Pedreschi and Ruggieri [22]. It allows us to state the following corollary.

Corollary 1. *Vanilla meta-interpreter preserves LD-termination.*

The proof of Theorem 2 sketched above, suggests the following methodology for proving that some meta-interpreter improves LD-termination. First, define an order on the set of calls to the meta-interpreter, that reflects its behaviour. Then, establish the relationship between a new order and the one that reflects term-acceptability w.r.t. a set of the interpreted program. Prove using this relationship that the newly defined order is well-defined, well-founded and reflects term-acceptability of the meta-program w.r.t. a corresponding set of calls. In order for the proofs to be correct one may need to assume (or to prove as a prerequisite) that the meta-interpreter is correct and that the set of calls of the interpreted program and of the meta-program correspond to each other. The opposite direction can be proved by using a similar methodology. Additional examples of methodology applications are provided in the next section.

5 Advanced meta-interpreters

In order for meta-interpreters to be useful in applications they should be able to cope with a richer language than the “vanilla” meta-interpreter, including, for example, negation. Moreover, typical applications of meta-interpreters, such as debuggers, will also require producing some additional output or performing some additional tasks during the execution, such as constructing proof trees or cutting “unlikely” branches for an uncertainty reasoner with cutoff. These extensions can and usually will influence termination properties of the meta-interpreter.

In this section we identify some most-popular classes of meta-interpreters and extend the methodology presented in the previous section to analyse them.

5.1 Extended meta-interpreters

The first class of meta-interpreters we identify is a class of *extended meta-interpreters* [21]. This class includes, among others, a proof trees

constructing meta-interpreter [26], that can be used as a basis for explanation facilities in expert system, meta-interpreters allowing reasoning about theories and provability [9, 20] or reasoning with uncertainty [26]. We slightly adapt the notation of [21].

Definition 9. *A definite program of the following form will be called extended meta-interpreter*

$$\begin{aligned}
& \text{solve}(\text{empty}, t_{11}, \dots, t_{1n}) \leftarrow C_{11}, \dots, C_{1m_1} \\
& \text{solve}((A, B), t_{21}, \dots, t_{2n}) \leftarrow \\
& \quad \text{solve}(A, t_{31}, \dots, t_{3n}), \\
& \quad \text{solve}(B, t_{41}, \dots, t_{4n}) \\
& \quad C_{21}, \dots, C_{2m_2} \\
& \text{solve}(A, t_{51}, \dots, t_{5n}) \leftarrow \\
& \quad \text{clause}(A, B, s_1, \dots, s_k), \\
& \quad \text{solve}(B, t_{61}, \dots, t_{6n}) \\
& \quad C_{31}, \dots, C_{3m_3}
\end{aligned}$$

where the t_{ij} terms are extra arguments of the solve predicate, s_p terms are extra arguments of the clause 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).

For the sake of simplicity we denote the predicate of C_{kl} by c_{kl} . Using this notation, we can say that the definition above implies:

- solve is the only meta-predicate and
- solve \sqsupset c_{kl} for every k and l .

The following theorem reduces the problem of reasoning on termination of meta-program to reasoning on termination of the object program and on termination of c_{kl} 's w.r.t. the meta-interpreter. Since there are no clauses defining c_{kl} and depending on solve, the later question can be answered by using well-known termination analysis techniques (see [12] for the survey). As before, the reduction is done by considering an order used for proving term-acceptability of the object program, constructing a new order for the meta-program that is based on the previous one and proving term-acceptability of the meta-program.

More formally, we obtain the following result.

Theorem 3. *Let P' be an interpreted program, E_P be the extended meta-interpreter, and $G \in B_{P'}^E$, such that P' is terminating w.r.t. G and E_P is*

terminating w.r.t. $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\}$
then $E_P \cup \gamma_{ce}(P')$ terminates w.r.t. $\text{solve}(G)$.

Proof. (sketch) Let M_P be the vanilla meta-interpreter. By Corollary 1 $M_P \cup \gamma_{ce}(P')$ terminates w.r.t. $\text{solve}(G)$. By Theorem 1 $M_P \cup \gamma_{ce}(P')$ is term-acceptable w.r.t. $\text{solve}(G)$. Let $>_1$ be a well-founded order, such that $M_P \cup \gamma_{ce}(P')$ is term-acceptable w.r.t. $\text{solve}(G)$ via it.

Similarly, let $>_2$ be a well-founded order such that E_P is term-acceptable w.r.t. $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\}$ via it.

Based on $>_1$ and $>_2$ we define a new relation $>$ on $B_{E_P \cup \gamma_{ce}(P')}^E$ as following for any terms $s_1, s_2, t_{11}, \dots, t_{1n}, t_{21}, \dots, t_{2n}$ and atoms a_1, a_2 :

- $\text{solve}(s_1, t_{11}, \dots, t_{1n}) > \text{solve}(s_2, t_{21}, \dots, t_{2n})$, if $\text{solve}(s_1) >_1 \text{solve}(s_2)$
- $a_1 > a_2$, if $\text{rel}(a_1) \neq \text{solve}$, $\text{rel}(a_2) \neq \text{solve}$ and $a_1 >_2 a_2$
- $\text{solve}(s_1, t_{11}, \dots, t_{1n}) > a_1$, if $\text{rel}(a_1) \neq \text{solve}$

We need to prove that the relation defined is an order, that this order is well-founded and that $E_P \cup \gamma_{ce}(P')$ is term-acceptable w.r.t. $\text{solve}(G)$ via it. The first two claims are immediate from the corresponding properties of $>_1$ and $>_2$ and from the observation that $\text{solve} \sqsupset r$, for every predicate $r \neq \text{solve}$.

Proving term-acceptability is slightly more elaborated. Let $A_0 \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G))$. We distinguish between the following cases:

- $\text{rel}(A_0) \neq \text{solve}$. If $\text{rel}(A_0) = \text{clause}$ term-acceptability condition holds immediately, since there are no recursive clauses defining this predicate. Otherwise, term-acceptability of E_P w.r.t. $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\}$ via $>_2$ implies that for any clause $A' \leftarrow B_1, \dots, B_s$, such that $\text{mgu}(A_0, A') = \theta$ exists, for any atom B_i , such that $\text{rel}(A_0) \simeq \text{rel}(B_i)$ and for any computed answer substitution σ for $\leftarrow (B_1, \dots, B_{i-1})\theta$ holds $A_0 >_2 B_i\theta\sigma$. Therefore, $A_0 > B_i\theta\sigma$ holds as well.
- $\text{rel}(A_0) = \text{solve}$. In this case, there are three different clauses $A' \leftarrow B_1, \dots, B_s$, such that $\text{mgu}(A_0, A') = \theta$ exists.
 - $A' \leftarrow B_1, \dots, B_s$ is $\text{solve}(\text{empty}, t_{11}, \dots, t_{1n}) \leftarrow C_{11}, \dots, C_{1m_1}$. In this case there are no recursive body subgoals and term-acceptability condition is satisfied.
 - $A' \leftarrow B_1, \dots, B_s$ is

$$\begin{aligned} &\text{solve}((A, B), t_{21}, \dots, t_{2n}) \leftarrow \\ &\quad \text{solve}(A, t_{31}, \dots, t_{3n}), \\ &\quad \text{solve}(B, t_{41}, \dots, t_{4n}) \\ &\quad C_{21}, \dots, C_{2m_2} \end{aligned}$$

Term-acceptability of $M_P \cup \gamma_{ce}(P')$ implies that $A'_0 >_1 \text{solve}(A\theta)$ and that $A'_0 >_1 \text{solve}(B\theta\sigma)$, where A'_0 is obtained from A_0 by dropping all the arguments except for the first one, θ is $\text{mgu}(A_0, \text{solve}((A, B)))$ and σ is a computed answer substitution for $\leftarrow \text{solve}(A\theta)$. Then, by definition of $>$, for any t_{31}, \dots, t_{3n} and t_{41}, \dots, t_{4n} holds $A_0 > \text{solve}(A, t_{31}, \dots, t_{3n})\theta$ and $A_0 > \text{solve}(B, t_{41}, \dots, t_{4n})\theta\sigma$. Moreover, assumption that $\text{solve} \sqsupset c_{kl}$ for all k and l implies that those are the only recursive body subgoals in the clause.

- $A' \leftarrow B_1, \dots, B_s$ is

$$\begin{aligned} & \text{solve}(A, t_{51}, \dots, t_{5n}) \leftarrow \\ & \quad \text{clause}(A, B), \\ & \quad \text{solve}(B, t_{61}, \dots, t_{6n}) \\ & \quad C_{31}, \dots, C_{3m_3} \end{aligned}$$

Similarly to the previous case. ■

Usually, extended meta-interpreters do not preserve termination.

Example 5.

$$\begin{aligned} & \text{solve}(\text{empty}, 0) \\ & \text{solve}((A, B), N) \leftarrow \text{solve}(A, N), \text{solve}(B, N) \\ & \text{solve}(A, s(N)) \leftarrow \text{clause}(A, B), \text{solve}(B, N) \end{aligned}$$

This meta-interpreter mimics the LD-refutation with the bounded depth, provided by the second argument. Thus, even if the interpreted program might be non-terminating, the meta-program always terminates, after applying not more than a predefined number of rules.

Alternatively, the same meta-interpreter can be seen as evaluating the maximal depth needed for execution. In this case the meta-interpreter will preserve LD-termination. The mode of the query determines which of these is used. □

This example illustrates the importance of additional properties of $>$ that prove term-acceptability of $E_P \cup \gamma_{ce}(P')$ w.r.t. $\text{solve}(G, t_1, \dots, t_n)$. If for $>$ it holds that $\text{solve}(A, t_1, \dots, t_n) > \text{solve}(B, s_1, \dots, s_n)$ if and only if $\text{solve}(A, t'_1, \dots, t'_n) > \text{solve}(B, s'_1, \dots, s'_n)$ for any $t_1, \dots, t_n, s_1, \dots, s_n, t'_1, \dots, t'_n, s'_1, \dots, s'_n$ then the following theorem can be proved. Note that this situation is typical, if all argument positions in the calls to solve , except for the first one, are occupied by a linear sequence of free variables.

Theorem 4. *Let P' be an interpreted program and E_P be an extended meta-interpreter, such that $E_P \cup \gamma_{ce}(P')$ is term-acceptable w.r.t. $\text{solve}(G)$ via an order $>$, such that $\text{solve}(A, t_1, \dots, t_n) > \text{solve}(B, s_1, \dots, s_n)$ if and only if $\text{solve}(A, t'_1, \dots, t'_n) > \text{solve}(B, s'_1, \dots, s'_n)$, for any $t_1, \dots, t_n, s_1, \dots, s_n, t'_1, \dots, t'_n, s'_1, \dots, s'_n$. Then,*

1. E_P terminates w.r.t. $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\}$ and
2. P' terminates w.r.t. G

Proof. The first claim of the theorem is almost immediate, since term-acceptability w.r.t. $\text{solve}(G)$ means termination w.r.t. $\text{solve}(G)$ or w.r.t. $\text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G))$ and $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\} \subset \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G))$. Thus, $E_P \cup \gamma_{ce}(P')$ terminates w.r.t. $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\}$. However, by definition of E_P , none of the definitions of c_{kl} does not involve predicate clause. Thus, the encoding of P' is superfluous, i.e., E_P terminates w.r.t. $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\}$.

In order to prove the second claim, we are going to use Theorem 2, i.e., we first prove that vanilla meta-interpreter, extended by $\gamma_{ce}(P')$, terminates w.r.t. $\text{solve}(G)$ and then, Theorem 2, will imply the desired result. However, the first claim is almost immediate since the same order that proves term-acceptability of $E_P \cup \gamma_{ce}(P')$ w.r.t. $\text{solve}(G)$ satisfy conditions for term-acceptability of $M_P \cup \gamma_{ce}(P')$ w.r.t. $\text{solve}(G)$ (since $\text{solve}(A, t_1, \dots, t_n) > \text{solve}(B, s_1, \dots, s_n)$ implies $A > B$, i.e., $\text{solve}(A) > \text{solve}(B)$). ■

5.2 Double extended meta-interpreters

One might extend the framework even further, by considering the following class of meta-interpreters:

Definition 10. *A definite program of the following form will be called a double extended meta-interpreter*

$$\begin{aligned}
&\text{solve}(\text{true}, t_{11}, \dots, t_{1n}) \leftarrow C_{11}, \dots, C_{1m_1} \\
&\text{solve}((A, B), t_{21}, \dots, t_{2n}) \leftarrow \\
&\quad D_{11}, \dots, D_{1k_1}, \text{solve}(A, t_{31}, \dots, t_{3n}), \\
&\quad D_{21}, \dots, D_{2k_2}, \text{solve}(B, t_{41}, \dots, t_{4n}) \\
&\quad C_{21}, \dots, C_{2m_2} \\
&\text{solve}(A, t_{51}, \dots, t_{5n}) \leftarrow
\end{aligned}$$

$$\begin{aligned}
&D_{31}, \dots, D_{3k_3}, \text{ clause}(A, B, s_1, \dots, s_k), \\
&D_{41}, \dots, D_{4k_4}, \text{ solve}(B, t_{61}, \dots, t_{6n}) \\
&C_{31}, \dots, C_{3m_3}
\end{aligned}$$

where t_{ij} terms are extra arguments of *solve/2*, s_p terms are extra arguments of the *clause/2* and C_{kl} and D_{pq} are extra-conditions in the body, together with defining clauses for any other predicates occurring in the C_{kl} and D_{pq} (none of which contain *solve* or *clause*).

Almost all the meta-interpreters discussed above belong to this class of the meta-interpreters. Moreover, this framework also describes a depth tracking tracer for Prolog, a reasoner with threshold cutoff [26] and a pure four port box execution model tracer [8]. Note, that despite the similarity with Example 4 the later one is not a double extended meta-interpreter, and this is due to the call to predicate *clause* in the definition of *check*.

Definition 11. Let $H \leftarrow B_1, \dots, B_i, \dots, B_n$ be a clause, and let V be a variable in B_i . We'll say that $\leftarrow B_1, \dots, B_{i-1}$ is irrelevant for V if for every computed answer substitution σ for $\leftarrow B_1, \dots, B_{i-1}$, $V\sigma = V$.

We'll call a double extended meta-interpreter *restricted* if for any i , $\leftarrow D_{i1}, \dots, D_{ik_i}$ is irrelevant for meta-variables A and B . Using the same methodology as in the proof of Theorem 3 allows to establish the following result.

Theorem 5. Let P' be an interpreted program, E_P be the restricted double extended meta-interpreter, and $G \in B_{P'}^E$, such that P' is terminating w.r.t. G and E_P is terminating w.r.t. $\{A \mid A \in \text{Call}(E_P \cup \gamma_{ce}(P'), \text{solve}(G)), \text{solve} \sqsupset \text{rel}(A)\}$ then $E_P \cup \gamma_{ce}(P')$ terminates w.r.t. $\text{solve}(G)$.

Proof. Proof follows the general outline of the proof of Theorem 3, based on three orders— $>_1$ for atoms of *solve*, $>_2$ for atoms of c_{kl} 's and $>_3$ for atoms of d_{pq} 's.

However, one major difference should be pointed out. If the predicate of the call A_0 is *solve* and the clause $A' \leftarrow B_1, \dots, B_s$ that its head is unified with A_0 is either the second or the third clause of the meta-interpreter.

We analyse only the case of the second clause. The third clause can be analysed in the same way. $A' \leftarrow B_1, \dots, B_s$ is

$$\begin{aligned}
&\text{solve}((A, B), t_{21}, \dots, t_{2n}) \leftarrow \\
&D_{11}, \dots, D_{1k_1}, \text{ solve}(A, t_{31}, \dots, t_{3n}),
\end{aligned}$$

$$D_{21}, \dots, D_{2k_2}, \text{solve}(B, t_{41}, \dots, t_{4n})$$

$$C_{21}, \dots, C_{2m_2}$$

Term-acceptability of $M_P \cup \gamma_{ce}(P')$ implies that $A'_0 >_1 \text{solve}(A\theta)$ and that $A'_0 >_1 \text{solve}(B\theta\sigma)$, where A'_0 is obtained from A_0 by dropping all the arguments except for the first one, $\theta = \text{mgu}(A_0, \text{solve}((A, B)))$ and σ is a computed answer substitution for $\leftarrow \text{solve}(A\theta)$.

By irrelevance of $D_{11}, \dots, D_{1k_1}, D_{21}, \dots, D_{2k_2}$ for A and B , we can conclude by using the definition of $>$, for any t_{31}, \dots, t_{3n} and t_{41}, \dots, t_{4n} $A_0 > \text{solve}(A, t_{31}, \dots, t_{3n})\theta$ and $A_0 > \text{solve}(B, t_{41}, \dots, t_{4n})\theta\sigma$. Moreover, assumption that $\text{solve} \sqsupset c_{kl}$ for all k and l implies that those are the only recursive body subgoals in the clause. ■

In general, the second direction of the theorem does not necessary hold. Indeed, consider the following uncertainty reasoner with cutoff, motivated by [26].

Example 6.

$$\begin{array}{ll}
\text{solve}(\text{true}, 1, T), & \text{solve}(A, C, T) \leftarrow \\
\text{solve}((A, B), C, T) \leftarrow & \text{clause}(A, B, C1), \\
\text{solve}(A, C1, T), & C1 > T, T1 \text{ is } T/C1, \\
\text{solve}(B, C2, T), & \text{solve}(B, C2, T1), \\
\text{minimum}(C1, C2, C) & C \text{ is } C1 * C2.
\end{array}$$

Let P' be the following uncertainty-program: $\text{clause}(r, r, 0.9)$. When executed as a usual program, ignoring uncertainty coefficients, the goal $\leftarrow r$ does not terminate. However, for any specified threshold $\text{Threshold} > 0$ the goal $\leftarrow \text{solve}(r, \text{Certainty}, \text{Threshold})$ finitely fails. □

5.3 Towards normal programs

So far we have considered only definite programs and meta-interpreters that are tailored to this type of programs. However, negation appears frequently in meta-interpreters and thus, should be considered as an important issue for termination analysis.

In order to prove that meta-interpreters with negation preserve termination we use correctness and completeness and a termination analysis framework based on term-acceptability. However, this framework was originally developed for definite programs only. Thus, we start by extending it to programs containing negation, and later discuss termination properties of the meta-interpreter.

First of all, instead of LD-derivations and trees LDNF-derivations and trees should be considered. Recall, that LDNF-derivation *flounders* if there occurs in it or in any of its subsidiary LDNF-trees a goal with the first literal being non-ground and negative.

Definition 12. *The program P is called non-floundering w.r.t. a set of queries S , if all its LDNF-derivations starting in queries $G \in S$ are non-floundering.*

We will say that a goal G *LDNF-terminates* for a program P , if the LDNF-tree for (P, G) is finite.

We would like to apply the term-acceptability criterion to prove termination. However, the well-founded order $>$, that is required in Definition 2, enforces decrease only along the branches of the LD-trees. We also need to force the relation between the negative literal selected ($\neg A$) and the the root of the subsidiary tree, created for it (A). We require that $>$ is such that for any atom A , such that its predicate is called negatively in some clause, $\neg A \geq A$.

More formally, we use the notation of [3] and denote by Neg_P the set of predicates which occur in a negative literal in a body of a clause from P . We also denote $N_P^E = \{\neg A \mid A \in B_P^E \text{ and } rel(A) \in Neg_P\}$.

We refine the notion of term-acceptability w.r.t. a set.

Definition 13. *Let S be a set of atomic queries and P a general program. P is term-acceptable w.r.t. S if there exists a well-founded order $>$ on $B_P^E \cup N_P^E \cup U_P^E$, such that*

- for any $\neg A \in N_P^E$: $\neg A \geq A$
- and
 - for any $A \in Call(P, S)$
 - for any clause $A' \leftarrow B_1, \dots, B_n$ in P , such that $mgu(A, A') = \theta$ exists,
 - for any atom B_i , such that $rel(B_i) \simeq rel(A)$
 - for any computed answer substitution σ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$A > B_i\theta\sigma$$

Observe, that if P is a definite program, N_P^E is an empty set, the first condition of definition above holds vacuously, and we return to the notion of term-acceptability for definite programs, defined in Definition 2. Similarly to the definite case, the following theorem holds.

Theorem 6. *Let P a general program, and S a set of queries, such that P is non-floundering w.r.t. S . P is LDNF-terminating if and only if P is term-acceptable w.r.t. S .*

We illustrate the application of the theorem, by analysing the following example:

Example 7.

$$\begin{array}{ll}
\text{holds}(P, s0) \leftarrow & \text{initially}(\text{alive}). \\
& \text{initially}(P). \quad \text{initiates}(A, S, \text{loaded}) \leftarrow \\
\text{holds}(P, \text{result}(A, S)) \leftarrow & A = \text{load}. \\
& \text{initiates}(A, S, P). \quad \text{terminates}(A, S, \text{loaded}) \leftarrow \\
\text{holds}(P, \text{result}(A, S)) \leftarrow & A = \text{shoot}. \\
& \text{holds}(P, S), \quad \text{terminates}(A, S, \text{alive}) \leftarrow \\
& \neg \text{terminates}(A, S, P). \quad A = \text{shoot}, \text{holds}(\text{loaded}, S).
\end{array}$$

This example is a description of the famous Yale shouting problem in the framework of Situation Calculus. Usually, we are interested to check if after *loading* the gun, *waiting* for some time and, finally, *shooting*, the turkey is still *alive*. More generally, one can ask if some property holds after performing some sequence of actions. States resulting after performing some actions can be recursively defined as:

$$\text{state} = \begin{cases} s0 \\ \text{result}(\text{action}, \text{state}), \text{ where } \text{action} \text{ is one of } \text{load}, \text{wait}, \text{shoot} \end{cases}$$

That is $S = \{\text{holds}(t_1, t_2) \mid t_1 \text{ is a variable } t_2 \text{ is a state}\}$.

Theorem 6 imposes the following to hold (dependence on the model is omitted for the sake of simplicity) for any terms t_1, t_2, t_3 :

$$\begin{array}{l}
\neg \text{terminates}(t_1, t_2, t_3) \geq \text{terminates}(t_1, t_2, t_3) \\
\text{holds}(t_1, \text{result}(t_2, t_3)) > \text{holds}(t_1, t_3) \\
\text{holds}(t_1, \text{result}(t_2, t_3)) > \neg \text{terminates}(t_2, t_3, t_1) \\
\text{terminates}(t_1, t_2, \text{alive})\theta > \text{holds}(\text{loaded}, t_2)\theta
\end{array}$$

An order $>$ having the following properties satisfies the decreases above:

1. ignores \neg , i.e., assumes $A =_> \neg A$ for all A .
2. has a subterm property for the second argument of *result*
3. is monotone in the second argument of *holds*

4. for all $u, t_1, t_2, t_3, t_4, t_5$ holds:

$$\text{holds}(t_1, \text{result}(t_2, u)) > \text{terminates}(t_3, u, t_4) > \text{holds}(t_5, u) > \dots$$

□

Note, that this type of order is useful also for other applications of Situation Calculus, such as plan verification.

Now the methodology presented earlier is sufficient for analysis of meta-interpreters with negation and normal object programs.

Example 8. This example, V_P , is an immediate extension of vanilla meta-interpreter to normal programs [15]. Correctness and completeness are proved in Theorem 2.3.3 [16].

$$\begin{aligned} & \text{solve}(\text{true}). \\ & \text{solve}((\text{Atom}, \text{Atoms})) \leftarrow \text{solve}(\text{Atom}), \text{solve}(\text{Atoms}). \\ & \text{solve}(\neg \text{Atom}) \leftarrow \neg \text{solve}(\text{Atom}) \\ & \text{solve}(\text{Head}) \leftarrow \text{clause}(\text{Head}, \text{Body}), \text{solve}(\text{Body}). \end{aligned}$$

Theorem 7. *Let P' be a normal program, S be a set of queries. Then P' is LDNF-terminating w.r.t. S if and only if $V_P \cup \gamma_{ce}(P')$ LDNF-terminates w.r.t. $\{\text{solve}(G) \mid G \in S\}$.*

Proof. We start by proving that V_P improves termination. To prove the claim of the theorem it is sufficient to construct a well-founded order on $N_{V_P \cup \gamma_{ce}(P')}^E \cup U_{V_P \cup \gamma_{ce}(P')}^E$, such that $V_P \cup \gamma_{ce}(P')$ will be term-acceptable w.r.t. $\{\text{solve}(G) \mid G \in S\}$ via it. We define \succ based on $>$ —an order, such that P is term-acceptable via it w.r.t. S .

We define a new relation \succ^* as a transitive closure of:

1. $\text{solve}(A) \succ \text{solve}((B_1\theta, \dots, B_n\theta))$ if there is a clause $A' \leftarrow B_1, \dots, B_n$ and $\text{mgu}(A, A') = \theta$ exists.
2. $\text{solve}((B_i, \dots, B_n)) \succ \text{solve}((B_{i+1}\sigma, \dots, B_n\sigma))$ if σ is a computed answer substitution for $\leftarrow B_i$.
3. $\text{solve}((B_i, \dots, B_n)) \succ \text{solve}(B_i)$
4. $\text{solve}(\neg A) \succ \neg \text{solve}(A)$
5. $\neg \text{solve}(A) \succ \text{solve}(A)$

We have to prove that \succ^* is an order relationship, i.e., it is well-defined. Then we will prove that \succ^* is well-founded and that $V_P \cup \gamma_{ce}(P')$ is term-acceptable w.r.t. $\text{solve}(S)$ via \succ^* .

We start by proving a small lemma.

Lemma 2. *Let A and B be two atoms. If $\text{solve}(A) \succ^* \text{solve}(B)$ then $A > B$.*

Proof. If $\text{solve}(A) \succ^* \text{solve}(B)$ then exists sequence of goals G_1, \dots, G_n such that $\text{solve}(A) \succ G_1 \succ \dots \succ G_n = \text{solve}(B)$.

Let G_i be a first goal that is an application of *solve* to an atomic goal of the object program. Call this atomic goal G . We assumed that B is an atomic goal, thus, i as required always exists. Observe that G is obtained by *exactly one* application of case (1), followed by a (possibly empty) sequence of applications of case (2), followed by zero or one applications of case (3) or by zero or one application of case (4) and (5). Thus, G_1 is $\text{solve}((B_1\theta, \dots, B_n\theta))$, where $A' \leftarrow B_1, \dots, B_n$ is a clause, such that $\text{mgu}(A, A') = \theta$ exists, and for some $1 \leq j \leq n$ $B_j\theta\sigma$, where σ is a computed answer substitution for $\leftarrow B_1, \dots, B_{j-1}$ is either G or $\neg G$. Term-acceptability of P' w.r.t. S implies that $A > B_j\theta\sigma$. If $B_j\theta\sigma = G$, then $A > G$. Otherwise, $A > \neg G$, and since by Definition 13 $\neq G \geq G$, $A > G$ still holds. The proof is completed inductively.

We have to prove that \succ^* is well-defined, well-founded and that it provides a proof for term-acceptability of $V_P \cup \gamma_{ce}(P')$ w.r.t. $\text{solve}(S)$. For well-definedness observe, that if $\text{solve}(A) \succ^* \text{solve}(A)$ for an atom A , the lemma above implies $A > A$, i.e., contradiction to the fact that $>$ is well-defined. If $\text{solve}(\neg A) \succ^* \text{solve}(\neg A)$ holds for some atom A , then, exists G , such that $\text{solve}(\neg A) \succ G \succ^* \text{solve}(\neg A)$. Case (4) is the only case that is applicable, thus G is $\neg \text{solve}(A)$ and this implies that $\neg \text{solve}(A) \succ^* \neg \text{solve}(A)$. By reasoning in similar way we conclude that $\text{solve}(A) \succ^* \text{solve}(A)$ holds as well, contradicting the previous result. Alternatively, if $\text{solve}((B_1, \dots, B_n)) \succ^* \text{solve}((B_1, \dots, B_n))$ then either exists some atom A , such that $\text{solve}((B_1, \dots, B_n)) \succ^* \text{solve}(A) \succ^* \text{solve}((B_1, \dots, B_n))$ (cases (2) and (3) reduce number of subgoals, thus, at some point case (1) should be applied) or exists some atom A , such that $\text{solve}((B_1, \dots, B_n)) \succ^* \text{solve}(\neg A) \succ^* \text{solve}((B_1, \dots, B_n))$. Reasoning similar to above shows that $\text{solve}(A) \succ^* \text{solve}(A)$ holds, implying contradiction. Well-foundedness is proved analogously.

Now we are going to prove term-acceptability of the meta-program. Let $A \in \text{Call}(V_P \cup \gamma_{ce}(P'), \text{solve}(S))$. Then, one of the following holds:

- $\text{rel}(A) = \text{clause}$. Since there is no recursive clause, defining *clause* the claim follows.
- $A \in \text{solve}(\text{Call}(P', S))$, i.e., for some atom $G \in \text{Call}(P', S)$ $A = \text{solve}(G)$. Then, the only clause that its head is unifiable with A is the last clause of V_P . Let θ be $\text{mgu}(A, \text{solve}(\text{Head}))$. Then, the restriction

of θ to variables of G and $Head$ is also most general unifier of G and $Head$.

We have to prove, thus, that for every computed answer substitution σ for $\leftarrow clause(Head, Body)\theta$, $A \succ^* solve(Body)\theta\sigma$. If σ is a computed answer substitution for $\leftarrow clause(Head, Body)\theta$, then $Body\theta\sigma$ is a body of a clause in P' , such that its head is unifiable with $Head\theta$. Thus, it is unifiable with $G\theta$, and with G . By case (1) of definition of \succ^* , $solve(G) \succ^* solve(Body\theta\sigma)$, completing the proof.

- A is $solve(\neg G)$, for some $G \in Atom_{P'}^E$. The only clause that its head is unifiable with A is $solve(\neg Atom) \leftarrow \neg solve(Atom)$. Let θ be $mgu(A, solve(\neg Atom))$. Since there are no intermediate body atoms in the clause we just have to show that $A \succ^* \neg solve(Atom)\theta$. However, $Atom$ is a fresh variable, thus, $Atom\theta = G$. By case (4) of definition of \succ , holds that $solve(\neg G) \succ \neg solve(G)$, i.e., $A \succ \neg solve(Atom)\theta$. In particular, $A \succ^* \neg solve(Atom)\theta$.
- For some atoms $G_1, \dots, G_n \in Atom_{P'}^E$, $A = solve((G_1, \dots, G_n))$. There are two recursive clause of V_P that can be unified with A .
 - If the atom A is resolved with is the first recursive clause of V_P , and θ is mgu of A and $solve((Atom, Atoms))$ then $Atom\theta = G_1\theta$ and $Atoms\theta = (G_2, \dots, G_n)\theta$. Moreover, $G_1\theta = G_1$ and $(G_2, \dots, G_n)\theta = (G_2, \dots, G_n)$. Since there are two recursive subgoals we need to prove the following:
 - * $A \succ^* solve(Atom)\theta$, i.e., $solve((G_1, \dots, G_n)) \succ^* solve(G_1)$, which is true by case (3) of definition of \succ^* .
 - * For every computed answer substitution σ for $\leftarrow solve(Atom)\theta$, $A \succ^* solve(Atoms)\theta\sigma$. V_P is complete (Theorem 2.13, [18]), thus, σ is also a computed answer substitution for $\leftarrow Atom\theta$. Thus, by case (2) of definition of \succ^* the decrease holds.
 - If the clause A is resolved with last recursive clause of V_P the term-acceptability condition is satisfied vacuously, since by definition of γ_{ce} there is no fact of predicate $clause$ that has a non-atomic expression as their first argument.

This completes the proof of term-acceptability of $V_P \cup \gamma_{ce}(P')$ w.r.t. $solve(S)$. Thus, V_P improves termination.

However, we can also prove that V_P preserves termination. To prove this, we start with term-acceptability of $V_P \cup \gamma_{ce}(P')$ w.r.t. $solve(S)$ and we'll prove term-acceptability of P' w.r.t. S . Let $>$ be an order defined by term-acceptability of $V_P \cup \gamma_{ce}(P')$ w.r.t. $solve(S)$. Then we define a new order on atoms of P' as following: $A \succ B$ if $solve(A) > solve(B)$. This

order is defined on $\{G \mid \text{solve}(G) \in \text{Call}(V_P \cup \gamma_{ce}(P'), \text{solve}(S))\}$. Since the set of calls is preserved by a meta-interpreter [16], this set coincides with $\text{Call}(P', S)$. The order is well-defined and well-founded. The only thing we need to prove is that P' is term-acceptable w.r.t. S via this order.

First of all, term-acceptability of $V_P \cup \gamma_{ce}(P')$ w.r.t. $\text{solve}(S)$ implies that $\text{solve}(\neg A) > \neg \text{solve}(A)$, and that $\neg \text{solve}(A) \geq \text{solve}(A)$. Thus, $\text{solve}(\neg A) > \text{solve}(A)$, i.e., $\neg A \succ A$.

Second we have to prove that the decrease condition holds. Indeed, let $A \in \text{Call}(P', S)$ and let $A' \leftarrow B_1, \dots, B_n$ be a clause in P' , such that $\text{mgu}(A, A') = \theta$ exists. Then, θ is also $\text{mgu}(\text{solve}(A), \text{solve}(A'))$. The computed answer substitution σ for $\leftarrow \text{clause}(A\theta, B\theta)$ will map $B\theta$ to $(B_1, \dots, B_n)\theta$. Thus, by term-acceptability of $V_P \cup \gamma_{ce}(P')$ w.r.t. $\text{solve}(S)$, $\text{solve}(A) > \text{solve}((B_1, \dots, B_n)\theta)$. Similarly, hold $\text{solve}((B_1, \dots, B_n)\theta) > \text{solve}(B_1\theta)$ and $\text{solve}((B_1, \dots, B_n)\theta) > \text{solve}((B_2, \dots, B_n)\theta\sigma_1)$, where σ_1 is a computed answer substitution for $\text{solve}(B_1\theta)$. By proceeding in this way we conclude, that for any atom B_i , $\text{solve}(A) > \text{solve}(B_i\theta\sigma_1 \dots \sigma_{i-1})$, where σ_j is a computed answer substitution for $\leftarrow \text{solve}(B_j\theta\sigma_1 \dots \sigma_{j-1})$. Thus, we've proved that $A \succ B_i\theta\sigma_1 \dots \sigma_{i-1}$.

Correctness and completeness imply the term-acceptability and complete the proof. ■

5.4 Ground representation

In all the examples we've seen so far, the non-ground representation was taken. However, a number of papers discuss also the *ground* representation of object programs, and the corresponding meta-interpreters.

Example 9. The following example was borrowed from [15]

<pre> idemo(P, X, Y) ← instance_of(X, Y), idemo1(P, Y). idemo1(⊥, true). idemo1(P, and(X, Y)) ← idemo1(P, X), idemo1(P, Y). idemo1(P, not(X)) ← ¬idemo1(P, X). idemo1(P, atom(Q, Xs)) ← member(Z, P), instance_of(Z, if(atom(Q, Xs), B)), idemo1(P, B). inst_term(v(N), X, [], [bind(N, X)]). inst_term(v(N), X, [bind(N, X) S], [bind(N, X) S]). inst_term(v(N), X, [bind(M, Y) S], [bind(M, Y) S1]) ← N ≠ M, inst_term(v(N), X, S, S1). inst_term(term(F, Xs), term(F, Ys), S, S1) ← inst_args(Xs, Ys, S, S1). </pre>	<pre> instance_of(X, Y) ← inst_formula(X, Y, [], ⊥). inst_formula(atom(Q, Xs), atom(Q, Ys), S, S1) ← inst_args(Xs, Ys, S, S1). inst_formula(and(X, Y), and(Z, W), S, S2) ← inst_formula(X, Z, S, S1), inst_formula(Y, W, S1, S2). inst_formula(if(X, Y), if(Z, W), S, S2) ← inst_formula(X, Z, S, S1), inst_formula(Y, W, S1, S2). inst_formula(not(X), not(Z), S, S1) ← inst_formula(X, Z, S, S1). inst_formula(true, true, S, S). inst_args([], [], S, S). inst_args([X Xs], [Y Ys], S, S2) ← inst_term(X, Y, S, S1), inst_args(Xs, Ys, S1, S2). </pre>
---	---

Existing termination techniques, such as [19] are powerful enough to prove termination of calls to $instance_of(t, v)$, where t is a term, being a ground representation of a term, atom or clause and v is a variable that will be bounded to the non-ground representation of the same object. However, they are not powerful enough to analyse correctly this example, both due to imprecise representation of all possible ground terms, in particular all possible ground representations of programs, by the same abstraction and due to the nature of $idemo1$ as a meta-interpreter. We can easily see that the “troublesome” part of this example is very similar to meta-interpreter V_P we’ve discussed.

6 Proving termination automatically

Recall once more the “vanilla” meta-interpreter, studied in Example 1. Note that there is no linear norm that can prove that LD-termination is improved. Indeed, let P be the following program:

$$\begin{aligned} l(X) &\leftarrow p(X), r(X). \\ p(X) &\leftarrow q(X, Y), p(Y). \\ r(f(X)) &\leftarrow s(Y), r(X). \\ q(f(Z), Z) &. \\ p(0).r(0).s(0). \end{aligned}$$

This program clearly terminates for $\leftarrow l(t)$ for every ground term t . Thus, $\text{solve}(l(t))$ terminates for every ground t as well. However, while trying use some linear norm $\|\cdot\|$ and to prove acceptability w.r.t. it among others following constraints are obtained:

$$\begin{aligned} \|\text{solve}(p(X))\| &> \|\text{solve}((q(X, Y), p(Y)))\| \\ \|\text{solve}(r(f(X)))\| &> \|\text{solve}((s(Y), r(X)))\| \\ \|\text{solve}((s(Y), r(X)))\| &> \|\text{solve}(r(X))\| \end{aligned}$$

Note, that in general, the last constraint should take in the consideration the intermediate body atom $s(Y)$ as well, but one can prove that it cannot affect $r(X)$.

Recalling the assumption of linearity of the norm, i.e., that $\|k(t_1, \dots, t_n)\| = c^k + \sum_{i=1}^n a_i^k \|t_i\|$ and for all k and for all i , $c^k \in \mathcal{N}_0$ and $a_i^k \in \mathcal{N}_0$ those can be reduced to the following (without loss of generality, $c^{\text{solve}} = 0$ and $a_1^{\text{solve}} = 1$). For functors having only one argument the subscript is dropped.

$$\begin{aligned} c^p + a^p \|X\| &> c^q + a_1^q c^q + a_1^q a_1^q \|X\| + a_1^q a_2^q \|Y\| + a_2^q c^p + a_2^q a^p \|Y\| \\ c^r + a^r c^f + a^r a^f \|X\| &> c^s + a_1^s c^s + a_1^s a^s \|Y\| + a_2^s c^r + a_2^s a^r \|X\| \\ c + a_1^s c^s + a_1^s a^s \|Y\| + a_2^s c^r + a_2^s a^r \|X\| &> c^r + a^r \|X\| \end{aligned}$$

Further reduction gives among, others, the following inequalities:

$$c^p > c + a_1 c^q + a_2 c^p \tag{1}$$

$$c^r + a^r c^f > c + a_1 c^s + a_2 c^r \tag{2}$$

$$c + a_1 c^s + a_2 c^r > c^r \tag{3}$$

$$a_2 a^r \geq a^r \tag{4}$$

(1) implies that $a_2 = 0$. Thus, $a^r = 0$ as well (4). However, (2) and (3) imply that $a^r c^f > 0$, which provides the desired contradiction.

Note that if the restriction of linearity is relaxed, termination, we proved using orders, implies existence of the norm, that also can be used for proving termination. However, this norm might be difficult or even impossible to generate automatically.

Note, that the trouble in this case arose from the fact that substitution needed for the intermediate body atom was “encapsulated” inside the one of the subgoals. To solve this we use the well-known technique of unfolding [7].

Unfortunately, as the following example shows, non-restricted unfolding can be non-terminating.

Example 10. Let M_P be the “vanilla” meta-interpreter and P' be the following

$$p(X) \leftarrow p(f(X))$$

Then, trying to unfold $\text{solve}(p(f(X)))$ will cause an attempt to unfold $\text{solve}(p(f(f(X))))$ and so on. \square

Thus, we perform the following transformation.

1. Unfold the predicate *clause*.
2. Repeatedly unfold the calls to $\text{solve}((A, B))$, until there are no subgoals of this form.

Observe, that by repeated unfolding $\text{solve}((B_1, \dots, B_n))$, in the body of $\text{solve}(A) \leftarrow \text{solve}((B_1, \dots, B_n))$, one obtains $\text{solve}(A) \leftarrow \text{solve}(B_1), \dots, \text{solve}(B_n)$.

This unfolding will always terminate, since a number of predicates and functors both in the meta-interpreter and in the interpreted program is finite. However, as shown in [7], in general, unfolding can improve termination, i.e., replace infinite branches by the failing ones. However, in our case, termination is not just improved, but preserved.

Lemma 3. *Let T be LD-tree of Q and P , let T' be LD-tree of Q and P' , where P' is obtained by unfolding clause. T is finite if and only if T' is finite.*

Proof. \Rightarrow Corollary 13 of [7].

\Leftarrow For each node N in T' with a query $\text{solve}(A)$ for some $A \in B_{P'}^E$, add between N and its children N_1, \dots, N_k new nodes M_1, \dots, M_k with a query $\text{clause}(A, V_i), \text{solve}(V_i)$, where V_i is a new fresh variable. In this way we reconstructed T from T' . Clearly, if T' is finite, then T is finite as well. ■

Lemma 4. *Let M_P be the “vanilla” meta-interpreter, P' be an interpreted program, $P = M_P \cup \gamma_{ce}(P')$, P' obtained from P by unfolding predicate clause and P'' is obtained from P' by repeated unfolding of $\text{solve}((A, B))$ as described above. Then, for every query $\text{solve}(Q)$, $\text{solve}(Q)$ terminates w.r.t. P' if and only if it terminates w.r.t. P'' .*

Proof. \Rightarrow Corollary 13 of [7].

\Leftarrow If $\text{solve}(Q)$ is non-terminating w.r.t. P' , then there are infinitely many calls to goals solve . We distinguish between the following cases:

- there is an infinite branch in the LD-tree of Q and P' , with finitely many calls to $\text{solve}((A, B))$. Then, there is a goal G , such that in the sub-LD-tree, rooted at G there are no calls to $\text{solve}((A, B))$ at all. Transformation described does not affect this sub-LD-tree, thus, the LD-tree of Q w.r.t. P'' is infinite as well.
- for all infinite branches in the LD-tree, there are infinitely many calls to $\text{solve}((B_1, \dots, B_n))$ on those branches.

The only clause in P' that can be resolved with this goal is $\text{solve}((A, B)) \leftarrow \text{solve}(A), \text{solve}(B)$. That is first A is resolved, and then further goals encapsulated in B . This behaviour is specified by the leftmost-selection rule used in the LD-tree for Q and P'' . Thus, there is correspondence between $\leftarrow \text{solve}((B_1, \dots, B_n))$ in the LD-tree for Q and P' and $\leftarrow \text{solve}(B_1), \dots, \text{solve}(B_n)$ in the LD-tree for Q and P'' , proving the lemma. ■

We summarise this to lemmas in the following theorem:

Theorem 8. *Let P be a meta-program, and let P'' be a program, obtained from P by unfolding predicate clause and repeated unfolding of $\text{solve}((A, B))$. Then for every Q , Q terminates w.r.t. P if and only if Q terminates w.r.t. P'' .*

Note, that termination analysis of P'' can be performed by existing methods. For example, if level mapping $|\cdot|$ and norm $\|\cdot\|$ are sufficient for proving acceptability of P' , $|\cdot|_{P''}$ defined as $|\text{solve}(X)|_{P''} = \|X\|$ can be used for proving acceptability of P'' . Alternatively, term-acceptability condition might be used.

7 Conclusion

We have presented a methodology for proving termination properties of meta-programs. The problem of termination was studied by a number of authors (see [12] for the survey, and more recent work can be found in [13, 14, 17, 19, 23, 25, 27, 29]).

Our methodology gains its power from using the integrated approach [24] that extends the traditional notion of acceptability [2] with the wide class of term-orderings that have been studied in the context of the term-rewriting systems. In this work we have shown that this approach allows to define relatively simple relation between the order that satisfies the requirements of term-acceptability for an object program and for the meta-interpreter extended by this object program and a corresponding set of goals. Thus, the methodology is useful to establish if the meta-interpreter improves or preserves termination. In particular, in the work on compositionality of termination proofs [4], level mappings for an object program cannot easily be reused for the meta-program.

Despite the intensive research on meta-programming inside the logic programming community [5, 18, 21] termination behaviour of meta-programs attracted less attention. Pedreschi and Ruggieri [22] use generic verification method, based on specifying preconditions and postconditions. Unfortunately, their termination results are restricted only to the “vanilla” meta-interpreter. It is not immediate how their results can be extended to alternative meta-interpreters, nor if the relationship between termination characterisation of the object program and the meta-program can be established.

We consider as a future work identifying additional classes of meta-interpreters, such as [10, 11, 28] and studying their termination behaviour.

8 Acknowledgement

Alexander Serebrenik is supported by GOA: “ LP^+ : a second generation logic programming language”.

References

1. K. R. Apt. *From Logic Programming to Prolog*. Prentice-Hall Int. Series in Computer Science. Prentice Hall, 1997.
2. K. R. Apt and D. Pedreschi. Studies in Pure Prolog: Termination. In J. W. Lloyd, editor, *Proc. Esprit Symp. on Comp. Logic*, pages 150–176. Springer Verlag, 1990.
3. K. R. Apt and D. Pedreschi. Proving termination of general Prolog programs. In T. Ito and A. Meyer, editors, *Theoretical Aspects of Computer Software, Int. Conf. TACS'91*, pages 265–289. Springer Verlag, 1991. LNCS 526.
4. K. R. Apt and D. Pedreschi. Modular termination proofs for logic and pure prolog programs. In G. Levi, editor, *Advances in Logic Programming Theory*, pages 183–229. Oxford University Press, 1994.
5. K. R. Apt and F. Turini, editors. *Meta-Logics and Logic Programming*. Logic Programming. The MIT Press, 1995.
6. T. Arts. *Automatically proving termination and innermost normalisation of term rewriting systems*. PhD thesis, Universiteit Utrecht, 1997.
7. A. Bossi and N. Cocco. Preserving universal termination through unfold/fold. In G. Levi and M. Rodríguez-Artalejo, editors, *Algebraic and Logic Programming*, pages 269–286. Springer Verlag, 1994. LNCS 850.
8. A. Bowles and P. Wilk. Tracing requirements for multi-layered meta-programming. In H. Abramson and M. H. Rogers, editors, *Meta-Programming in Logic Programming*, pages 205–216. The MIT Press, 1989.
9. A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini. Composition operators for logic theories. In J. W. Lloyd, editor, *Proc. Esprit Symp. on Comp. Logic*, pages 117–134. Springer Verlag, 1990.
10. M. Bruynooghe, D. De Schreye, and B. Martens. A general criterion for avoiding infinite unfolding during partial deduction. *New Generation Computing*, 11(1):47–79, 1992.
11. M. H. Cheng, M. H. van Emden, and P. A. Strooper. Complete sets of frontiers in logic-based program transformation. In H. Abramson and M. H. Rogers, editors, *Meta-Programming in Logic Programming*, pages 283–297. The MIT Press, 1989.
12. D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *J. Logic Programming*, 19/20:199–260, May/July 1994.
13. S. Decorte and D. De Schreye. Termination analysis: some practical properties of the norm and level mapping space. In J. Jaffar, editor, *Proc. of the 1998 Joint Int. Conf. and Symp. on Logic Programming*, pages 235–249. MIT Press, June 1998.
14. S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based termination analysis of logic programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(6):1137–1195, November 1999.
15. P. Hill and J. Gallagher. Meta-programming in logic programming. In D. M. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of logic in Artificial Intelligence and Logic Programming*, pages 421–498. Clarendon press, 1998. volume 5. Logic Programming.
16. P. Hill and J. Lloyd. Analysis of meta-programs. In H. Abramson and M. H. Rogers, editors, *Meta-Programming in Logic Programming*, pages 23–52. The MIT Press, 1989.
17. S. Hoarau. *Inférer et compiler la terminaison des programmes logiques avec contraintes*. PhD thesis, Université de La Réunion, 1999.
18. G. Levi and D. Ramundo. A formalization of metaprogramming for real. In D. S. Warren, editor, *Logic Programming, Proceedings of the Tenth International Conference on Logic Programming*, pages 354–373. MIT Press, 1993.

19. N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In L. Naish, editor, *Proc. of the Fourteenth Int. Conf. on Logic Programming*, pages 63–77. MIT Press, July 1997.
20. B. Martens and D. De Schreye. Two semantics for definite meta-programs, using the non-ground representation. In K. Apt and F. Turini, editors, *Meta-Logics and Logic Programming*, pages 57–81. MIT Press, Cambridge, MA, 1995. ISBN: 0-262-01152-2.
21. B. Martens and D. De Schreye. Why untyped nonground metaprogramming is not (much of) a problem. *Journal of Logic Programming*, 22(1):47–99, January 1995.
22. D. Pedreschi and S. Ruggieri. Verification of meta-interpreters. *Journal of Logic and Computation*, 7(2):267–303, November 1997.
23. S. Ruggieri. *Verification and validation of logic programs*. PhD thesis, Università di Pisa, 1999.
24. A. Serebrenik and D. De Schreye. Non-transformational termination analysis of logic programs, based on general term-orderings. In K.-K. Lau, editor, *Pre-Proceedings of Tenth International Workshop on Logic-based Program Synthesis and Transformation, 2000*, pages 45–54. University of Manchester, 2000. Technical Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-00-6-1, URL : <http://www.cs.man.ac.uk/cstechrep/titles00.html>.
25. J.-G. Smaus. *Modes and Types in Logic Programming*. PhD thesis, University of Kent, 1999.
26. L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, 1994.
27. C. Taboch. A semantic basis for termination analysis of logic programs. Master's thesis, Ben-Gurion University of the Negev, 1998.
28. F. van Harmelen. A classification of meta-level architectures. In H. Abramson and M. H. Rogers, editors, *Meta-Programming in Logic Programming*, pages 103–122. The MIT Press, 1989.
29. S. Verbaeten. *Static verification of compositionality and termination for logic programming languages*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, June 2000. v+265+xxvii.

9 Appendix A—Proof of the Calls Preservation Lemma

Lemma 1 *Let P' be an interpreted program, M_P be the vanilla meta-interpreter and $G \in B_{P'}^E$, then*

$$\{solve(A) \mid A \in Call(P', G)\} \equiv Call(M_P \cup \gamma_{ce}(P'), solve(G)) \cap \{solve(A) \mid A \in B_{P'}^E\}$$

where \equiv means equality up to variable renaming.

Proof. We prove the set-equality by proving containment in both directions.

\subseteq Clearly, $Call(P', G) \subseteq B_{P'}^E$. Thus, $\{solve(A) \mid A \in Call(P', G)\} \subseteq \{solve(A) \mid A \in B_{P'}^E\}$. To prove the inclusion we need, therefore, to

prove that $\{solve(A) \mid A \in Call(P', G)\} \subseteq Call(M_P \cup \gamma_{ce}(P'), solve(G))$. To prove this we prove that every element of the right-hand side set is also an element of the left-hand side set.

Let $K \in \{solve(A) \mid A \in Call(P', G)\}$. That is, $K = solve(K')$ for some $K' \in Call(P', G)$. The proof is inductive and based on the derivation of K' . Note, that we need to prove stronger claim than we actually need: we prove not only the membership claimed, but also that for every goal G_0 in the LD-tree of P' and G there is a goal in $M_P \cup \gamma_{ce}(P')$ and $solve(G)$ such that its arguments form a partition of G_0 up to variable renaming.

- *Induction base* $K' = G$ and $K = solve(G)$, implying $K \in Call(M_P \cup \gamma_{ce}(P'), solve(G))$. The second part of the claim is also immediate.
- *Inductive assumption* Assume that for some goal $\leftarrow G_1, \dots, G_k$ in the LD-tree of P' and G , $solve(G_1) \in Call(M_P \cup \gamma_{ce}(P'), solve(G))$ and there exists a goal G_0 in the LD-tree of $M_P \cup \gamma_{ce}(P')$ and $solve(G)$, such that G_0 is

$$\leftarrow solve((G_{1,1}, \dots, G_{1,n_1})), \dots, solve((G_{m,1}, \dots, G_{m,n_m}))$$

and $(G_{1,1}, \dots, G_{1,n_1}), \dots, (G_{m,1}, \dots, G_{m,n_m})$ forms a partition of G_1, \dots, G_k up to variable renaming.

- *Inductive step* We need to prove two claims.
 - * First, let $H \leftarrow H_1, \dots, H_l$ be a clause in P' , and $H' \leftarrow H'_1, \dots, H'_l$ be its renamed apart version. Then, the goal $\leftarrow (H'_1, \dots, H'_l, G_2, \dots, G_k)\theta$, is a resolvent of the previous goal and a clause, where $\theta = mgu(G_1, H)$. We have to prove that $solve(H'_1\theta) \in Call(M_P \cup \gamma_{ce}(P'), solve(G))$. We also have to prove that there is a goal, such that its arguments form a partition of $\leftarrow (H'_1, \dots, H'_l, G_2, \dots, G_k)\theta$ up to variable renaming.
 - * Second, we prove $solve(G_2\sigma) \in Call(M_P \cup \gamma_{ce}(P'), solve(G))$, where σ is a computed answer substitution for G_1 . We also have to prove that there is a goal, such that its arguments form a partition of $solve(G_2\sigma, \dots, G_k\sigma)$ up to variable renaming.

We prove these claims.

- * By inductive assumption,

$$solve(G_1) \in Call(M_P \cup \gamma_{ce}(P'), solve(G))$$

The only clause of M_P that is applicable is

$$solve(Head) \leftarrow clause(Head, Body), solve(Body).$$

Thus, the descendant of a goal with the selected atom $solve(G_1)$, will be $\leftarrow clause(Head\tau, Body\tau), solve(Body\tau), G_2\tau, \dots, G_k\tau$, where $\tau = mgu(solve(G_1), Head)$. However, $Head$ is a variable. Thus, $Head\tau = G_1\tau = G_1$. Moreover, by the definition of τ , $Body\tau$ remains a free variable $Body$. Similarly, observe, that τ does not bind G_2, \dots, G_k .

Observe, that by definition of γ_{ce} , $\gamma_{ce}((H \leftarrow H_1, \dots, H_l)) \in \gamma_{ce}(P')$. Thus, $clause(G_1, Body)$, can be unified with a renamed apart instance of $\gamma_{ce}((H \leftarrow H_1, \dots, H_l))$. Call this instance $clause(H'', (H''_1, \dots, H''_l))$. Then, the unification can be done via variably-renamed θ combined with the mapping of $Body$ to properly instantiated H''_1, \dots, H''_l , say θ' . Thus, $H'_i\theta$ and $H''_i\theta'$ are identical up to variable renaming for all i .

Thus, the next call in the LD-tree of $M_P \cup \gamma_{ce}(P')$ and $solve(G)$ is $\leftarrow solve((H''_1\theta', \dots, H''_l\theta'), G_2, \dots, G_k)$. By the remark on θ' arguments of this goal form a partition of $\leftarrow H'_1\theta, \dots, H'_l\theta, G_2\theta, \dots, G_k\theta$.

Selected atom of this goal is $solve((H''_1\theta', \dots, H''_l\theta'))$. We distinguish between the following cases:

- $m = 1$. In this case the selected atom is $solve(H''_1\theta')$. Thus, there is a variant of $solve(H'_1\theta)$ that is contained in the set of calls of $M_P \cup \gamma_{ce}(P')$ and $solve(G)$, completing the proof.
- $m > 1$. In this case both recursive clauses of M_P can be unified with the selected atom. If the same clause of M_P as above is selected then the attempt to resolve a goal $clause((H''_1\theta', \dots, H''_l\theta'), Body')$ will fail, since by definition of γ_{ce} , the first argument position of $clause/2$ is always occupied by an atom of $B_{P'}^E$ and $,/2$ is assumed not to be a predicate of P' . Otherwise, the clause

$$solve((Atom, Atoms)) \leftarrow solve(Atom), solve(Atoms).$$

is used. $Atom$ is a fresh variable. Thus, after the unification via an mgu δ , $Atom\delta = H''_1\theta'\delta$, i.e., $H''_1\theta'$. That is the next atom that will be selected is $solve(Atom\delta)$, i.e., $solve(H''_1\theta')$ and once more we conclude by claiming that there is a variant of $solve(H'_1\theta)$ that is contained in the set of calls of $M_P \cup \gamma_{ce}(P')$ and $solve(G)$.

- * Now we prove that $solve(G_2\sigma) \in Call(M_P \cup \gamma_{ce}(P'), solve(G))$, where σ is a computed answer substitution for G_1 . By inductive assumption there is a goal in the LD-tree of $M_P \cup \gamma_{ce}(P')$ and

$solve(G)$, say $\leftarrow solve((G_{1,1}, \dots, G_{1,n_1}), \dots, solve((G_{m,1}, \dots, G_{m,n_m}))$ such that $(G_{1,1}, \dots, G_{1,n_1}), \dots, (G_{m,1}, \dots, G_{m,n_m})$ forms a partition of G_1, \dots, G_k up to variable renaming.

We distinguish between the following cases.

- $n_1 = 1$. By definition of partition $G_{1,1} = G_1$ (up to variable renaming). By completeness and correctness result of Levi and Ramundo [18], σ is also obtained as a computed answer substitution for $\leftarrow solve(G_1)$. Thus, the next atom that will be selected is $solve((G_{2,1}, \dots, G_{2,n_2}))\sigma$. Sequence of sequences $(G_{2,1}, \dots, G_{2,n_2})\sigma, \dots, (G_{m,1}, \dots, G_{m,n_m})\sigma$ forms a partition of $G_2\sigma, \dots, G_k\sigma$ up to variable renaming and $G_{1,2} = G_2$ (up to variable renaming). Reasoning similar to above shows that $solve(G_2)$ becomes selected either immediately ($n_2 = 1$) or after additional application of the second clause of M_P .
- $n_1 > 1$. The only clause that can be applied to this goal is the second clause of M_P . Goals that originate from it are $\leftarrow solve((G_{1,2}, \dots, G_{1,n_1}))\sigma', \dots, solve((G_{m,1}, \dots, G_{m,n_m}))\sigma'$, and $\leftarrow solve(G_{1,1})$, where σ' is a computed answer substitution for $\leftarrow solve(G_{1,1})$.

By definition of partition $G_{1,1} = G_1$ (up to variable renaming). By completeness and correctness result of Levi and Ramundo [18], σ' is also a computed answer substitution for G_1 and σ is a computed answer substitution for $\leftarrow solve(G_1)$. Thus, we can identify σ and σ' , and the second goal can be written as

$$\leftarrow solve((G_{1,2}, \dots, G_{1,n_1}))\sigma, \dots, solve((G_{m,1}, \dots, G_{m,n_m}))\sigma$$

Arguments of this goal form a partition of $G_2\sigma, \dots, G_k\sigma$ up to variable renaming and that $G_{1,2} = G_2$ (up to variable renaming).

By reasoning similar to above one can see that $solve(G_2)$ becomes selected either immediately ($n_1 = 2$) or after additional application of the second clause of M_P .

\supseteq Now we are going to prove that

$$\{solve(A) \mid A \in Call(P', G)\} \supseteq Call(M_P \cup \gamma_{ce}(P'), solve(G)) \cap \{solve(A) \mid A \in B_{P'}^E\}$$

Let $K \in Call(M_P \cup \gamma_{ce}(P'), solve(G)) \cap \{solve(A) \mid A \in B_{P'}^E\}$. Then, the $K = solve(K')$ for some $K' \in B_{P'}^E$. We need to show that $K' \in Call(P', G)$.

As earlier, we are going to prove this claim inductively and, similarly, we need to prove more strong claim than we actually need. We are going to prove that for every goal

$$\leftarrow \text{solve}((G_{1,1}, \dots, G_{1,n_1}), \dots, \text{solve}((G_{m,1}, \dots, G_{m,n_m})))$$

in the LD-tree of $M_P \cup \gamma_{ce}(P')$ and $\text{solve}(G)$, there is a goal $\leftarrow G_1, \dots, G_k$ in the LD-tree of P' and G , such that the sequence of sequences $((G_{1,1}, \dots, G_{1,n_1}), \dots, (G_{m,1}, \dots, G_{m,n_m}))$ forms a partition of G_1, \dots, G_k and that $G_{1,1} \in \text{Call}(P', G)$.

- *Induction base* If $K = \text{solve}(G)$, then $K' = G$, and $K' \in \text{Call}(P', G)$. The second claim is immediate as well.

- *Inductive step* As above we need to prove two claims.

* Let $\leftarrow \text{solve}((G_{1,1}, \dots, G_{1,n_1}), \dots, \text{solve}((G_{m,1}, \dots, G_{m,n_m})))$ be a goal in the LD-tree of $M_P \cup \gamma_{ce}(P')$ and $\text{solve}(G)$. We distinguish between the following cases:

- $n_1 = 1$. Then, the only clause that can be applied to the goal is the third clause of M_P , i.e.,

$$\text{solve}(\text{Head}) \leftarrow \text{clause}(\text{Head}, \text{Body}), \text{solve}(\text{Body}).$$

Similarly, to the reasoning in the proving the other direction of containment, observe that Head is a variable, thus, if θ is a mgu of $\text{solve}(\text{Head})$ and $\text{solve}(G_{1,1})$, then $\text{solve}(G_{1,1})\theta = \text{solve}(G_{1,1})$.

Let τ be a computed answer substitution for $\text{clause}(G_{1,1}, \text{Body})$.

That is there is a fact $\text{clause}(H, (H_1, \dots, H_l))$, such that $H\tau = G_{1,1}\tau$ and $\text{Body}\tau = (H_1, \dots, H_l)\tau$. Thus, the resolvent is $\leftarrow \text{solve}((H_1, \dots, H_l)\tau), \dots, \text{solve}((G_{m,1}, \dots, G_{m,n_m}))\tau$.

By our assumption, there is $\leftarrow G_1, \dots, G_k$ in the LD-tree of P' and G , such that $(G_{1,1}, \dots, G_{1,n_1}), \dots, (G_{m,1}, \dots, G_{m,n_m})$ form a partition of G_1, \dots, G_k . Thus, $G_1 = G_{1,1}$ (up to variable renaming).

Since $\text{clause}(H, (H_1, \dots, H_l)) \in \gamma_{ce}(P')$, $(H \leftarrow H_1, \dots, H_l) \in P'$. Moreover, G_1 and H are unifiable via τ , and τ restricted to variables of G_1 and H , is the most general unifier for them. Thus, the next goal in the LD-tree of P' and G is the resolvent, i.e., $\leftarrow H_1\tau, \dots, H_l\tau, G_2\tau, \dots, G_k\tau$. This proves that there is a goal in the LD-tree of P' and G such that a sequence of sequences $(H_1\tau, \dots, H_l\tau), \dots, (G_{m,1}\tau, \dots, G_{m,n_m}\tau)$ forms a partition of it. Selected atom of $\leftarrow H_1\tau, \dots, H_l\tau, G_2\tau, \dots, G_k\tau$

is $H_1\tau$. Thus, $H_1\tau \in \text{Call}(P', G)$, i.e., the first argument of the first subgoal in the goal $\leftarrow \text{solve}((H_1, \dots, H_l)\tau), \dots, \text{solve}((G_{m,1}, \dots, G_{m,n_m}))\tau$ appears in $\text{Call}(P', G)$.

- $n_1 > 1$. In this case the only clause that is applicable is the second clause of M_P . After application of the clause the case is reduced to the previous one.

* We prove that if

$$\leftarrow \text{solve}((G_{1,1}, \dots, G_{1,n_1})), \dots, \text{solve}((G_{m,1}, \dots, G_{m,n_m}))$$

has the properties specified above, then the following goal: $\leftarrow \text{solve}((G_{1,2}, \dots, G_{1,n_1}))\sigma, \dots, \text{solve}((G_{m,1}, \dots, G_{m,n_m}))\sigma$ has the same properties as well (σ is a computed answer substitution of $\text{solve}(G_{1,1})$).

First of all, as in the previous proof we can assume $n_1 = 1$, since the second case can be reduced to this one after one application of the second clause of M_P .

By our assumption, there is a goal $\leftarrow G_1, \dots, G_k$ in the LD-tree of P' and G , such that $G_{1,1}, (G_{2,1}, \dots, G_{2,n_2}), \dots, (G_{m,1}, \dots, G_{m,n_m})$ forms its partition and $G_{1,1} = G_1$ (up to variable renaming).

By correctness and completeness result of Levi and Ramundo [18] σ is also computed answer substitution of G_1 . Thus, the goal $\leftarrow G_2\sigma, \dots, G_k\sigma$ is a goal in the LD-tree of P' and G , such that $(G_{2,1}, \dots, G_{2,n_2})\sigma, \dots, (G_{m,1}, \dots, G_{m,n_m})\sigma$ forms its partition and $G_{2,1}\sigma = G_2\sigma$. Since $G_2\sigma$ is selected atom in $\leftarrow G_2\sigma, \dots, G_k\sigma$, we claim that $G_{2,1}\sigma \in \text{Call}(P', G)$.