Termination of Logic Programs Using Various Dynamic Selection Rules*

Jan–Georg Smaus

Institut für Informatik
Universität Freiburg
Georges-Köhler-Allee 52, 79110 Freiburg im Breisgau, Germany
smaus@informatik.uni-freiburg.de

15th September 2004

Technical Report No. 203

Abstract

We study termination of logic programs with dynamic scheduling, as it can be realised using delay declarations. Following previous work, our minimum assumption is that derivations are *input-consuming*, a notion introduced to define dynamic scheduling in an abstract way. Since this minimum assumption is sometimes insufficient to ensure termination, we consider here various *additional* assumptions on the permissible derivations. In one dimension, we consider derivations parametrised by any property that the selected atoms must have, for example being ground in the input positions. In another dimension, we consider both *local* and non-local derivations. In all cases, we give sufficient and necessary criteria for termination. The dimensions can freely be combined, yielding the most comprehensive approach so far to termination of logic programs with dynamic scheduling.

1 Introduction

Termination of logic programs has been widely studied for the LD selection rule, that is, derivations where the leftmost atom in a query is selected [1, 4, 8, 9, 10, 16, 17]. This rule is adequate for many applications, but there are situations, for example in the context of parallel executions or the test-and-generate paradigm, that require *dynamic scheduling*, that is, some mechanism to determine at runtime which atom is selected. Dynamic scheduling can be realised by *delay declarations* [12, 24], specifying that an atom must be instantiated to a certain degree to be selected.

Termination of logic programs with dynamic scheduling has been studied for about a decade [3, 6, 7, 13, 14, 15, 18, 20, 23]; the research started with observations of some surprisingly complex (non-)termination behaviour of simple programs such as `APPEND` or `PERMUTE` with delay declarations [18]. In our own research [7, 20, 23], we found that *modes* (input and output), while arguably compromising the "pure logical" view of logic programming, are the key to understanding this behaviour and achieving or verifying termination. We have proposed *input-consuming derivations* (where in each resolution step, the input arguments of the selected atom do not become instantiated) as a reasonable minimum assumption about the selection rule, which abstracts away from the technical details of delay declarations.

In this report, we study termination of logic programs for input-consuming derivations with various *additional* assumptions about the selection rule, for example saying that the selected

*This technical report is the extended version of [22] and contains the proofs missing there for space reasons. The results in Subsec. 6.3 are new.

atom must be ground in its input positions. Some authors have considered termination under such strong assumptions [13, 14, 15, 18, 23], partly because certain programs do not terminate for input-consuming derivations, but also because termination for input-consuming derivations is so hard to show: termination proofs usually use *level mappings*, which measure the size of an atom; an atom is *bounded* if its level is invariant under instantiation. Now, the usual reasoning that there is a decrease in level mapping between a clause head and the clause body atoms does not readily apply to derivations where selected atoms are not necessarily bounded.

After intensive studies of the semantics [6], and restricting to a class of programs that is well-behaved with respect to the modes, we now have a sufficient and necessary criterion for termination of input-consuming derivations [7]. The key concept of that approach is a special notion of *model*, bottom-up computable by a variant of the well-known T_P-operator. The notion reflects the answer substitutions computed by input-consuming derivations. We build on this work here.

We consider additional assumptions in two dimensions. *Certain* additional assumptions about derivations, for example the one above, can be formulated in terms of the selected atoms alone. We do this abstractly by saying that each selected atom must have a property $\mathcal{P}$. There are some natural conditions on $\mathcal{P}$ mentioned later. It turns out that the approach of [7] can be easily adapted to give a *sufficient* criterion for termination of $\mathcal{P}$-derivations [21]. In this report we give a criterion that is also *necessary*. To this end, the approach of [7] required some small modifications in many places. More specifically, the model notion had to be modified.

Other additional assumptions about derivations cannot be expressed in terms of the selected atoms alone. We consider here one such assumption, that of derivations being *local*, meaning that in a resolution step, the most recently introduced atoms must be resolved first [14]. Clearly, this is not a property of a single atom, but of atoms in the context of a derivation. To deal with local selection rules, we modify the model notion of [7] so that it reflects the substitutions computed by local derivations. Based on such models, in [22] we gave a sufficient criterion for termination of *local* derivations, parametrised by a $\mathcal{P}$ as before. Here, we modify this criterion so that it is sufficient and necessary.

We thus present a framework for showing termination of logic programs with dynamic scheduling. The initial motivation for this work was our impression that while stronger assumptions than that of input-consuming derivations are sometimes required, *locality* is too strong. More specifically, by instantiating the framework appropriately, we can now make the following five points:

1. There is a class of recursive clauses, using a natural pattern of programming, that narrowly misses the property of termination for input-consuming derivations. Put simply, theses clauses have the form $p(\mathtt{X}) \leftarrow q(\mathtt{X},\mathtt{Y}),\ p(\mathtt{Y})$, where the mode is $p(\mathit{input})$, $q(\mathit{input}, \mathit{output})$. Due to the variable in the head, it follows that an atom using p may always be selected, and hence we have non-termination. Sometimes, just requiring the argument of p to be at least non-variable is enough to ensure termination. This can be captured in our approach by setting (the relevant subset of) $\mathcal{P}$ to $\{p(t) \mid t \text{ is non-variable}\}$ (Ex. 4.8).

2. Some programs require for termination that selected atoms must be bounded with respect to a level mapping $|.|$. This is related to *speculative output bindings*, and the `PERMUTE` program is the standard example [18]. This can be captured in our approach by setting $\mathcal{P}$ to the set of bounded atoms (Ex. 4.4).

3. For some programs it is useful to consider "hybrid" selection rules, where differently strong assumptions are made for different predicates. For example, one might require ground input positions for some predicates but no additional assumptions for other predicates. This can be captured by setting $\mathcal{P}$ accordingly (Ex. 4.9).

4. A method for showing termination of programs with delay declarations has been proposed in [14], assuming local selection rules. In our opinion, this assumption is unsatisfactory. No implementation of local selection rules is mentioned. Local selection rules do not permit any coroutining. But most importantly, while "the class of local selection rules

[...] supports simple tools for proving termination" [14], in practice, it does not seem to make programs terminate that would not terminate otherwise. In fact, we can show termination for `PERMUTE` without requiring local selection rules (Ex. 4.4).

5. In spite of point 4, there are programs that crucially rely on the assumption of local selection rules for termination. We are only aware of artificial examples, but our treatment of local selection rules helps to understand the role this assumption plays in proving termination and why this assumption is not required for more realistic examples (Ex. 6.3).

The rest of this report is organised as follows. The next section gives some preliminaries. In Sec. 3, we adapt the semantics approach of [7] to $\mathcal{P}$-derivations. In Sec. 4, we study termination for such derivations. In Sec. 5, we adapt the semantics approach to local selection rules, and in Sec. 6, we study termination in this case. In Sec. 7, we conclude.

2 Preliminaries

We assume familiarity with the basic notions and results of logic programming [1]. For $m, n \in \mathbb{N}_0$, $m \leq n$, the set $\{m, \ldots, n\}$ is denoted by $[m..n]$. For any kind of object that we commonly denote with a certain letter, we use the same letter in boldface to denote a finite *sequence* of such objects [1].

We denote by $Term$ and $Atom$ the set of terms and atoms of the language in which the programs and queries in question are written. The arity n of a predicate symbol p is indicated by writing p/n. We use typewriter font for logical variables, for example $\mathtt{X}, \mathtt{Ys}$, and lower case letters for arbitrary terms, for example t, s, xs.

For any syntactic object o, we denote by $Vars(o)$ the set of variables occurring in o. A syntactic object is **linear** if every variable occurs in it at most once.

A **substitution** is a finite mapping from variables to terms. The domain (resp., set of variables in the range) of a substitution σ is denoted as $Dom(\sigma)$ (resp., $Ran(\sigma)$). We denote by $\sigma_{|o}$ the restriction of a substitution σ to $Vars(o)$. The result of the application of a substitution σ to a term t is called an **instance** of t and it is denoted by $t\sigma$. We say t is a **variant** of t' and write $t \approx t'$ if t and t' are instances of each other. A substitution σ is a **unifier** of terms t and t' if $t\sigma = t'\sigma$. We denote by $mgu(t, t')$ any *most general unifier* (mgu) of t and t'.

A **query** is a finite sequence of atoms $A_1, \ldots, A_m$, denoted $\Box$ when $m = 0$. If $1 \leq i_1 < \ldots < i_k \leq m$, $k \geq 0$, then $A_{i_1}, \ldots, A_{i_k}$ is a **subquery** of $A_1, \ldots, A_m$. A **clause** is a formula $H \leftarrow \mathbf{B}$ where H is an atom (the **head**) and $\mathbf{B}$ is a query (the **body**). $H \leftarrow \Box$ is simply written $H \leftarrow$. A **program** is a finite set of clauses. We denote atoms by H, A, B, C, D, E, queries by $\mathbf{Q}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}$, clauses by c, d and programs by P. We often suppress reference to P.

If $\mathbf{A}, B, \mathbf{C}$ is a query and $c = H \leftarrow \mathbf{B}$ is a fresh variant of a clause, and B and H unify with mgu σ, then $(\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma$ is called a **resolvent of $\mathbf{A}, B, \mathbf{C}$ and $H \leftarrow \mathbf{B}$ with selected atom B and mgu σ**. We call $\mathbf{A}, B, \mathbf{C} \overset{\sigma}{\Longrightarrow}_c (\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma$ a **derivation step**, in short: **step**. If c is irrelevant then we drop the reference. A sequence $\delta = \mathbf{Q}_0 \overset{\sigma_1}{\Longrightarrow}_{c_1} \mathbf{Q}_1 \overset{\sigma_2}{\Longrightarrow}_{c_2} \cdots$ is called a **derivation of $P \cup \{\mathbf{Q}_0\}$**.

If $\delta = \mathbf{Q}_0 \overset{\sigma_1}{\Longrightarrow}_{c_1} \cdots \overset{\sigma_n}{\Longrightarrow}_{c_n} \mathbf{Q}_n$ is a finite derivation, we also denote it as $\delta = \mathbf{Q}_0 \overset{\sigma}{\longrightarrow} \mathbf{Q}_n$ where $\sigma = \sigma_1 \cdots \sigma_n$. We call $len(\delta) = n$ the **length** of δ.

Let $\mathbf{A}, B, \mathbf{C} \overset{\sigma}{\Longrightarrow} (\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma$ be a step. We say each atom in $\mathbf{B}\sigma$ is a **direct descendant of** B, and for each atom D in $(\mathbf{A}, \mathbf{C})$, $D\sigma$ is a **direct descendant** of D. We say that D is a **descendant** of E if the pair (D, E) is in the reflexive, transitive closure of the relation *is a direct descendant of*.

Consider a derivation $\mathbf{Q}_0 \overset{\sigma_1}{\Longrightarrow} \cdots \overset{\sigma_i}{\Longrightarrow} \mathbf{Q}_i \cdots \overset{\sigma_j}{\Longrightarrow} \mathbf{Q}_j \overset{\sigma_{j+1}}{\Longrightarrow} \mathbf{Q}_{j+1} \cdots$. We say that $\mathbf{Q}_j \overset{\sigma_{j+1}}{\Longrightarrow} \mathbf{Q}_{j+1} \cdots$ is a **D-step** if $\mathbf{D}$ is a subquery of $\mathbf{Q}_i$ and the selected atom in $\mathbf{Q}_j$ is a descendant of an atom in $\mathbf{D}$.

A derivation **proceeds left-to-right** if whenever an atom B is selected in a query $\mathbf{A}, B, \mathbf{C}$, then no $\mathbf{A}$-step is performed afterwards.

2.1 Moded Programs

For a predicate p/n, a **mode** is an atom $p(m_1, \ldots, m_n)$, where $m_i \in \{I, O\}$ for $i \in [1..n]$. Positions with I (O) are called **input** (**output**) positions of p. To simplify the notation, an atom $p(\mathbf{s}, \mathbf{t})$ means: $\mathbf{s}$ is the vector of terms filling in the input positions, and $\mathbf{t}$ is the vector of terms filling in the output positions. Alternatively, we denote by $In(\mathbf{A})$ (resp. $Out(\mathbf{A})$) the sequence of terms filling in the input (resp. output) positions of predicates in $\mathbf{A}$.

We assume that the mode of each predicate is unique. One way of ensuring this is to rename predicates whenever multiple modes are desired.

Several notions of "modedness" have been proposed, for example *nicely-modedness* and *well-modedness* [1]. We assume here *simply* moded programs [2], a special case of nicely moded programs. Most practical programs are simply moded [7], although we will also give an example of a clause that is not—at least in one of its two plausible modes.

Note that the use of the letters $\mathbf{s}$ and $\mathbf{t}$ is reversed for clause heads. We believe that this notation naturally reflects the data flow within a clause.

Definition 2.1 A clause $p(\mathbf{t}_0, \mathbf{s}_{n+1}) \leftarrow p_1(\mathbf{s}_1, \mathbf{t}_1), \ldots, p_n(\mathbf{s}_n, \mathbf{t}_n)$ is **simply moded** if $\mathbf{t}_1, \ldots, \mathbf{t}_n$ is a linear vector of variables and for all $i \in [1..n]$

$$Vars(\mathbf{t}_i) \cap Vars(\mathbf{t}_0) = \emptyset \quad \text{and} \quad Vars(\mathbf{t}_i) \cap \bigcup_{j=1}^{i} Vars(\mathbf{s}_j) = \emptyset.$$

A query $\mathbf{B}$ is **simply moded** if the clause $\texttt{dummy} \leftarrow \mathbf{B}$ is simply moded. A program is **simply moded** if all of its clauses are simply moded.

Thus, a clause is simply moded if the output positions of body atoms are filled in by distinct variables, and every variable occurring in an output position of a body atom does not occur in an earlier input position.

As an example of a clause that is *not* simply moded, consider the clause

```
reverse([X|Xs],Ys) ← append(Zs,[X],Ys), reverse(Xs,Zs).
```

in mode $\texttt{reverse}(O, I), \texttt{append}(O, O, I)$: `[X]` is not a variable. In Ex. 4.7, we give a slightly modified version of the `NAIVE_REVERSE` program that is simply moded. *Robustly typed* programs [23] are in some sense a generalisation of simply moded programs, and include the above clause. However, the results of this report have so far not been generalised to robustly typed programs.

2.2 Norms and Level Mappings

Proofs of termination usually rely on the notions of *norm* and *level mapping* for measuring the size of terms and atoms. These concepts were originally defined for ground objects [10], but here we define them for arbitrary objects (in [19], we call such norms and level mappings *generalised*). To show termination of moded programs, it is natural to use *moded* level mappings, where the level of an atom depends only on its input positions [7].

Definition 2.2 A **norm** is a function $|.| : Term \to \mathbb{N}_0$, and a **level mapping** is a function $|.| : Atom \to \mathbb{N}_0$, both invariant under renaming. A **moded level mapping** is a level mapping where for any $\mathbf{s}$, $\mathbf{t}$ and $\mathbf{u}$, $|p(\mathbf{s}, \mathbf{t})| = |p(\mathbf{s}, \mathbf{u})|$.

An atom A is **bounded** with respect to the level mapping $|.|$ if there exists $k \in \mathbb{N}$ such that for every substitution σ, we have $k > |A\sigma|$.

Our method of showing termination, following [7], inherently relies on measuring the size of atoms that are not bounded. In Defs. 4.3 and 6.2, a decrease in level mapping must be shown (also) for such atoms. So it is important to understand that stating $|A| = k$ is different from stating that A is bounded by k.

One commonly used norm is the term size norm, defined as

$$\begin{array}{rcll} |f(t_1, \ldots, t_n)| & = & 1 + |t_1| + \ldots + |t_n| & \text{if } n > 0, \\ |t| & = & 0 & \text{if } t \text{ is a constant or variable.} \end{array}$$

Another widely used norm is the list-length function, defined as

$$\begin{array}{rcll} |[t|ts]| & = & 1 + |ts|, & \\ |t| & = & 0 & \text{if } t \neq [_|_] \text{ (in particular, if } t \text{ is a variable).} \end{array}$$

For a nil-terminated list $[t_1, \ldots, t_n]$, the list-length is n.

2.3 Selection Rules in the Literature

A selection rule is some rule stating which atom in a query may be selected in each step. We do not give any formal definition here; instead we define various kinds of derivations and state our formal results in terms of those.

The notion of *input-consuming derivation* was introduced in [20] as formalism for describing dynamic scheduling in an abstract way.

Definition 2.3 A derivation step $\mathbf{A}, p(\mathbf{s}, \mathbf{t}), \mathbf{C} \stackrel{\sigma}{\Longrightarrow} (\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma$ is **input-consuming** if $\mathbf{s}\sigma = \mathbf{s}$. A derivation is **input-consuming** if all its steps are input-consuming.

Local derivations were treated in [14]. Consider a query, containing atoms A and B, in a derivation ξ. Then A is **introduced more recently** than B if the step introducing A comes after the step introducing B, in ξ.

Definition 2.4 A derivation is **local** if in each step, there is no more recently introduced atom in the current query than the selected atom.

Intuitively, in a local derivation, once an atom is selected, that atom must be resolved away completely before any of its siblings may be selected.

3 Input-Consuming $\mathcal{P}$-Derivations

We consider derivations restricted by some property $\mathcal{P}$ of the selectable atoms. There are two conditions on $\mathcal{P}$. Some of our results would hold without these restrictions, but the conditions are so natural that we do not bother with this. Note that the atoms of a simply moded query have variables in their output positions, and so it would clearly be pathological to require a particular instantiation of the output arguments of an atom for that atom to be selected.[1]

Definition 3.1 A **$\mathcal{P}$-derivation** is a derivation such that each selected atom is in $\mathcal{P}$, where

1. $\mathcal{P}$ is a set of atoms closed under instantiation;
2. for any $\mathbf{s}$, $\mathbf{t}$ and $\mathbf{u}$, $p(\mathbf{s}, \mathbf{t}) \in \mathcal{P}$ implies $p(\mathbf{s}, \mathbf{u}) \in \mathcal{P}$.

This abstract way of defining derivations is an original notion of [22] (considering only published papers). Of course, a $\mathcal{P}$-derivation can be qualified further by saying *input-consuming* $\mathcal{P}$-derivation etc.

Input-consuming ($\mathcal{P}$-)derivations may end in a query where no atom can be selected. This situation is called **deadlock**. It is a form of termination. As a consequence, the usual completeness of logic programming does not hold for input-consuming ($\mathcal{P}$-)derivations.

[1] This argument would be slightly weakened if we considered relaxations of simply-modedness such as robustly-typedness mentioned above.

3.1 Renaming and Switching

We now generalise some results of [6, 7] to $\mathcal{P}$-derivations. We first cite a fundamental proposition about input-consuming steps [6, Remark 9].

Proposition 3.2 If P and $\mathbf{A}, B, \mathbf{C}$ are a simply moded program and query and $\mathbf{A}, B, \mathbf{C} \stackrel{\sigma}{\Longrightarrow} (\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma$ is an input-consuming step in P, then $\mathbf{A}\sigma = \mathbf{A}$.

The next lemma shows that input-consuming $\mathcal{P}$-derivations are invariant under renaming.

Lemma 3.3 Let P be a program, $\mathbf{Q}$ be a query and $\delta : \mathbf{Q} \stackrel{\theta}{\longrightarrow} \mathbf{Q}'$ be an input-consuming $\mathcal{P}$-derivation of $P \cup \{\mathbf{Q}\}$. Then, for any renaming ρ there exists an input-consuming $\mathcal{P}$-derivation $\delta' : \mathbf{Q}\rho \stackrel{\vartheta}{\longrightarrow} \mathbf{Q}'\rho$ where $\vartheta = \rho^{-1}\theta\rho$.

Proof. The lemma was shown for input-consuming derivations in [7, Lemma 2.9]. It generalises to $\mathcal{P}$-derivations since by Def. 3.1 (1), $A \in \mathcal{P}$ implies $A\rho \in \mathcal{P}$. □

The Left-Switching Lemma states that steps in an input-consuming derivation can always be switched so that atoms more to the left are selected earlier [6, Lemma 13]. Here we give a generalisation to $\mathcal{P}$-derivations.

Lemma 3.4 Let the program P and the query $\mathbf{Q}$ be simply moded. Let δ be an input-consuming $\mathcal{P}$-derivation of $P \cup \{\mathbf{Q}\}$ of the form (the selected atom is underlined for clarity):

$$\mathbf{Q} \longrightarrow \mathbf{A}, B, \mathbf{C}, \underline{D}, \mathbf{E} \stackrel{\theta}{\Longrightarrow}_c (\mathbf{A}, \underline{B}, \mathbf{C}, \mathbf{D}, \mathbf{E})\theta \stackrel{\sigma}{\Longrightarrow}_d (\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E})\theta\sigma \longrightarrow \ldots$$

where $c = \tilde{D} \leftarrow \mathbf{D}$ and $d = \tilde{B} \leftarrow \mathbf{B}$ are clauses from P.

Then, there exists an input-consuming $\mathcal{P}$-derivation δ' of the form

$$\mathbf{Q} \longrightarrow \mathbf{A}, \underline{B}, \mathbf{C}, D, \mathbf{E} \stackrel{\sigma'}{\Longrightarrow}_d (\mathbf{A}, \mathbf{B}, \mathbf{C}, \underline{D}, \mathbf{E})\sigma' \stackrel{\theta'}{\Longrightarrow}_c (\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E})\sigma'\theta' \longrightarrow \ldots,$$

such that $\theta\sigma = \sigma'\theta'$ and

- δ and δ' coincide up to the resolvent $\mathbf{A}, B, \mathbf{C}, D, \mathbf{E}$,
- $B = B\theta$,
- δ and δ' coincide after the resolvent $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E})\theta\sigma$.

Proof. The statement $B = B\theta$ follows from Prop. 3.2. Moreover, $D\sigma'$ is at least as instantiated as D. By Def. 3.1 (1) and since δ is a $\mathcal{P}$-derivation, it thus follows that δ' is a $\mathcal{P}$-derivation. Apart from that, the statement is identical to [6, Lemma 13] (but we use a more concise notation). □

The following statement, originally stated in [7, Cor. 2.12] as a corollary of [6, Lemma 13], is also generalised here to $\mathcal{P}$-derivations.

Corollary 3.5 Let the program P and the query $\mathbf{A}, \mathbf{B}$ be simply moded. Suppose that $\delta = \mathbf{A}, \mathbf{B} \stackrel{\sigma}{\longrightarrow} \mathbf{C}$ is an input-consuming $\mathcal{P}$-derivation of $P \cup \{(\mathbf{A}, \mathbf{B})\}$.

Then there exist $\mathbf{C}_1$ and $\mathbf{C}_2$ and an input-consuming $\mathcal{P}$-derivation δ' that proceeds left-to-right of the form

$$\delta' = \mathbf{A}, \mathbf{B} \stackrel{\sigma_1}{\longrightarrow} \mathbf{C}_1, \mathbf{B}\sigma_1 \stackrel{\sigma_2}{\longrightarrow} \mathbf{C}_1, \mathbf{C}_2$$

such that $len(\delta) = len(\delta')$, and $\mathbf{C} = \mathbf{C}_1, \mathbf{C}_2$, and $\sigma = \sigma_1\sigma_2$, and all the $\mathbf{A}$-steps are performed in the prefix $\mathbf{A}, \mathbf{B} \stackrel{\sigma_1}{\longrightarrow} \mathbf{C}_1, \mathbf{B}\sigma_1$, and all the $\mathbf{B}$-steps are performed in the suffix $\mathbf{C}_1, \mathbf{B}\sigma_1 \stackrel{\sigma_2}{\longrightarrow} \mathbf{C}_1, \mathbf{C}_2$ and $\mathbf{C}_1\sigma_2 = \mathbf{C}_1$.

```
permute([],[]).
permute(Ys,[X|Xs]) ←
   delete(Ys,Zs,X),
   permute(Zs,Xs).

delete([X|Xs],Xs,X).
delete([Y|Zs],[Y|Xs],X) ←
   delete(Zs,Xs,X), call_late.
call_late.
```

Figure 1: PERMUTE

3.2 Simply Local Substitutions

The contents of this subsection are taken from [7]. We define *simply local* substitutions, which reflect the way simply moded clauses become instantiated in input-consuming derivations. Given a clause $c = p(\mathbf{t}_0, \mathbf{s}_{n+1}) \leftarrow p_1(\mathbf{s}_1, \mathbf{t}_1), \ldots, p_n(\mathbf{s}_n, \mathbf{t}_n)$, first $\mathbf{t}_0$ becomes instantiated, and the range of that substitution contains only variables from outside of c. Then, by resolving $p_1(\mathbf{s}_1, \mathbf{t}_1)$, $\mathbf{t}_1$ becomes instantiated, and the range of that substitution contains variables from outside of c and from $\mathbf{s}_1$. Continuing in the same way, finally, $\mathbf{t}_n$ becomes instantiated, and the range of that substitution contains variables from outside of c and from $\mathbf{s}_1 \ldots \mathbf{s}_n$.

Definition 3.6 The substitution σ is **simply local** with respect to the clause $c = p(\mathbf{t}_0, \mathbf{s}_{n+1}) \leftarrow p_1(\mathbf{s}_1, \mathbf{t}_1), \ldots, p_n(\mathbf{s}_n, \mathbf{t}_n)$ if there are substitutions $\sigma_0, \sigma_1 \ldots, \sigma_n$ and disjoint sets $V_0, V_1, \ldots, V_n$ consisting of fresh (with respect to c) variables such that $\sigma = \sigma_0\sigma_1 \cdots \sigma_n$, where for $i \in [0..n]$,

- $Dom(\sigma_i) \subseteq Vars(\mathbf{t}_i)$,
- $Ran(\sigma_i) \subseteq Vars(\mathbf{s}_i\sigma_0\sigma_1 \cdots \sigma_{i-1}) \cup V_i$.[2]

σ is **simply local** with respect to a query $\mathbf{B}$ if σ is simply local with respect to the clause $\mathtt{dummy} \leftarrow \mathbf{B}$.

Note that in the case of a simply local substitution with respect to a query, σ_0 is the empty substitution.

Example 3.7 Consider the recursive clause for `delete` in Fig. 1, with mode `delete`(I, O, O). The substitution $\sigma = \{\mathtt{Y/V}, \mathtt{Zs/[W]}, \mathtt{Xs/[]}, \mathtt{X/W}\}$ is simply local with respect to it: let $\sigma_0 = \{\mathtt{Y/V}, \mathtt{Zs/[W]}\}$, $\sigma_1 = \{\mathtt{X/W}, \mathtt{Xs/[]}\}$, and $\sigma_2 = \emptyset$; then $Dom(\sigma_0) \subseteq \{\mathtt{Y}, \mathtt{Zs}\}$, $Ran(\sigma_0) \subseteq V_0$ where $V_0 = \{\mathtt{V}, \mathtt{W}\}$, $Dom(\sigma_1) \subseteq \{\mathtt{Xs}, \mathtt{X}\}$, and $Ran(\sigma_1) \subseteq Vars(\mathtt{Zs}\,\sigma_0)$.

The following property follows immediately from Definition 3.6.

Proposition 3.8 [7, Prop. 3.3] Let the clause c be simply moded and ρ be a renaming. If the substitution θ is simply local with respect to c then the substitution $\rho^{-1}\theta\rho$ is simply local with respect to $c\rho$.

It is just a technical detail, but we must be precise about what "freshness" means:

Definition 3.9 Let θ_1 be a substitution simply local with respect to $\mathbf{A}$ and θ_2 be simply local with respect to $\mathbf{B}\theta_1$. Then θ_1 and θ_2 are **variable compatible** with respect to $\mathbf{A}$ and $\mathbf{B}$ if

- the set of fresh variables of θ_1 is disjoint from the set of fresh variables of θ_2,
- $Var(\mathbf{A}, \mathbf{B})$ is disjoint from the set of fresh variables of θ_1 and θ_2.

When two substitutions are variable compatible then we have a way of combining them as described below.

Lemma 3.10 [7, Lemma 3.5] Let the query $\mathbf{A}, \mathbf{B}$ be simply moded. There exists a substitution θ simply local with respect to $\mathbf{A}, \mathbf{B}$ iff $\theta = \theta_1\theta_2$ where

[2] Note that $\mathbf{s}_0$ is undefined. By abuse of notation, $Vars(\mathbf{s}_0 \ldots) = \emptyset$.

- $\theta_1 = \theta_{|\mathbf{A}} = \theta_{|Out(\mathbf{A})}$ is simply local with respect to $\mathbf{A}$,
- $\theta_2 = \theta_{|\mathbf{B}} = \theta_{|Out(\mathbf{B})}$ simply local with respect to $\mathbf{B}\theta_1$,
- θ_1 and θ_2 are variable compatible with respect to $\mathbf{A}$ and $\mathbf{B}$.

Analogously, one can prove the following result which allows us to combine simply local substitutions applied to a clause rather than to a query.

Lemma 3.11 [7, Lemma 3.6] Let the clause $c : H \leftarrow \mathbf{B}$ be simply moded. There exists a substitution θ simply local with respect to c iff $\theta = \theta_0\theta_1$ where

- $\theta_0 = \theta_{|H} = \theta_{|In(H)}$ is simply local with respect to $H \leftarrow$,
- $\theta_1 = \theta_{|\mathbf{B}} = \theta_{|Out(\mathbf{B})}$ simply local with respect to $\mathbf{B}\theta_0$,
- θ_0 and θ_1 are variable compatible with respect to H and $\mathbf{B}$.

The following definition introduces a property of mgu's which can be naturally satisfied by input-consuming derivations, as shown in the subsequent lemma.

Definition 3.12 Let the atoms A and H be variable disjoint, A be simply moded and θ be a mgu of A and H such that $In(A\theta) = In(A)$. We say that θ is a **simply local mgu of A and H** if $\theta = \sigma_0\sigma_1$ where σ_0 is simply local with respect to the clause $H \leftarrow$ and σ_1 is simply local with respect to the atom A.

Lemma 3.13 [7, Lemma 3.8] Let the atoms A and H be variable disjoint and A be simply moded. Suppose that there exists $\vartheta = mgu(A, H)$ such that $In(A\vartheta) = In(A)$. Then there exists a simply local mgu θ of A and H.

As a consequence of the lemma, we can safely assume that all the mgu's employed in an input-consuming derivation of a simply moded program with a simply moded query are simply local, that is to say: if $\mathbf{A}, B, \mathbf{C} \Longrightarrow (\mathbf{A}, \mathbf{B}, \mathbf{C})\sigma$ is an input-consuming step using clause $c = H \leftarrow \mathbf{B}$, then $\sigma = \sigma_0\sigma_1$ and $\sigma_0 (= \sigma_{|H})$ is simply local with respect to the clause $H \leftarrow$ and $\sigma_1 (= \sigma_{|B})$ is simply local with respect to the atom[3] B.

3.3 Models

In [7], a particular notion of model is defined, which reflects the substitutions that can be computed by input-consuming derivations. According to this notion, a model is a set of *not necessarily ground* atoms. Here, we generalise this notion so that it reflects the substitutions that can be computed by input-consuming $\mathcal{P}$-derivations. So in this entire subsection, results from [7] are generalised to $\mathcal{P}$-derivations.

Definition 3.14 Let $M \subseteq Atom$. We say that M is a **simply local $\mathcal{P}$-model** of $c = H \leftarrow B_1, \ldots, B_n$ if for every substitution σ simply local with respect to c,

$$\text{if } B_1\sigma, \ldots, B_n\sigma \in M \text{ and } H\sigma \in \mathcal{P} \text{ then } H\sigma \in M. \tag{1}$$

M is a **simply local $\mathcal{P}$-model** of a program P if M is a simply local $\mathcal{P}$-model of each clause of P.

An interpretation is a set of atoms closed under variance. Abusing notation, we sometimes identify a query $\mathbf{A}$ with its set of atoms and write $\mathbf{A} \subseteq I$.

We denote the set of all simply moded atoms[3] for the program P by SM_P.

Least simply local $\mathcal{P}$-models, possibly containing SM_P, can be computed by a variant of the well-known T_P-operator [7].

[3]We sometimes say "atom" for "query containing only one atom."

Definition 3.15 Given a program P and $I \subseteq Atom$, we define

$$\begin{array}{rcl} T_P^{sl\mathcal{P}}(I) & = & \{H\sigma \mid \exists\, c = H \leftarrow B_1, \ldots, B_n \text{ variant of a clause in } P, \\ & & \quad \sigma \text{ is simply local with respect to } c,\ B_1\sigma, \ldots, B_n\sigma \in I,\ H\sigma \in \mathcal{P}\}, \\ T_P^{SL\mathcal{P}}(I) & = & I \cup T_P^{sl\mathcal{P}}(I). \end{array}$$

We denote the least simply local $\mathcal{P}$-model of P containing SM_P by $PM_P^{SL\mathcal{P}}$.

The following proposition is in analogy to [7, Prop. 4.3].

Proposition 3.16 Let P be a simply moded program and $I \subseteq SM_P$ be an interpretation. Then $T_P^{SL\mathcal{P}} \uparrow \omega(I)$ is the least simply local $\mathcal{P}$-model of P containing I.

Example 3.17 Consider the program `DELETE` (see Fig. 1) where we ignore the `call_late` predicate. Recall Ex. 3.7. Let $\mathcal{P}$ be the set containing *all* atoms using `delete`. SM_P consists of all atoms of the form $\texttt{delete}(vs, \texttt{Us}, \texttt{U})$ where $\texttt{Us}, \texttt{U} \notin Vars(vs)$. To construct $PM_P^{SL\mathcal{P}}$, we iterate $T_P^{SL\mathcal{P}}$ starting from any atom in SM_P (the resulting atoms are written on the l.h.s. below) and the fact clause (r.h.s.). Each line below corresponds to one iteration of $T_P^{SL\mathcal{P}}$. We have $PM_P^{SL\mathcal{P}} =$

$$\begin{array}{ll} \{\ \texttt{delete}(vs, \texttt{Us}, \texttt{U}), & \\ \quad \texttt{delete}(\texttt{[}y_1|vs\texttt{]}, \texttt{[}y_1|\texttt{Us]}, \texttt{U}), & \texttt{delete}(\texttt{[}x_1|xs_1\texttt{]}, xs_1, x_1), \\ \quad \texttt{delete}(\texttt{[}y_2, y_1|vs\texttt{]}, \texttt{[}y_2, y_1|\texttt{Us]}, \texttt{U}), & \texttt{delete}(\texttt{[}y_1, x_1|xs_1\texttt{]}, \texttt{[}y_1|xs_1\texttt{]}, x_1), \\ \quad \ldots & \ldots \\ \quad \mid vs, xs_1, x_1, y_1, y_2, \ldots \text{arbitrary where } \texttt{Us}, \texttt{U} \notin Vars(vs)\}. & \end{array} \qquad (2)$$

Observe the variable occurrences of `U`, `Us` in the atoms on the l.h.s. In Ex. 4.4, we will see the importance of such variable occurrences.

If we are very fussy, the terms $y_1, y_2, \ldots$ in (2) are not completely arbitrary: each y_i must not contain any variables occurring in the clause variant used by $T_P^{SL\mathcal{P}}$ to introduce $\texttt{delete}(\texttt{[}y_i, \ldots, y_1|vs\texttt{]}, \texttt{[}y_i, \ldots, y_1|\texttt{Us]}, \texttt{U})$; speaking in terms of Def. 3.6, $Vars(y_i) \subseteq V_0$. But since one can assume that each application of $T_P^{SL\mathcal{P}}$ always uses a fresh clause copy, there is no need to worry about this point.

In the above example, we assume that $\mathcal{P}$ is the set of *all* atoms, and so the simply local $\mathcal{P}$-model is in fact a *simply local model* [7]. In order to obtain a *necessary* termination criterion, the approach of [7] required some small modifications in many places, one of them being the generalisation of simply local models to simply local $\mathcal{P}$-models. However, we are not aware of a practical situation where one has to consider a simply local $\mathcal{P}$-model that is not a simply local model.

The following lemma states that simply local $\mathcal{P}$-models indeed capture the partial answer substitutions that can be computed by input-consuming $\mathcal{P}$-derivations. It is shown in analogy to [7, Lemma 4.4].

Lemma 3.18 Let the program P and the query $\mathbf{A}$ be simply moded and $I \subseteq SM_P$ be an interpretation, and M be the least simply local $\mathcal{P}$-model of P containing I. Then the following statements are equivalent:

(i) there exists an input-consuming $\mathcal{P}$-derivation $\delta = \mathbf{A} \xrightarrow{\sigma} \mathbf{C}$ with $\mathbf{C} \subseteq I$,

(ii) there exists a substitution θ simply local with respect to $\mathbf{A}$, such that $\mathbf{A}\theta \subseteq M$,

where $\mathbf{A}\sigma \approx \mathbf{A}\theta$. For the "(i) $\Rightarrow$ (ii)" direction, we can choose $\theta = \sigma_{|\mathbf{A}}$.

Proof. This lemma differs from [7, Lemma 4.4] in that we generalise to $\mathcal{P}$-derivations. We will point out the few places where our proof differs significantly.

Throughout the proof we will use $T_P^{SL\mathcal{P}} \uparrow \omega(I) = M$ (Prop. 3.16).

We assume that all the mgu's employed are simply local, as stated after Lemma 3.13. We refer to the assumption that derivations always use fresh clause variants as *freshness*.

(i) $\Rightarrow$ (ii). We proceed by induction on the length of δ. In this "(i) $\Rightarrow$ (ii)" part, we assume that δ proceeds left-to-right and employs only simply local mgu's and prove that: $\sigma_{|\mathbf{A}}$ is simply local with respect to $\mathbf{A}$ and $\mathbf{A}\sigma \subseteq T_P^{SLP} \uparrow \omega(I)$. The general case follows from Corollary 3.5 (note that this had to be adapted from [7, Cor. 2.12]) and [1, Theorem 3.18] on derivations employing different mgu's.

Base: $len(\delta) = 0$. In this case $\mathbf{A} = \mathbf{C} \subseteq I$ and $\sigma = \epsilon$ (the empty substitution). The thesis follows from the fact that, by definition of T_P^{SLP}, $I \subseteq T_P^{SLP} \uparrow \omega(I)$.

Induction step: $len(\delta) > 0$. Let $\mathbf{A} = \mathbf{L}, A, \mathbf{R}$ and A be the leftmost atom of $\mathbf{A}$ such that there is some A-step in δ (and hence there are no $\mathbf{L}$-steps in δ). Assume also that $c = H \leftarrow \mathbf{B}$ is the clause used in the first derivation step of δ and σ_1 is the simply local mgu of A and H used in this step. Since δ proceeds left-to-right, we can write

$$\delta = \mathbf{A} \overset{\sigma_1}{\Longrightarrow} (\mathbf{L}, \mathbf{B}, \mathbf{R})\sigma_1 \xrightarrow{\sigma_2} \mathbf{L}, \mathbf{C}' \tag{3}$$

such that $\mathbf{C} = \mathbf{L}, \mathbf{C}'$, $\sigma = \sigma_1\sigma_2$ and $\mathbf{L}\sigma_1 = \mathbf{L}\sigma_1\sigma_2 = \mathbf{L}$.

Hence

$$\mathbf{L} \subseteq I \subseteq T_P^{SLP} \uparrow \omega(I) \tag{4}$$

and there exists an input-consuming $\mathcal{P}$-derivation $\delta' = (\mathbf{B}, \mathbf{R})\sigma_1 \xrightarrow{\sigma_2} \mathbf{C}'$ where $len(\delta') = len(\delta) - 1$ and $(\mathbf{B}, \mathbf{R})\sigma_1$ is simply moded (by the persistence of simply-modedness [7, Lemma 2.7]).

By the inductive hypothesis, σ_2 is simply local with respect to $(\mathbf{B}, \mathbf{R})\sigma_1$ and

$$(\mathbf{B}, \mathbf{R})\sigma_1\sigma_2 \subseteq T_P^{SLP} \uparrow \omega(I). \tag{5}$$

Note also that since $\sigma_1\sigma_2$ is computed in a derivation of $(A, \mathbf{R})$, by freshness and [7, Lemma 3.10][4] we have that

$$(\sigma_1\sigma_2)_{|(A,\mathbf{R})} \text{ is simply local with respect to} (A, \mathbf{R}), \tag{6}$$

and since $(\sigma_1\sigma_2)_{|\mathbf{L}} = \varepsilon$,

$$(\sigma_1\sigma_2)_{|(\mathbf{L},A,\mathbf{R})} \text{ is simply local with respect to} (\mathbf{L}, A, \mathbf{R}). \tag{7}$$

To conclude the proof it remains to be shown that

$$A\sigma_1\sigma_2 \subseteq T_P^{SLP} \uparrow \omega(I). \tag{8}$$

Then, the result will follow from (4), (5), (7) and (8).

By the assumption that mgu's are simply local, $\sigma_{1|H}$ is simply local with respect to $H \leftarrow$. Moreover, by Lemma 3.10, $\sigma_{2|\mathbf{B}\sigma_1}$ is simply local with respect to $\mathbf{B}\sigma_1$. Note also that, by freshness, $\sigma_{1|H}$ and $\sigma_{2|\mathbf{B}\sigma_1}$ are variable compatible with respect to H and $\mathbf{B}$. Hence, by Lemma 3.11,

$$\sigma_{1|H}\sigma_{2|\mathbf{B}\sigma_1} = (\sigma_1\sigma_2)_{|c} \text{ is simply local with respect to } c. \tag{9}$$

By (3), A is selected in a $\mathcal{P}$-derivation, hence $A \in \mathcal{P}$, and so by Def. 3.1 (1), $H\sigma_1\sigma_2 = A\sigma_1\sigma_2 \in \mathcal{P}$ (in the proof of [7, Lemma 4.4], this condition is missing). By Definition 3.15 and (5, 9), this implies that

$$H\sigma_1\sigma_2 = A\sigma_1\sigma_2 \subseteq T_P^{SLP} \uparrow \omega(I).$$

(ii) $\Rightarrow$ (i). Let $\mathbf{A}\theta \subseteq T_P^{SLP} \uparrow \omega(I)$ with $\mathbf{A} = A_1, \ldots, A_n$. Let k be the minimum index such that $\mathbf{A}\theta \in T_P^{SLP} \uparrow k(I)$. The proof proceeds by induction on k.

Base: $k = 0$. In this case, $\mathbf{A}\theta \subseteq T_P^{SLP} \uparrow 0(I) = I$ with θ simply local with respect to $\mathbf{A}$. Since both $\mathbf{A}$ and $\mathbf{A}\theta$ consist of simply moded atoms, and θ is a simply local substitution with respect to $\mathbf{A}$, it follows that θ is just a renaming of the output variables of $\mathbf{A}$, and so since I is an interpretation and thus closed under variance, $\mathbf{A} \subseteq I$. The thesis follows by taking σ to be the empty substitution and δ to be the derivation of length zero.

[4] In short, the lemma states that if $A, \mathbf{R} \overset{\sigma}{\Longrightarrow} (\mathbf{B}, \mathbf{R})\sigma$ is an input-consuming step and θ is simply local with respect to $(\mathbf{B}, \mathbf{R})\sigma$, then $(\sigma\theta)_{|A,\mathbf{R}}$ is simply local with respect to $A, \mathbf{R}$.

Induction step: $k > 0$. We proceed by another induction, this time on n, the number of atoms in the query.

Base: $n = 1$. In this case $\mathbf{A} = A$, θ is simply local with respect to A and $A\theta \in T_P^{SL\mathcal{P}} \uparrow k(I)$. By definition of $T_P^{SL\mathcal{P}}$, there exist a variant $c = H \leftarrow \mathbf{B}$ of a clause of P variable disjoint from A and a substitution ϑ such that

$$\vartheta \text{ is simply local with respect to } c, \tag{10}$$
$$\mathbf{B}\vartheta \subseteq T_P^{SL\mathcal{P}} \uparrow (k-1)(I), \tag{11}$$
$$A\theta = H\vartheta. \tag{12}$$

By (10) and Lemma 3.11 there exist ϑ_0 and ϑ_1 such that $\vartheta = \vartheta_0\vartheta_1$, $\vartheta_0 = \vartheta_{|H}$ is simply local with respect to $H \leftarrow$ and ϑ_1 is simply local with respect to $\mathbf{B}\vartheta_0$. Hence, by (11) and the inductive hypothesis on k, there exists an input-consuming $\mathcal{P}$-derivation

$$\delta' = \mathbf{B}\vartheta_0 \xrightarrow{\sigma_1} \mathbf{C} \tag{13}$$

where $\mathbf{C} \subseteq I$ and $\mathbf{B}\vartheta_0\sigma_1 \approx \mathbf{B}\vartheta_0\vartheta_1$.

Since θ is simply local with respect to A and A is simply moded,

$$In(A) = In(A\theta), \tag{14}$$

and so since $A\theta \in \mathcal{P}$, by Def. 3.1 (2) we also have

$$A \in \mathcal{P} \tag{15}$$

(in the proof of [7, Lemma 4.4], this argument is of course missing).

Note also that

$$H\vartheta_0\sigma_1 \approx H\vartheta_0\vartheta_1, \tag{16}$$

since the only variables of $H\vartheta_0$ which can be affected by ϑ_1 or σ_1 are those occurring also in $\mathbf{B}\vartheta_0$.

By Proposition 3.8 we can assume $Var(A) \cap Var(c) = \emptyset$. The fact that ϑ_0 is simply local with respect to H and (14) together imply that ϑ_0 is the minimal substitution such that $In(H)\vartheta_0 = In(A)$. Since A is simply moded, there exists a minimal substitution θ_0 such that $Out(A)\theta_0 = Out(H)\vartheta_0$. Moreover, $Dom(\vartheta_0) \cap Vars(A) = \emptyset$, $Dom(\theta_0) \cap Vars(In(A)) = \emptyset$, and $Dom(\theta_0) \cap Vars(c\vartheta_0) = \emptyset$. Therefore, $\vartheta_0\theta_0$ is a simply local mgu (Def. 3.12) of A and H, and

$$A \overset{\vartheta_0\theta_0}{\Longrightarrow} \mathbf{B}\vartheta_0\theta_0$$

in an input-consuming derivation step, which by (15) is a $\mathcal{P}$-derivation step. By (13) and since $\mathbf{B}\vartheta_0\theta_0 = \mathbf{B}\vartheta_0$,

$$\delta = A \overset{\vartheta_0\theta_0}{\Longrightarrow} \mathbf{B}\vartheta_0 \xrightarrow{\sigma_1} \mathbf{C}$$

is an input-consuming $\mathcal{P}$-derivation. Since $\vartheta_0\theta_0$ is an mgu of A and H, we have $A\vartheta_0\theta_0\sigma_1 = H\vartheta_0\theta_0\sigma_1$. Since $Dom(\theta_0) \cap Vars(c\vartheta_0) = \emptyset$, we have $H\vartheta_0\theta_0\sigma_1 = H\vartheta_0\sigma_1$. By (16), we have $H\vartheta_0\sigma_1 \approx H\vartheta_0\vartheta_1$. By (12), $H\vartheta_0\vartheta_1 = H\vartheta = A\theta$. Summarising we have $A\vartheta_0\theta_0\sigma_1 \approx A\theta$, which shows "(ii) ⇒ (i)" in this case.

Induction step: $n > 1$. In this case $\mathbf{A} = A, \mathbf{R}$ and $\mathbf{A}\theta \subseteq T_P^{SL\mathcal{P}} \uparrow k(I)$. By Lemma 3.10 there exist θ_1 and θ_2 such that $\theta = \theta_1\theta_2$, and $\theta_1 = \theta_{|A}$ is simply local with respect to A and θ_2 is simply local with respect to $\mathbf{R}\theta_1$, and θ_1 and θ_2 are variable compatible with respect to A and $\mathbf{R}$. By the inductive hypothesis on n, there exists an input-consuming $\mathcal{P}$-derivation

$$\delta_1 = A \xrightarrow{\sigma_1} \mathbf{C}_1$$

where $\mathbf{C}_1 \subseteq I$ and

$$A\sigma_1 \text{ and } A\theta_1 = A\theta \text{ are variant.} \tag{17}$$

Again by the inductive hypothesis on n, there exists an input-consuming $\mathcal{P}$-derivation

$$\delta_2' = \mathbf{R}\theta_1 \xrightarrow{\sigma_2'} \mathbf{C}_2'$$

where $\mathbf{C}'_2 \subseteq I$ and $\mathbf{R}\theta_1\sigma'_2 \approx \mathbf{R}\theta_1\theta_2$. The only variables of $\mathbf{R}$ that can be affected by θ_1 or σ_1 are those occurring also in A_1, and so by (17) we have $\mathbf{R}\theta_1 \approx \mathbf{R}\sigma_1$. Therefore by Lemma 3.3 there exists an input-consuming $\mathcal{P}$-derivation $\delta_2 = \mathbf{R}\sigma_1 \xrightarrow{\sigma_2} \mathbf{C}_2$ where $\mathbf{R}\sigma_1\sigma_2 \approx \mathbf{R}\theta_1\sigma'_2$. Since I is closed under variance and $\mathbf{C}'_2 \subseteq I$, we have $\mathbf{C}_2 \subseteq I$. Without loss of generality, we can assume that the clauses used in δ_2 are renamed apart with respect to δ_1.

Summarising, there exists an input-consuming $\mathcal{P}$-derivation

$$\delta = A, \mathbf{R} \xrightarrow{\sigma_1} \mathbf{C}_1, \mathbf{R}\sigma_1 \xrightarrow{\sigma_2} \mathbf{C}_1, \mathbf{C}_2$$

such that $\mathbf{A}\sigma_1\sigma_2 \approx \mathbf{A}\theta$. □

The results on semantics of input-consuming $\mathcal{P}$-derivations presented in this section are the basis for the termination criteria presented in the subsequent sections.

4 Termination without Requiring Local Derivations

4.1 Simply $\mathcal{P}$-Acceptable Programs

The following concept is adopted from Apt [1].

Definition 4.1 Let p, q be predicates in a program P. We say that p **refers to** q if there is a clause in P with p in its head and q in its body, and p **depends on** q (written $p \sqsupseteq q$) if (p, q) is in the reflexive, transitive closure of *refers to*. We write $p \sqsupset q$ if $p \sqsupseteq q$ and $q \not\sqsupseteq p$, and $p \simeq q$ if $p \sqsupseteq q$ and $q \sqsupseteq p$.

We extend this notation to *atoms*, for example $p(\mathbf{s}, \mathbf{t}) \simeq q(\mathbf{u}, \mathbf{v})$ if $p \simeq q$.

Definition 4.2 A program is **input $\mathcal{P}$-terminating** if all input-consuming $\mathcal{P}$-derivations starting in a simply-moded query are finite.

Previously, we had defined *input-termination*, which is input $\mathcal{P}$-termination for $\mathcal{P}$ being the set of all atoms [7]. We now give a sufficient and necessary criterion for input $\mathcal{P}$-termination.

Definition 4.3 Let P be a simply moded program, $|.|$ a moded level mapping and M a simply local $\mathcal{P}$-model of P containing SM_P. A clause $H \leftarrow B_1, \ldots, B_n$ is **simply $\mathcal{P}$-acceptable by $|.|$ and M** if for every substitution σ simply local with respect to it, for all $i \in [1..n]$,

$$B_1\sigma, \ldots, B_{i-1}\sigma \in M \text{ and } H \simeq B_i \text{ and } H\sigma \in \mathcal{P} \text{ and } B_i\sigma \in \mathcal{P} \quad \text{imply} \quad |H\sigma| > |B_i\sigma|. \quad (18)$$

The program P is **simply $\mathcal{P}$-acceptable by $|.|$ and M** if each clause of P is simply $\mathcal{P}$-acceptable by $|.|$ and M.

The reference to $|.|$ or M or both is often omitted, for example "simply $\mathcal{P}$-acceptable by M" means "simply $\mathcal{P}$-acceptable by some level mapping and M."

The difference to the definition of a *simply acceptable* clause [7] is in the conditions $B_i\sigma \in \mathcal{P}$ and $H\sigma \in \mathcal{P}$. The condition $B_i\sigma \in \mathcal{P}$ may seem more natural than the condition $H\sigma \in \mathcal{P}$, since σ, due to the model condition, reflects the degree of instantiation that B_i may have when it is selected. But for $i = 1$, σ reflects the degree of instantiation of the entire clause obtained by unification when the clause is used for an input-consuming $\mathcal{P}$-derivation step. Technically, the condition $H\sigma \in \mathcal{P}$ is important for showing *necessity* (Subsec. 4.3).

Note that a decrease between the head and body atoms must be shown only for the atoms where $H \simeq B_i$. The idea is that termination is shown incrementally, so we assume that for the B_i where $H \sqsupset B_i$, termination has been shown already. One can go further and explicitly give *modular* termination results [7, 10], but this is a side issue for us and we refrain from it in order to avoid unnecessary complication.

The following is the standard example of a program that requires boundedness as additional condition on selected atoms (see point 2 in the Introduction).

Example 4.4 Consider PERMUTE (Fig. 1) with mode permute(I, O), delete(I, O, O). Recall Ex. 3.17. As norm we take the list-length function, and we define the level mapping as $|\texttt{permute}(zs, xs)| = |zs|$ and $|\texttt{delete}(xs, zs, x)| = |xs|$. Now for all atoms $\texttt{delete}(ys, zs, x) \in PM_P^{SL\mathcal{P}}$, we have $|ys| \geq |zs|$; for the ones on the r.h.s. even $|ys| > |zs|$. Let $\mathcal{P}$ be the set of bounded atoms with respect to $|.|$.

Now let us look at the recursive clause for permute. We verify that the second body atom fulfils the requirement of Def. 4.3, where M is $PM_P^{SL\mathcal{P}}$. So we have to consider all simply local substitutions σ such that $\texttt{delete}(\texttt{Ys}, \texttt{Zs}, \texttt{X})\sigma \in PM_P^{SL\mathcal{P}}$. For the atoms on the l.h.s. in (2), this means that

$$\sigma \supseteq \{\texttt{Ys}/[y_n, \ldots, y_1|vs], \texttt{Zs}/[y_n, \ldots, y_1|\texttt{Us}], \texttt{X}/\texttt{U}\} \qquad (n \geq 0).$$

Clearly, $\texttt{permute}(\texttt{Zs}, \texttt{Xs})\sigma \notin \mathcal{P}$, and hence no proof obligation arises. For the atoms on the r.h.s. in (2), this means that

$$\sigma \supseteq \{\texttt{Ys}/[y_n, \ldots, y_1, x_1|xs_1], \texttt{Zs}/[y_n, \ldots, y_1|xs_1], \texttt{X}/x_1\} \qquad (n \geq 0).$$

But then $|\texttt{permute}(\texttt{Ys}, [\texttt{X}|\texttt{Xs}])\sigma| > |\texttt{permute}(\texttt{Zs}, \texttt{Xs})\sigma|$.

The other clauses are trivial to check, and so PERMUTE is simply $\mathcal{P}$-acceptable. Observe that only the model of DELETE played a role in our argument, not the model of PERMUTE.

The atom call_late only serves the purpose of allowing for non-local derivations, to emphasise that locality is not needed for termination (see point 4 in Sec. 1). Without this atom, all $\mathcal{P}$-derivations would automatically be local.

The following derivation (ignoring the call_late predicate) demonstrates that the program does not input-terminate:

$$\underline{\texttt{permute}([1], \texttt{W})} \Longrightarrow$$
$$\underline{\texttt{delete}([1], \texttt{Zs}', \texttt{X}')}, \texttt{permute}(\texttt{Zs}', \texttt{Xs}') \Longrightarrow \texttt{delete}([], \texttt{Xs}'', \texttt{X}'), \underline{\texttt{permute}([1|\texttt{Xs}''], \texttt{Xs}')} \Longrightarrow$$
$$\texttt{delete}([], \texttt{Xs}'', \texttt{X}'), \underline{\texttt{delete}([1|\texttt{Xs}''], \texttt{Zs}''', \texttt{X}''')}, \texttt{permute}(\texttt{Zs}''', \texttt{Xs}') \Longrightarrow$$
$$\texttt{delete}([], \texttt{Xs}'', \texttt{X}'), \texttt{delete}(\texttt{Xs}'', \texttt{Xs}'''', \texttt{X}'''), \underline{\texttt{permute}([1|\texttt{Xs}'''], \texttt{Xs}')} \Longrightarrow \ldots$$

4.2 Sufficiency of Simply $\mathcal{P}$-Acceptability

In [7], we find a result stating that simply acceptable programs are input-terminating. We will show that simply $\mathcal{P}$-acceptable programs are input $\mathcal{P}$-terminating.

The following statement allows us to restrict our attention to atomic queries. Originally given in [6, Cor. 23], it generalises easily to $\mathcal{P}$-derivations because it rests on Cor. 3.5 in the same way that [6, Cor. 23] rests on [7, Cor. 2.12].

Corollary 4.5 Let P be a simply moded program. P is input $\mathcal{P}$-terminating if and only if for each simply moded atomic query A all input-consuming $\mathcal{P}$-derivations of $P \cup \{A\}$ are finite.

Theorem 4.6 Let P be a simply moded program. Let M be a simply local $\mathcal{P}$-model of P containing SM_P. Suppose that P is simply $\mathcal{P}$-acceptable by a moded level mapping $|.|$ and M. Then P is input $\mathcal{P}$-terminating.

Proof. For each predicate p, define $dep(p)$ to be the number of predicates p depends on: $dep(p) = \#\{q | p \sqsupset q\}$. Clearly, $dep(p)$ is always finite. Moreover, if $p \simeq q$ then $dep(p) = dep(q)$, and if $p \sqsupset q$ then $dep(p) > dep(q)$. Just like $\sqsupset$ etc., we extend this notation to atoms.

By Cor. 4.5, it is sufficient to prove that for any simply moded atomic query A, all input-consuming $\mathcal{P}$-derivations of $P \cup \{A\}$ are finite.

For the purpose of deriving a contradiction, assume that δ is an infinite input-consuming $\mathcal{P}$-derivation of $P \cup \{A\}$. Then

$$\delta = A \overset{\sigma_1}{\Longrightarrow} (B_1, \ldots, B_n)\sigma_1 \longrightarrow \cdots$$

where $c = H \leftarrow B_1, \ldots, B_n$ is the clause used in the first step and $\sigma_1 = mgu(A, H)$. Clearly, $(B_1, \ldots, B_n)\sigma_1$ has an infinite input-consuming $\mathcal{P}$-derivation. By Cor. 3.5, for some $i \in [1..n]$ and for some substitution σ_2,

```
reverse([X|Xs],Ys) ←                    append_sing([X|Xs],Y,[X|Zs]) ←
   append_sing(Zs,X,Ys),                   append_sing(Xs,Y,Zs).
   reverse(Xs,Zs).                      append_sing([],Y,[Y]).
reverse([],[]).
```

Figure 2: NAIVE_REVERSE

1. there exists an infinite input-consuming $\mathcal{P}$-derivation of $P \cup \{A\}$ of the form

$$A \stackrel{\sigma_1}{\Longrightarrow} (B_1, \ldots, B_n)\sigma_1 \stackrel{\sigma_2}{\longrightarrow} \mathbf{C}, (B_i, \ldots, B_n)\sigma_1\sigma_2 \longrightarrow \cdots;$$

2. there exists an infinite input-consuming $\mathcal{P}$-derivation of $P \cup \{B_i\sigma_1\sigma_2\}$

both employing only simply local mgu's (Lemma 3.13).

Let $\sigma = (\sigma_1\sigma_2)_{|c}$. Using [7, Proposition A.1], one can easily show that σ is simply local with respect to c. By the persistence of simply-modedness [7, Lemma 2.7], $\mathbf{C}, (B_i, \ldots, B_n)\sigma_1\sigma_2$ is simply moded, and so letting $I = SM_P$, by Lemma 3.18 (i) $\Rightarrow$ (ii), $(B_1, \ldots, B_{i-1})\sigma \subseteq M$.

Consider the ordering $\succ$ defined by: $\langle m, n\rangle \succ \langle m', n'\rangle$ if either $m > m'$ or $m = m'$ and $n > n'$. We will show that $\langle dep(A), |A|\rangle \succ \langle dep(B_i\sigma), |B_i\sigma|\rangle$. By an obvious inductive argument, this is a contradiction to point 2.

We distinguish two cases: *(a)* $H \sqsupset B_i$, and *(b)* $H \simeq B_i$. Consider the derivation in point 1 above, recalling that $\sigma = (\sigma_1\sigma_2)_{|c}$. By [6, Lemma 10][5] and since $|.|$ is a moded level mapping, we have $|A| = |A\sigma| = |H\sigma|$.

In case *(a)* we have $dep(A) = dep(H\sigma) > dep(B_i\sigma)$. Therefore,

$$\langle dep(A), |A|\rangle = \langle dep(H\sigma), |H\sigma|\rangle \succ \langle dep(B_i\sigma), |B_i\sigma|\rangle. \tag{19}$$

Now consider case *(b)*. P is simply $\mathcal{P}$-acceptable by $|.|$ and M. We shall check that the premises in (18) are fulfilled. We have already stated that σ is simply local with respect to c and that $(B_1, \ldots, B_{i-1})\sigma \subseteq M$. Now since $H\sigma_1 = A\sigma_1$ and $A\sigma_1$ was selected in an input-consuming $\mathcal{P}$-derivation step, and $\sigma = \sigma_1\sigma_2$, by Def. 3.1 (1) it follows that $H\sigma \in \mathcal{P}$. Finally, by hypothesis 2, $B_i\sigma \in \mathcal{P}$. Thus it follows that $|H\sigma| > |B_i\sigma|$. Hence, (19) holds also in this case. □

We give three further examples. The first supports point 2 in the Introduction.

Example 4.7 The program NAIVE_REVERSE (Fig. 2) in mode $\texttt{reverse}(O, I), \texttt{append}(O, O, I)$ is not input-terminating, but it is input $\mathcal{P}$-terminating for $\mathcal{P}$ chosen in analogy to Ex. 4.4.

The next example illustrates point 1 in the Introduction.

Example 4.8 Let PERMUTE2 be the program obtained from PERMUTE by replacing the recursive clause for delete by its *most specific variant* [18]:

```
delete([Y,H|T],[Y|Xs],X) ← delete([H|T],Xs,X).
```

Assume $|.|$ and the modes as in Ex. 4.4. As in Ex. 3.17 we have $SM_P = \{\texttt{delete}(vs, \texttt{Us}, \texttt{U}) \mid vs \text{ arbitrary}\}$, but when we start applying $T_P^{SL\mathcal{P}}$ to the atoms in SM_P, then due to the modified clause above, only the atoms of the form $\texttt{delete}([v|vs'], \texttt{Us}, \texttt{U})$ contribute; $\texttt{delete}([], \texttt{Us}, \texttt{U})$ does not contribute. We get:

$$\begin{array}{lll} PM_P^{SL\mathcal{P}} = \{ & \texttt{delete}(vs, \texttt{Us}, \texttt{U}), & \\ & \texttt{delete}([y_1, v, vs'], [y_1|\texttt{Us}], \texttt{U}), & \texttt{delete}([x_1|xs_1], xs_1, x_1), \\ & \texttt{delete}([y_2, y_1, v|vs'], [y_2, y_1|\texttt{Us}], \texttt{U}), & \texttt{delete}([y_1, x_1|xs_1], [y_1|xs_1], x_1), \\ & \ldots & \ldots \\ & \mid vs, v, vs', xs_1, x_1, y_1, y_2, \ldots \text{arbitrary where } \texttt{Us}, \texttt{U} \notin \mathit{Vars}(vs, v, vs')\}. & \end{array}$$

[5]The lemma states that a variable in a query must occur in some output position to become instantiated.

We show that the program is simply $\mathcal{P}$-acceptable where $\mathcal{P}$ is the set of atoms that are at least non-variable in their input positions. As in Ex. 4.4, we concentrate on showing that the second body atom of the recursive clause for `permute` fulfils the requirement of Def. 4.3. We have to consider all simply local substitutions σ such that $\texttt{delete}(\texttt{Ys},\texttt{Zs},\texttt{X})\sigma \in PM_P^{SL\mathcal{P}}$, and moreover $\texttt{permute}(\texttt{Zs},\texttt{Xs})\sigma \in \mathcal{P}$. It is easy to see that for all such σ, we have $|\texttt{permute}(\texttt{Ys},\texttt{[X|Xs]})\sigma| > |\texttt{permute}(\texttt{Zs},\texttt{Xs})\sigma|$. The important point is the following: since $PM_P^{SL\mathcal{P}}$ contains, among others, atoms of the form $\texttt{delete}(vs,\texttt{Us},\texttt{U})$, we have to consider all simply local substitutions σ such that $\texttt{delete}(\texttt{Ys},\texttt{Zs},\texttt{X})\sigma = \texttt{delete}(vs,\texttt{Us},\texttt{U})$. But in this case $\texttt{Zs}\sigma = \texttt{Us}$, and $\texttt{permute}(\texttt{Zs},\texttt{Xs})\sigma \notin \mathcal{P}$ since `Us` is a variable. Therefore the atoms of the form $\texttt{delete}(vs,\texttt{Us},\texttt{U})$ do not give rise to an obligation to show a decrease.

The following is an example of "hybrid" selection rules (point 3 in Sec. 1).

Example 4.9 We only sketch this example. A program for the well-known n-queens problem has the following main clause:

```
nqueens(N,Sol) ←
   sequence(N,Seq), permute(Seq,Sol), safe(Sol).
```

We could implement `permute` as in Ex. 4.4 or as in Ex. 4.8. In either case, we have a non-trivial $\mathcal{P}$. In contrast, $\mathcal{P}$ may contain *all* atoms using `safe`. In fact, for efficiency reasons, atoms using `safe` should be selected as early as possible.

Note that such a hybrid selection rule can effectively be realised by means of the *default left-to-right selection rule* [23]. To this end, the second and third atoms must be swapped. Since any results in this report do not actually depend on the textual position of atoms, they still apply to the thus modified program.

4.3 Necessity of Simply $\mathcal{P}$-Acceptability

We now prove the converse of Theorem 4.6, namely that our criterion for proving input $\mathcal{P}$-termination is also necessary. We proceed in close analogy to [7] which in turn is based on [5].

IC$\mathcal{P}$-trees are defined in obvious analogy to IC-trees [7], which in turn resemble SLD-trees.

Definition 4.10 Let P be a program and $\mathbf{Q}$ be a query. An **IC$\mathcal{P}$-tree** for $P \cup \{\mathbf{Q}\}$ is a tree such that

- its root is $\mathbf{Q}$,
- every node $\mathbf{Q}'$ has exactly one child $\mathbf{Q}''$ for every atom A of $\mathbf{Q}'$ and every clause c in P such that $\mathbf{Q}''$ is an input-consuming $\mathcal{P}$-resolvent (that is, a resolvent in an input-consuming $\mathcal{P}$-derivation) of $\mathbf{Q}'$ with respect to A and c.

Informally, an IC$\mathcal{P}$-tree for $P \cup \{\mathbf{Q}\}$ groups all the input-consuming $\mathcal{P}$-derivations of $P \cup \{\mathbf{Q}\}$ modulo the choices of the renaming of the program clauses used and the choices of the mgu's. Notice that it can happen that a node contains no selectable atom, in which case it has no children.

The following characterisation of input $\mathcal{P}$-termination in terms of IC$\mathcal{P}$-trees is in obvious analogy to [7, Lemma 5.11].

Lemma 4.11 An IC$\mathcal{P}$-tree for $P \cup \{\mathbf{Q}\}$ is finite if and only if all input-consuming consuming $\mathcal{P}$-derivations of $P \cup \{\mathbf{Q}\}$ are finite.

We measure atoms by counting the number of nodes in the corresponding IC$\mathcal{P}$-tree. For a program P and a query $\mathbf{Q}$, we denote by $nodes_P^{ic\mathcal{P}}(\mathbf{Q})$ the number of nodes in an IC$\mathcal{P}$-tree for $P \cup \{\mathbf{Q}\}$. This is undefined if the tree is infinite. The following lemma adapts [7, Lemma 5.12] to IC$\mathcal{P}$-trees.

Lemma 4.12 Let the program P and the query $\mathbf{A}, B$ be simply moded. Suppose that P is input $\mathcal{P}$-terminating and that $\mathbf{A}\theta \subseteq PM_P^{SL\mathcal{P}}$, where θ is a simply local substitution with respect to $\mathbf{A}$. Then $nodes_P^{ic\mathcal{P}}(\mathbf{A}, B) \geq nodes_P^{ic\mathcal{P}}(B\theta)$.

Proof. The proof is obtained from the proof of [7, Lemma 5.12] by using Lemma 3.18 instead of [7, Lemma 4.4] and inserting $\mathcal{P}$ in the obvious way:

Consider an IC$\mathcal{P}$-tree T for $P \cup \{\mathbf{A}, B\}$. By the hypothesis that $\mathbf{A}\theta \subseteq PM_P^{SL\mathcal{P}}$, it follows that there exists a substitution σ such that—by Lemma 3.18 (ii) $\Rightarrow$ (i) letting $I = SM_P$—$\mathbf{A} \xrightarrow{\sigma} \mathbf{C}$ is an input-consuming $\mathcal{P}$-derivation and $\mathbf{A}\theta \approx \mathbf{A}\sigma$. Hence there exists an input-consuming $\mathcal{P}$-derivation $\mathbf{A}, B \xrightarrow{\sigma} \mathbf{C}, B\sigma$ and $B\theta \approx B\sigma$. Clearly, by definition of IC$\mathcal{P}$-tree, $nodes_P^{ic\mathcal{P}}(\mathbf{A}, B) \geq nodes_P^{ic\mathcal{P}}(B\sigma) = nodes_P^{ic\mathcal{P}}(B\theta)$. Hence the thesis. □

Let us give some intuition about the above lemma and [7, Lemma 5.12]. The original lemma states first that if $\mathbf{A}, B$ is simply moded and θ is simply local, then the IC-tree of $B\theta$ is a subtree of the IC-tree of $\mathbf{A}, B$. This is so since θ is a substitution that may be computed by means of an input-consuming derivation. This subtree relationship holds of course even if the IC-tree of $\mathbf{A}, B$ is infinite. Now, assuming in addition that P is input-terminating, the IC-tree of $\mathbf{A}, B$ is in fact finite and so we have $nodes_P^{ic}(\mathbf{A}, B) \geq nodes_P^{ic\mathcal{P}}(B\theta)$. What should be noted here is that the IC$\mathcal{P}$-trees are obtained from the respective IC-trees by cutting away certain branches corresponding to derivations where the selected atoms are not in $\mathcal{P}$: the $\mathcal{P}$ restricts the possible derivations. The subtree relationship holds for the IC$\mathcal{P}$-trees just like it holds for the IC-trees. But the IC$\mathcal{P}$-tree may be *finite* where the respective IC-tree is infinite, and then the inequality $nodes_P^{ic\mathcal{P}}(\mathbf{A}, B) \geq nodes_P^{ic\mathcal{P}}(B\theta)$ will hold where nothing of the like can be said for the IC-trees.

Observe moreover that if in the above lemma, $B\theta \notin \mathcal{P}$, then $nodes_P^{ic\mathcal{P}}(B\theta) = 0$.

Theorem 4.13 Let P be a simply moded program and $\mathcal{P}$ a set of atoms according to Def. 3.1. If P is input $\mathcal{P}$-terminating then P is simply $\mathcal{P}$-acceptable.

In particular, P is simply $\mathcal{P}$-acceptable by $PM_P^{SL\mathcal{P}}$, the least simply local $\mathcal{P}$-model of P containing SM_P.

Proof. We show that there exists a moded level mapping $|.|$ for P such that P is simply $\mathcal{P}$-acceptable by $|.|$ and $PM_P^{SL\mathcal{P}}$.

For an atom $p(\mathbf{s}, \mathbf{t})$, we define the following moded level mapping for P:

$$|p(\mathbf{s}, \mathbf{t})| = nodes_P^{ic\mathcal{P}}(p(\mathbf{s}, \mathbf{y})),$$

where $\mathbf{y}$ is a vector of distinct variables disjoint from $\mathbf{s}$. Note that $p(\mathbf{s}, \mathbf{y})$ is simply moded, and so since P is input $\mathcal{P}$-terminating, all the input-consuming $\mathcal{P}$-derivations of $P \cup \{p(\mathbf{s}, \mathbf{y})\}$ are finite. Therefore, by Lemma 4.11, $nodes_P^{ic\mathcal{P}}(p(\mathbf{s}, \mathbf{y}))$ is defined, and thus $|A|$ is defined for every atom A.

We now prove that P is simply $\mathcal{P}$-acceptable by $|.|$ and $PM_P^{SL\mathcal{P}}$.

Let $c = p(\mathbf{v}, \mathbf{u}) \leftarrow \mathbf{A}, B, \mathbf{C}$ be a clause of P and σ a simply local substitution with respect to c. We show that

$$\text{if } \mathbf{A}\sigma \subseteq PM_P^{SL\mathcal{P}} \text{ and } p(\mathbf{v}, \mathbf{u}) \simeq B \text{ and } p(\mathbf{v}, \mathbf{u})\sigma \in \mathcal{P} \text{ and } B\sigma \in \mathcal{P} \text{ then } |p(\mathbf{v}, \mathbf{u})\sigma| > |B\sigma|, \quad (20)$$

so let us assume that the premises of this implication hold.

Consider a renaming ρ of c and a vector of distinct variables $\mathbf{y}$, of length equal to the number of p's output arguments, such that

$$Dom(\rho) = Vars(c), \quad Ran(\rho) \text{ is fresh}, \quad \mathbf{y} \text{ is fresh}. \quad (21)$$

Fresh means: variable disjoint from any object mentioned in this proof so far.

Then $\rho^{-1}\sigma_{|\mathbf{v}}$ is an mgu[6] of $\mathbf{v}\rho$ and $\mathbf{v}\sigma$, in fact a matcher: $(\mathbf{v}\rho)\rho^{-1}\sigma_{|\mathbf{v}} = \mathbf{v}\sigma$.[7] Moreover, $\{\mathbf{y}/\mathbf{u}\sigma_{|\mathbf{v}}\}$ is an mgu of $\mathbf{y}$ and $(\mathbf{u}\rho)(\rho^{-1}\sigma_{|\mathbf{v}})$, in fact a matcher: $\mathbf{y}\{\mathbf{y}/\mathbf{u}\sigma_{|\mathbf{v}}\} = (\mathbf{u}\rho)(\rho^{-1}\sigma_{|\mathbf{v}})$.

[6] Here, $\rho^{-1}\sigma_{|\mathbf{v}}$ stands for $\rho^{-1}(\sigma_{|\mathbf{v}})$, not $(\rho^{-1}\sigma)_{|\mathbf{v}}$.

[7] We use here that σ is idempotent, since it is a simply local substitution. Otherwise $\rho^{-1}\sigma_{|\mathbf{v}}$ might not be a unifier.

```
even(X) ← minus2(X,Y), even(Y).        minus2(X,s(X)) ← fail.
even(0).                               minus2(s(s(X)),X).
```

Figure 3: The EVEN program

Concluding, $\mu := (\rho^{-1}\sigma_{|\mathbf{v}})\{\mathbf{y}/\mathbf{u}\sigma_{|\mathbf{v}}\}$ is a simply local mgu of $p(\mathbf{v}\sigma, \mathbf{y})$ and $p(\mathbf{v}, \mathbf{u})\rho$. By hypothesis $p(\mathbf{v}, \mathbf{u})\sigma \in \mathcal{P}$, and so by Def. 3.1 (2),

$$p(\mathbf{v}\sigma, \mathbf{y}) \stackrel{\mu}{\Longrightarrow} (\mathbf{A}, B, \mathbf{C})\rho\mu$$

is an input-consuming $\mathcal{P}$-derivation step using clause $c\rho$, that is, $(\mathbf{A}, B, \mathbf{C})\rho\mu$ is a child of $p(\mathbf{v}\sigma, \mathbf{y})$ in the IC$\mathcal{P}$-tree for $P \cup \{p(\mathbf{v}\rho, \mathbf{y})\}$.

Moreover, using (21), it follows that

$$(\mathbf{A}, B, \mathbf{C})\rho\mu = (\mathbf{A}, B, \mathbf{C})\rho(\rho^{-1}\sigma_{|\mathbf{v}})\{\mathbf{y}/\mathbf{u}\sigma_{|\mathbf{v}}\} = (\mathbf{A}, B, \mathbf{C})\sigma_{|\mathbf{v}}\{\mathbf{y}/\mathbf{u}\sigma_{|\mathbf{v}}\} = (\mathbf{A}, B, \mathbf{C})\sigma_{|\mathbf{v}}. \quad (22)$$

Let us decompose σ (as it is simply local) as $\sigma_{|\mathbf{v}}\sigma_{|Out(\mathbf{A})}\sigma_{|Out(B,\mathbf{C})}$ and write B as $p(\mathbf{s}, \mathbf{z})$ (recall that since B is a simply moded atom, it has a vector of distinct variables in its output positions). Hence, by Lemmas 3.10 and 3.11, $\sigma_{|Out(\mathbf{A})}$ is simply local with respect to $\mathbf{A}\sigma_{|\mathbf{v}}$. Therefore, we have that

$$\begin{array}{lcll}
|p(\mathbf{v}, \mathbf{u})\sigma| & = & nodes_P^{ic\mathcal{P}}(p(\mathbf{v}\sigma, \mathbf{y})) & \text{(by definition of } |.|) \\
& > & nodes_P^{ic\mathcal{P}}((\mathbf{A}, p(\mathbf{s}, \mathbf{z}), \mathbf{C})\rho\mu) & \text{(by definition of IC}\mathcal{P}\text{-tree)} \\
& = & nodes_P^{ic\mathcal{P}}((\mathbf{A}, p(\mathbf{s}, \mathbf{z}), \mathbf{C})\sigma_{|\mathbf{v}}) & \text{(by (22))} \\
& \geq & nodes_P^{ic\mathcal{P}}((\mathbf{A}, p(\mathbf{s}, \mathbf{z}))\sigma_{|\mathbf{v}}) & \text{(by definition of IC}\mathcal{P}\text{-tree)} \\
& \geq & nodes_P^{ic\mathcal{P}}((p(\mathbf{s}, \mathbf{z})\sigma_{|\mathbf{v}}\sigma_{|Out(\mathbf{A})}) & \text{(by Lemma 4.12)} \\
& = & nodes_P^{ic\mathcal{P}}(p(\mathbf{s}\sigma, \mathbf{z})) & \text{(since } \sigma \text{ is simply local with respect to } c\text{)} \\
& = & |p(\mathbf{s}, \mathbf{z})\sigma| & \text{(by definition of } |.|\text{).}
\end{array}$$

□

Note that of the four premises in (20), $p(\mathbf{v}, \mathbf{u}) \simeq B$ and $B\sigma \in \mathcal{P}$ were not actually used. Let us define a **simply $\mathcal{P}$-acceptable*** clause as in Def. 4.3 with (18) replaced by

$$B_1\sigma, \ldots, B_{i-1}\sigma \in M \text{ and } H\sigma \in \mathcal{P} \quad \text{imply} \quad |H\sigma| > |B_i\sigma|.$$

By definition, the simply $\mathcal{P}$-acceptable* programs are a subset of the simply $\mathcal{P}$-acceptable programs. The above proof shows that every input $\mathcal{P}$-terminating program is simply $\mathcal{P}$-acceptable*. Theorem 4.6 says that every simply $\mathcal{P}$-acceptable program is input $\mathcal{P}$-terminating, and hence simply $\mathcal{P}$-acceptable* is the same as simply $\mathcal{P}$-acceptable. However, for applying Theorem 4.6, the definition (18) is to be preferred since a decrease must be shown in fewer cases. Also note that we do not claim in general that a program can be shown to be $\mathcal{P}$-acceptable* and $\mathcal{P}$-acceptable by the same norm and model.

5 Local Input-Consuming $\mathcal{P}$-Derivations

In this section, we adapt the results of the two previous sections to local selection rules. First note that local derivations genuinely need special treatment, since one cannot express locality as a property $\mathcal{P}$ of the selected atoms. Note also that *local* and *simply local* (Subsection 3.2) are completely different concepts.

We give an example of a program for which locality is crucial for termination.

Example 5.1 Consider the program in Fig. 3 in mode $\texttt{even}(I), \texttt{minus2}(I, O)$. Let

$$\mathcal{P} = \{\texttt{even}(x) \mid x \text{ is non-variable}\} \cup \{\texttt{minus2}(x, y) \mid x, y \text{ arbitrary}\}$$

(one might as well require the input positions to be ground). Then we have an infinite *input-consuming $\mathcal{P}$-derivation*, which is *not local*:

$$\underline{\texttt{even(0)}} \Longrightarrow \underline{\texttt{minus2(0,Y')}},\ \texttt{even(Y')} \Longrightarrow \texttt{fail},\ \underline{\texttt{even(s(0))}} \Longrightarrow$$
$$\texttt{fail},\ \underline{\texttt{minus2(s(0),Y'')}},\ \texttt{even(Y'')} \Longrightarrow \texttt{fail},\ \texttt{fail},\ \underline{\texttt{even(s(s(0)))}} \Longrightarrow \ldots$$

Assuming local selection rules is helpful for showing termination, since one can exploit model information almost in the same way as for LD derivations [4]. In fact, some arguments are simpler here than in the previous two sections.

We make a simple observation.

Proposition 5.2 Let $\mathbf{A}, B, \mathbf{C} \xrightarrow{\sigma} (\mathbf{A}', B, \mathbf{C}')\sigma \overset{\theta}{\Longrightarrow} (\mathbf{A}', \mathbf{B}, \mathbf{C}')\sigma\theta$ be a local derivation. Then $\mathbf{A}'$ is a subquery of $\mathbf{A}$ and $\mathbf{C}'$ is a subquery of $\mathbf{C}$.

Proof. If $\mathbf{A}'$ is not a subquery of $\mathbf{A}$, it contains a proper descendant of $\mathbf{A}$, which is more recent in the derivation than B. But this is a contradiction to the derivation being local. Likewise for $\mathbf{C}$.

An important observation is that Lemma 3.4 does not generalise to local derivations—switching steps may destroy the locality property. Corollary 4.5 rests on the Left-Switching Lemma. But it turns out that for local derivations, the analogous statement is a corollary of the following lemma.

Lemma 5.3 Let the program P and the query $\mathbf{Q} = A_1, \ldots, A_n$ be simply moded. Suppose there is an infinite local input-consuming $\mathcal{P}$-derivation δ of $P \cup \{\mathbf{Q}\}$. Then for some $i \in [1..n]$, δ has the form

$$\mathbf{Q} \xrightarrow{\sigma} (\mathbf{B}, A_i, \mathbf{C})\sigma \longrightarrow \ldots$$

where the suffix after $(\mathbf{B}, A_i, \mathbf{C})\sigma$ contains only A_i-steps, and there exists an infinite local input-consuming $\mathcal{P}$-derivation of $P \cup \{A_i\sigma\}$.

Proof. Since δ is infinite, it contains an infinite number of A_i-steps for some $i \in [1..n]$. Moreover, since δ is local, there can only be one such i. Hence δ contains only a finite number of A_j-steps for all $j \neq i$, and again since δ is local, each A_j, where $j \neq i$, is either resolved completely (and before the first A_i-step) or not at all. We write $A_1, \ldots, A_n$ as $\mathbf{A}_0, A_{j_1}, \mathbf{A}_1, \ldots, A_{j_k}, \mathbf{A}_k$ where $A_{j_1}, \ldots, A_{j_k}$ are the resolved atoms and $\mathbf{A}_0, \ldots, \mathbf{A}_k$ are the sequences of unresolved atoms in between. Note that $i = j_l$ for some $l \in [1..k]$ since A_i is (partially) resolved. Then δ has the desired form

$$A_1, \ldots, A_n \xrightarrow{\sigma} \underbrace{(\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_{l-1}, A_{j_l}, \mathbf{A}_l, \ldots, \mathbf{A}_k)\sigma \longrightarrow \ldots}_{=:\delta'}$$

where δ' contains only A_{j_l}-, that is, A_i-steps. But then an infinite local input-consuming $\mathcal{P}$-derivation of $A_i\sigma$ is obtained from δ' simply by removing the sequences $\mathbf{A}_0, \ldots, \mathbf{A}_k$. □

We now adapt the results concerning models to the case of local derivations. To reflect the substitutions that can be computed by local derivations, we need the union of a simply local $\mathcal{P}$-model (for the completely resolved atoms) and the set of simply moded atoms (for the unresolved atoms).

Let M be the least simply local $\mathcal{P}$-model of P, that is, $T_P^{SL\mathcal{P}} \uparrow \omega(\emptyset)$. We define $LM_P^{SL\mathcal{P}} := SM_P \cup M$. So $LM_P^{SL\mathcal{P}}$ contains SM_P, but unlike $PM_P^{SL\mathcal{P}}$, does not involve applications of $T_P^{SL\mathcal{P}}$ to atoms in SM_P.

Example 5.4 Consider again Ex. 5.1. We have

$$\begin{aligned} SM_P &= \{\texttt{even}(x),\ \texttt{minus2}(x,\texttt{Z}),\ \texttt{fail} \mid x \text{ arbitrary where } \texttt{Z} \notin \mathit{Vars}(x)\} \\ PM_P^{SL\mathcal{P}} &= SM_P \cup \{\texttt{even(0)},\ \texttt{minus2}(\texttt{s}(\texttt{s}(x)),x),\ \texttt{minus2}(x,\texttt{s}(x)), \\ &\qquad \texttt{even}(\texttt{s}(\texttt{s}(x))),\ \texttt{even}(x) \mid x \text{ arbitrary}\}. \end{aligned}$$

The minimal simply local model of P, not containing SM_P, is the following:

$$M = \{\texttt{even}(\texttt{s}^{2n}(\texttt{0})),\ \texttt{minus2}(\texttt{s}(\texttt{s}(x)), x) \mid n \in \mathbb{N}_0,\ x \text{ arbitrary}\}.$$

Then $LM_P^{SL\mathcal{P}} = SM_P \cup M$. In contrast to $PM_P^{SL\mathcal{P}}$ we have $\texttt{minus2}(x, \texttt{s}(x)) \notin LM_P^{SL\mathcal{P}}$. This reflects that in a local derivation, resolving an atom with the clause head $\texttt{minus2}(\texttt{X}, \texttt{s}(\texttt{X}))$ will definitely lead to finite failure. For this program, locality is crucial for termination (see point 5 in Sec. 1).

The example is contrived since the first clause for `minus2` is completely unnecessary, yet natural enough to suggest that there might be a "real" example.

The following lemma adapts Lemma 3.18 to local derivations.

Lemma 5.5 Let the program P and the query $\mathbf{A}$ be simply moded and $I \subseteq SM_P$ be an interpretation, and M be the least simply local $\mathcal{P}$-model of P. Then the following statements are equivalent:

(i) there exists a local input-consuming $\mathcal{P}$-derivation $\delta = \mathbf{A} \xrightarrow{\sigma} \mathbf{C}\sigma$ where $\mathbf{C}\sigma \subseteq I$ and $\mathbf{C}$ is a subquery of $\mathbf{A}$,

(ii) there exists a substitution θ simply local with respect to $\mathbf{A}$, such that $\mathbf{A}\theta \subseteq M \cup I$,

where $\mathbf{A}\sigma \approx \mathbf{A}\theta$. For the "(i) ⇒ (ii)" direction, we can choose $\theta = \sigma_{|\mathbf{A}}$.

Proof. (i) ⇒ (ii). By the assumption that $\mathbf{C}$ is a subquery of $\mathbf{A}$, all atoms in $\mathbf{A}$ are resolved completely or not at all. We write $\mathbf{A}$ as $\mathbf{A}_0, A_1, \mathbf{A}_1, \ldots, A_n, \mathbf{A}_n$ where $A_1, \ldots, A_n$ are the resolved atoms and $\mathbf{A}_0, \ldots, \mathbf{A}_n$ are the sequences of unresolved atoms in between. Then $\mathbf{C} = (\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_n)$.

By Cor. 3.5, setting $\mathbf{B}$ and $\mathbf{C}_1$ to $\Box$, there is an (local, but this is not used) input-consuming $\mathcal{P}$-derivation

$$\delta' = \mathbf{A}_0, A_1, \mathbf{A}_1, \ldots, A_n, \mathbf{A}_n \xrightarrow{\sigma_1} (\mathbf{A}_0, \mathbf{A}_1, A_2, \ldots, A_n, \mathbf{A}_n)\sigma_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_n} \mathbf{C}\sigma_1 \cdots \sigma_n$$

which proceeds left-to-right, and $\sigma = \sigma_1 \cdots \sigma_n$.

Now for each $i \in [1..n]$, there exists an input-consuming $\mathcal{P}$-derivation $A_i\sigma_1 \ldots \sigma_{i-1} \xrightarrow{\sigma_i} \Box\sigma_i$, and so by Lemma 3.18 (i) ⇒ (ii), $A_i\sigma_1 \ldots \sigma_i \in M$. By Prop. 3.2, $A_i\sigma_1 \ldots \sigma_i = A_i\sigma$, and so $A_i\sigma \in M$. Moreover, by hypothesis, $\mathbf{C}\sigma \subseteq I$, that is, for each $i \in [0..n]$, $\mathbf{A}_i\sigma \subseteq I$.

Finally, by (the proof of) Lemma 3.18 (i) ⇒ (ii), $\sigma_{|\mathbf{A}}$ is simply local with respect to $\mathbf{A}$, and so the claim follows.

(ii) ⇒ (i). The proof is by induction on the number n of atoms of $\mathbf{A}$.

Base: $n = 1$. In this case $\mathbf{A} = A$ and θ is simply local with respect to A. We distinguish whether $A\theta \in M$ or $A\theta \in I$.

Suppose $A\theta \in M$. Then by Lemma 3.18 (ii) ⇒ (i), there exists an input-consuming $\mathcal{P}$-derivation $\delta = \mathbf{A} \xrightarrow{\sigma} \Box\sigma$. By Cor. 3.5, there exists an input-consuming $\mathcal{P}$-derivation $\delta' = \mathbf{A} \xrightarrow{\sigma} \Box\sigma$ which proceeds left-to-right, which implies that δ' is in fact an LD derivation and hence a local derivation. Moreover, clearly $\Box$ is a subquery of A.

Now suppose $A\theta \in I$. Since both A and $A\theta$ are simply moded atoms, and θ is a simply local substitution with respect to A, it follows that θ is just a renaming of the output variables of A, and so since I is closed under variance, $A \in I$. The thesis follows by taking σ to be the empty substitution and δ to be the derivation of length zero, which is trivially a local input-consuming $\mathcal{P}$-derivation.

Induction step: $n > 1$. In this case $\mathbf{A} = A, \mathbf{R}$. By Lemma 3.10 there exist θ_1 and θ_2 such that $\theta = \theta_1\theta_2$, $\theta_1 = \theta_{|A}$ is simply local with respect to A and θ_2 is simply local with respect to $\mathbf{R}\theta_1$, and θ_1 and θ_2 are variable compatible with respect to A and $\mathbf{R}$.

By the inductive hypothesis, there exists a local input-consuming $\mathcal{P}$-derivation $\delta_1 = A \xrightarrow{\sigma_1} \mathbf{C}_1\sigma_1$ where $\mathbf{C}_1\sigma_1 \subseteq I$ and $\mathbf{C}_1$ is a subquery of A, that is, either $\mathbf{C}_1 = \Box$ or $\mathbf{C}_1 = A$, and $A\sigma_1$ and $A\theta_{|A} = A\theta$ are variant.

Again by the inductive hypothesis on n (skipping the reasoning about $\mathbf{R}\theta_1$ and $\mathbf{R}\sigma_1$ being variants, which works exactly as in the proof of Lemma 3.18), there exists a local input-consuming $\mathcal{P}$-derivation $\delta_2 = \mathbf{R}\sigma_1 \xrightarrow{\sigma_2} \mathbf{C}_2\sigma_1\sigma_2$ where $\mathbf{C}_2\sigma_1\sigma_2 \subseteq M \cup I$, and $\mathbf{C}_2$ is a subquery of $\mathbf{R}$. By composing the two derivations, one obtains an input-consuming $\mathcal{P}$-derivation

$$\delta = A, \mathbf{R} \xrightarrow{\sigma_1} (\mathbf{C}_1, \mathbf{R})\sigma_1 \xrightarrow{\sigma_2} (\mathbf{C}_1, \mathbf{C}_2)\sigma_1\sigma_2$$

which is clearly local. Moreover, by a reasoning that works exactly as in the proof of Lemma 3.18, $\mathbf{A}\sigma_1\sigma_2 \approx \mathbf{A}\theta$. □

6 Termination of Local Derivations

6.1 Local $\mathcal{P}$-Acceptable Programs

Definition 6.1 A program is **local $\mathcal{P}$-terminating** if all local input-consuming $\mathcal{P}$-derivations starting in a simply-moded query are finite.

We now give a sufficient and necessary criterion for local $\mathcal{P}$-termination.

Definition 6.2 Let P be a simply moded program, $|.|$ a moded level mapping and M a superset of $LM_P^{SL\mathcal{P}}$. A clause $H \leftarrow B_1, \ldots, B_n$ is **local $\mathcal{P}$-acceptable by $|.|$ and M** if for every substitution σ simply local with respect to it, for all $i \in [1..n]$,

$$B_1\sigma, \ldots, B_{i-1}\sigma \in M \text{ and } H \simeq B_i \text{ and } H\sigma \in \mathcal{P} \text{ and } B_i\sigma \in \mathcal{P} \quad \text{imply} \quad |H\sigma| > |B_i\sigma|. \quad (23)$$

The program P is **local $\mathcal{P}$-acceptable by $|.|$ and M** if each clause of P is local $\mathcal{P}$-acceptable by $|.|$ and M.

Note that the above definition differs from the one in [22] in that the condition $H\sigma \in \mathcal{P}$ has been added. This is crucial for showing *necessity* (Subsec. 6.3). In [22], necessity is not claimed.

Example 6.3 Consider again the program in Fig. 3 and Ex. 5.4, in particular the recursive clause. Let $\mathcal{P}$ be the set of atoms where all input arguments are non-variable and $|\mathtt{even}(x)| = |\mathtt{minus2}(x, y)| = |x|$ where $|.|$ is the term size norm. We verify that the second body atom fulfils the requirement of Def. 6.2, taking $M = LM_P^{SL\mathcal{P}}$. We have to consider all simply local σ such that $\mathtt{minus2}(\mathtt{X}, \mathtt{Y})\sigma \in LM_P^{SL\mathcal{P}}$. So

$$\sigma \supseteq \{\mathtt{X}/x, \mathtt{Y}/\mathtt{Z}\} \quad \text{or} \quad \sigma \supseteq \{\mathtt{X}/\mathtt{s}(\mathtt{s}(x)), \mathtt{Y}/x\}.$$

In the first case, $\mathtt{even}(\mathtt{Y})\sigma \notin \mathcal{P}$ and hence no proof obligation arises. In the second case, $|\mathtt{even}(\mathtt{X})\sigma| > |\mathtt{even}(\mathtt{Y})\sigma|$. Hence the clause is local $\mathcal{P}$-acceptable. Note that the clause is not simply $\mathcal{P}$-acceptable (due to $\mathtt{minus2}(x, \mathtt{s}(x)) \in PM_P^{SL\mathcal{P}}$).

Observe that unlike [14], we do not require that the selected atoms must be bounded. In our formalism, the instantiation requirements of the selected atom and the locality issue are two separate dimensions.

6.2 Sufficiency of Local $\mathcal{P}$-Acceptability

We now give the promised corollary of Lemma 5.3.

Corollary 6.4 Let P be a simply moded program. P is local $\mathcal{P}$-terminating if and only if for each simply moded atomic query A all local input-consuming $\mathcal{P}$-derivations of $P \cup \{A\}$ are finite.

Theorem 6.5 Let P be a simply moded program. Let M be a superset of $LM_P^{SL\mathcal{P}}$. Suppose that P is local $\mathcal{P}$-acceptable by a moded level mapping $|.|$ and M. Then P is local $\mathcal{P}$-terminating.

Proof. We define $dep(p)$ as in the proof of Thm. 4.6. By Cor. 6.4, it is sufficient to prove that for any simply moded atomic query A, all local input-consuming $\mathcal{P}$-derivations of $P \cup \{A\}$ are finite.

For the purpose of deriving a contradiction, assume that δ is an infinite local input-consuming $\mathcal{P}$-derivation of $P \cup \{A\}$. Then, spelling out the first step and using Lemma 5.3, for some $i \in [1..n]$, we have

$$\delta = A \stackrel{\sigma_1}{\Longrightarrow} (B_1, \ldots, B_n)\sigma_1 \stackrel{\sigma_2}{\longrightarrow} (\mathbf{A}, B_i, \mathbf{C})\sigma_1\sigma_2 \longrightarrow \ldots,$$

where $c = H \leftarrow B_1, \ldots, B_n$ is the clause used in the first step, and

$$\text{there is an infinite local input-consuming } \mathcal{P}\text{-derivation of } P \cup \{B_i\sigma_1\sigma_2\}. \tag{24}$$

Let $\sigma = (\sigma_1\sigma_2)_{|c}$. Using [7, Proposition A.1], one can easily show that σ is simply local with respect to c.

By the persistence of simply-modedness [7, Lemma 2.7], $(\mathbf{A}, B_i, \mathbf{C})\sigma_1\sigma_2$ is simply moded, and so letting $I = SM_P$, by Lemma 5.5 (i) $\Rightarrow$ (ii), since M is a superset of $LM_P^{SL\mathcal{P}}$ and hence of both SM_P and of the least simply local $\mathcal{P}$-model of P, we have $(B_1, \ldots, B_{i-1})\sigma \subseteq M$.

Define $\succ$ as in the proof of Thm. 4.6. We will show that $\langle dep(A), |A|\rangle \succ \langle dep(B_i\sigma), |B_i\sigma|\rangle$. By an obvious inductive argument, this contradicts (24).

As in the proof of Thm. 4.6 we have $|A| = |A\sigma| = |H\sigma|$, and we distinguish two cases: *(a)* $H \sqsupset B_i$, and *(b)* $H \simeq B_i$.

In case *(a)* we have $dep(A) = dep(H\sigma) > dep(B_i\sigma)$. Therefore,

$$\langle dep(A), |A|\rangle = \langle dep(H\sigma), |H\sigma|\rangle \succ \langle dep(B_i\sigma), |B_i\sigma|\rangle. \tag{25}$$

In case *(b)*, from the hypothesis that P is local $\mathcal{P}$-acceptable by $|.|$ and M, σ is simply local with respect to c and $(B_1, \ldots, B_{i-1})\sigma \subseteq M$, and moreover $B_i\sigma \in \mathcal{P}$, it follows that $|H\sigma| > |B_i\sigma|$. Hence, (25) holds also in this case.

Now consider case *(b)*. P is local $\mathcal{P}$-acceptable by $|.|$ and M. We shall check that the premises in (23) are fulfilled. We have already stated that σ is simply local with respect to c and that $(B_1, \ldots, B_{i-1})\sigma \subseteq M$. Now since $H\sigma_1 = A\sigma_1$ and $A\sigma_1$ was selected in an input-consuming $\mathcal{P}$-derivation step, and $\sigma = \sigma_1\sigma_2$, by Def. 3.1 (1) it follows that $H\sigma \in \mathcal{P}$. Finally, by (24) $B_i\sigma \in \mathcal{P}$. Thus it follows that $|H\sigma| > |B_i\sigma|$. Hence, (19) holds also in this case. □

6.3 Necessity of Local $\mathcal{P}$-Acceptability

We now prove the converse of Theorem 6.5, namely that our criterion for proving local $\mathcal{P}$-termination is also necessary. We proceed in analogy to Subsec. 4.3.

To construct trees that capture the local derivations of a query, we have to label each atom in a query with a natural number to record its "age." We put a superscript to atoms for this purpose. For a query $\mathbf{Q}$, the notation $\mathbf{Q}^n$ means that all atoms of $\mathbf{Q}$ are labelled n.

Definition 6.6 Let P be a program and $\mathbf{Q}$ be a query. A **LIC$\mathcal{P}$-tree** for $P \cup \{\mathbf{Q}\}$ is a tree such that

- its root is $\mathbf{Q}^0$,
- every node $\mathbf{Q}'$ has exactly one child for every atom A^n of $\mathbf{Q}'$, where n is a maximal label in $\mathbf{Q}'$, and every clause c in P such that there is an input-consuming $\mathcal{P}$-resolvent of $\mathbf{Q}'$ with respect to A and c. The child is the resolvent, where the new atoms are labelled $n+1$ and the old atoms are labelled as in $\mathbf{Q}'$.

The selected atom must always have a maximal label, that is, it must be among the most recently introduced atoms. This clearly models local derivations.

Lemma 6.7 An LIC$\mathcal{P}$-tree for $P \cup \{\mathbf{Q}\}$ is finite iff all local input-consuming consuming $\mathcal{P}$-derivations of $P \cup \{\mathbf{Q}\}$ are finite.

For a program P and a query $\mathbf{Q}$, we denote by $nodes_P^{lic\mathcal{P}}(\mathbf{Q})$ the number of nodes in an LIC$\mathcal{P}$-tree for $P \cup \{\mathbf{Q}\}$. This is undefined if the tree is infinite. The following lemma adapts Lemma 4.12 to LIC$\mathcal{P}$-trees.

Lemma 6.8 Let the program P and the query $\mathbf{A}, B$ be simply moded. Suppose that P is local $\mathcal{P}$-terminating and that $\mathbf{A}\theta \subseteq LM_P^{SL\mathcal{P}}$, where θ is a simply local substitution with respect to $\mathbf{A}$. Then $nodes_P^{lic\mathcal{P}}(\mathbf{A}, B) \geq nodes_P^{lic\mathcal{P}}(B\theta)$.

Proof. Consider an LIC$\mathcal{P}$-tree T for $P \cup \{\mathbf{A}, B\}$. By the hypothesis that $\mathbf{A}\theta \subseteq LM_P^{SL\mathcal{P}}$, it follows that there exists a substitution σ such that—by Lemma 5.5 (ii) $\Rightarrow$ (i) letting $I = SM_P$—$\mathbf{A} \xrightarrow{\sigma} \mathbf{C}\sigma$ is a local input consuming $\mathcal{P}$-derivation where $\mathbf{C}$ is a subquery of $\mathbf{A}$ and $\mathbf{A}\theta \approx \mathbf{A}\sigma$. Hence there exists a local input-consuming $\mathcal{P}$-derivation $\mathbf{A}, B \xrightarrow{\sigma} (\mathbf{C}, B)\sigma$ and $B\theta \approx B\sigma$. By the construction of LIC$\mathcal{P}$-trees, T contains a node $(\mathbf{C}, B)\sigma$, and since $\mathbf{C}, B$ is a subquery of $\mathbf{A}, B$, all atoms in the node $(\mathbf{C}, B)\sigma$ are labelled 0. Therefore the subtree with root $(\mathbf{C}, B)\sigma$ is an LIC$\mathcal{P}$-tree for $(\mathbf{C}, B)\sigma$. Now the LIC$\mathcal{P}$-tree for $B\sigma$ is obtained from the LIC$\mathcal{P}$-tree for $(\mathbf{C}, B)\sigma$ by cutting away all branches where an atom in $\mathbf{C}$ is resolved and removing the prefix $\mathbf{C}$ in the remaining nodes. Consequently, $nodes_P^{lic\mathcal{P}}(\mathbf{A}, B) \geq nodes_P^{lic\mathcal{P}}(B\sigma) = nodes_P^{lic\mathcal{P}}(B\theta)$. Hence the thesis. □

The crucial point is that the node $(\mathbf{C}, B)\sigma$ has all atoms labelled 0. If it contained an atom with a label greater than 0, it would be wrong to claim that the tree for $B\sigma$ is a subtree of the tree for $(\mathbf{C}, B)\sigma$, since the children of $(\mathbf{C}, B)\sigma$ could not contain resolvents of $B\sigma$.

Theorem 6.9 Let P be a simply moded program and $\mathcal{P}$ a set of atoms according to Def. 3.1. If P is local $\mathcal{P}$-terminating then P is local $\mathcal{P}$-acceptable.

In particular, P is local $\mathcal{P}$-acceptable by $LM_P^{SL\mathcal{P}}$.

Proof. We proceed in analogy to the proof of Theorem 4.13 so we refer to that proof to keep the present proof concise.

For an atom $p(\mathbf{s}, \mathbf{t})$, we define $|p(\mathbf{s}, \mathbf{t})| = nodes_P^{lic\mathcal{P}}(p(\mathbf{s}, \mathbf{y}))$, in analogy to $nodes_P^{ic\mathcal{P}}(p(\mathbf{s}, \mathbf{y}))$. By Lemma 6.7, in analogy to Lemma 4.11, $|A|$ is defined for every atom A. We now prove that P is local $\mathcal{P}$-acceptable by $|.|$ and $LM_P^{SL\mathcal{P}}$.

Let $c = p(\mathbf{v}, \mathbf{u}) \leftarrow \mathbf{A}, B, \mathbf{C}$ be a clause of P and σ a simply local substitution with respect to c. We show that

$$\text{if } \mathbf{A}\sigma \subseteq LM_P^{SL\mathcal{P}} \text{ and } p(\mathbf{v}, \mathbf{u}) \simeq B \text{ and } p(\mathbf{v}, \mathbf{u})\sigma \in \mathcal{P} \text{ and } B\sigma \in \mathcal{P} \text{ then } |p(\mathbf{v}, \mathbf{u})\sigma| > |B\sigma|, \quad (26)$$

so let us assume that the premises of this implication hold.

Consider a renaming ρ of c and a vector $\mathbf{y}$, and let $\mu = (\rho^{-1}\sigma_{|\mathbf{v}})\{\mathbf{y}/\mathbf{u}\sigma_{|\mathbf{v}}\}$, just as in the proof of Theorem 4.13. Then

$$p(\mathbf{v}\sigma, \mathbf{y}) \overset{\mu}{\Longrightarrow} (\mathbf{A}, B, \mathbf{C})\rho\mu$$

is a *local* input-consuming $\mathcal{P}$-derivation step using clause $c\rho$, that is, $(\mathbf{A}^1, B^1, \mathbf{C}^1)\rho\mu$ is a child of $p(\mathbf{v}\sigma, \mathbf{y})^0$ in the LIC$\mathcal{P}$-tree for $P \cup \{p(\mathbf{v}\rho, \mathbf{y})\}$. The LIC$\mathcal{P}$-tree for $(\mathbf{A}, B, \mathbf{C})\rho\mu$ can be obtained from the tree with root $(\mathbf{A}^1, B^1, \mathbf{C}^1)\rho\mu$ by decreasing each label by 1. Hence

$$nodes_P^{lic\mathcal{P}}(p(\mathbf{v}\sigma, \mathbf{y})) = nodes_P^{lic\mathcal{P}}((\mathbf{A}, B, \mathbf{C})\rho\mu) + 1 \quad (27)$$

Proceeding in analogy to the proof of Theorem 4.13, we have

$$\begin{array}{lcll}
|p(\mathbf{v},\mathbf{u})\sigma| & = & nodes_P^{lic\mathcal{P}}(p(\mathbf{v}\sigma,\mathbf{y})) & \text{(by definition of } |.|) \\
& > & nodes_P^{lic\mathcal{P}}((\mathbf{A},p(\mathbf{s},\mathbf{z}),\mathbf{C})\rho\mu) & \text{(by (27))} \\
& = & nodes_P^{lic\mathcal{P}}((\mathbf{A},p(\mathbf{s},\mathbf{z}),\mathbf{C})\sigma_{|\mathbf{v}}) & \text{(by (22))} \\
& \geq & nodes_P^{lic\mathcal{P}}((\mathbf{A},p(\mathbf{s},\mathbf{z}))\sigma_{|\mathbf{v}}) & \text{(by definition of LIC}\mathcal{P}\text{-tree)} \\
& \geq & nodes_P^{lic\mathcal{P}}((p(\mathbf{s},\mathbf{z})\sigma_{|\mathbf{v}}\sigma_{|Out(\mathbf{A})}) & \text{(by Lemma 6.8)} \\
& = & |p(\mathbf{s},\mathbf{z})\sigma|. &
\end{array}$$

□

7 Conclusion

We have presented a framework for proving termination of logic programs with dynamic scheduling. We have considered various assumptions about the selection rule, in addition to the assumption that derivations must be input-consuming. On the one hand, derivations can be restricted by giving a property $\mathcal{P}$ that the selected atoms must fulfil. On the other hand, derivations may or may not be required to be local. These aspects can be combined freely. We now refer back to the five points in the Introduction.

Some programs terminate under an assumption about the selection rule that is just slightly stronger than assuming input-consuming derivations (point 1). Others need what we call *strong* assumptions: the selected atom must be bounded with respect to a level mapping (point 2). Different versions of `PERMUTE`, which is the standard example of a program that tends to loop for dynamic scheduling [18], are representatives of these program classes. Then there are programs for which one should make hybrid assumptions about the selection rule: depending on the predicate, an atom should be bounded in its input positions or not (point 3). Considering our work together with [7], it is no longer true that "the termination behaviour of 'delay until nonvar' is poorly understood" [14].

The authors of [14] have assumed local selection rules. The question of whether there are programs for which this assumption is genuinely needed for termination is not answered in [14]. We answer it affirmatively (point 5). When is locality needed for termination? Abstractly, this is the case for a query $A_1,\ldots,A_n$ where for some atom A_i and some clause c, the subderivations associated with A_i and c all fail, but at the same time, the unification between A_i and c's head produces a substitution that may trigger an infinite derivation for some atom A_j, where $j > i$. In this case, locality ensures failure of A_i before the infinite derivation of A_j can happen. The main technical aspect of the treatment of local derivations is the notion of model: we take into account that atoms are either resolved completely or not at all (see Ex. 5.4).

On the other hand, we are not aware of a *realistic* program where the locality assumption is crucial (point 4). As an obvious consequence, we have no realistic program that we can show to terminate for local derivations and the method of [14] cannot. But the better understanding of the role of locality may direct the search for such an example.

Both termination criteria given in this report are sufficient and necessary for their respective notions of termination. For local derivations, the necessity is an important advance over [14], since the criterion given there is known not to be necessary.

The concepts of *input-consuming derivations* and $\mathcal{P}$-derivations are both meant to be abstract descriptions of dynamic scheduling. Delay declarations that check for arguments being at least non-variable, or at least non-variable in some sub-argument [12, 24], are often adequate for ensuring input-consuming derivations with $\mathcal{P}$ stating that the input arguments are at least non-variable (see Ex. 4.8). Delay declarations that check for groundness are adequate for ensuring boundedness of atoms (see Ex. 4.4). In general groundness is stronger than boundedness, but we are not aware of delay declarations that could check for boundedness, for example check for a list being nil-terminated. This deficiency has been mentioned previously [13]. Hybrid selection rules can be realised with delay declarations combined with the default left-to-right selection rule (see Ex. 4.9).

Concerning automation of our method, the problems are not so different from the ones encountered when proving left-termination: we have to reason about infinite models—to do so, abstract interpretation approaches, where terms are abstracted as their norms, may be useful [11, 16]. It seems that in our case automation is additionally complicated because we have to consider infinitely many simply local substitutions. But looking at Ex. 3.17, we have terms $y_1, y_2, \ldots$ that are arbitrary and whose form does not affect the termination problem. Therefore it might be sufficient to consider *most general* substitutions in applications of $T_P^{SL\mathcal{P}}$.

Another topic for future work is, of course, a practical evaluation, looking at a larger program suite. In this context, it would be desirable to infer a $\mathcal{P}$, as unrestrictive as possible, automatically. Also, we should consider the following issues: (1) possible generalisations of the results in Sections 5 and 6, concerning derivations that are local but not necessarily input-consuming; (2) a termination criterion that would capture programs that terminate for certain (intended) queries, but not for all queries; (3) relaxing the condition of simply moded programs.

Acknowledgements

The author thanks Lee Naish for useful discussions about this work. Felix Klaedtke has inspired Ex. 5.1 driven by healthful skepticism about statements of the form "there is no natural X such that ...."

References

[1] K. R. Apt. *From Logic Programming to Prolog*. Prentice Hall, 1997.

[2] K. R. Apt and S. Etalle. On the unification free Prolog programs. In A. Borzyszkowski and S. Sokolowski, editors, *Proc. of the 18th International Symposium on Mathematical Foundations of Computer Science*, volume 711 of *LNCS*, pages 1–19. Springer-Verlag, 1993.

[3] K. R. Apt and I. Luitjes. Verification of logic programs with delay declarations. In V. S. Alagar and M. Nivat, editors, *Proc. of the 4th International Conference on Algebraic Methodology and Software Technology*, volume 936 of *LNCS*, pages 66–90. Springer-Verlag, 1995.

[4] K. R. Apt and D. Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106(1):109–157, 1993.

[5] 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.

[6] A. Bossi, S. Etalle, and S. Rossi. Properties of input-consuming derivations. *Theory and Practice of Logic Programming*, 2(2):125–154, 2002.

[7] A. Bossi, S. Etalle, S. Rossi, and J.-G. Smaus. Semantics and termination of simply moded logic programs with dynamic scheduling. *Transactions on Computational Logic*, 5(3):470–507, 2004.

[8] D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, 19/20:199–260, 1994.

[9] N. Dershowitz, N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. A general framework for automatic termination analysis of logic programs. *Applicable Algebra in Engineering, Communication and Computing*, 2001(1/2):117–156, 2001.

[10] S. Etalle, A. Bossi, and N. Cocco. Termination of well-moded programs. *Journal of Logic Programming*, 38(2):243–257, 1999.

[11] S. Genaim, M. Codish, J. Gallagher, and V. Lagoon. Combining norms to prove termination. In A. Cortesi, editor, *Proc. of the 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation*, volume 2294 of *LNCS*, pages 126–138. Springer-Verlag, 2002.

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

[13] S. Lüttringhaus-Kappel. Control generation for logic programs. In D. S. Warren, editor, *Proceedings of the 10th International Conference on Logic Programming*, pages 478–495. MIT Press, 1993.

[14] E. Marchiori and F. Teusink. On termination of logic programs with delay declarations. *Journal of Logic Programming*, 39(1-3):95–124, 1999.

[15] J. Martin and A. King. Generating efficient, terminating logic programs. In M. Bidoit and M. Dauchet, editors, *Proc. of the 7th International Conference on Theory and Practice of Software Development*, volume 1214 of *LNCS*, pages 273–284. Springer-Verlag, 1997.

[16] F. Mesnard and U. Neumerkel. Applying static analysis techniques for inferring termination conditions of logic programs. In P. Cousot, editor, *Proc. of the 8th Static Analysis Symposium*, volume 2126 of *LNCS*, pages 93–110. Springer-Verlag, 2001.

[17] F. Mesnard, É. Payet, and U. Neumerkel. Detecting optimal termination conditions of logic programs. In M. V. Hermenegildo and G. Puebla, editors, *Proc. of the 9th Static Analysis Symposium*, volume 2477 of *LNCS*, pages 509–526. Springer-Verlag, 2002.

[18] L. Naish. Coroutining and the construction of terminating logic programs. Technical Report 92/5, Department of Computer Science, University of Melbourne, 1992.

[19] D. Pedreschi, S. Ruggieri, and J.-G. Smaus. Characterisations of termination in logic programming. In M. Bruynooghe and K. Lau, editors, *Program Development in Computational Logic*, volume 3049 of *LNCS*, pages 376–431. Springer-Verlag, 2004.

[20] J.-G. Smaus. Proving termination of input-consuming logic programs. In D. De Schreye, editor, *Proc. of the International Conference on Logic Programming*, pages 335–349. MIT Press, 1999.

[21] J.-G. Smaus. Termination of logic programs for various dynamic selection rules. Technical Report 191, Institut für Informatik, Universität Freiburg, 2003.

[22] J.-G. Smaus. Termination of logic programs using various dynamic selection rules. In B. Demoen and V. Lifschitz, editors, *Proceedings of the 20th International Conference on Logic Programming*, volume 3132 of *LNCS*, pages 43–57. Springer-Verlag, 2004.

[23] J.-G. Smaus, P. M. Hill, and A. M. King. Verifying termination and error-freedom of logic programs with `block` declarations. *Theory and Practice of Logic Programming*, 1(4):447–486, 2001.

[24] Swedish Institute of Computer Science. *SICStus Prolog User's Manual*, 2003. `http://www.sics.se/isl/sicstuswww/site/documentation.html`.