ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
CONSIGLIO NAZIONALE DELLE RICERCHE

A. Pettorossi, M. Proietti, S. Renault

DERIVATION OF EFFICIENT LOGIC PROGRAMS BY SPECIALIZATION AND REDUCTION OF NONDETERMINISM

R. 551 Luglio 2001

Alberto Pettorossi – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy, and Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy.
Email : adp@iasi.rm.cnr.it. URL : http://www.iasi.rm.cnr.it/~adp.

Maurizio Proietti – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy. Email : proietti@iasi.rm.cnr.it.
URL : http://www.iasi.rm.cnr.it/~proietti.

Sophie Renault – European Patent Office, Rijswijk, The Netherlands.
Email : srenault@epo.org.

A preliminary version of this paper appears as: Reducing Nondeterminism while Specializing Logic Programs. *Proceedings of the 24th Annual ACM Symposium on Principles of Programming Languages, Paris, France, January 15–17, 1997*, ACM Press, 1997, pp. 414–427.

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica, CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.rm.cnr.it
URL: http://www.iasi.rm.cnr.it

Abstract

Program specialization is a program transformation methodology which improves program efficiency by exploiting the information about the input data which are available at compile time. We show that current techniques for program specialization based on partial evaluation do not perform well on nondeterministic logic programs. We then consider a set of transformation rules which extend the ones used for partial evaluation, and we propose a strategy for guiding the application of these extended rules so to derive very efficient specialized programs. The efficiency improvements which sometimes are exponential, are due to the reduction of nondeterminism and to the fact that the computations which are performed by the initial programs in different branches of the computation trees, are performed by the specialized programs within single branches. In order to reduce nondeterminism we also make use of mode information for guiding the unfolding process. To exemplify our technique, we show that we can automatically derive very efficient semideterministic matching programs and semideterministic parsers for regular languages. The derivations we have performed could not have been done by previously known partial evaluation techniques.

Key words: Automatic program derivation, program transformation, logic programming, transformation rules and strategies

1. Introduction

The goal of *program specialization* [15] is the adaptation of a generic program to a specific context of use. *Partial evaluation* [4, 15] is a well established technique for program specialization which from a program and its *static input* (that is, the portion of the input which is known at compile time), allows us to derive a new, more efficient program in which the portion of the output which depends on the static input, has already been computed. Partial evaluation has been applied in several areas of computer science, and it has been applied also to logic programs [10, 20, 23], where it is also called *partial deduction*. In this paper we follow a *rule-based* approach to program specialization. In particular, we consider definite logic programs [22] and we propose new program specialization techniques based on unfold/fold transformation rules [3, 37]. In our approach, the process of program specialization can be viewed as the construction of a sequence, say $P_0, \ldots, P_n$, of programs, where P_0 is the program to be specialized, P_n is the specialized program, and every program of the sequence is obtained from the previous one by applying a transformation rule.

As shown in [29, 34], partial evaluation of logic programs can be viewed as a particular rule-based program transformation technique using the familiar definition, unfolding, and folding rules [37] with the following two restrictions: (i) each new predicate introduced by the definition rule is defined by precisely one non-recursive clause whose body consists of precisely one atom (in this sense, according to the terminology of [13], partial evaluation of logic programs is said to be *monogenetic*), and (ii) the folding rule uses only clauses introduced by the definition rule. In what follows the definition and folding rules which comply with restrictions (i) and (ii), are called *atomic definition* and *atomic folding*, respectively.

In Section 2 we will see that the use of these restricted transformation rules makes it easier to automate the partial evaluation process, but it may limit the program improvements which can be achieved during program specialization. In particular, when we perform partial evaluation of nondeterministic programs using atomic definition, unfolding, and atomic folding, it is impossible to combine information present in different branches of the computation trees, and as a consequence, it is often the case that we cannot reduce the nondeterminism of the programs.

This weakness of partial evaluation is demonstrated in Section 2.2 where we revisit the familiar problem of looking for occurrences of a pattern in a string. It has been shown in [8, 10, 12] that by partial evaluation of a string matching program, we may derive a deterministic finite automaton, similarly to what is done by the Knuth-Morris-Pratt algorithm [16]. However, in [8, 10, 12] the string matching program to which partial evaluation is applied, is *deterministic*. We show that by the partial evaluation of a *nondeterministic* version of the matching program, one cannot derive a specialized program which is deterministic, and thus, one cannot get a program which corresponds to a deterministic finite automaton.

In the subsequent sections we propose a specialization technique which enhances partial evaluation by making use of more powerful transformation rules. In particular, we consider a version of the definition introduction rule so that a new predicate may be introduced by means of several non-recursive clauses whose bodies consist of several atoms, and we allow folding steps which use these predicate definitions consisting of several clauses. We also consider the following extra rules which are used to derive *mutually exclusive* clauses: *head generalization*, *case split*, *equation elimination*, and *disequation replacement*. These rules may introduce equations and negated equations between terms. We prove that our extended set of program transformation rules preserves both the least Herbrand model semantics and the operational semantics with left-to-right selection rule.

We then develop a strategy for applying our transformation rules in an automatic way, so to specialize programs and reduce their nondeterminism. Our strategy will be based on the *modes* associated with predicate calls [40].

Finally, we show by means of some examples which refer to parsing and matching problems, that our strategy is more powerful than standard partial evaluation. In particular, given a nondeteministic version of the matching program, one can derive by using our strategy a specialized program which corresponds to a deterministic finite automaton.

2. Partial Evaluation of Logic Programs via Unfold/Fold Transformations

In this section we illustrate some limitations of the standard techniques of the rule-based approach to partial evaluation of logic programs [30, 34]. These limitations motivates the introduction of the new, enhanced rules and strategies for program specialization which we will propose in Sections 3, 4, and 5.

We begin by recalling in Section 2.1 the basic concepts of the rule-based partial evaluation of logic programs. We consider *definite* logic programs [22] (also called *programs*, for short), that is, programs whose clauses do not contain negative literals in their body.

We will use the following notations. *Atoms* and *Clauses* are the sets of all atoms and all clauses, respectively. $Clauses^*$ is the set of all finite sequences of clauses and $\mathcal{P}(Clauses)$ is the powerset of *Clauses*. For any clause C, $hd(C)$ and $bd(C)$ denote the head and the body of C, respectively. By $vars(A)$ we denote the set of variables occurring in an atom A. An atom A_1 is a *variant* of an atom A_2 iff there exists a bijective mapping ρ from $vars(A_1)$ onto $vars(A_2)$ such that $A_1\rho = A_2$. $M(P)$ denotes the least Herbrand model of a program P. For concepts not defined here we refer to [22].

2.1. Transformation Rules and Strategies for Partial Evaluation

In the rule-based approach to the partial evaluation we will use the following transformation rules. Suppose that we are given a program P that we want to partially evaluate.

Rule PE1 (Atomic Definition Introduction) We introduce a clause, called an *atomic definition clause*, of the form

$$newp(X_1, \ldots, X_h) \leftarrow A$$

where (i) *newp* is a new predicate, that is, it occurs neither in P nor in previously introduced atomic definition clauses, (ii) A is an atom whose predicate occurs in program P, and (iii) $\{X_1, \ldots, X_h\} = vars(A)$.

Rule PE2 (Unfolding). Let C be a clause of the form $H \leftarrow G_1, A, G_2$, where A is an atom and G_1 and G_2 are (possibly empty) conjunctions of atoms. Let $C_1, \ldots, C_n$, with $n \geq 0$, be the clauses of P such that A is unifiable with the head of C_i with most general unifier ϑ_i, for $i = 1, \ldots, n$. By *unfolding C w.r.t. A* we derive the clauses $(H \leftarrow G_1, bd(C_i), G_2)\vartheta_i$, for $i = 1, \ldots, n$.

Rule PE3 (Atomic Folding). Let C be a clause of the form $H \leftarrow G_1, A\vartheta, G_2$ where G_1 and G_2 are (possibly empty) conjunctions of atoms, A is an atom, and ϑ is a substitution, and let D be an atomic definition clause of the form $N \leftarrow A$. By *folding C w.r.t. $A\vartheta$ using D* we introduce the atom $N\vartheta$ and we derive the clause $H \leftarrow G_1, N\vartheta, G_2$.

The partial evaluation of a program P may be realized by applying the atomic definition introduction, unfolding, and atomic folding rules, according to the strategy which we will specify below. Our partial evaluation strategy uses two subsidiary strategies: (1) an *Unfold* strategy, which derives sequences of clauses by repeatedly applying the unfolding rule, and (2) a *Define-Fold* strategy, which introduces new atomic definition clauses and it folds the clauses derived by the previous *Unfold* strategy.

In order to implement these subsidiary strategies, the partial evaluation strategy requires the following two functions: (1) *Select* and (2) *Gen*.

(1) The *unfolding selection function* $Select : Clauses^* \times Clauses \longrightarrow Atoms \cup \{halt\}$ is defined for any sequence $C_1, \ldots, C_n$ of clauses and for any clause C such that C is derived from C_n by unfolding and for $k = 1, \ldots, n-1$, clause C_{k+1} is derived from clause C_k by unfolding. (For this reason in the partial evaluation strategy below, we have called $Ancestors(C)$ the first argument of *Select*.) When applying the *Unfold* strategy the *Select* function is used as follows: (i) if $Select((C_1, \ldots, C_n), C) = A$ where A is an atom in the body of clause C, then C is unfolded w.r.t. A, and (ii) if $Select((C_1, \ldots, C_n), C) = halt$ then C is not unfolded.

(2) The *generalization function* $Gen : \mathcal{P}(Clauses) \times Atoms \longrightarrow Clauses$ is defined for any set *Defs* of atomic definition clauses and for any atom A. $Gen(Defs, A)$ is a clause of the form $g(X_1, \ldots, X_h) \leftarrow GenA$, where: (i) $\{X_1, \ldots, X_h\} = vars(GenA)$, (ii) A is an instance of $GenA$, and (iii) either $Gen(Defs, A)$ is a clause in *Defs* or g is a new predicate, that is, it occurs neither in P nor in *Defs*.

When applying the *Define-Fold* strategy the generalization function *Gen* is used as follows: when we want to fold a clause C w.r.t. an atom A in its body, we consider the set *Defs* of all atomic definition clauses introduced so far and we apply the folding rule using $Gen(Defs, A)$. This application of the folding rule is indeed possible because, by construction, A is an instance of the body of $Gen(Defs, A)$.

Partial Evaluation Strategy

Input: A program P and an atomic goal $p(t_1, \ldots, t_h)$ w.r.t. which we want to specialize P.
Output: A program P_{pe} and an atom $p_{pe}(X_1, \ldots, X_r)$, with $\{X_1, \ldots, X_r\} = vars(p(t_1, \ldots, t_h))$, such that for every ground substitution $\vartheta = \{X_1/u_1, \ldots, X_r/u_r\}$, $M(P) \models p(t_1, \ldots, t_h)\vartheta$ iff $M(P_{pe}) \models p_{pe}(X_1, \ldots, X_r)\vartheta$.

Initialize: Let S be the clause $p_{pe}(X_1, \ldots, X_r) \leftarrow p(t_1, \ldots, t_h)$ and let $Ancestors(S)$ be the empty sequence of clauses.
$Defs := \{S\}$; $P_{pe} := \emptyset$; $Cls := \{S\}$;
while $Cls \neq \emptyset$ **do**

(1) *Unfold*:

while there exists a clause $C \in Cls$ with $Select(Ancestors(C), C) \neq halt$ **do**
Let $Unf(C) = \{E \mid E$ is derived by unfolding C w.r.t. $Select(Ancestors(C), C)\}$.
$Cls := (Cls - \{C\}) \cup Unf(C)$;
for each $E \in Unf(C)$ let $Ancestors(E)$ be the sequence $Ancestors(C)$ followed by C
end-while ;

(2) *Define-Fold*:

$NewCls := \emptyset$;

while there exists a clause $C \in Cls$ and there exists an atom $A \in bd(C)$ which has not been introduced by folding **do**
Let G be the atomic definition clause $Gen(Defs, A)$ and F be the clause derived by folding C w.r.t. A using G.
$Cls := (Cls - \{C\}) \cup \{F\}$;
if $G \notin Defs$ **then** ($Defs := Defs \cup \{G\}$; $NewCls := NewCls \cup \{G\}$)
end-while ;
$P_{pe} := P_{pe} \cup Cls$; $Cls := NewCls$
end-while

A given unfolding selection function *Select* is said to be *progressive* iff for the empty sequence () of clauses and for any clause C with nonempty body, we have that $Select((), C) \neq halt$.

We have the following correctness result which is derived from Tamaki and Sato's correctness results [37] of the unfold/fold transformation rules w.r.t. the least Herbrand model semantics.

Theorem 2.1 (Correctness of Partial Evaluation) Let *Select* be a progressive unfolding selection function. Given a definite program P and an atomic goal $p(t_1, \ldots, t_h)$, if the Partial Evaluation Strategy using *Select* terminates with output program P_{pe} and output atom $p_{pe}(X_1, \ldots, X_r)$, then for every ground substitution $\vartheta = \{X_1/u_1, \ldots, X_r/u_r\}$,

$M(P) \models p(t_1, \ldots, t_h)\vartheta$ iff $M(P_{pe}) \models p_{pe}(X_1, \ldots, X_r)\vartheta$.

We say that an unfolding selection function *Select* is *halting* iff for any infinite sequence $C_1, C_2, \ldots$ of clauses, there exists a non-negative integer n such that $Select((C_1, C_2, \ldots, C_n), C_{n+1}) = halt$.

Given an infinite sequence $A_1, A_2, \ldots$ of atoms, its *image* under the generalization function *Gen*, is the sequence of sets of clauses defined as follows:

$G_1 = \{newp(X_1, \ldots, X_n) \leftarrow A_1\}$, where $\{X_1, \ldots, X_n\} = vars(A_1)$
$G_{i+1} = G_i \cup \{ Gen(G_i,\ A_{i+1})\}$ for $i \geq 1$.

We say that *Gen* is *stabilizing* iff for any infinite sequence $A_1, A_2, \ldots$ of atoms whose image under *Gen* is $G_1, G_2, \ldots$, there exists a positive integer n such that $G_k = G_n$ for all $k \geq n$.

We have the following theorem whose proof is similar to the one in [19].

Theorem 2.2 (Termination of Partial Evaluation) Let *Select* be a halting unfolding selection function and *Gen* be a stabilizing generalization function. Then for any input program P and atomic goal $p(t_1, \ldots, t_h)$, the Partial Evaluation Strategy using *Select* and *Gen* terminates.

2.2. An Example of Partial Evaluation: String Matching

In this section we illustrate an example of partial evaluation based on a string matching program. Given a program for searching a pattern in a string, and a fixed ground pattern p, we want to derive a new, specialized program for searching the pattern p in a given string. Now we present a general, deterministic program, called *Match*, for searching a pattern P in a string S in $\{a, b\}^*$. It is a variant of the ones presented in [8, 10]. Sequences in $\{a, b\}^*$ are denoted by lists of a's and b's.

Program *Match* (initial, deterministic)

1. $match(P, S) \leftarrow match1(P, S, P, S)$
2. $match1([\,], S, Y, Z) \leftarrow$
3. $match1([C|P], [C|S], Y, Z) \leftarrow match1(P, S, Y, Z)$
4. $match1([a|P], [b|S], Y, [C|Z]) \leftarrow match1(Y, Z, Y, Z)$
5. $match1([b|P], [a|S], Y, [C|Z]) \leftarrow match1(Y, Z, Y, Z)$

This program is deterministic in the sense that given some ground values for P and S, at most one clause head unifies with an atomic goal at run time. Let us assume that we want to specialize this program *Match* w.r.t. the goal $match([a, a, b], S)$, that is, we want to derive a program which tells us whether or not the pattern $[a, a, b]$ occurs in the string S.

We apply our partial evaluation strategy with the following selection and generalization functions.

(1) The unfolding selection function $DetU : Clauses^* \times Clauses \longrightarrow Atoms \cup \{halt\}$ is defined as follows:

(i) $DetU((), C) = A$ if A is the leftmost atom in the body of clause C,

(ii) $DetU((C_1, C_2, \ldots, C_n), C) = A$ if $n \geq 1$ and A is the leftmost atom the body of C such that A is unifiable with at most one clause head in the program to be partially evaluated, and

(iii) $DetU((C_1, C_2, \ldots, C_n), C) = halt$ if there exists no atom in the body of C which is unifiable with at most one clause head in the program to be partially evaluated.

(2) The generalization function $Variant : \mathcal{P}(Clauses) \times Atoms \longrightarrow Clauses$ is defined as follows:

(i) $Variant(Defs, A)$ is a clause C such that $bd(C)$ is a variant of A, if in $Defs$ there exists any such clause C, and

(ii) $Variant(Defs, A)$ is the clause $newp(X_1, \ldots, X_h) \leftarrow A$, where $newp$ is a new predicate symbol and $\{X_1, \ldots, X_h\} = vars(A)$, otherwise.

The function $DetU$ corresponds to the *determinate unfolding rule* considered in [10]. In general, we have that $DetU$ is not halting and $Variant$ is not stabilizing. Nevertheless, in our example, as the reader may verify, the partial evaluation strategy using $DetU$ and $Variant$ terminates and it generates the following specialized program:

Program $Match_{pe}$ (specialized, deterministic)

6. $match_{pe}(S) \leftarrow new1(S)$
7. $new1([a|S]) \leftarrow new2(S)$
8. $new1([b|S]) \leftarrow new1(S)$
9. $new2([a|S]) \leftarrow new3(S)$
10. $new2([b|S]) \leftarrow new1(S)$
11. $new3([b|S]) \leftarrow$
12. $new3([a|S]) \leftarrow new3(S)$

The program $Match_{pe}$ is deterministic and it corresponds to a deterministic finite automaton in the sense that: (i) each predicate corresponds to a state, (ii) each clause, except for clause 6

and 11, corresponds to a transition from the state corresponding to the predicate of the head to the state corresponding to the predicate of the body, (iii) each transition is labelled by the symbol (either a or b) occurring in the head of the corresponding clause, (iv) clause 6 enforces that $new1$ is the initial state in the hypothesis that the intended queries are of the of the form $match_{pe}(w)$, for a ground list w in $\{a, b\}^*$, and (v) clause 11 corresponds to a transition to a final state where all words of $\{a, b\}^*$ are accepted.

Thus, via partial evaluation we may derive a deterministic finite automaton from a deterministic string matching program, and the derived program corresponds to the Knuth-Morris-Pratt (KMP) string matching algorithm [16]. For this reason in [36] partial evaluation of logic programs (referred to as partial deduction), is said to *pass the* KMP *test*.

2.3. Some Limitations of Partial Evaluation

We argue that the ability of partial evaluation of logic programs to pass the KMP test is related to the fact that the initial string matching program *Match* is rather sophisticated and, indeed, the correctness proof of the program *Match* is not straightforward. Actually, partial evaluation does *not* pass the KMP test if we consider, instead of the program *Match*, the following naive, nondeterministic initial program for string matching:

Program *Naive_Match* (initial, nondeterministic)

1. $naive\_match(P, S) \leftarrow append(X, R, S),\ append(L, P, X)$
2. $append([\,], Y, Y) \leftarrow$
3. $append([A|X], Y, [A|Z]) \leftarrow append(X, Y, Z)$

The correctness of this program is straightforward because for a given pattern P and a string S, *Naive_Match* checks that P occurs in S by looking in a nondeterministic way for two strings L and R such that S is the concatenation of L, P, and R in this order.

The reader may verify that partial evaluation does not pass the KMP test when starting from the program *Naive_Match*. Indeed, if one specializes *Naive_Match* w.r.t. the goal $naive\_match([a, a, b], S)$ by applying the Partial Evaluation Strategy given above, using the unfolding selection function *DetU* and the generalization function *Variant*, then one derives the following program $Naive\_Match_{pe}$ which does *not* correspond to a deterministic finite automaton and it is nondeterministic:

Program $Naive\_Match_{pe}$ (specialized, nondeterministic)

4. $naive\_match_{pe}(S) \leftarrow new1(X, R, S),\ new2(L, X)$
5. $new1([\,], Y, Y) \leftarrow$
6. $new1([A|X], Y, [A|Z]) \leftarrow new1(X, Y, Z)$
7. $new2([\,], [a, a, b]) \leftarrow$
8. $new2([A|X], [A|Z]) \leftarrow new2(X, Z)$

This $Naive\_Match_{pe}$ program, in fact, looks in a nondeterministic way for two strings L and R such that S is the concatenation of L, $[a, a, b]$, and R. If the pattern $[a, a, b]$ is not found within the string S at a given position, then the search for $[a, a, b]$ is restarted after a shift of one character to the right w.r.t. that position.

The failure of partial evaluation to pass the KMP test when starting from the program *Naive_Match*, is due to some limitations which can be overcome by using the enhanced transformation rules which we will present in the next section. By applying these enhanced rules we can

define a new predicate by introducing *several* clauses whose bodies are *non-atomic* goals, while by applying the standard rules, a new predicate can be defined by introducing *one* clause only and the body of that clause is required to be an *atomic* goal. By folding using definition clauses of the enhanced form, we can derive specialized programs where nondeterminism is reduced and intermediate data structures are avoided. Our enhanced rules will also include the so called *case split rule* which can reduce nondeterminism, because given a clause, it produces two mutually exclusive instances of that clause by introducing negated equations.

By applying the enhanced transformation rules, one can automatically specialize the nondeterministic program *Naive_Match* w.r.t. the goal $naive\_match([a,a,b],S)$ thereby deriving the following deterministic program (the derivation which we do not present here, is similar to the one presented in Section 6.1):

Program $Naive\_Match_s$ (specialized, deterministic)

9. $naive\_match_s(S) \leftarrow new1(S)$
10. $new1([a|S]) \leftarrow new2(S)$
11. $new1([C|S]) \leftarrow C \neq a, new1(S)$
12. $new2([a|S]) \leftarrow new3(S)$
13. $new2([C|S]) \leftarrow C \neq a, new1(S)$
14. $new3([b|S]) \leftarrow new4(S)$
15. $new3([a|S]) \leftarrow new3(S)$
16. $new3([C|S]) \leftarrow C \neq b, C \neq a, new1(S)$
17. $new4(S) \leftarrow$

$Naive\_Match_s$ corresponds in a straightforward way to a deterministic finite automaton. Due to the introduction of negated equations, the derived program $Naive\_Match_s$ may be used also when the alphabet is a superset of $\{a,b\}$. This was not the case for the program $Match_{pe}$.

The main limitations of the Partial Evaluation Strategy presented in Section 2.1, are the following ones:

(i) it fails to eliminate intermediate data structures (see the intermediate list X in clause 4 of $Naive\_Match_{pe}$) because a *sequence* of computations cannot be combined into a *single* computation (indeed, atomic definition clauses have *one* atom only in the body);

(ii) it fails to reduce nondeterminism, because it cannot combine the partial evaluations of multiple *alternative* computations (indeed, atomic definition clauses are made out of *one* clause only); and

(iii) it does not incorporate any reasoning by cases, and it does not introduce negated equations which may take into account unification failures.

These limitations of the Partial Evaluation Strategy are not due to the choice of the subsidiary strategies, that is, the unfolding and generalization functions, and instead, they are due to the restricted way in which partial evaluation can introduce new predicates. As the expert reader may notice, this restricted way which allows us to use a single clause with an atomic body for each new predicate, corresponds to the fact that standard partial evaluation [23] is performed w.r.t. atomic goals only.

3. Transformation Rules for Logic Programs with Equations and Disequations between Terms

In this section we introduce an extension of definite logic programs with equations and negated equations between terms. Negated equations will also be called *disequations*. The introduction of equations and disequations during program specialization allows us to derive mutually exclusive clauses. We also present the enhanced program transformation rules which we use for program specialization. These rules extend to our language the unfold/fold rules considered in [11, 33, 37]. In particular, similarly to [11, 33], we consider a folding rule by which we can fold several clauses at a time. In addition, we consider the subsumption rule and the following transformation rules for introducing and eliminating equations and disequations: (i) head generalization, (ii) case split, (iii) equation elimination, and (iv) disequation replacement. Our rules preserve the least Herbrand model as indicated in Theorem 3.3 below.

3.1. Syntax

The syntax of our language is defined starting from the following infinite and pairwise disjoint sets:
(i) *variables*: $X, Y, Z, X_1, X_2, \ldots$,
(ii) *function symbols* (with arity): $f, f_1, f_2, \ldots$, and
(iii) *predicate symbols* (with arity): *true*, $=$, $\neq$, $p, p_1, p_2, \ldots$ The predicate symbols *true*, $=$, and $\neq$ are said to be *basic*, and the other ones are said to be *non-basic*.

We also consider the following sets: (iv) *Terms*: $t, t_1, t_2, \ldots$, (v) *Basic atoms*: $B, B_1, B_2, \ldots$, (vi) *Non-basic atoms*: $A, A_1, A_2, \ldots$, and (vii) *Goals*: $G, G_1, G_2, \ldots$ Their syntax is as follows:

Terms :	$t ::= X \mid f(t_1, \ldots, t_n)$
Basic Atoms :	$B ::= \mathit{true} \mid t_1 = t_2 \mid t_1 \neq t_2$
Non-basic Atoms :	$A ::= p(t_1, \ldots, t_m)$
Goals :	$G ::= B \mid A \mid G_1, G_2$

Basic and non-basic atoms are collectively called *atoms*. Goals made out of basic atoms only are said to be *basic goals*. Goals with at least one non-basic atom are said to be *non-basic goals*. The binary operator ',' denotes *conjunction* and it is assumed to be associative with neutral element *true*. Thus, a goal G is the same as goal (true, G), and it is also the same as goal (G, true). Parentheses are used for grouping goals.

In what follows we will feel free to use different symbols to denote our syntactic expressions, and in particular, we will also denote non-basic atoms by $H, H_1, \ldots$, and goals by $K, K_1, \mathit{Body}, \mathit{Body}_1, \ldots$

Clauses $C, C_1, C_2, \ldots$ have the following syntax:

$C ::= H \leftarrow G$

Given a clause C of the form: $H \leftarrow G$, the atom H, which is non-basic, is called the *head* of C and it is denoted by $hd(C)$, and the goal G is called the *body* of C and it is denoted by $bd(C)$. A clause $H \leftarrow G$ where G is a basic goal, is called a *unit clause*. We write a unit clause of the form: $H \leftarrow \mathit{true}$ also as: $H \leftarrow$.
We say that C *is a clause for* a predicate p iff C is a clause of the form $p(\ldots) \leftarrow \mathit{Body}$.

Programs $P, P_1, P_2, \ldots$ are *sets* of clauses.

Given a program P, we say that a predicate p *depends on* a predicate q (not necessarily distinct from p) iff *either* there exists in P a clause for p whose body contains an occurrence of q *or*

there exists a predicate r such that p depends on r and r depends on q. We say that a predicate p depends on a clause C iff *either* C is a clause for p *or* C is a clause for a predicate q and p depends on q.

Terms, atoms, goals, clauses, and programs are collectively called *expressions*, ranged over by $e, e_1, e_2, \ldots$ By $vars(e)$ we denote the set of variables occurring in an expression e. By $e[e_1]$ we denote an expression where we point out an occurrence of the subexpression e_1. $e[\_]$ is the expression $e[e_1]$ where we have dropped that occurrence of the subexpression e_1. We say that X is a *local* variable of e_1 in $e[e_1]$ iff $X \in vars(e_1) - vars(e[\_])$, and X is a *linking* variable of e_1 in $e[e_1]$ iff $X \in vars(e_1) \cap vars(e[\_])$.

A *variable renaming* is a bijective mapping from the set of variables onto itself. An expression e_1 is a *variant* of an expression e_2 iff there exists a bijective mapping ρ from $vars(e_1)$ onto $vars(e_2)$ such that $e_1\rho = e_2$. We allow the silent application of variable renamings to clauses. Any variable renaming preserves the least Herbrand model semantics which we define below. Given a clause C, a *renamed apart* clause C' is any clause obtained from C by applying a variable renaming, so that each variable of C' does not appear in any other expression. In this sense we say that the variables of C' are *new* variables.

For the notions of *substitution*, *domain* of a substitution, *composition* of substitutions, *instance*, *most general unifier* (abbreviated as *mgu*), *ground expression*, *ground substitution*, and for other notions not defined here, we refer to [22]. For any two unifiable terms t_1 and t_2, there exists at least one mgu ϑ which is *relevant* (that is, each variable occurring in ϑ also occurs in $vars(t_1) \cup vars(t_2)$) and *idempotent* (that is, $\vartheta\vartheta = \vartheta$) [1]. Without loss of generality, we assume that all mgu's considered in this paper are relevant and idempotent.

3.2. Declarative Semantics

The declarative semantics of a program P is its least Herbrand model. The definition of the least Herbrand model of a program which we now give, differs from the usual one [22] because the predicates *true*, $=$, and $\neq$ have a fixed interpretation.

We say that a ground basic atom A *holds* iff: (i) A is *true*, or (ii) A is $t = t$ for some term t, or (iii) A is $t_1 \neq t_2$ for some different terms t_1 and t_2. We say that a ground basic goal G holds iff G is the conjunction of ground basic atoms each of which holds. The *Herbrand base* is the set $\mathcal{HB}$ of all ground *non-basic* atoms. An *Herbrand interpretation* is a subset of $\mathcal{HB}$. Given an Herbrand interpretation I, a ground goal G *is true in* I, written as $I \models G$, iff: (i) G is a basic goal and G holds, (ii) G is a non-basic atom which belongs to I, or (iii) G is a conjunction (G_1, G_2) and both G_1 and G_2 are true in I. G *is false in* I iff it is not the case that G is true in I. A ground clause C is true in I iff either $hd(C)$ is true in I or $bd(C)$ is false in I. A non ground clause is true in I iff all its ground instances are true in I. A program P is true in I iff all its clauses are true in I.

The Herbrand interpretation I is said to be an *Herbrand model* of a program P iff P is true in I. Similarly to [22], we can prove the following important property.

Theorem 3.1. For any program P there exists an Herbrand model of P which is the least (w.r.t. set inclusion) Herbrand model.

The least Herbrand model of a program P is denoted by $M(P)$. For any program P and any ground atom A, we have that: $M(P) \models A$ iff *either* A is a basic atom and A holds *or* A is a non-basic atom and $P \vdash A$ holds, where $\vdash$ denotes the usual entailment relation in first order predicate calculus.

3.3. Operational Semantics

We define the operational semantics of our programs by introducing, for each program P, a *derivability* relation $\mapsto_P$ between goals, defined as follows:

(1) $(t_1 = t_2,\ G) \mapsto_P G\vartheta$ iff t_1 and t_2 are unifiable via an mgu ϑ
(2) $(t_1 \neq t_2,\ G) \mapsto_P G$ iff t_1 and t_2 are not unifiable
(3) $(A, G) \mapsto_P (G_1, G)\vartheta$ iff (i) A is a non-basic atom,
(ii) $A_1 \leftarrow G_1$ is a renamed apart clause in P, and
(iii) A and A_1 are unifiable via an mgu ϑ.

The relation $\mapsto_P^*$ is the reflexive and transitive closure of $\mapsto_P$. Given two goals G_0 and G_1 such that $G_0 \mapsto_P^* G_1$ holds, we say that G_1 is *derivable* from G_0 using P. We say that the goal G *succeeds* in P iff $G \mapsto_P^* \mathit{true}$. We will feel free to omit the reference to program P when it is clear from the context.

A sequence $G_0 \mapsto_P \ldots \mapsto_P G_n$, with $n \geq 0$, is called a *derivation* using P. If G_n is *true* then the derivation is said to be *successful*. Given two goals (A_0, G_0) and (A_n, G_n), where A_0 and A_n are non-basic atoms and $n > 0$, we write $(A_0, G_0) \Rightarrow_P (A_n, G_n)$ iff there exists a derivation $(A_0, G_0) \mapsto_P (A_1, G_1) \mapsto_P \ldots \mapsto_P (A_{n-1}, G_{n-1}) \mapsto_P (A_n, G_n)$, where $A_1, \ldots, A_{n-1}$, if any, are basic atoms.

Our operational semantics can be viewed as an abstraction of the usual Prolog semantics, because: (i) given a goal G_1, in order to derive a goal G_2 such that $G_1 \mapsto_P G_2$, we consider the leftmost atom in G_1, (ii) the predicate $=$ is interpreted as unifiability of terms, and (iii) the predicate $\neq$ is interpreted as non-unifiability of terms. Similarly to [22], we have the following relationship between the declarative and the operational semantics.

Theorem 3.2. For any program P and ground goal G, we have that *if* G succeeds in P *then* $M(P) \models G$.

The converse of Theorem 3.2 does not hold. Indeed, consider the program P consisting of the clause $p(1) \leftarrow X \neq 0$ only. We have that $M(P) \models p(1)$ because there exists a value for X, namely 1, which is syntactically different from 0. However, $p(1)$ does not succeed in P, because X and 0 are unifiable terms.

Remark Let us observe that given a program P, a function symbol f, and a ground term r, (i) in our declarative semantics $r \neq f(X)$ holds, that is, $M(P) \models r \neq f(X)$, iff *there exists* a ground term s such that r is different from $f(s)$, and (ii) in our operational semantics, $r \neq f(X)$ succeeds in P, that is, $r \neq f(X) \mapsto_P \mathit{true}$, iff *for all* ground terms s we have that r is different from $f(s)$.

3.4. Transformation Rules

The process of program transformation can be viewed as the construction of a sequence of programs, called *transformation sequence*. A transformation sequence $P_0, \ldots, P_n$ is constructed from a given initial program P_0 by applications of the transformation rules 1–9 given below, as follows. For $k = 0, \ldots, n-1$, program P_{k+1} is derived from program P_k by: (i) selecting a (possibly empty) subset γ_1 of clauses of P_k, (ii) deriving a set γ_2 of clauses by applying a transformation rule to γ_1, and (iii) replacing γ_1 by γ_2 in P_k.

Now we assume that we have constructed from the initial program P_0 the sequence $P_0, \ldots, P_k$ of programs. We derive program P_{k+1} which is the next one in the sequence, by applying one of the following rules.

Rule 1 (Definition Introduction) We introduce m ($\geq$1) new clauses of the form:

$$\begin{cases} D_1.\ \ newp(X_1, \ldots, X_h) \leftarrow Body_1 \\ \qquad \ldots \\ D_m.\ newp(X_1, \ldots, X_h) \leftarrow Body_m \end{cases}$$

where: (i) *newp* is a non-basic predicate symbol not occurring in $P_0, \ldots, P_k$, (ii) the variables $X_1, \ldots, X_h$ are all distinct and for all $i \in \{1, \ldots, h\}$ there exists $j \in \{1, \ldots, m\}$ such that X_i occurs in the goal $Body_j$, (iii) for all $j \in \{1, \ldots, m\}$, every non-basic predicate occurring in $Body_j$ also occurs in P_0, and (iv) for all $j \in \{1, \ldots, m\}$, there exists at least one non-basic atom in $Body_j$.
Program P_{k+1} is the program $P_k \cup \{D_1, \ldots, D_m\}$.

We denote by $Defs_k$ the set of clauses, called *definition clauses*, which have been introduced by the definition introduction rule during the construction of the transformation sequence $P_0, \ldots, P_k$. In particular, we have that $Defs_0 = \emptyset$.

Rule 2 (Definition Elimination) Let p be a predicate symbol. By *definition elimination w.r.t. p* we derive program P_{k+1} from program P_k by keeping only the clauses of P_k on which p depends.

Rule 3 (Unfolding) Let C be a clause in program P_k of the form: $H \leftarrow G_1, A, G_2$, where A is a non-basic atom. Let $C_1, \ldots, C_m$, with $m \geq 0$, be the clauses of P_k such that, for all $i \in \{1, \ldots, m\}$ A is unifiable with the head of C_i via the mgu ϑ_i. For $i = 1, \ldots, m$, let D_i be the clause $(H \leftarrow G_1, bd(C_i), G_2)\vartheta_i$. By *unfolding C w.r.t. A* we derive the clauses $D_1, \ldots, D_m$.
Program P_{k+1} is the program $(P_k - \{C\}) \cup \{D_1, \ldots, D_m\}$.

Notice that an application of the unfolding rule to clause C amounts to the deletion of C iff $m = 0$. Sometimes in the literature this particular instance of the unfolding rule is treated as an extra rule.

Rule 4 (Folding) Let

$$\begin{cases} C_1.\ \ H \leftarrow\ G_1, Body_1\vartheta, G_2 \\ \ldots \\ C_m.\ H \leftarrow\ G_1, Body_m\vartheta, G_2 \end{cases}$$

be some clauses in P_k, for a suitable substitution ϑ, and let

$$\begin{cases} D_1.\ \ newp(X_1, \ldots, X_h) \leftarrow Body_1 \\ \ldots \\ D_m.\ \ newp(X_1, \ldots, X_h) \leftarrow Body_m \end{cases}$$

be all clauses in $Defs_k$ which have *newp* as head predicate. Suppose that for $i = 1, \ldots, m$, the following condition holds: for every variable X occurring in the goal $Body_i$ and not in $\{X_1, \ldots, X_h\}$, we have that: (i) $X\vartheta$ is a variable which does not occur in (H, G_1, G_2), and (ii) $X\vartheta$ does not occur in $Y\vartheta$, for any variable Y occurring in $Body_i$ and different from X. By *folding* $C_1, \ldots, C_m$ *using* $D_1, \ldots, D_m$ we derive the single clause C: $H \leftarrow G_1, newp(X_1, \ldots, X_h)\vartheta, G_2$.
Program P_{k+1} is the program $(P_k - \{C_1, \ldots, C_m\}) \cup \{C\}$.

For instance, the clauses C_1: $p(X) \leftarrow q(t(X), Y), r(Y)$ and C_2: $p(X) \leftarrow s(X), r(Y)$ can be folded (via the substitution $\vartheta = \{U/X, V/Y\}$) using the two clauses D_1: $a(U, V) \leftarrow q(t(U), V)$ and D_2: $a(U, V) \leftarrow s(U)$, and we replace C_1 and C_2 by the clause C: $p(X) \leftarrow a(X, Y), r(Y)$.

Rule 5 (Subsumption) (i) Given a substitution ϑ, we say that a clause $H \leftarrow G_1$ *subsumes* a clause $(H \leftarrow G_1, G_2)\vartheta$.
Program P_{k+1} is derived from program P_k by deleting a clause which is subsumed by another clause in P_k.

Rule 6 (Head Generalization) Let C be a clause of the form: $H\{X/t\} \leftarrow Body$ in P_k, where $\{X/t\}$ is a substitution such that X occurs in H and X does not occur in C. By *head generalization*, we derive the clause $GenC$: $H \leftarrow X{=}t, Body$.
Program P_{k+1} is the program $(P_k - \{C\}) \cup \{GenC\}$.

Rule 6 is a particular case of the rule of *generalization + equality introduction* considered, for instance, in [31].

Rule 7 (Case Split) Let C be a clause in P_k of the form: $H \leftarrow Body$. By *case split of* C *w.r.t.* the binding X/t where X does not occur in t, we derive the following two clauses:

C_1. $(H \leftarrow Body)\{X/t\}$
C_2. $H \leftarrow X{\neq}t, Body$.

Program P_{k+1} is the program $(P_k - \{C\}) \cup \{C_1, C_2\}$. Notice that X need not occur in C.

According to our operational semantics, if $G \mapsto_{P_{k+1}} G_1$ using clause C_1 and X occurs in H, then no G_2 exists such that $G \mapsto_{P_{k+1}} G_2$ using clause C_2. The same holds by interchanging C_1 and C_2. We will return to this property in Definitions 7 (Semideterminism) and 11 (Mutual Exclusion) below.

Rule 8 (Equation Elimination) Let C_1 be a clause in P_k of the form:

C_1. $H \leftarrow G_1,\ t_1{=}t_2,\ G_2$

If t_1 and t_2 are unifiable via the most general unifier ϑ, then by equation elimination we derive the following clause:

C_2. $(H \leftarrow G_1, G_2)\vartheta$

Program P_{k+1} is the program $(P_k - \{C_1\}) \cup \{C_2\}$.
If t_1 and t_2 are not unifiable then by equation elimination we derive program P_{k+1} which is $P_k - \{C_1\}$.

Rule 9 (Disequation Replacement) Let C be a clause in program P_k. Program P_{k+1} is derived from P_k by either removing C or replacing C as we now indicate:

9.1 if C is of the form: $H \leftarrow G_1, t_1{\neq}t_2, G_2$ and t_1 and t_2 are not unifiable, then C is replaced by $H \leftarrow G_1, G_2$

9.2 if C is of the form: $H \leftarrow G_1,\ f(t_1, \ldots, t_m){\neq}f(u_1, \ldots, u_m),\ G_2$, then C is replaced by the following m (≥ 0) clauses: $H \leftarrow G_1, t_1{\neq}u_1, G_2, \quad \ldots, \quad H \leftarrow G_1, t_m{\neq}u_m, G_2$

9.3 if C is of the form: $H \leftarrow G_1, X{\neq}X, G_2$, then C is removed from P_k

9.4 if C is of the form: $H \leftarrow G_1, t{\neq}X, G_2$, then C is replaced by $H \leftarrow G_1, X{\neq}t, G_2$

9.5 if C is of the form: $H \leftarrow G_1, X{\neq}t_1, G_2, X{\neq}t_2, G_3$ and there exists a substitution ρ which is a bijective mapping from the set of the local variables of t_1 in C onto the set of the local variables of t_2 in C such that $t_1\rho = t_2$, then C is replaced by $H \leftarrow G_1, X \neq t_1, G_2, G_3$. In particular, if a disequation has two occurrences in the body of a clause, then we can remove the rightmost one.

Notice that no rule is given for replacing a single disequation of the form $X \neq t$ when t is unifiable with X.

The following example shows that by removing any one of two identical atoms in the body of a clause, the operational semantics is not preserved.

Example 1. Let us consider the program P:

1. $p \leftarrow q(X,Y),\ q(X,Y),\ X \neq Y$
2. $q(X,b) \leftarrow$
3. $q(a,Y) \leftarrow$

and the program Q obtained from P by replacing clause 1 by the following clause:

4. $p \leftarrow q(X,Y),\ X \neq Y$

The goal p succeeds in P, while it does not succeed in Q. Indeed, (i) for program P we have that:
$p \ \mapsto_P \ q(X,Y), q(X,Y), X \neq Y \ \mapsto_P \ q(X,b), X \neq b \ \mapsto_P \ a \neq b \ \mapsto_P \ \mathit{true}$, and (ii) for program Q we have that: *either* $p \mapsto_Q X \neq b$ *or* $p \mapsto_Q a \neq Y$. In Case (ii), since X and Y are unifiable with b and a, respectively, we have that $p \mapsto_Q^* \mathit{true}$ does not hold.

3.5. Correctness of the Transformation Rules w.r.t. the Declarative Semantics

In this section we show that, under suitable hypotheses, our transformation rules preserve the declarative semantics presented in Section 3.2. In that sense we also say that our transformation rules are *correct*) w.r.t. the given declarative semantics. Our correctness result extends to our language similar results holding for definite logic programs [11, 33, 37].

Theorem 3.3 (Correctness w.r.t. the Least Herbrand Model) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9 and let p be a non-basic predicate in P_n. Let us assume that:

1. *if* the folding rule is applied for the derivation of a clause C in program P_{k+1} from clauses $C_1, \ldots, C_m$ in program P_k using clauses $D_1, \ldots, D_m$ in Defs_k, with $0 \leq k < n$,
   then for every $i \in \{1, \ldots, m\}$ there exists $j \in \{1, \ldots, n-1\}$ such that D_i occurs in P_j and P_{j+1} is derived from P_j by unfolding D_i.

2. during the transformation sequence $P_0, \ldots, P_n$ the definition elimination rule *either* is never applied *or* it is applied w.r.t. predicate p once only, in the last step, that is, when deriving P_n from P_{n-1}.

Then, for every ground atom A with predicate p, $M(P_0 \cup \mathit{Defs}_n) \models A$ iff $M(P_n) \models A$.

Proof: It is a simple extension of a similar result presented in [11] for the case where we use the unfolding, folding, and *generalization + equality introduction* rules. The proof technique used in [11] can be adapted to prove also the correctness of our extended set of rules. □

We would like to notice that our transformation rules do *not* preserve the operational semantics, as shown by the following examples.

Example 2. Let us consider the following program P_1 :

1. $p(X) \leftarrow q(X),\ X \neq a$
2. $q(X) \leftarrow$
3. $q(X) \leftarrow X = b$

By rule 5 we may delete clause 3 which is subsumed by clause 2 and we derive a new program P_2. Now, we have that $p(X)$ succeeds in P_1, while it does not succeed in P_2.

Example 3. Let us consider the following program P_3 :

1. $p(X) \leftarrow$

By the case split rule we may replace clause 1 by the two clauses:

2. $p(a) \leftarrow$
3. $p(X) \leftarrow X \neq a$

and we derive a new program P_4. The goal $p(X), X = b$ succeeds in P_3, while it does not succeed in P_4.

Example 4. Let us consider the following program P_5 :

1. $p \leftarrow X \neq a,\ X = b$

By rule 8 we may replace clause 1 by:

2. $p \leftarrow b \neq a$

and we derive a new program P_6. The goal p does not succeed in P_5, while it succeeds in P_6.

In the next section we will introduce a class of programs and a class of goals for which our transformation rules preserve both the declarative semantics and the operational semantics. In order to do so, we associate a *mode* with every predicate. A mode of a predicate specifies the *input* arguments of that predicate, and we assume that whenever the predicate is called, its input arguments are bound to ground terms. We will see that, if some suitable conditions are satisfied, compliance to modes guarantees the preservation of the operational semantics. This fact is illustrated by the above Examples 2 and 3, and indeed, in either of them, if we restrict ourselves to calls of the predicate p with ground arguments, then the initial program and the derived program have the same operational semantics.

Notice, however, that the incorrectness of the transformation of Example 4 does not depend on the modes. Thus, to ensure correctness w.r.t. the operational semantics we have to rule out clauses such as clause 1 of program P_5. Indeed, as we will see in the next section, the clauses we will consider satisfy the following condition: each variable which occurs in a disequation *either* occurs in an input argument of the head predicate *or* it is a local variable of the disequation.

4. Program Transformations based on Modes

Program modes provide an abstract interpretation framework which allows us: (i) to specify classes of programs and goals w.r.t. which the transformation rules we have presented in Section 3.4, preserve both the declarative semantics (see Section 3.2) and the operational semantics (see Section 3.3), and (ii) to design our strategy for specializing programs and reducing nondeterminism.

4.1. Modes

A *mode for a non-basic predicate* p of arity h (≥ 0) is an expression of the form $p(m_1, \ldots, m_h)$, where for $i = 1, \ldots, h$, m_i is either $+$ (denoting *any ground* term) or $?$ (denoting *any* term). Thus, if $h{=}0$, p has a unique mode which is p itself. Given a mode $p(m_1, \ldots, m_h)$ and an atom of the form $p(t_1, \ldots, t_h)$,
(1) for $i = 1, \ldots, h$, the term t_i is said to be an *input argument* of p iff m_i is $+$, and
(2) a variable of $p(t_1, \ldots, t_h)$ with an occurrence in an input argument of p, is said to be an *input variable* of $p(t_1, \ldots, t_h)$.

A *mode for a program* P is a set of modes for non-basic predicates containing exactly one mode for every distinct, non-basic predicate p occurring in P. Notice that a mode for a program P may or may not contain modes for non-basic predicates *not* occurring in P.

Example 5. Given the program P:

$p(0, 1) \leftarrow$
$p(0, Y) \leftarrow q(Y)$

the set $M_1 = \{p(+, ?), q(?)\}$ is a mode for P. $M_2 = \{p(+, ?), q(+), r(+)\}$ is different mode for P.

Definition 1. Let M be a mode for a program P and p a non-basic predicate. We say that an atom $p(t_1, \ldots, t_h)$ *satisfies* the mode M iff (1) a mode for p belongs to M and (2) for $i = 1, \ldots, h$, if the argument t_i is an input argument of p according to M, then t_i is a ground term. In particular, when $h{=}0$, we have that p *satisfies* M iff $p \in M$.
The program P *satisfies* the mode M iff for each non-basic atom A_0 which satisfies M, and for each non-basic atom A and goal G such that $A_0 \mapsto_P^* (A, G)$, we have that A satisfies M.

With reference to Example 5 above, program P satisfies mode M_1, but it does *not* satisfy mode M_2. Often the property that a program satisfies a mode can be automatically verified by abstract interpretation methods (see, for instance, [6]).

4.2. Correctness of the Transformation Rules w.r.t. the Operational Semantics

We now introduce a class of programs, called *safe* programs, and we prove that if the transformation rules are applied to a safe program with suitable restrictions, then the given program and the derived program are equivalent w.r.t. the operational semantics.

Definition 2 (Safe Programs) Let M be a mode for a program P. We say that a clause C in P is *safe* w.r.t. M iff for each disequation $t_1 \neq t_2$ in the body of C, we have that: for each variable X occurring in $t_1 \neq t_2$ *either* X is an input variable of $hd(C)$ *or* X is a local variable of $t_1 \neq t_2$ in C. Program P is safe w.r.t. M iff all its clauses are safe w.r.t. M.

For instance, let us consider the mode $M = \{p(+), q(?)\}$. Clause $p(X) \leftarrow X \neq f(Y)$ is safe w.r.t. M and clause $p(X) \leftarrow X \neq f(Y),\ q(Y)$ is not safe w.r.t. M because Y occurs both in $f(Y)$ and in $q(Y)$.

When mentioning the safety property w.r.t. a given mode M, we feel free to omit the reference to M, if it is irrelevant or it is understood from the context.

In order to get our desired correctness results (see Theorem 4.1 below), we also need to restrict the use of our transformation rules as follows.

Definition 3 (Safe Unfolding) Let P_k be a program and M be a mode for P_k. Let us consider an application of the unfolding rule (see Rule 3 in Section 3.4) whereby from the following clause of P_k:

$H \leftarrow G_1, A, G_2$

we derive the clauses:

$$\begin{cases} D_1.\ \ (H \leftarrow G_1, bd(C_1), G_2)\vartheta_1 \\ \cdots \\ D_m.\ (H \leftarrow G_1, bd(C_m), G_2)\vartheta_m \end{cases}$$

where $C_1, \ldots, C_m$ are the clauses in P_k such that, for $i \in \{1, \ldots, m\}$, A is unifiable with the head of C_i via the mgu ϑ_i.

We say that this application of the unfolding rule is *safe* w.r.t. mode M iff for all $i = 1, \ldots, m$, for all disequations d in $bd(C_i)$, and for all variables X occurring in $d\vartheta_i$, we have that either X is an input variable of $H\vartheta_i$ or X is a local variable of $d\vartheta_i$ in D_i.

Definition 4 (Safe Folding) Let us consider a program P_k and a mode M for P_k. Let us also consider an application of the folding rule (see Rule 4 in Section 3.4) whereby from the following clauses in P_k:

$$\begin{cases} C_1.\ H \leftarrow\ \ G_1, (A_1, K_1)\vartheta, G_2 \\ \cdots \\ C_m.\ H \leftarrow\ \ G_1, (A_m, K_m)\vartheta, G_2 \end{cases}$$

and the following definition clauses in $Defs_k$:

$$\begin{cases} D_1.\ newp(X_1, \ldots, X_h) \leftarrow A_1, K_1 \\ \cdots \\ D_m.\ newp(X_1, \ldots, X_h) \leftarrow A_m, K_m \end{cases}$$

we derive the new clause:

$H \leftarrow G_1, newp(X_1, \ldots, X_h)\vartheta, G_2$

We say that this application of the folding rule is *safe* w.r.t. mode M iff the following Property Σ holds:

(*Property* Σ) Each input variable of $newp(X_1, \ldots, X_h)\vartheta$, is also an input variable of at least one of the non-basic atoms occurring in $(H, G_1, A_1\vartheta, \ldots, A_m\vartheta)$.

Definition 5 (Safe Head Generalization) Let us consider a program P_k and a mode M for P_k. We say that an application of the head generalization rule (see Rule 6 in Section 3.4) to a clause of P_k is *safe* iff $H\{X/t\}$ and H have the same set of input variables w.r.t. M.

Definition 6 (Safe Case Split) Let us consider a program P_k and a mode M for P_k. Let us consider also an application of the case split rule (see Rule 7 in Section 3.4) whereby from a clause C in P_k of the form: $H \leftarrow Body$ we derive the following two clauses:

$C_1.\ (H \leftarrow Body)\{X/t\}$
$C_2.\ H \leftarrow X \neq t, Body.$

We say that this application of the case split rule is *safe* w.r.t. mode M iff X is an input variable of H, X does not occur in t, and for all variables $Y \in vars(t)$, *either* Y is an input variable of H *or* Y does not occur in C.

When applying the safe case split rule, X occurs in H and thus, given a goal G, it is not the case that for some goals G_1 and G_2, we have both $G \mapsto G_1$ using clause C_1 and $G \mapsto G_2$ using clause C_2. In Definition 11 below, we will formalize this property by saying that the clauses C_1 and C_2 are *mutually exclusive*.

In Theorem 4.1 below we will show that if we apply our transformation rules and their safe versions in a restricted way, then a program P which satisfies a mode M and is safe w.r.t. M, is transformed into a new program, say Q, which satisfies M and is safe w.r.t. M. Moreover, the programs P and Q have the same operational semantics.

Theorem 4.1 (Correctness w.r.t. the Operational Semantics) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9 and let p be a non-basic predicate in P_n. Let M be a mode for $P_0 \cup \mathit{Defs}_n$ such that: (i) $P_0 \cup \mathit{Defs}_n$ is safe w.r.t. M, (ii) $P_0 \cup \mathit{Defs}_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are all safe w.r.t. M. Suppose also that Conditions 1 and 2 of Theorem 3.3 hold. Then: (i) P_n is safe w.r.t. M, (ii) P_n satisfies M, and (iii) for each atom A which has predicate p and satisfies mode M, A succeeds in $P_0 \cup \mathit{Defs}_n$ iff A succeeds in P_n.

Proof: See Appendix. □

We now prove that the promotion of disequations in the body of a safe clause preserves the operational semantics.

Proposition 4.2 (Correctness of Disequation Promotion) Let M be a mode for a program P_1. Let us assume that P_1 is safe w.r.t. M and P_1 satisfies M. Let C_1: $H \leftarrow G_1,\ G_2,\ t_1 \neq t_2,\ G_3$ be a clause in P_1. Let P_2 be the program derived from P_1 by replacing clause C_1 by clause C_2: $H \leftarrow G_1,\ t_1 \neq t_2,\ G_2,\ G_3$. Then: (i) P_2 is safe w.r.t. M, (ii) P_2 satisfies M, and (iii) for each non-basic atom A which satisfies mode M, A succeeds in P_1 iff A succeeds in P_2.

Proof: Point (i) follows from the fact that safety does not depend on the position of the disequation in a clause. Moreover, the evaluation of goal G_2 in program P_1 according our operational semantics, does not bind any variable in $t_1 \neq t_2$, and thus, we get Point (ii). Point (iii) is a consequence of Points (i) and (ii) and the fact that the evaluation of $t_1 \neq t_2$ does not bind any variable in the goals G_2 and G_3. □

The above proposition does not hold if we interchange clause C_1 and C_2. Consider, in fact, the following clause which is safe w.r.t. mode $M = \{p(+), q(+)\}$:

C_3. $p(X) \leftarrow X \neq Y,\ q(Z)$

This clause satisfies M because for all derivations starting from a ground instance $p(t)$ of $p(X)$ the atom $t \neq Y$ does not succeed. In contrast, if we use the clause C_4: $p(X) \leftarrow q(Z), X \neq Y$, we have that in the derivation starting from $p(t)$, the variable Z is not bound to a ground term and thus, clause C_4 does not satisfy the mode M which has the element $q(+)$.

4.3. Semideterministic Programs

Now we introduce the concept of *semideterminism* which characterizes a class of programs we will consider in this paper.

Definition 7 (Semideterminism) A program P is *semideterministic* w.r.t. a non-basic atom A iff for each goal (A_0, G_0) where A_0 is a non-basic atom and $A \mapsto_P^* (A_0, G_0)$, there exists *at most one* goal (A_1, G_1) where A_1 is a non-basic atom, such that $(A_0, G_0) \Rightarrow_P (A_1, G_1)$.
Given a mode M for a program P, we say that P is *semideterministic* w.r.t. M iff for each non-basic atom A which satisfies M, P is semideterministic w.r.t. A.

Notice that given a program P which is semideterministic w.r.t. a given mode M, a non-basic predicate p in P defines on the Herbrand base a relation which is not necessarily functional w.r.t. the input arguments of p. For instance, the program P:

$p(0, X) \leftarrow q(0)$
$p(1, X) \leftarrow q(X)$
$q(1) \leftarrow$
$q(2) \leftarrow$

is semideterministic w.r.t. the mode $\{p(+, ?), q(?)\}$, and p denotes a relation which is *not* functional w.r.t. its first argument, because $M(P) \models p(1, 1)$ and $M(P) \models p(1, 2)$. Notice that program P is *not* semideterministic w.r.t. the mode $\{p(?, ?), q(?)\}$.

Now we give a simple sufficient condition which ensures semideterminism. It is based on the concept of *mutually exclusive* clauses which we introduce below. We need some preliminary definitions.

Definition 8 (Satisfiability of Disequations w.r.t. a Set of Variables) Given a set V of variables, we say that a conjunction D of disequations, is *satisfiable w.r.t. V* iff there exists a ground substitution σ with domain V, such that every ground instance of $D\sigma$ holds (see Section 3.2). In particular, D is satisfiable w.r.t. $\emptyset$ iff every ground instance of D holds.

The satisfiability of a conjunction D of disequations w.r.t. a given set V of variables, can be checked by using the following algorithm defined by structural induction:

(1) *true*, i.e., the empty conjunction of disequations, is satisfiable w.r.t. V,

(2) (D_1, D_2) is satisfiable w.r.t. V iff both D_1 and D_2 are satisfiable w.r.t. V,

(3) $X \neq t$ is satisfiable w.r.t. V iff X occurs in V and t is either a non-variable term or a variable occurring in V distinct from X,

(4) $t \neq X$ is satisfiable w.r.t. V iff $X \neq t$ is satisfiable w.r.t. V,

(5) $f(\ldots) \neq g(\ldots)$, where f and g are distinct function symbols, is satisfiable w.r.t. V, and

(6) $f(t_1, \ldots, t_m) \neq f(u_1, \ldots, u_m)$ is satisfiable w.r.t. V iff at least one disequation among $t_1 \neq u_1, \ldots, t_m \neq u_m$ is satisfiable w.r.t. V.

The correctness of this algorithm relies on the fact that the set of function symbols is infinite (see Section 3.1).

Definition 9 (Linearity) A program P is said to be *linear* iff every clause of P has at most one non-basic atom in its body.

Definition 10 (Guard of a Clause) The *guard* of a clause C, denoted $grd(C)$, is $bd(C)$ if all atoms in $bd(C)$ are disequations, otherwise $grd(C)$ is the (possibly empty) conjunction of the disequations occurring in $bd(C)$ to the left of the leftmost atom which is not a disequation.

Definition 11 (Mutually Exclusive Clauses) Let us consider a mode M for the following two, renamed apart clauses:

$C_1.\ p(t_1, u_1) \leftarrow G_1$
$C_2.\ p(t_2, u_2) \leftarrow G_2$

where: (i) p is a predicate of arity k (≥ 0) whose first h arguments, with $0 \leq h \leq k$, are input arguments according to M, (ii) t_1 and t_2 are h-tuples of terms denoting the input arguments of p, and (iii) u_1 and u_2 are $(k-h)$-tuples of terms.
We say that C_1 and C_2 are *mutually exclusive* w.r.t. mode M iff either (i) t_1 is not unifiable with t_2 or (ii) t_1 and t_2 are unifiable via an mgu ϑ and $(grd(C_1), grd(C_2))\vartheta$ is not satisfiable w.r.t. $vars(t_1, t_2)$.
If $h = 0$ we stipulate that the empty tuples t_1 and t_2 are unifiable via an mgu which is the identity substitution.

We have the following lemmata.

Lemma 4.3. Let us consider a program P and a conjunction D of disequations. D succeeds in P *iff* every ground instance of D holds.

Proof: Let us consider the conjunction $(r_1 \neq s_1, \ldots, r_k \neq s_k)$ of disequations. Every ground instance of $(r_1 \neq s_1, \ldots, r_k \neq s_k)$ holds iff for $i = 1, \ldots, k$, and for every ground substitution σ, $r_i\sigma \neq s_i\sigma$ holds iff for $i = 1, \ldots, k$, and for every ground substitution σ, $r_i\sigma$ is a ground term different from $s_i\sigma$ iff for $i = 1, \ldots, k$, it does not exist a ground substitution σ such that $r_i\sigma$ and $s_i\sigma$ are the same ground term iff for $i = 1, \ldots, k$, r_i and s_i are not unifiable iff $(r_1 \neq s_1, \ldots, r_k \neq s_k)$ succeeds in P. □

Lemma 4.4. Let P be a program which is safe w.r.t. mode M and satisfies mode M. Let the non-unit clauses of P be pairwise mutually exclusive w.r.t. mode M. Given any non-basic atom A_0 which satisfies M, and any basic goal G_0, there exists at most one goal (A_1, G_1) such that A_1 is a non-basic atom and $(A_0, G_0) \Rightarrow_P (A_1, G_1)$.

Proof: By the definition of the $\Rightarrow_P$ relation (see Section 3.3), we need to prove that for any non-basic atom A_0 which satisfies M, and any basic goal G_0, there exists at most one goal (A_1, G_1) where A_1 is a non-basic atom, such that: (i) $(A_0, G_0) \mapsto_P^* (A_1, G_1)$, and (ii) the relation $\mapsto_P^*$ is constructed by first applying exactly once Point (3) of our operational semantics, and then applying to the resulting goal Points (1) and (2) of our operational semantics, as many times as required to evaluate the leftmost basic atoms, if any.

Since the non-unit clauses of P are pairwise mutually exclusive w.r.t. M, for any given non-basic atom A_0 which satisfies M, there exists at most one non-unit clause, say C, of P such that A_0 unifies with $hd(C)$ via an mgu, say μ, and $grd(C)\mu$ succeeds in P. In fact, suppose to the contrary, that there were two such non-unit clauses, say C_1 and C_2. Suppose that, for $j = 1, 2$, clause C_j is renamed apart and it is of the form:

$C_j.\ p(t_j, u_j) \leftarrow grd_j,\ K_j,$

where: (i) t_j is a tuple of terms denoting the input arguments of p and (ii) the goal grd_j is the guard of C_j, that is, a conjunction of disequations such that the leftmost atom of the goal K_j is not a disequation.

Suppose that for $j = 1, 2$, $hd(C_j)$ unifies with A_0 via the mgu ϑ_j. Since A_0 satisfies M, for $j = 1, 2$, the input variables of $hd(C_j)$ are bound by ϑ_j to ground terms. Since t_1 and t_2 have

a common ground instance, namely $t_1\vartheta_1 (= t_2\vartheta_2)$, they have a relevant mgu ϑ whose domain is a subset of $vars(t_1, t_2)$, and there exists a ground substitution σ with domain $vars(t_1, t_2)$ such that $t_1\vartheta_1 = t_1\vartheta\sigma (= t_2\vartheta_2 = t_2\vartheta\sigma)$. Moreover, since the clauses C_1 and C_2 are renamed apart, we have that:

(Property α) for $j = 1, 2$, if restrict $\vartheta\sigma$ to $vars(t_j)$ then $\vartheta_j = \vartheta\sigma$.

By hypothesis, both $grd_1\vartheta_1$ and $grd_2\vartheta_2$ succeed in P. Thus, by Lemma 4.3, every ground instance of $grd_1\vartheta_1$ and $grd_2\vartheta_2$ holds. (Recall that the goals $grd_1\vartheta_1$ and $grd_2\vartheta_2$ are ground goals, except for the local variables of each disequation occurring in them.)

Since P is safe w.r.t. M, for $j = 1, 2$, every variable occurring in a disequation of grd_j either occurs in t_j or it is a local variable of that disequation in C_j. Thus, by Property (α), $grd_1\vartheta_1 = grd_1\vartheta\sigma$ and $grd_2\vartheta_2 = grd_2\vartheta\sigma$. Since every ground instance of $grd_1\vartheta_1$ and $grd_2\vartheta_2$ holds, we have that every ground instance of $(grd_1\vartheta\sigma, grd_2\vartheta\sigma)$ holds. In other words, there exists a ground substitution σ whose domain is $vars(t_1, t_2)$, such that every ground instance of $(grd_1, grd_2)\vartheta\sigma$ holds. By definition, this means that $(grd_1, grd_2)\vartheta$ is satisfiable w.r.t. $vars(t_1, t_2)$. This contradicts the fact that the non-unit clauses of P are mutually exclusive w.r.t. M.

We conclude that for any given non-basic atom A_0 which satisfies M, A_0 unifies via an mgu, say μ, with the head of at most one non-unit clause, say C, of P such that $grd(C)\mu$ succeeds in P.

Now there are two cases: (Case i) A_0 unifies with the head of the clauses in $\{C, D_1, \ldots, D_n\}$, where $n \geq 0$, C is a non-unit clause, and clauses $D_1, \ldots, D_n$ are all unit clauses, and (Case ii) A_0 unifies with the head of the clauses in $\{D_1, \ldots, D_n\}$, where $n \geq 0$ and these clauses are all unit clauses.

Let us consider Case (i). Let clause C be of the form: $H \leftarrow K$ for some non-basic goal K. For any basic goal G_0, by applying once Point (3) of our operational semantics, we have that: $(A_0, G_0) \mapsto_P (K, G_0)\mu$. Thus, $(K, G_0)\mu$ is of the form (Bs, G_2) where Bs is a conjunction of basic atoms and the leftmost atom of G_2 is non-basic. Since for any basic atom B and goal G_3, there exists at most one goal G_4 such that $(B, G_3) \mapsto_P G_4$, by using Points (1) and (2) of our operational semantics, we have that there exists at most one goal (A_1, G_1) such that $(Bs, G_2) \mapsto^*_P (A_1, G_1)$, where the atom A_1 is non-basic.

Every other derivation starting from (A_0, G_0) by applying Point (3) of our operational semantics using a clause in $\{D_1, \ldots, D_n\}$, is such that if for some goal G_5 we have that $(A_0, G_0) \mapsto^*_P G_5$, then G_5 is a basic goal, because from a basic goal we cannot derive a non-basic one. This concludes the proof of the Lemma in Case (i). The proof in Case (ii) is analogous to that of Case (i). □

The following proposition allows us to prove that a program is semideterministic.

Proposition 4.5 (Sufficient Condition for Semideterminism) If (i) P is a linear program, (ii) P is safe w.r.t. a given mode M, (iii) P satisfies M, and (iv) the non-unit clauses of P are pairwise mutually exclusive w.r.t. M, then P is semideterministic w.r.t. M.

Proof: Take a non-basic atom A which satisfies M. Every non-basic atom A_0 such that $A \mapsto^*_P (A_0, G_0)$ for some goal G_0, satisfies M because P satisfies M. Since P is linear, G_0 is a basic goal. By Lemma 4.4 there exists at most one goal (A_1, G_1) where A_1 is a non-basic atom, such that $(A_0, G_0) \Rightarrow_P (A_1, G_1)$. This means that P is semideterministic w.r.t. M. □

In Section 5, we will present a strategy for deriving specialized programs which satisfies the hypotheses of the above Proposition 4.5, and thus, these derived programs are semideterministic.

The following examples show that in Proposition 4.5 no hypothesis on program P can be discarded.

Example 6. Consider the following program P and the mode $M = \{p, q\}$ for P:

1. $p \leftarrow q,\ q$
2. $q \leftarrow$
3. $q \leftarrow q$

P is not linear, but P is safe w.r.t. M and P satisfies M. The non-unit clauses of P which are the clauses 1 and 3, are pairwise mutually exclusive. However, P is not semideterministic w.r.t. M, because $p \mapsto_P^* (q, q)$, and there exist two non-basic goals, namely q and (q, q), such that $(q, q) \Rightarrow_P q$ and $(q, q) \Rightarrow_P (q, q)$.

Example 7. Consider the following program Q and the mode $M = \{p(?), q_1, q_2\}$ for Q:

1. $p(X) \leftarrow X \neq 0,\ q_1$
2. $p(1) \leftarrow q_2$

Q is linear and it satisfies M, but Q is not safe w.r.t. M because X is not an input variable of p. Clauses 1 and 2 are mutually exclusive w.r.t. M, because the set of input variables in $p(X)$ is empty and $X \neq 0$ is not satisfiable w.r.t. $\emptyset$. However, Q is not semideterministic w.r.t. M, because $p(1) \mapsto_Q^* p(1)$, and there exist two non-basic goals, namely q_1 and q_2, such that $p(1) \Rightarrow_Q q_1$ and $p(1) \Rightarrow_Q q_2$.

Example 8. Consider the following program R and the mode $M = \{p, r(+), r_1, r_2\}$ for R:

1. $p \leftarrow r(X)$
2. $r(1) \leftarrow r_1$
3. $r(2) \leftarrow r_2$

R is linear and safe w.r.t. M, but R does not satisfy M, because $p \mapsto_R r(X)$ and X is not a ground term. Clauses 1, 2, and 3 are pairwise mutually exclusive. However, R is not semideterministic w.r.t. M, because $p \mapsto_R^* r(X)$ and there exist two non-basic goals, namely r_1 and r_2, such that $r(X) \Rightarrow_R r_1$ and $r(X) \Rightarrow_R r_2$.

Example 9. Consider the following program S and the mode $M = \{p, r_1, r_2\}$ for S:

1. $p \leftarrow r_1$
2. $p \leftarrow r_2$

S is linear and safe w.r.t. M, and S satisfies M. Clauses 1 and 2 are not pairwise mutually exclusive. S is not semideterministic w.r.t. M, because $p \mapsto_S^* p$, and there exist two non-basic goals, namely r_1 and r_2, such that $p \Rightarrow_S r_1$ and $p \Rightarrow_S r_2$.

5. A Transformation Strategy for Specializing Programs and Reducing Nondeterminism

In this section we present a strategy, called *Determinization*, for guiding the application of the transformation rules presented in Section 3.4 so to derive efficient, specialized programs with reduced nondeterminism. In particular, we get derived programs which are semideterministic. As we will indicate in Section 7, our strategy is an enhancement of the specialization technique [5], called *conjunctive partial deduction*, which allows the introduction of new predicates defined in terms of conjunctions of atoms.

Our Determinization Strategy is based upon three subsidiary strategies: (i) the *Unfold-Simplify* subsidiary strategy, which uses the safe unfolding, equation elimination, disequation replacement, and subsumption rules, (ii) the *Partition* subsidiary strategy, which uses the safe case split, equation elimination, disequation replacement, subsumption, and safe head generalization rules, and (iii) the *Define-Fold* subsidiary strategy which uses the definition introduction and safe folding rules. More details on these subsidiary strategies will be given below in Sections 5.1, 5.2, and 5.3, respectively.

Given an initial program P, a mode M for P, and an atom $p(t_1, \ldots, t_h)$ w.r.t. which we want to specialize P, we introduce by the definition introduction rule, the clause

S: $p_s(X_1, \ldots, X_r) \leftarrow p(t_1, \ldots, t_h)$

where $X_1, \ldots, X_r$ are the distinct variables occurring in $p(t_1, \ldots, t_h)$.
We also define a mode $p_s(m_1, \ldots, m_r)$ for the predicate p_s by stipulating that: for any $j = 1, \ldots, r$, m_j is $+$ iff X_j is an input variable of $p(t_1, \ldots, t_h)$ according to the mode M. We assume that the program P is safe w.r.t. M. Thus, also program $P \cup \{S\}$ is safe w.r.t. $M \cup \{p_s(m_1, \ldots, m_r)\}$. We also assume that P satisfies mode M and thus, program $P \cup \{S\}$ satisfies mode $M \cup \{p_s(m_1, \ldots, m_r)\}$.

Our Determinization Strategy is presented below as an iterative procedure that, at each iteration, manipulates the following three sets of clauses: (1) *Defs*, which is the set of clauses introduced by the definition introduction rule, (2) *Cls*, which is the set of clauses to be transformed during the current iteration, and (3) *TransfP*, which is the set of clauses from which we will construct the specialized program. Initially, *Cls* consists of the single clause S: $p_s(X_1, \ldots, X_r) \leftarrow p(t_1, \ldots, t_h)$ which is constructed as we have indicated above. From the set *Cls* a new, semideterministic set of clauses, namely *UnitCls* $\cup$ *FoldedCls*, is derived by applying the transformation rules according to the Unfold-Simplify, Partition, and Define-Fold subsidiary strategies, and this new set is added to *TransfP*. The derivation of *UnitCls* $\cup$ *FoldedCls* may require the introduction of new predicates by applying the definition introduction rule. At the end of each iteration, the set *NewDefs* of clauses that define the new predicates introduced during that iteration, is added to *Defs*, and the value of the set *Cls* is updated to *NewDefs*. The transformation strategy terminates when *Cls* $= \emptyset$, that is, no new predicate is introduced during the current iteration.

Determinization Strategy

Input: A program P, an atom $p(t_1, \ldots, t_h)$ w.r.t. which we want to specialize P, and a mode M for P such that P is safe w.r.t. M and P satisfies M.

Output: A specialized program P_s, and an atom $p_s(X_1, \ldots, X_r)$, with $\{X_1, \ldots, X_r\} = vars(p(t_1, \ldots, t_h))$ such that: (i) for every ground substitution $\vartheta = \{X_1/u_1, \ldots, X_r/u_r\}$, $M(P) \models p(t_1, \ldots, t_h)\vartheta$ iff $M(P_s) \models p_s(X_1, \ldots, X_r)\vartheta$, and (ii) for every substitution $\sigma = \{X_1/v_1, \ldots, X_r/v_r\}$ such that the atom $p(t_1, \ldots, t_h)\sigma$ satisfies mode M, we have that: (ii.1) $p(t_1, \ldots, t_h)\sigma$ succeeds in P iff $p_s(X_1, \ldots, X_r)\sigma$ succeeds in P_s, and (ii.2) P_s is semideterministic w.r.t. $p_s(X_1, \ldots, X_r)\sigma$.

Initialize. Let S be the clause $p_s(X_1, \ldots, X_r) \leftarrow p(t_1, \ldots, t_h)$.
Defs $:= \{S\}$; *Cls* $:= \{S\}$; *TransfP* $:= P$; $M_s := M \cup \{p_s(m_1, \ldots, m_r)\}$, where for any $j = 1, \ldots, r$, $m_j = +$ iff X_j is an input variable of $p(t_1, \ldots, t_h)$ according to the mode M;

while *Cls* $\neq \emptyset$ **do**

(1) *Unfold-Simplify.* We apply the safe unfolding, equation elimination, disequation replacement, and subsumption rules according to the Unfold-Simplify Strategy given in Section 5.1

below, and from *Cls* we derive a new set of clauses *UnfoldedCls*.

(2) *Partition.* Let *UnitCls* be the unit clauses occurring in *UnfoldedCls*, and *NonunitCls* be the set of non-unit clauses in *UnfoldedCls*.

We apply the safe case split, equation elimination, disequation replacement, and safe head generalization rules according to the Partition Strategy given in Section 5.2 below, and from *NonunitCls* we derive a set *PartitionedCls* of clauses which is the union of disjoint subsets of clauses. Each subset is called a *packet*. The packets of *PartitionedCls* enjoy the following properties:

(2a) each packet is a set of clauses of the form (modulo variable renaming):

$$\begin{cases} H \leftarrow \mathit{Diseqs}, G_1 \\ \quad \ldots \\ H \leftarrow \mathit{Diseqs}, G_m \end{cases}$$

where *Diseqs* is a conjunction of disequations and for $k = 1, \ldots, m$, no disequation occurs in G_k, and

(2b) for any two clauses C_1 and C_2, if the packet of C_1 is different from the packet of C_2, then C_1 and C_2 are mutually exclusive w.r.t. mode M_s.

(3) *Define-Fold.* We apply the definition introduction and the safe folding rules according to the Define-Fold subsidiary strategy given in Section 5.3 below. According to that strategy, we introduce a minimal (possibly empty) set *NewDefs* of new definition clauses and a set M_{new} of modes such that:

(3a) in M_{new} there exists exactly one mode for each distinct head predicate in *NewDefs*, and

(3b) by applying the folding rule, which is safe w.r.t. M_{new}, using the clauses in $\mathit{Defs} \cup \mathit{NewDefs}$, from each packet in *PartitionedCls* we derive a single clause of the form: $H \leftarrow \mathit{Diseqs}, \mathit{newp}(\ldots)$.

Let *FoldedCls* be the set of clauses derived by folding the packets in *PartitionedCls*.

(4) $\mathit{Defs} := \mathit{Defs} \cup \mathit{NewDefs}$; $\mathit{Cls} := \mathit{NewDefs}$; $\mathit{TransfP} := \mathit{TransfP} \cup \mathit{UnitCls} \cup \mathit{FoldedCls}$; $M_s := M_s \cup M_{new}$

end-while

We derive the specialized program P_s by applying the definition elimination rule and keeping only the clauses of *TransfP* on which p_s depends.

We now show that, if the Determinization Strategy terminates, then the least Herbrand model and the operational semantics are preserved. Moreover, the derived specialized program P_s is semideterministic w.r.t. $p_s(X_1, \ldots, X_r)\sigma$ as stated in the following theorem.

Theorem 5.1 (Correctness of the Determinization Strategy) Let us consider a program P, a non-basic atom $p(t_1, \ldots, t_h)$, and a mode M for P such that: (1) P is safe w.r.t. M and (2) P satisfies M. If the Determinization Strategy terminates with output program P_s and output atom $p_s(X_1, \ldots, X_r)$ where $\{X_1, \ldots, X_r\} = \mathit{vars}(p(t_1, \ldots, t_h))$, then

(i) for every ground substitution $\vartheta = \{X_1/u_1, \ldots, X_r/u_r\}$,

$M(P) \models p(t_1, \ldots, t_h)\vartheta$ iff $M(P_s) \models p_s(X_1, \ldots, X_r)\vartheta$ and

(ii) for every substitution $\sigma = \{X_1/v_1, \ldots, X_r/v_r\}$ such that the atom $p(t_1, \ldots, t_h)\sigma$ satisfies mode M,

(ii.1) $p(t_1, \ldots, t_h)\sigma$ succeeds in P iff $p_s(X_1, \ldots, X_r)\sigma$ succeeds in P_s, and

(ii.2) P_s is semideterministic w.r.t. $p_s(X_1, \ldots, X_r)\sigma$.

Proof: Let *Defs* and P_s be the set of definition clauses and the specialized program obtained at the end of the Determinization Strategy.

(i) Since $p_s(X_1, \ldots, X_r) \leftarrow p(t_1, \ldots, t_h)$ is the only clause for p_s in $P \cup \mathit{Defs}$ and $\{X_1, \ldots, X_r\} = \mathit{vars}(p(t_1, \ldots, t_h))$, for every ground substitution $\vartheta = \{X_1/u_1, \ldots, X_r/u_r\}$ we have that $M(P) \models p(t_1, \ldots, t_h)\vartheta$ iff $M(P \cup \mathit{Defs}) \models p_s(X_1, \ldots, X_r)\vartheta$. By the correctness of the transformation rules w.r.t. the least Herbrand model (see Theorem 3.3), we have that $M(P \cup \mathit{Defs}) \models p_s(X_1, \ldots, X_r)\vartheta$ iff $M(P_s) \models p_s(X_1, \ldots, X_r)\vartheta$.

Point (ii.1) follows from Theorem 4.1 because during the Determinization Strategy, each application of the unfolding, folding, head generalization, and case split rule is safe.

(ii.2) We first observe that, by construction, for every substitution σ, the atom $p(t_1, \ldots, t_h)\sigma$ satisfies mode M iff $p_s(X_1, \ldots, X_r)\sigma$ satisfies mode M_s, where M_s is the mode obtained from M at the end of the Determinization Strategy. Thus, Point (ii.2) can be shown by proving that P_s is semideterministic w.r.t. M_s. In order to prove this fact, it is enough to prove that $\mathit{TransfP}_w - P$ is semideterministic w.r.t. M_s, where $\mathit{TransfP}_w$ is the set of clauses which is the value of the variable *TransfP* at the end of the while-do statement of the Determinization Strategy. Indeed, P_s is equal to $\mathit{TransfP}_w - P$ because, by construction, p_s does not depend on any clause of P, and thus, by the final application of the definition elimination rule, all clauses of P are removed from $\mathit{TransfP}_w$.

By Lemma 4.5, it is enough to prove that: (a) $\mathit{TransfP}_w - P$ is linear, (b) $\mathit{TransfP}_w - P$ is safe w.r.t. M_s, (c) $\mathit{TransfP}_w - P$ is satisfies M_s, and (d) the non-unit clauses of $\mathit{TransfP}_w - P$ are pairwise mutually exclusive w.r.t. M_s.

Property (a) holds because according to the Determinization Strategy, after every application of the safe folding rule we get a clause of the form: $H \leftarrow \mathit{Diseqs}, \mathit{newp}(\ldots)$, where a single non-basic atom occurs in the body. All other clauses in $\mathit{TransfP}_w - P$ are unit clauses.

Properties (b) and (c) follow from Theorem 4.1 recalling that the application of the folding, head generalization, and case split rules are all safe.

Property (d) can be proved by computational induction. (*Basis*) Initially, *TransfP*–P is empty and thus, all its non-unit clauses are pairwise mutually exclusive w.r.t. M_s. (*Step*) At the end of each execution of the body of the while-do (see Point (4) of the strategy), the non-unit clauses which are added to the current value of *TransfP* are the elements of the set *FoldedCls* and those non-unit clauses are derived by applying the Partition and Define-Fold subsidiary strategies at Points (3) and (4), respectively. By construction, the clauses in *FoldedCls* are pairwise mutually exclusive w.r.t. M_{new}, and their head predicates do not occur in *TransfP*. Thus, the clauses of $\mathit{TransfP} \cup \mathit{UnitCls} \cup \mathit{FoldedCls}$ are pairwise mutually exclusive w.r.t. $M_s \cup M_{new}$. As a consequence, after the two assignments (see Point (4) of the strategy) $\mathit{TransfP} := \mathit{TransfP} \cup \mathit{UnitCls} \cup \mathit{FoldedCls}$ and $M_s := M_s \cup M_{new}$, we have that Property (d) holds at the beginning of the next execution of the body of the while-do, and the proof of the Step case is completed. □

We now describe the three subsidiary strategies for realizing the *Unfold-Simplify*, *Partition*, and *Define-Fold* transformations as specified by the Determinization Strategy. We will see these subsidiary strategies in action in the examples of Section 6.

During the application of our subsidiary strategies it will be convenient to rewrite every safe clause into its *normal form*. The normal form N of a safe clause can be constructed by performing disequation replacements and disequation promotions, so that the following Properties N1–N5 hold:

(N1) every disequation is of the form: $X \neq t$, with t different from X and unifiable with X,
(N2) every disequation occurs in $bd(N)$ to the left of every atom different from a disequation,
(N3) if $X \neq Y$ occurs in $bd(N)$ and both X and Y are input variables of $hd(N)$, then in $hd(N)$ the leftmost occurrence of X is to the left of the leftmost occurrence of Y,
(N4) for every disequation of the form $X \neq Y$ where Y is an input variable, we have that also X is an input variable, and
(N5) for any pair of disequations d_1 and d_2 in $bd(N)$, it does not exist a substitution ρ which is a bijective mapping from the set of the local variables of d_1 in N onto the set of the local variables of d_2 in N such that $d_1\rho = d_2$. Thus, in particular, no two equal disequations occur in the normal form of a safe clause.

We have that: (i) the normal form of a safe clause is unique, modulo variable renaming and disequation promotion, and (ii) given a program P and a mode M for P such that P is safe w.r.t. M and P satisfies M, if we rewrite a clause of P into its normal form, then the least Herbrand model semantics and the operational semantics are preserved (this fact is a consequence of Theorem 3.3, Theorem 4.1, and Proposition 4.2).

A safe clause for which Properties N1–N5 hold, is said to be *in normal form*. If a clause C is in normal form, then by Property N2, every disequation in $bd(C)$ occurs also in $grd(C)$.

5.1. The Unfold-Simplify Subsidiary Strategy

The Unfold-Simplify Strategy first produces instances of the clauses in *Cls* by unfolding them w.r.t. the leftmost atom in their body, and then it keeps unfolding the derived clauses as long as input variables are not instantiated. Now, for giving a formal definition of the Unfold-Simplify Strategy we introduce the following concept.

Definition 12 (Consumer Atom) Let P be a program and M a mode for P. A non-basic atom $q(t_1, \ldots, t_k)$ is said to be a *consumer atom* iff for every non-unit clause in P whose head unifies with that non-basic atom via an mgu ϑ, we have that for $i = 1, \ldots, k$, if t_i is an input argument of q then $t_i\vartheta$ is a variant of t_i.

The Unfold-Simplify Strategy is realized by the following *Unfold-Simplify* procedure, where the expression *Simplify*(S) denotes the set of clauses derived from a given set S of clauses, by: (1) first, applying whenever possible, the equation elimination rule to the clauses in S, (2) then, rewriting the derived clauses into their normal form, and (3) finally, applying as long as possible the subsumption rule.

Procedure *Unfold-Simplify*(*Cls*, *UnfoldedCls*).
Input: A set *Cls* of clauses in a program P and a mode M_s for P. P is safe w.r.t. M_s and for each $C \in$ *Cls*, the input variables of the leftmost non-basic atom in the body of C are input variables of the head of C.
Output: A new set *UnfoldedCls* of clauses which are derived from *Cls* by applying the safe unfolding, equation elimination, disequation replacement, and subsumption rules. The clauses in *UnfoldedCls* are safe w.r.t. M_s.

Unfold w.r.t. Leftmost Non-basic Atom:

UnfoldedCls := {E | there exists a clause $C \in$ *Cls* and clause E is derived by unfolding C w.r.t. the leftmost non-basic atom in its body};
UnfoldedCls := *Simplify*(*UnfoldedCls*)

Unfold w.r.t. Leftmost Consumer Atom:

while there exists a clause $C \in$ *UnfoldedCls* whose body has a leftmost consumer atom, say A, such that the unfolding of C w.r.t. A is safe **do**
UnfoldedCls := (*UnfoldedCls* − {C}) ∪ {E | E is derived by unfolding C w.r.t. A};
UnfoldedCls := *Simplify*(*UnfoldedCls*)

end-while

Notice that, by the hypotheses on the input program P and clauses *Cls*, the first unfolding step performed by the *Unfold-Simplify* procedure is safe.

This strategy differs from usual unfolding strategies for partial evaluation of logic programs (see, for instance, [10]), because mode information is used. We have found this strategy very effective on several examples as shown in the following Section 6. However, our strategy has a drawback in that it may fail to terminate. This drawback can be remedied by applying techniques for *finite unfolding* such as those based on *well-quasi orders* [17] or *well-founded measures* [24].

5.2. The Partition Subsidiary Strategy

The Partition Strategy is realized by the following procedure, where we will write $p(t, u)$ to denote an atom with non-basic predicate p of arity k (≥ 0), such that: (i) t is an h-tuple of terms, with $0 \leq h \leq k$, denoting the h input arguments of p, and (ii) u is a $(k-h)$-tuple of terms denoting the arguments of p which are *not* input arguments.

Procedure *Partition*(*NonunitCls*, *PartitionedCls*).
Input: A set *NonunitCls* of non-unit clauses in normal form and without variables in common. A mode M_s for *NonunitCls*. The clauses in *NonunitCls* are safe w.r.t. M_s.
Output: A set *PartitionedCls* of clauses which is the union of disjoint packets of clauses such that:
(2a) each packet is a set of clauses of the form (modulo variable renaming):

$$\begin{cases} H \leftarrow Diseqs, G_1 \\ \quad \ldots \\ H \leftarrow Diseqs, G_m \end{cases}$$

where *Diseqs* is a conjunction of disequations and for $k = 1, \ldots, m$, no disequation occurs in G_k, and
(2b) for any two clauses C_1 and C_2, if the packet of C_1 is different from the packet of C_2, then C_1 and C_2 are mutually exclusive w.r.t. mode M_s.
The clauses in *PartitionedCls* are in normal form and they are safe w.r.t. M_s.

while there exist in *NonunitCls* two clauses of the form:

C_1. $p(t_1, u_1) \leftarrow Body_1$
C_2. $p(t_2, u_2) \leftarrow Body_2$

such that: (i) C_1 and C_2 are not mutually exclusive w.r.t. mode M_s, and *either*
(ii.1) t_1 is not a variant of t_2 *or*
(ii.2) t_1 is a variant of t_2 via an mgu ϑ such that $t_1\vartheta = t_2$, and for any substitution ρ which is a bijective mapping from the set of local variables of $grd(C_1\vartheta)$ in $C_1\vartheta$ onto the set of local variables of $grd(C_2)$ in C_2, $grd(C_1\vartheta\rho)$ cannot be made equal to $grd(C_2)$ by applying disequation promotion **do**

We take a binding X/r as follows.

(Case 1) Suppose that t_1 is *not* a variant of t_2. In this case, since C_1 and C_2 are not mutually exclusive, we have that t_1 and t_2 are unifiable and, for some $i, j \in \{1, 2\}$, with $i \neq j$, there exists an mgu ϑ of t_i and t_j and a binding Y/t_a in ϑ such that $t_j\{Y/t_a\}$ is not a variant of t_j. Without loss of generality we may assume that $i = 1$ and $j = 2$. Then we take the binding X/r to be Y/t_a.

(Case 2) Suppose that t_1 is a variant of t_2 via an mgu ϑ. Now every clause whose normal form has a disequation of the form $X \neq t$ where X is a local variable, is mutually exclusive w.r.t. any other clause. Thus, for some $i, j \in \{1, 2\}$, with $i \neq j$, there exists a disequation $(Y \neq t_a)\vartheta$ in $grd(C_i\vartheta)$ where $Y\vartheta$ is an input variable of $hd(C_i\vartheta)$, such that for any substitution ρ which is a bijective mapping from the set of local variables of $grd(C_i\vartheta)$ in $C_i\vartheta$ onto the set of local variables of $grd(C_j\vartheta)$ in $C_j\vartheta$ and for every disequation $(Z \neq t_b)\vartheta$ in $grd(C_j\vartheta)$, we have that $(Y \neq t_a)\vartheta\rho$ is different from $(Z \neq t_b)\vartheta$. We also have that $Y\vartheta$ is an input variable of $hd(C_j\vartheta)$. Without loss of generality we may assume that $i = 1$, $j = 2$, $t_1\vartheta = t_2$, and $C_2\vartheta = C_2$. Then we take the binding X/r to be $(Y/t_a)\vartheta$.

We apply the case split rule to clause C_2 w.r.t. X/r, that is, we derive the two clauses:

C_{21}. $(p(t_2, u_2) \leftarrow Body_2)\{X/r\}$
C_{22}. $p(t_2, u_2) \leftarrow X \neq r, Body_2$

This application of the case split rule is safe because: (i) clauses C_1 and C_2 are safe w.r.t. M_s, (ii) X is an input variable of $hd(C_{22})$ (recall that our choice of X/r in Case 2 ensures that X is an input variable of $hd(C_2)$), and (iii) each variable in r is either an input variable of $hd(C_{22})$ or a local variable of $X \neq r$ in C_{22}. Thus, clauses C_{21} and C_{22} are safe w.r.t. mode M_s and they are also mutually exclusive w.r.t. M_s.

We update the value of *NonunitCls* as follows:
$NonunitCls := (NonunitCls - \{C_2\}) \cup \{C_{21}, C_{22}\}$
$NonunitCls := Simplify(NonunitCls)$.

end-while

Now the set *NonunitCls* is partitioned into subsets of clauses and after suitable variable renaming and disequation promotion, each subset is of the form:

$$\begin{cases} p(t, u_1) \leftarrow Diseqs, Goal_1 \\ \quad \cdots \\ p(t, u_m) \leftarrow Diseqs, Goal_m \end{cases}$$

where *Diseqs* is a conjunction of disequations and for $k = 1, \ldots, m$, no disequation occurs in $Goal_k$, and any two clauses in different subsets are mutually exclusive w.r.t. mode M_s.

Then we process every subset of clauses we have derived, by applying the safe head generalization rule so to replace the non-input arguments in the heads of the clauses belonging to the same

subset by their most specific common generalization. Thus, every subset of clauses will eventually take the form:

$$\begin{cases} p(t,u) \leftarrow Eqs_1, Diseqs, Goal_1 \\ \quad \ldots \\ p(t,u) \leftarrow Eqs_m, Diseqs, Goal_m \end{cases}$$

where u is the most specific common generalization of the terms $u_1, \ldots, u_m$ and, for $k = 1, \ldots, m$, the goal Eqs_k is a conjunction of the equations $V_1 = v_1, \ldots, V_r = v_r$ such that $u\{V_1/v_1, \ldots, V_r/v_r\} = u_k$.

Finally, we move all disequations to the leftmost positions of the body of every clause whereby getting the set *PartitionedCls*.

The following property is particularly important for the mechanization of our Determinization strategy.

Theorem 5.2. The Partition procedure terminates.

Proof: It is enough to show that the while-do statement in the Partition procedure terminates. To see this, let us first consider the set $NonunitCls_{in}$ which is the value of the set *NonunitCls* at the beginning of the execution of the while-do statement. $NonunitCls_{in}$ can be partitioned into maximal sets of clauses such that: (i) two clauses which belong to two distinct sets, are mutually exclusive, and (ii) if two clauses, say C_0 and C_{n+1}, belong to the same set, then there exists a sequence of clauses $C_0, C_1, \ldots, C_{n+1}$, with $n \geq 0$, such that for $i = 0, \ldots, n$, clauses C_i and C_{i+1} are not mutually exclusive.

For our termination proof it is enough to show the termination of the Partition procedure when starting from exactly one maximal set, say K, of the partition of $NonunitCls_{in}$ because during the execution of the Partition procedure, the replacement of a clause, say C_2, by the clauses, say C_{21} and C_{22}, satisfies the following property: if clauses C_2 and D are mutually exclusive then C_{21} and D are mutually exclusive and also C_{22} and D are mutually exclusive.

Let every clause of K be renamed apart and written in a form, called *equational form*, where the input arguments are generalized to new variables and these new variables are bound by equations in the body. The equational form of a clause C will be denoted by C^{eq}. For instance, given the clause C: $p(f(X), r(Y,Y), r(X,U)) \leftarrow Body$, with mode $p(+,+,?)$ for p, we have that C^{eq} is: $p(V,W,r(X)) \leftarrow V = f(X), W = r(Y,Y), Body$.

Let K^{eq} be the set $\{C^{eq} \mid C \in K\}$. Thus, K^{eq} has the following form:

$$\begin{cases} p(v_1,u_1) \leftarrow Eqs_1, Diseqs_1, Body_1 \\ \quad \ldots \\ p(v_n,u_n) \leftarrow Eqs_n, Diseqs_n, Body_n \end{cases}$$

where, for $i = 0, \ldots, n$: (1) v_i denotes a tuple of variables which are the input arguments of p, (2) u_i denotes a tuple of arguments of p which are *not* input arguments, (3) Eqs_i denotes a conjunction of equations of the form $X = t$, which bind the variables in v_i, (4) $Diseqs_i$ denotes a conjunction of disequations, and (5) $Body_i$ denotes a conjunction of atoms which are different from disequations (recall that the clauses in $NonunitCls_{in}$ are in normal form). Equations may occur also in $Body_i$, but they do not bind any input variable of $p(v_i, u_i)$.

Let us now introduce the following set $T = \{t \mid t$ is a term or a subterm occurring in Eqs_i or $Diseqs_i$ for some $i = 1, \ldots, n\}$.

Every execution of the body of the while-do statement of the Partition procedure works by replacing a safe clause, say C_2, by two new safe clauses, say C_{21} and C_{22}. We will prove the

termination of the Partition procedure by: (i) mapping the replacements it performs, onto the corresponding replacements of the clauses written in equational form in the set K^{eq}, and (ii) showing that the set K^{eq} cannot undergo an infinite number of such replacements.

Let us then consider the equational forms C_2^{eq}, C_{21}^{eq}, and C_{22}^{eq} of the clauses C_2, C_{21}, and C_{22}, respectively. We have that: (i) $bd(C_{21}^{eq})$ has one more equation of the form $X = r$ w.r.t. $bd(C_2^{eq})$, and (ii) $bd(C_{22}^{eq})$ has one more disequation of the form $X \neq r$ w.r.t. $bd(C_2^{eq})$. We also have that there exists only a finite number of pairs $\langle X, r \rangle$, because X is a variable symbol occurring in K^{eq} and r is a term occurring in the finite set $T \cup \{t \mid t$ is a term or a subterm occurring in an mgu of a *finite* number of elements of $T\}$. (We have considered mgu's of a finite number of elements of T, rather than mgu's of two elements only, because a finite number of clause heads in K may have the same common instance.)

Thus, in order to conclude the proof, it remains to show that before the replacement of C_2 by C_{21} and C_{22}, neither $X = r$ nor $X \neq r$ occurs in $bd(C_2^{eq})$. Here and in the rest of the proof, the notion of occurrence of an equation or a disequation is modulo renaming of the local variables. Indeed,
(1.1) $X \neq r$ does not occur in $bd(C_2^{eq})$ because X/r is a binding of an mgu of the input arguments of $hd(C_1)$ and $hd(C_2)$, and clauses C_1 and C_2 are not mutually exclusive, and thus, $X \neq r$ does not occur in $bd(C_2)$, and (1.2) $X = r$ does not occur in $bd(C_2^{eq})$ because X/r is, by construction, a binding of an mgu between the input arguments of the heads of the clauses C_1 and C_2 and these clauses are obtained as a result of the *Simplify* function which eliminates every occurrence of the variable X from C_2, and
– in Case (2): (2.1) $X = r$ does not occur in $bd(C_2^{eq})$ because, by hypothesis, a variant of $X \neq r$ occurs in $bd(C_1)$ and clauses C_1 and C_2 are not mutually exclusive, and (2.2) $X \neq r$ does not occur in $bd(C_2^{eq})$ because $X \neq r$ does not occur in $bd(C_2)$ (indeed, we choose $X \neq r$ precisely to satisfy this condition). □

When the Partition procedure terminates, it returns a set *PartitionedCls* of clauses which is the union of packets of clauses enjoying Properties (2a) and (2b) indicated in the Output specification of that procedure. These properties are a straightforward consequence of the termination condition of the while-do statement of that same procedure.

5.3. The Define-Fold Subsidiary Strategy

The Define-Fold Strategy is realized by the following procedure.

Procedure *Define-Fold*(*PartitionedCls*, *Defs*, *NewDefs*, *FoldedCls*).
Input: (i) A mode M_s, (ii) a set *PartitionedCls* of clauses which are safe w.r.t. M_s, and (iii) a set *Defs* of definition clauses. *PartitionedCls* is the union of the disjoint packets of clauses computed by the Partition subsidiary strategy.
Output: (i) A minimal, possibly empty, set *NewDefs* of definition clauses, together with a mode M_{new} consisting of exactly one mode for each distinct head predicate in *NewDefs*. For each $C \in$ *NewDefs*, the input variables of the leftmost non-basic atom in the body of C are input variables of the head of C. (ii) A set *FoldedCls* of folded clauses.
$NewDefs := \emptyset$; $M_{new} := \emptyset$; $FoldedCls := \emptyset$;
while there exists in *PartitionedCls* a packet Q of the form:

$$\begin{cases} H \leftarrow Diseqs,\ G_1 \\ \quad \cdots \\ H \leftarrow Diseqs,\ G_m \end{cases}$$

where *Diseqs* is a conjunction of disequations and for $k = 1, \ldots, m$, no disequation occurs in G_k, **do** $PartitionedCls := PartitionedCls - Q$ and apply the definition and safe folding rules as follows.

(Case α) Let us suppose that the set *Defs* of the available definition clauses contains a subset of clauses of the form:

$$\begin{cases} newq(X_1, \ldots, X_h) \leftarrow G_1 \\ \quad \ldots \\ newq(X_1, \ldots, X_h) \leftarrow G_m \end{cases}$$

such that: (i) they are all clauses in *Defs* for predicate *newq*, (ii) $X_1, \ldots, X_h$ include every variable which occurs in one of the goals $G_1, \ldots, G_m$ and also occurs in one of the goals H, *Diseqs* (this property is needed for the correctness of folding, see Section 3.4), and (iii) for $i = 1, \ldots, h$, if X_i is an input argument of *newq* then X_i is either an input variable of H (according to the given mode M_s) or an input variable of the leftmost non-basic atom of one of the goals $G_1, \ldots, G_m$. Then we fold the given packet and we get:

$FoldedCls := FoldedCls \cup \{H \leftarrow Diseqs, newq(X_1, \ldots, X_h)\}$

(Case β) If in *Defs* there is no set of definition clauses satisfying the conditions described in Case (α), then we add to *NewDefs* the following clauses for a new predicate *newr*:

$$\begin{cases} newr(X_1, \ldots, X_h) \leftarrow G_1 \\ \quad \ldots \\ newr(X_1, \ldots, X_h) \leftarrow G_m \end{cases}$$

where, for $i = 1, \ldots, h$, either (i) X_i occurs in one of the goals $G_1, \ldots, G_m$ and also occurs in one of the goals H, *Diseqs*, or (ii) X_i is an input variable of the leftmost non-basic atom of one of the goals $G_1, \ldots, G_m$. We add to M_{new} the mode $newr(m_1, \ldots, m_h)$ such that for $i = 1, \ldots, h$, $m_i = +$ iff X_i is either an input variable of H or an input variable of the leftmost non-basic atom of one of the goals $G_1, \ldots, G_m$. We then fold the packet under consideration and we get:

$FoldedCls := FoldedCls \cup \{H \leftarrow Diseqs, newr(X_1, \ldots, X_h)\}$

end-while

Notice that the post-conditions on the set *NewDefs* which is an output of the Define-Fold procedure (see Point (i) of the Output of the procedure), ensure the satisfaction of the pre-conditions on the set *Cls* which is an input of the Unfold-Simplify procedure. Indeed, recall that the set *Cls* is constructed during the Determinization strategy by the assignment $Cls := NewDefs$. Recall also that these pre-conditions are needed to ensure that the first unfolding step performed by the Unfold-Simplify procedure is safe.

Notice also that each application of the folding rule is safe (see Definition 4). This fact is implied in Case (α) by Condition (iii), and in Case (β) by the definition of the mode for *newr*.

Finally, notice that the Define-Fold subsidiary strategy does not guarantee the termination of the specialization process. To avoid non-termination we may use generalization techniques similar to those used in partial evaluation (see the Partial Evaluation Strategy in Section 1 and [10, 15, 18]). We do not discuss further this issue here.

6. Examples of Application of the Transformation Strategy

In this section we will present some examples of program specialization where we will see in action our Determinization Strategy together with the Unfold-Simplify, Partition, and Define-Fold subsidiary strategies.

6.1. A Complete Derivation: Computing the Occurrences of a Pattern in a String

Let us consider the following generalization of the problem of Section 2.2: Given a pattern P and a string S we want to compute the *position*, say N, of an occurrence of P in S, that is, we want to find two strings L and R such that: (i) S is the concatenation of L, P, and R, and (ii) the length of L is N. A program that for given P and S computes N is as follows.

Program *Match_Pos* (initial, nondeterministic)

1. $match\_pos(P, S, N) \leftarrow append(Y, R, S),\ append(L, P, Y),\ length(L, N)$
2. $length([\,], 0) \leftarrow$
3. $length([H|T], s(N)) \leftarrow length(T, N)$
4. $append([\,], Y, Y) \leftarrow$
5. $append([A|X], Y, [A|Z]) \leftarrow append(X, Y, Z)$

Given a pattern P and a string S, the *Match_Pos* program is nondeterministic and *either* (i) it computes no position if P does not occur in S, *or* (ii) it computes one or more positions (one position for each occurrence of P in S). The mode M for the program *Match_Pos* is $\{match\_pos(+, +, ?),\ append(?, ?, +),\ length(+, ?)\}$. We leave it to the reader to verify that *Match_Pos* satisfies M.

The derivation we will perform using the Determinization Strategy is more challenging than the ones presented in the literature (see, for instance, [12, 35]) because an occurrence of the pattern in the string is specified in the initial program via the *append* predicate which denotes list concatenation.

We want to specialize the *Match_Pos* program w.r.t. the atom $match\_pos([a, a, b], S, N)$. Thus, we introduce the definition clause:

6. $sp\_match\_pos(S, N) \leftarrow match\_pos([a, a, b], S, N)$

The mode of the new predicate is $sp\_match\_pos(+, ?)$ because S is an input argument of *match_pos* and N is not an input argument. Our transformation strategy starts off with the following initial values: $Defs = Cls = \{6\}$, $TransfP = Match\_Pos$, and $M_s = M \cup \{sp\_match\_pos(+, ?)\}$.

First iteration

Unfold-Simplify. By unfolding clause 6 w.r.t. the leftmost atom in its body we derive:

7. $sp\_match\_pos(S, N) \leftarrow append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

The body of clause 7 has no consumer atoms (notice that, for instance, the mgu of the atom $append(Y, R, S)$ and the head of clause 5 has the binding $S/[A|Z]$ where S is an input variable). Thus, the Unfold-Simplify subsidiary strategy terminates. We have: $UnfoldedCls = \{7\}$.

Partition. *NonunitCls* is made out of clause 7 only, and thus, the Partition subsidiary strategy immediately terminates and produces a set *PartitionedCls* which consists of a single packet made out of clause 7.

Define-Fold. In order to fold clause 7 in *PartitionedCls*, the Define-Fold subsidiary strategy introduces the following definition clause:

8. $new1(S, N) \leftarrow append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

The mode of $new1$ is $new1(+, ?)$. By folding clause 7 using clause 8 we derive:

9. $sp\_match\_pos(S, N) \leftarrow new1(S, N)$

Thus, the first iteration of the Determinization Strategy terminates with $Defs = \{6, 8\}$, $Cls = \{8\}$, $TransfP = Match\_Pos \cup \{9\}$, and $M_s = M \cup \{sp\_match\_pos(+, ?),\ new1(+, ?)\}$.

Second iteration

Unfold-Simplify. We follow the subsidiary strategy described in Section 5.1 and we first unfold clause 8 in *Cls* w.r.t. the leftmost atom in its body. We get:

10. $new1(S, N) \leftarrow \underline{append(L, [a, a, b], [\,])},\ length(L, N)$
11. $new1([C|S], N) \leftarrow append(Y, R, S),\ \underline{append(L, [a, a, b], [C|Y])},\ length(L, N)$

Now we unfold clauses 10 and 11 w.r.t. the leftmost consumer atom of their body (see the underlined atoms). The unfolding of clause 10 amounts to its deletion, because the selected atom is not unifiable with any head in program *Match_Pos*. The unfolding of clause 11 yields two new clauses that are further unfolded according to the Unfold-Simplify subsidiary strategy. After some unfolding steps, we derive the following clauses:

12. $new1([a|S], 0) \leftarrow append([a, b], R, S)$
13. $new1([C|S], s(N)) \leftarrow append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

Partition. We apply the safe case split rule to clause 13 w.r.t. to the binding C/a, because the input argument in the head of this clause is unifiable with the input argument in the head of clause 12 via the mgu $\{C/a\}$. We derive the following two clauses:

14. $new1([a|S], s(N)) \leftarrow append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$
15. $new1([C|S], s(N)) \leftarrow C \neq a,\ append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

Now, the set of clauses derived so far by the Partition subsidiary strategy can be partitioned into two packets: the first one is made out of clauses 12 and 14, where the input argument of the head predicate is of the form $[a|S]$, and the second one is made out of clause 15 only, where the input argument of the head predicate is of the form $[C|S]$ with $C \neq a$.

The Partition subsidiary strategy terminates by applying the safe head generalization rule to clauses 12 and 14, so to replace the second arguments in their heads by the most specific common generalization of those arguments, that is, a variable. We get the packet:

16. $new1([a|S], M) \leftarrow M = 0,\ append([a, b], R, S)$
17. $new1([a|S], M) \leftarrow M = s(N),\ append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

For the packet made out of clause 15 only, no application of the safe head generalization rule is performed. Thus, we have derived the set of clauses *PartitionCls* which is the union of the two packets $\{16, 17\}$ and $\{15\}$.

Define-Fold. Since there is no set of definition clauses which can be used to fold the packet $\{16, 17\}$, we are in Case (β) of the Define-Fold subsidiary strategy. Thus, we introduce a new predicate $new2$ as follows:

18. $new2(S, M) \leftarrow M = 0,\ append([a, b], R, S)$
19. $new2(S, M) \leftarrow M = s(N),\ append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

The mode of $new2$ is $new2(+,?)$ because S is an input variable of the head of each clause of the corresponding packet. By folding clauses 16 and 17 using clauses 18 and 19 we derive the following clause:

20. $new1([a|S], M) \leftarrow new2(S, M)$

We then consider the packet made out of clause 15 only. This packet can be folded using clause 8 in *Defs*. Thus, we are in Case (β) of the Define-Fold subsidiary strategy. By folding clause 15 we derive the following clause:

21. $new1([C|S], s(N)) \leftarrow C \neq a,\ new1(S, N)$

Thus, *FoldedCls* is the set $\{20, 21\}$.

After these folding steps we conclude the second iteration of the Determinization Strategy with the following assignments: $Defs := Defs \cup \{18, 19\}$; $Cls := \{18, 19\}$; $TransfP := TransfP \cup \{20, 21\}$; $M_s := M_s \cup \{new2(+,?)\}$.

Third iteration

Unfold-Simplify. From *Cls*, that is, clauses 18 and 19, we derive the set *UnfoldedCls* made out of the following clauses:

22. $new2([a|S], 0) \leftarrow append([b], R, S)$
23. $new2([a|S], s(0)) \leftarrow append([a, b], R, S)$
24. $new2([C|S], s(s(N))) \leftarrow append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

Partition. The set *NonunitCls* is identical to *UnfoldedCls*. From *NonunitCls* we derive the set *PartitionedCls* made out of the following clauses:

25. $new2([a|S], M) \leftarrow M = 0,\ append([b], R, S)$
26. $new2([a|S], M) \leftarrow M = s(0),\ append([a, b], R, S)$
27. $new2([a|S], M) \leftarrow M = s(s(N)),\ append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$
28. $new2([C|S], s(s(N))) \leftarrow C \neq a,\ append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

Define-Fold. We introduce the following definition clauses:

29. $new3(S, M) \leftarrow M = 0,\ append([b], R, S)$
30. $new3(S, M) \leftarrow M = s(0),\ append([a, a], R, S)$
31. $new3(S, M) \leftarrow M = s(s(N)),\ append(Y, R, S),\ append(L, [a, a, b], Y),\ length(L, N)$

where the mode for $new3$ is $new3(+,?)$. By folding, from *PartitionedCls* we derive the following two clauses:

32. $new2([a|S], M) \leftarrow new3(S, M)$
33. $new2([C|S], s(s(N))) \leftarrow C \neq a,\ new1(S, N)$

which constitute the set *FoldedCls*.

The third iteration of the Determinization Strategy terminates by performing the following updates: $Defs := Defs \cup \{29, 30, 31\}$; $Cls := \{29, 30, 31\}$; $TransfP := TransfP \cup \{32, 33\}$; $M_s := M_s \cup \{new3(+,?)\}$.

Fourth iteration

Unfold-Simplify. From *Cls* we derive the new set *UnfoldedCls* made out of the following clauses:

34. $new3([b|S], 0) \leftarrow append([\,], R, S)$
35. $new3([a|S], s(0)) \leftarrow append([b], R, S)$

36. $new3([a|S], s(s(0))) \leftarrow append([a,b], R, S)$
37. $new3([C|S], s(s(s(N)))) \leftarrow append(Y, R, S),\ append(L, [a,a,b], Y),\ length(L, N)$

Partition. The set *NonunitCls* is identical to *UnfoldedCls*. From *NonunitCls* we derive the new set *PartitionedCls* made out of the following clauses:

38. $new3([a|S], s(M)) \leftarrow M = 0,\ append([b], R, S)$
39. $new3([a|S], s(M)) \leftarrow M = s(0),\ append([a,b], R, S)$
40. $new3([a|S], s(M)) \leftarrow M = s(s(N)), append(Y, R, S), append(L, [a,a,b], Y), length(L, N)$
41. $new3([b|S], M) \leftarrow M = 0,\ append([\,], R, S)$
42. $new3([b|S], M) \leftarrow M = s(s(s(N))), append(Y, R, S), append(L, [a,a,b], Y), length(L, N)$
43. $new3([C|S], s(s(s(N)))) \leftarrow C \neq a, C \neq b, append(Y, R, S), append(L, [a,a,b], Y),$ $length(L, N)$

Define-Fold. We introduce two new predicates by the following definition clauses:

44. $new4(S, M) \leftarrow M = 0,\ append([\,], R, S)$
45. $new4(S, M) \leftarrow M = s(s(s(N))),\ append(Y, R, S),\ append(L, [a,a,b], Y),\ length(L, N)$

We now fold the clauses in *PartitionedCls* and we derive the set *FoldedCls* made out of the following clauses:

46. $new3([a|S], s(M)) \leftarrow new3(R, S)$
47. $new3([b|S], M) \leftarrow new4(R, S)$
48. $new3([C|S], s(s(s(N)))) \leftarrow C \neq a,\ C \neq b,\ new1(S, N)$

The fourth iteration terminates by performing the following updates: $Defs := Defs \cup \{44, 45\}$; $Cls := \{44, 45\}$; $TransfP := TransfP \cup \{46, 47, 48\}$; $M_s := M_s \cup \{new4(+, ?)\}$.

Fifth iteration

Unfold-Simplify. From *Cls* we derive the new set *UnfoldedCls* made out of the following clauses:

49. $new4(S, 0) \leftarrow$
50. $new4([a|S], s(s(s(0)))) \leftarrow append([a,b], R, S)$
51. $new4([C|S], s(s(s(s(N))))) \leftarrow append(Y, R, S),\ append(L, [a,a,b], Y),\ length(L, N)$

Partition. The set *NonunitCls* is made out of clauses 50 and 51. From *NonunitCls* we derive the new set *PartitionedCls* made out of the following clauses:

52. $new4([a|S], s(s(s(M)))) \leftarrow M = 0,\ append([a,b], R, S)$
53. $new4([a|S], s(s(s(M)))) \leftarrow M = s(N),\ append(Y, R, S),\ append(L, [a,a,b], Y),$ $length(L, N)$
54. $new4([C|S], s(s(s(s(N))))) \leftarrow C \neq a,\ append(Y, R, S),\ append(L, [a,a,b], Y),$ $length(L, N)$

Define-Fold. We are able to perform all required folding steps without introducing new definition clauses (see Case (α) of the Define-Fold procedure). In particular, we fold: (i) clauses 52 and 53 using clauses 18 and 19, and (ii) clause 54 using clause 8. Since no new definition is introduced, the set *Cls* is empty and the transformation strategy stops. Our final specialized program is the following:

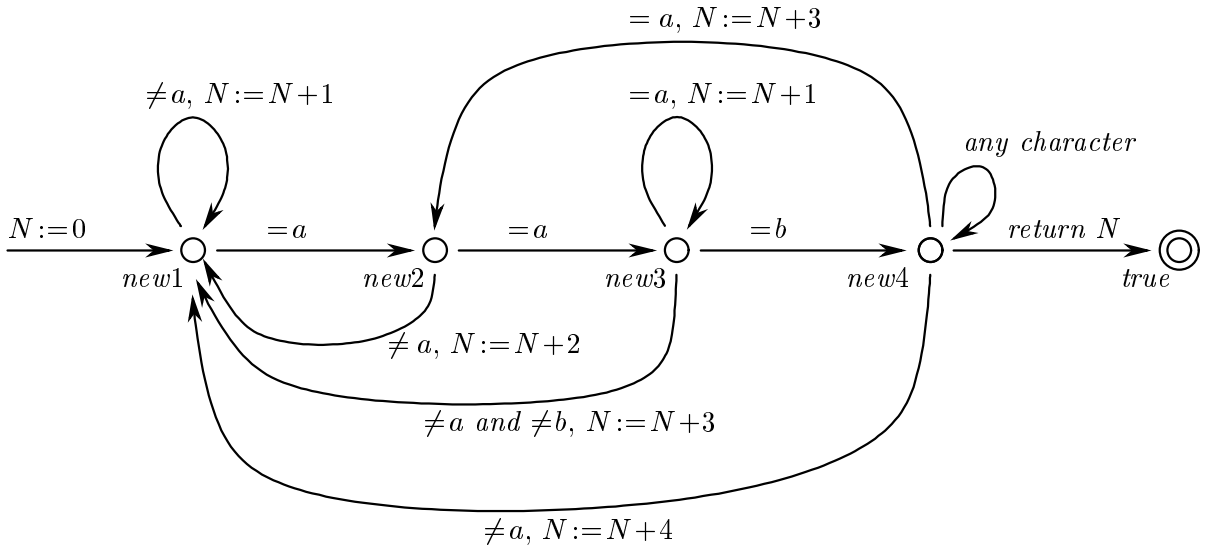

Figure 1: Finite automaton with counter N, corresponding to $Match\_Pos_s$

Program $Match\_Pos_s$ (specialized, semideterministic)

9. $sp\_match\_pos(S, N) \leftarrow new1(S, N)$
20. $new1([a|S], M) \leftarrow new2(S, M)$
21. $new1([C|S], s(N)) \leftarrow C \neq a,\ new1(S, N)$
32. $new2([a|S], M) \leftarrow new3(S, M)$
33. $new2([C|S], s(s(N))) \leftarrow C \neq a,\ new1(S, N)$
46. $new3([a|S], s(M)) \leftarrow new3(R, S)$
47. $new3([b|S], M) \leftarrow new4(R, S)$
48. $new3([C|S], s(s(s(N)))) \leftarrow C \neq a,\ C \neq b,\ new1(S, N)$
49. $new4(S, 0) \leftarrow$
55. $new4([a|S], s(s(s(M)))) \leftarrow new2(S, M)$
56. $new4([C|S], s(s(s(s(N))))) \leftarrow C \neq a,\ new1(S, N)$

This final program is semideterministic and it corresponds to the finite automaton with one counter depicted in Fig. 1. The predicates correspond to the states of the automaton and the clauses correspond to the transitions. The predicate $new1$ corresponds to the initial state, because the program is intended to be used for goals of the form $sp\_match\_pos(S, N)$, where S is bound to a list of characters, and by clause 1 $sp\_match\_pos(S, N)$ calls $new1(S, N)$. Notice that this finite automaton is deterministic except for the state corresponding to the predicate $new4$, where the automaton can either (i) accept the input string by returning the value of N and moving to the final state *true*, even if the input string has not been completely scanned (see clause 49), or (ii) move to the state corresponding to $new2$, if the symbol of the input string which is scanned is a (see clause 55), or (iii) move to the state corresponding to $new1$, if the symbol of the input string which is scanned is different from a (see clause 56).

6.2. Multiple Pattern Matching

Now we consider a generalization of the problem described in Section 6.1. Given a list Ps of patterns and a string S we want to compute the position, say N, of any occurrence in S of a pattern which is a member of Ps. For any given Ps and S the following program computes N in a nondeterministic way.

Program *Mmatch* (initial, nondeterministic)

1. $mmatch([P|Ps], S, N) \leftarrow match\_pos(P, S, N)$
2. $mmatch([P|Ps], S, N) \leftarrow mmatch(Ps, S, N)$

The atom $mmatch(Ps, S, N)$ holds iff there exists a pattern in the list Ps of patterns which occurs in the string S at position N. The predicate *match_pos* is defined as in program *Match_Pos* of Section 6.1, and its clauses are not listed here. We consider the following mode for the program *Mmatch*: $\{mmatch(+,+,?),\ match\_pos(+,+,?),\ append(?,?,+),\ length(+,?)\}$.

We want to specialize this multi-pattern matching program w.r.t. the list $[[a,a,a],\ [a,a,b]]$ of patterns. Thus, we introduce the following definition clause:

3. $sp\_mmatch(S, N) \leftarrow mmatch([[a,a,a],[a,a,b]], S, N)$

The mode of the new predicate is $sp\_mmatch(+,?)$ because S is an input argument of *mmatch*, and N does not. Thus, our transformation strategy starts off with the following initial values: $Defs = Cls = \{3\}$, $TransfP = Mmatch$, and $M_s = M \cup \{sp\_mmatch(+,?)\}$.

The output of the transformation strategy is the following program.

Program $Mmatch_s$ (specialized, semideterministic)

4. $sp\_mmatch(S, N) \leftarrow new1(S, N)$
5. $new1([a|S], M) \leftarrow new2(S, M)$
6. $new1([C|S], s(N)) \leftarrow C \neq a,\ new1(S, N)$
7. $new2([a|S], M) \leftarrow new3(S, M)$
8. $new2([C|S], s(s(N))) \leftarrow C \neq a,\ new1(S, N)$
9. $new3([a|S], M) \leftarrow new4(S, M)$
10. $new3([b|S], M) \leftarrow new5(S, M)$
11. $new3([C|S], s(s(s(N)))) \leftarrow C \neq a,\ C \neq b,\ new1(S, N)$
12. $new4(S, 0) \leftarrow$
13. $new4([a|S], s(N)) \leftarrow new4(S, N)$
14. $new4([b|S], s(N)) \leftarrow new5(S, N)$
15. $new4([C|S], s(s(s(s(N))))) \leftarrow C \neq a,\ C \neq b,\ new1(S, N)$
16. $new5(S, 0) \leftarrow$
17. $new5([a|S], s(s(s(N)))) \leftarrow new2(S, N)$
18. $new5([C|S], s(s(s(s(N))))) \leftarrow C \neq a,\ new1(S, N)$

Similarly to the single-pattern string matching example of the previous Section 6.1, this specialized, semideterministic program corresponds to a finite automaton with counters which is deterministic, except for the states corresponding to the predicates *new*4 and *new*5 where all strings are accepted. A similar derivation cannot be performed by usual partial evaluation techniques without a prior transformation into *failure continuation passing style* [35].

6.3. From Regular Expressions to Finite Automata

In this example we show the derivation of a deterministic finite automaton by specializing a general parser for regular expressions w.r.t. a given regular expression. The initial program *Reg_Expr* for testing whether or not a string belongs to the language denoted by a regular expression over the alphabet $\{a, b\}$, is the one given below.

Program *Reg_Expr* (initial, nondeterministic)

1. $in\_language(E, S) \leftarrow string(S),\ accepts(E, S)$
2. $string([\,]) \leftarrow$
3. $string([a|S]) \leftarrow string(S)$
4. $string([b|S]) \leftarrow string(S)$
5. $accepts(E, [E]) \leftarrow symbol(E)$
6. $accepts(E_1E_2, S) \leftarrow append(S, S_1, S_2),\ accepts(E_1, S_1),\ accepts(E_2, S_2)$
7. $accepts(E_1{+}E_2, S) \leftarrow accepts(E_1, S)$
8. $accepts(E_1{+}E_2, S) \leftarrow accepts(E_2, S)$
9. $accepts(E^*, [\,])$
10. $accepts(E^*, S) \leftarrow ne\_append(S_1, S_2, S),\ accepts(E, S_1),\ accepts(E^*, S_2)$
11. $symbol(a) \leftarrow$
12. $symbol(b) \leftarrow$
13. $ne\_append([A], Y, [A|Y]) \leftarrow$
14. $ne\_append([A|X], Y, [A|Z]) \leftarrow ne\_append(X, Y, Z)$

We have that $in\_language(E, S)$ holds iff S is a string in $\{a, b\}^*$ and S belongs to the language denoted by the regular expression E. In the *Reg_Expr* program we have used the predicate $ne\_append(S_1, S_2, S)$ which holds iff the non-empty string S is the concatenation of the *nonempty* string S_1 and the string S_2. The use the $ne\_append(S_1, S_2, S)$ in clause 10 is motivated by the objective of writing a *terminating* program, that is, a program for which we cannot have an infinite derivation starting from a ground goal. Indeed, if in clause 10 we replace $ne\_append(S_1, S_2, S)$ by $append(S_1, S_2, S)$, then we may construct an infinite derivation because from a goal of the form $accepts(E^*, S)$ we can derive a new goal of the form $(accepts(E, [\,]),\ accepts(E^*, S))$.

We consider the following mode for the program *Reg_Expr*:
$\{in\_language(+, +),\ string(+),\ accepts(+, +),\ symbol(+),\ ne\_append(?, ?, +),\ append(?, ?, +)\}$.

We use our Determinization Strategy to specialize the program *Reg_Expr* w.r.t. the atom $in\_language(a(a{+}b)^*b, S)$. Thus, we begin by introducing the definition clause:

15. $sp\_in\_language(S) \leftarrow in\_language(a(a{+}b)^*b, S)$

We derive the following specialized program Reg_Expr_s.

Program Reg_Expr_s (specialized, semideterministic)

16. $sp\_in\_language(S) \leftarrow new1(S)$
17. $new1([a|S]) \leftarrow new2(S)$
18. $new2([b]) \leftarrow$
19. $new2([a|S]) \leftarrow new3(S)$
20. $new2([b|S]) \leftarrow new3(S)$
21. $new3([b]) \leftarrow$
22. $new3([a|S]) \leftarrow new3(S)$
23. $new3([b|S]) \leftarrow new3(S)$

This specialized program corresponds to a finite automaton, which is deterministic except in

the states corresponding to the predicates $new2$ and $new3$ when it is scanning the symbol b. In these states the finite automaton may either enter a final state (see clauses 18 and 21) or remain in the same state (see clauses 20 and 23).

6.4. Matching Regular Expressions

We consider the following problem of matching a regular expression: Given a regular expression E and a string S we want to test whether or not there exists a substring of S which belongs to the language denoted by the regular expression E. The following program solves our matching problem in a nondeterministic way.

Program *Reg_Expr_Match* (initial, nondeterministic)

1. $re\_match(E, S) \leftarrow append(Y, R, S),\ append(L, P, Y),\ accepts(E, P)$

where the predicates *append* and *accepts* are defined as in the programs *Match_Pos* and *Reg_Expr*, respectively, and their clauses are not listed here. The atom $re\_match(E, S)$ holds iff there exists a string P which occurs as a substring of S such that P belongs to the language denoted by the regular expression E. We consider the following mode for the program *Reg_Expr_Match*: $\{append(?, ?, +),\ accept(+, +),\ re\_match(+, +)\}$.

We want to specialize the program *Reg_Expr_Match* w.r.t. the regular expression aa^*b. Thus, we introduce the following definition clause:

2. $sp\_re\_match(S) \leftarrow re\_match(aa^*b,\ S)$

The mode of the new predicate is $sp\_re\_match(+)$ because S is an input argument of re_match. Thus, our transformation strategy starts off with the following initial values: $Defs = Cls = \{2\}$, $TransfP = re\_Match$, and $M_s = M \cup \{sp\_re\_match(+)\}$.

The output of the transformation strategy is the following program.

Program $Reg\_Expr\_Match_s$ (specialized, semideterministic)

3. $sp\_re\_match(S) \leftarrow new1(S)$
4. $new1([a|S]) \leftarrow new2(S)$
5. $new1([C|S]) \leftarrow C \neq a,\ new1(S)$
6. $new2([a|S]) \leftarrow new3(S)$
7. $new2([C|S]) \leftarrow C \neq a,\ new1(S)$
8. $new3([a|S]) \leftarrow new4(S)$
9. $new3([b|S]) \leftarrow new3(S)$
10. $new3([C|S]) \leftarrow C \neq a,\ C \neq b,\ new1(S)$
11. $new4(S) \leftarrow$

Similarly to the single-pattern string matching example of Section 2.2, this specialized, semideterministic program corresponds to a deterministic finite automaton.

6.5. Specializing Context-free Parsers to Regular Grammars

Let us consider the following program for parsing context-free languages:

Program *CF_Parser* (initial, nondeterministic)

1. $string\_parse(G, A, W) \leftarrow string(W),\ parse(G, A, W)$
2. $string([\,]) \leftarrow$
3. $string([0|W]) \leftarrow string(W)$
4. $string([1|W]) \leftarrow string(W)$
5. $parse(G, [\,], [\,]) \leftarrow$
6. $parse(G, [A|X], [A|Y]) \leftarrow terminal(A),\ parse(G, X, Y)$
7. $parse(G, [A|X], Y) \leftarrow nonterminal(A),\ member(A \rightarrow B, G),$
   $append(B, X, Z),\ parse(G, Z, Y)$
8. $member(A, [A|X]) \leftarrow$
9. $member(A, [B|X]) \leftarrow member(A, X)$

together with the clauses for the predicate *append* defined as in program *Match_Pos* of Section 6.1, and the unit clauses stating that 0 and 1 are terminals and s, u, v, and w are nonterminals. The first argument of *parse* is a context-free grammar, the second argument is a list of terminal and nonterminal symbols, and the third argument is a word represented as a list of terminal symbols. We assume that a context-free grammar is represented as a list of productions of the form $x \rightarrow y$, where x is a nonterminal symbol and y is a list of terminal and nonterminal symbols. We have that $parse(G, [s], W)$ holds iff from the symbol s we can derive the word W using the grammar G. We consider the following mode for the program *CF_Parser*: $\{parse(+,+,+),\ terminal(+),\ nonterminal(+),\ member(?,+),\ append(+,+,?)\}$.

We want to specialize our parsing program w.r.t. the following regular grammar:

$$\begin{array}{lll} s \rightarrow 0\,u & s \rightarrow 0\,v & s \rightarrow 0\,w \\ u \rightarrow 0 & u \rightarrow 0\,u & u \rightarrow 0\,v \\ v \rightarrow 0 & v \rightarrow 0\,v & v \rightarrow 0\,u \\ w \rightarrow 1 & w \rightarrow 0\,w & \end{array}$$

To this aim we apply our Determinization Strategy starting from the following definition clause:

10. $sp\_string\_parse(W) \leftarrow parse([\ s \rightarrow [0,u],\ \ s \rightarrow [0,v],\ \ s \rightarrow [0,w],$
    $u \rightarrow [0],\ \ u \rightarrow [0,u],\ \ u \rightarrow [0,v],$
    $v \rightarrow [0],\ \ v \rightarrow [0,v],\ \ v \rightarrow [0,u],$
    $w \rightarrow [1],\ \ w \rightarrow [0,w]\ \ ],\ [s],\ W)$

The mode for the newly introduced predicate is $sp\_string\_parse(+)$. We derive the following specialized program CF_Parser_s.

Program CF_Parser_s (specialized, semideterministic)

11. $sp\_string\_parse(W) \leftarrow new1(W)$
12. $new1([0|W]) \leftarrow new2(W)$
13. $new2([0|W]) \leftarrow new3(W)$
14. $new2([1|W]) \leftarrow new4(W)$
15. $new3([\,]) \leftarrow$
16. $new3([0|W]) \leftarrow new5(W)$
17. $new3([1|W]) \leftarrow new4(W)$
18. $new4([\,]) \leftarrow$
19. $new5([\,]) \leftarrow$
20. $new5([0|W]) \leftarrow new3(W)$
21. $new5([1|W]) \leftarrow new4(W)$

This program corresponds to a deterministic finite automaton.

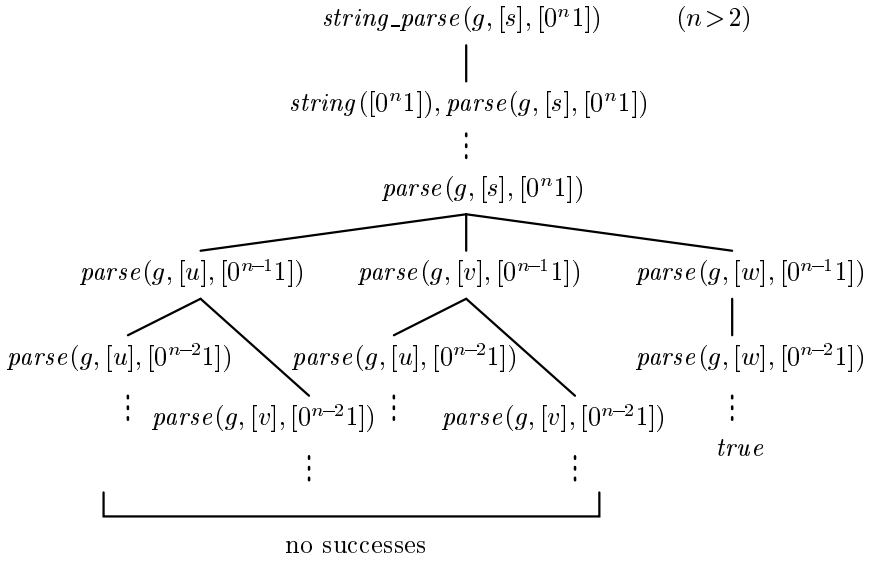

Figure 2: Derivation tree T_1 for $string\_parse(g,[s],[0^n1])$

Now, we would like to discuss the improvements we achieved in this example by applying our Determinization Strategy. Let us consider the *derivation tree* T_1 (see Fig. 2) generated by the initial program *CF_Parser* starting from the goal $string\_parse(g,[s],[0^n1])$, where g denotes the grammar w.r.t. which we have specialized the *CF_Parser* program and $[0^n1]$ denotes the list $[0,\ldots,0,1]$ with n occurrences of 0. The nodes of T_1 are labeled by the goals derivable from $string\_parse(g,[s],[0^n1])$. In particular, the root of the derivation tree is labeled by $string\_parse(g,[s],[0^n1])$ and a node labeled by a goal G has n children labeled by the goals $G_1,\ldots,G_n$ which can be derived from G. The tree T_1 has a number of nodes which is $O(2^n)$. Thus, by using the initial program *CF_Parser* it takes $O(2^n)$ number of steps to search for a derivation from the root goal $string\_parse(g,[s],[0^n1])$ to the goal *true*. (Indeed, this is the case if one uses a Prolog compiler.) In contrast, by using the specialized program CF_Parser_s, it takes $O(n)$ steps to search for a derivation from the goal $sp\_string\_parse([0^n1])$ to *true*, because the derivation tree T_2 has a number of nodes which is $O(n)$ (see Fig. 3).

The improvement of performance is due to the fact that our Determinization Strategy is able to avoid repeated derivations by introducing new definition clauses whose bodies have goals with common subgoals. Then, by performing folding steps using these definition clauses, we reduce the search space when looking for the goal *true* starting from a given initial goal.

For instance, our strategy introduces the predicate *new*2 defined by the following clauses:

$new2(W) \leftarrow string(W),\ parse(g,[u],W)$
$new2(W) \leftarrow string(W),\ parse(g,[v],W)$
$new2(W) \leftarrow string(W),\ parse(g,[w],W)$

whose bodies are goals from which common subgoals are derived for $A = [0^{n-1}1]$ and $n > 1$. Indeed, for instance, $parse(g,[u],[0^{n-2}])$ can be derived from both $parse(g,[u],[0^{n-1}])$ and $parse(g,[v],[0^{n-1}])$ (see Fig. 2). The reader may verify that by using the specialized program CF_Parser_s no repeated goal is derived from $sp\_string\_parse(g,[s],[0^n1])$.

The ability of our Determinization Strategy of putting together various computations performed by the initial program in different branches of the computation tree, so that common repeated subcomputations are avoided, is based on the same ideas which motivate the *tupling*

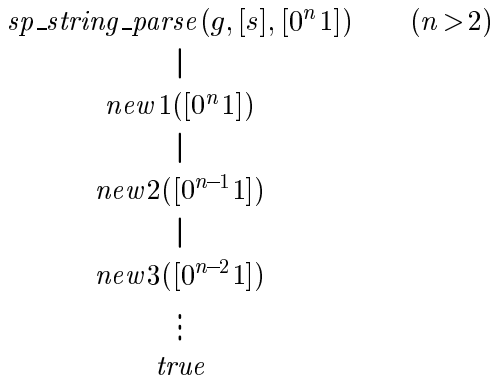

Figure 3: Derivation tree T_2 for $sp\_string\_parse([0^n 1])$

strategy [26], first proposed as a transformation technique for functional languages.

6.6. Discussion of the Specialization Examples

All program specialization examples presented in this Section were worked out in a fully automatic way by using the MAP program transformation system [32].

The performance of the specialized programs may be further improved by several post-processing techniques which preserve the operational semantics and the semideterminism of the programs. In particular, if the specialized programs are to be run by a standard Prolog system, we may: (i) reorder the clauses so that unit clauses appear before non-unit clauses, and (ii) remove disequations by introducing cuts instead. For a systematic treatment of cut introduction, we refer the reader to [7]. As an example we now show the program obtained from $Match\_Pos_s$ of Example 6.1 after the above post-processing transformations have been performed.

Program $Match\_Pos_{cut}$ (specialized, with cuts)

$sp\_match\_pos(S, N) \leftarrow new1(S, N)$
$new1([a|S], M) \leftarrow !,\ new2(S, M)$
$new1([C|S], s(N)) \leftarrow new1(S, N)$
$new2([a|S], M) \leftarrow !,\ new3(S, M)$
$new2([C|S], s(s(N))) \leftarrow new1(S, N)$
$new3([a|S], s(M)) \leftarrow !,\ new3(R, S)$
$new3([b|S], M) \leftarrow !,\ new4(R, S)$
$new3([C|S], s(s(s(N)))) \leftarrow new1(S, N)$
$new4(S, 0) \leftarrow$
$new4([a|S], s(s(s(M)))) \leftarrow !,\ new2(S, M)$
$new4([C|S], s(s(s(s(N))))) \leftarrow new1(S, N)$

In Columns 4 and 5 of the following Table 1 we show the speedups achieved by specializing the programs listed in Column 1 according to our Determinization Strategy (Column 4), and also by removing disequations and introducing cuts (Column 5). Every speedup is computed as the ratio between the timing of the initial program and the timing of the specialized program. For our experimental results we used SICStus Prolog 3.8.5. To clarify the content of that Table 1 let us remark that:

(1) Program	(2) Static Input	(3) Dynamic Input	(4) Speedups (Det)	(5) Speedups (Det & Cut)
Naive_Match	$naive\_match([aab], S)$	$\|S\| = 1000$	0.9×10^3	1.6×10^3
Naive_Match	$naive\_match([aab], S)$	$\|S\| = 4000$	2.7×10^3	6.7×10^3
Naive_Match	$naive\_match([aaaaaaaaab], S)$	$\|S\| = 1000$	0.7×10^3	1.6×10^3
Naive_Match	$naive\_match([aaaaaaaaab], S)$	$\|S\| = 4000$	2.9×10^3	8.3×10^3
Match_Pos	$match\_pos([aab], S, N)$	$\|S\| = 1000$	0.4×10^3	0.9×10^3
Match_Pos	$match\_pos([aab], S, N)$	$\|S\| = 4000$	1.7×10^3	3.8×10^3
Match_Pos	$match\_pos([aaaaaaaaab], S, N)$	$\|S\| = 1000$	1.0×10^3	2.4×10^3
Match_Pos	$match\_pos([aaaaaaaaab], S, N)$	$\|S\| = 4000$	3.7×10^3	9.7×10^3
Mmatch	$mmatch([[aaa], [aab]], S, N)$	$\|S\| = 1000$	0.9×10^3	1.5×10^3
Mmatch	$mmatch([[aaa], [aab]], S, N)$	$\|S\| = 4000$	3.5×10^3	6.2×10^3
Mmatch	$mmatch([[aa], [aaa], [aab]], S, N)$	$\|S\| = 1000$	1.5×10^3	2.1×10^3
Mmatch	$mmatch([[aa], [aaa], [aab]], S, N)$	$\|S\| = 4000$	6.1×10^3	8.2×10^3
Reg_Expr	$in\_language(a(a+b)^*b, S)$	$\|S\| = 100$	1.6×10^4	1.7×10^4
Reg_Expr	$in\_language(a(a+b)^*b, S)$	$\|S\| = 200$	6.9×10^4	7.4×10^4
Reg_Expr	$in\_language((aa^*b)^*, S)$	$\|S\| = 100$	4.7×10^5	4.7×10^5
Reg_Expr	$in\_language((aa^*b)^*, S)$	$\|S\| = 200$	3.9×10^6	3.7×10^6
Reg_Expr_Match	$re\_match(aa^*b, S)$	$\|S\| = 100$	3.2×10^6	3.7×10^6
Reg_Expr_Match	$re\_match(aa^*b, S)$	$\|S\| = 200$	5.3×10^7	5.8×10^7
Reg_Expr_Match	$re\_match((aa)+(a^*b), S)$	$\|S\| = 50$	7.3×10^4	8.2×10^4
Reg_Expr_Match	$re\_match((aa)+(a^*b), S)$	$\|S\| = 100$	6.1×10^5	7.2×10^5
CF_Parser	$string\_parse(g, [s], W)$	$\|W\| = 1000$	23	23
CF_Parser	$string\_parse(g, [s], W)$	$\|W\| = 10000$	178	178
CF_Parser	$string\_parse(g_1, [s], W)$	$\|W\| = 1000$	19	19
CF_Parser	$string\_parse(g_1, [s], W)$	$\|W\| = 10000$	170	170

Table 1. Speedups (Det): after applying the Determinization Strategy.
Speedups (Det & Cut): after applying the Determinization Strategy and Cut Introduction.

1. Column 1 shows the names of the initial programs with reference to Example 2.2 and Examples 6.1–6.5.
2. Column 2 shows the Static Input, that is, the goal w.r.t. which we have specialized the program of Column 1. The argument $[aab]$ denotes the list $[a, a, b]$. Similar notation has been used for the other static input arguments. The argument g of the first two *string_parse* atoms denotes the regular grammar considered in Example 6.5. The argument g_1 of the last two *string_parse* atoms denotes the regular grammar:
$\{s \to 0\,u,\ \ s \to 1\,v,\ \ u \to 0,\ \ u \to 0\,v,\ \ u \to 0\,w,\ \ v \to 1,\ \ v \to 0\,v,\ \ v \to 1\,u,\ \ w \to 1,\ w \to 1\,w\}$.
3. Column 3 shows the size of the Dynamic Input, that is, the size of the argument which is supplied to the initial and to the specialized programs after the specialization process for measuring the speedups.

4. Column 4, called Speedups (Det), shows the speedups we have obtained after the application of our Determinization Strategy.
5. Column 5, called Speedups (Det & Cut), shows the speedups we have obtained after the application of our Determinization Strategy followed by the removal of disequations and the introduction of cuts. Notice that for some programs (see, for instance, the entries for *Reg_Expr* and *CF_Parser* in Columns 4 and 5), the speedups obtained in this way are not better than the speedups after the application of the Determinization Strategy alone. There are two reasons for this: (i) the first one is the absence of disequations in the specialized program, and (ii) the second one is the ability of our compiler of exploiting the presence of mutually exclusive clauses through *clause indexing*, so that the introduction of cuts does not improve efficiency.

Further post-processing techniques are applicable. For instance, similarly to the familiar case of finite automata, we may eliminate clauses corresponding to *epsilon*-transitions where no input symbols are consumed (such as clause 9 in program $Match\_Pos_s$), and we may minimize the number of predicate symbols (this corresponds to the minimization of the number of states). We do not present here these post-processing techniques which are outside the scope of the paper.

7. Concluding Remarks and Related Work

We have proposed a specialization technique for logic programs based on an automatic strategy which makes use of the following transformation rules: (1) definition introduction, (2) definition elimination, (3) unfolding, (4) folding, (5) subsumption, (6) head generalization, (7) case split, (8) equation elimination, and (9) disequation replacement. (Actually, we make use of the safe versions of the rules 4, 6, 7, and 8.) We have also shown that our strategy may reduce the amount of nondeterminism in the specialized programs and it may achieve exponential gains in time complexity.

To get these results, we allow new predicates to be introduced by *one or more* non-recursive definition clauses whose bodies may contain *more than one* atom. We also allow folding steps using these definition clauses. By a folding step a program where a predicate, say p, occurs in several clause heads, can be transformed into an equivalent program where the predicate p occurs in one clause only, thereby reducing nondeterminism.

The use of the subsumption rule is motivated by the desire of increasing efficiency by avoiding redundant computations. Head generalizations are used for deriving clauses with equal heads and thus, they allow us to perform folding steps. The case split rule is the crucial rule for the derivation of programs with reduced nondeterminism because it replaces a clause, say C, by several clauses which correspond to exhaustive and mutually exclusive instantiations of the head of C. To get exhaustiveness and mutual exclusion, we allow the introduction of disequalities. Then, to increase program efficiency when using a Prolog-like evaluator, these disequalities may be removed in favour of cuts.

We have assumed that the initial program to specialize is given to us together with a mode of use of its predicates. Our transformation strategy makes use of this mode information for directing the various transformation steps, and in particular, the unfolding and case split steps. Moreover, if our strategy terminates, then we derive specialized programs which are semideterministic w.r.t. the given mode. This notion has been formally defined in Section 4.3. Although semideterminism is not in itself a guarantee for efficiency improvement, it is often the case that

efficiency is increased by our strategy because it reduces nondeterminism and it also avoids redundant computations.

The transformation strategy we have proposed may not terminate. This may be due both to the unfolding strategy, which allows for infinitely many steps, and to the definition and folding strategy, which allows for the introduction of infinitely many new predicates. The finiteness of our unfolding strategy can be guaranteed by applying techniques already developed for partial evaluation, and in particular, some generalization methods for partial evaluation (see, for instance, [10, 19]) can be used for avoiding the definition of an infinite number of new predicates. Notice, however, that the adaptation of these methods to our framework may not be straightforward, because our definition and folding rules are more complex than the ones used in partial evaluation.

We have implemented our proposed strategy and we have tested it by performing several program specializations of string matching and parsing programs. In these cases our strategy is able to automatically derive programs which behaves like Knuth-Morris-Pratt algorithm, in the sense that they generate a finite automaton from a general pattern matcher and any given pattern. This was done also in the case of programs for matching sets of patterns and programs for matching regular expressions.

In these examples the improvement over similar derivations performed by partial evaluation techniques [8, 10, 35] consists in the fact that we have started from naive, nondeterministic initial programs, while the corresponding derivations by partial evaluation described in the literature, use initial programs which already incorporate some ingenuity. A similar remark also applies to the derivation performed by using *supercompilation* with *perfect driving* [12, 38] and *generalized partial computation* [9].

A formal derivation of the Knuth-Morris-Pratt algorithm for pattern matching has also been presented in [2]. This derivation follows the *calculational* approach which consists in applying equivalences of higher order functions. On the one hand the calculational derivation is more general than ours, because it takes into consideration a generic pattern, not a fixed one (the string $[a, a, b]$ in our Example 2.2), on the other hand the calculational derivation is more specific than ours, because it deals with single-pattern string matching only, whereas our strategy is able to automatically derive programs in a much larger class which also includes multi-pattern matching, matching with regular expressions, and parsing.

We have shown that the transformation rules we use for program specialization, are correct w.r.t. the declarative semantics of logic programs based on the least Herbrand model. The proof of this correctness result is similar to the proofs of the correctness results which are presented in [11, 33, 37].

We have also considered an operational semantics for our logic language where a disequation $t_1 \neq t_2$ holds iff t_1 and t_2 are not unifiable. This operational semantics is sound, but not complete w.r.t. the declarative semantics. Indeed, if a goal operationally succeeds in a program, then it is true in the least Herbrand model of the program, but not vice versa. Thus, the proof of correctness of our transformation rules w.r.t. the operational semantics cannot be based on previous results and it is much more elaborate. Indeed, it requires some restrictions on the programs and applicability conditions on the transformation rules which are related to the modes of the predicates.

In Section 2 we have extensively discussed the fact that our specialization technique is more powerful than partial evaluation [15, 23] (also referred to as *partial deduction* in the case of logic programs). The main reason of the greater power of our technique is that it uses more powerful transformation rules. In particular, partial evaluation corresponds to the use the

definition introduction, definition elimination, unfolding, and folding transformation rules, with the restriction that we may only fold a single atom at a time in the body of a clause.

Our extended rules allow us to introduce and transform new predicates defined in terms of *disjunctions of conjunctions of atoms* (recall that a set of clauses with the same head is equivalent to a single clause whose premise is the disjunction of the bodies of the clauses in the given set). Thus, our technique improves over the one of *conjunctive partial deduction* [5], which is a specialization technique where new predicates are defined in terms of conjunctions of atoms.

The use of the case split rule is a form of reasoning *by cases*, which is a very well-known technique in mechanical theorem proving (see, for instance, the Edinburgh LCF theorem prover [14]). Forms of reasoning by cases have been incorporated in program specialization techniques such as the already mentioned supercompilation with perfect driving [12, 38] and generalized partial computation [9]. However, the strategy presented in this paper is the first fully automatic transformation technique which uses case reasoning to reduce nondeterminism of logic programs.

Besides specializing programs and reducing nondeterminism, our strategy is able to eliminate intermediate data structures. Indeed, the initial programs of our examples in Section 6 all have intermediate lists, while the specialized programs do not have them. Thus, our strategy can be regarded as an extension of the transformation strategies for the elimination of intermediate data structures (see the *deforestation* technique [39] for the case of functional programs and the strategy for *eliminating unnecessary variables* [31] for the case of logic programs [31]). Moreover, our strategy derives specialized programs which avoid repeated subcomputations (see the Context-free Parsing example of Section 6.5). In this respect our strategy is similar to the *tupling strategy* for functional programs [26].

Finally, our specialization strategy is related to the program derivation techniques called *finite differencing* [25] and *incrementalization* [21]. These techniques use program invariants to avoid costly, repeated calculations of function calls. Our specialization strategy implicitly discovers and exploits program invariants by using the folding rule. It should be noticed, however, that it is difficult to establish in a rigorous way the formal connection between the basic ideas underlying our specialization strategy and the above mentioned invariant-based program derivation methods. They, in fact, are presented in very different frameworks.

This paper is an improved version of [27, 28].

Acknowledgments

We would like to thank D. De Schreye, J. Gallagher, R. Glück, N. D. Jones, M. Leuschel, B. Martens, and M. H. Sørensen for stimulating discussions about partial evaluation and logic program specialization. We also acknowledge very useful comments by the anonymous referees of LOPSTR'96 and POPL'97. This work has been partially supported by the EC under the HCM Project 'Logic Program Synthesis and Transformation', the INTAS Project 93-1702, MURST 40% (Italy), and Progetto Coordinato 'Programmazione Logica' of the CNR (Italy).

Appendix. Proof of Theorem 4.1

For the reader's convenience, we rewrite the statement of Theorem 4.1.

Theorem 4.1 (Correctness w.r.t. the Operational Semantics) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9 and let p be a non-basic predicate in P_n. Let M be a mode for $P_0 \cup \mathit{Defs}_n$ such that: (i) $P_0 \cup \mathit{Defs}_n$ is safe w.r.t. M, (ii) $P_0 \cup \mathit{Defs}_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are all safe w.r.t. M. Suppose also that:

1. *if* the folding rule is applied for the derivation of a clause C in program P_{k+1} from clauses $C_1, \ldots, C_m$ in program P_k using clauses $D_1, \ldots, D_m$ in Defs_k, with $0 \leq k < n$,
   then for every $i \in \{1, \ldots, m\}$ there exists $j \in \{1, \ldots, n-1\}$ such that D_i occurs in P_j and P_{j+1} is derived from P_j by unfolding D_i.

2. during the transformation sequence $P_0, \ldots, P_n$ the definition elimination rule *either* is never applied *or* it is applied w.r.t. predicate p once only, when deriving P_n from P_{n-1}.

Then: (i) P_n is safe w.r.t. M, (ii) P_n satisfies M, and (iii) for each atom A which has predicate p and satisfies mode M, A succeeds in $P_0 \cup \mathit{Defs}_n$ iff A succeeds in P_n.

The proof of Theorem 4.1 will be divided in four parts, corresponding to Propositions 7.1, 7.5, 7.8, and 7.18 presented below.

Proposition 7.1 (*Preservation of Safety*) shows that program P_n derived according to the hypotheses of Theorem 4.1, is safe w.r.t. mode M (that is, Point (i) of the thesis of Theorem 4.1). Proposition 7.5 (*Preservation of Modes*) shows that P_n satisfies M (that is, Point (ii) of the thesis of Theorem 4.1). Propositions 7.8 (*Partial Correctness*) and 7.18 (*Completeness*) show the *if* part and the *only-if* part, respectively, of Point (iii) of the thesis of Theorem 4.1. For proving these propositions we will use various notions and lemmata which we introduce below.

Preservation of Safety

In this section we prove that, if the transformation rules are applied according to the restrictions indicated in Theorem 4.1, then from a program which is safe w.r.t. a given mode we derive a program which is safe w.r.t. the same mode.

Proposition 7.1 (Preservation of Safety) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9. Let M be a mode for $P_0 \cup \mathit{Defs}_n$ such that: (i) $P_0 \cup \mathit{Defs}_n$ is safe w.r.t. M and (ii) the applications of the unfolding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are safe w.r.t. M. Then, for $k = 0, \ldots, n$, the program P_k is safe w.r.t. M.

Proof: The proof proceeds by induction on k. During the proof we will omit the reference to mode M. In particular, we will simply say that a program (or a clause) is safe, instead of saying that a program (or a clause) is safe w.r.t. M.

For $k = 0$ the thesis follows directly from the hypothesis that $P_0 \cup \mathit{Defs}_n$ is safe and thus, P_0 is safe. Let us now assume that, for $k < n$, program P_k is safe. We will show that also P_{k+1} is safe. We consider the following cases, corresponding to the rule which is applied to derive P_{k+1} from P_k.

Case 1: P_{k+1} is derived by applying the definition introduction rule. P_{k+1} is safe because P_k is safe and, by hypothesis, every definition clause in $Defs_n$ is safe.

Case 2: P_{k+1} is derived by applying the definition elimination rule. Then P_{k+1} is safe because P_k is safe and $P_{k+1} \subseteq P_k$.

Case 3: P_{k+1} is derived by a safe application of the unfolding rule (see Definition 3). Let us consider a clause D_i in P_{k+1} which has been derived by unfolding a clause C in P_k of the form: $H \leftarrow G_1, A, G_2$ w.r.t. the atom A. Then there exists a clause C_i in P_k such that (i) A is unifiable with $hd(C_i)$ via the mgu ϑ_i, and (ii) clause D_i in P_{k+1} of the form $(H \leftarrow G_1, bd(C_i), G_2)\vartheta_i$.

Let us now show that D_i is safe. We take a variable X occurring in a disequation $t_1 \neq t_2$ in the body of D_i, and we prove that X is either an input variable of $hd(D_i)$ or a local variable of $t_1 \neq t_2$ in D_i. We have that $t_1 \neq t_2$ is of the form $(u_1 \neq u_2)\vartheta_i$, where $u_1 \neq u_2$ is a disequation occurring in $G_1, bd(C_i), G_2$. We consider two cases:

Case A: $u_1 \neq u_2$ occurs in G_1 or G_2. Since $t_1 \neq t_2$ is of the form $(u_1 \neq u_2)\vartheta_i$, there exists a variable $Y \in vars(u_1 \neq u_2)$ such that $X \in vars(Y\vartheta)$. By the inductive hypothesis, C is safe and thus, Y is either an input variable of $hd(C)$ or a local variable of $u_1 \neq u_2$ in C. We have that: (i) if Y is an input variable of $hd(C)$ then X is an input variable of $hd(D_i)$, and (ii) if Y is a local variable of $u_1 \neq u_2$ in C then $X = Y = Y\vartheta_i$ and X is a local variable of $t_1 \neq t_2$ in D_i.

Case B: $u_1 \neq u_2$ occurs in $bd(C_i)$. From the definition of safe unfolding we have that X is either an input variable of $hd(D_i)$ or a local variable of $t_1 \neq t_2$ in D_i.

Case 4: P_{k+1} is derived by applying the folding rule. Let us consider a clause P_{k+1} of the form:

$$C.\ H \leftarrow G_1, newp(X_1, \ldots, X_h)\vartheta, G_2$$

which has been derived by folding the following clauses in P_k:

$$\begin{cases} C_1.\ H \leftarrow\ G_1, (A_1, K_1)\vartheta, G_2 \\ \ldots \\ C_m.\ H \leftarrow\ G_1, (A_m, K_m)\vartheta, G_2 \end{cases}$$

using the following definition clauses in $Defs_k$:

$$\begin{cases} D_1.\ newp(X_1, \ldots, X_h) \leftarrow A_1, K_1 \\ \ldots \\ D_m.\ newp(X_1, \ldots, X_h) \leftarrow A_m, K_m \end{cases}$$

Now we take a variable X occurring in a disequation $t_1 \neq t_2$ in the body of C, and we prove that X is either an input variable of H or a local variable of $t_1 \neq t_2$ in C.

The disequation $t_1 \neq t_2$ occurs in G_1 or G_2 and, by the hypothesis that P_k is safe, either X is an input variable of H or, for $i = 1, \ldots, m$, X is a local variable of $t_1 \neq t_2$ in C_i. If for $i = 1, \ldots, m$, X is a local variable of $t_1 \neq t_2$ in C_i, then X is a local variable of $t_1 \neq t_2$ in C, because by the definition of the folding rule (see Rule 4) X does not occur in $newp(X_1, \ldots, X_h)\vartheta$.

Case 5: P_{k+1} is derived by applying the subsumption rule. P_{k+1} is safe because $P_{k+1} \subseteq P_k$.

Case 6: P_{k+1} is derived by a safe application of the head generalization rule (see Definition 5). Let $GenC$ be a clause in P_{k+1} of the form:

$H \leftarrow Y = t, Body$

derived from a clause C in P_k of the form:

$H\{Y/t\} \leftarrow Body$

where $\{Y/t\}$ is a substitution such that Y occurs in H and Y does not occur in C.

Let us now prove that $GenC$ is safe. Let X be a variable occurring in a disequation $t_1 \neq t_2$ in $Body$. By inductive hypothesis C is safe and thus, X is either an input variable of $H\{Y/t\}$ or a local variable of $t_1 \neq t_2$ in C. If X is an input variable of $H\{Y/t\}$, then it is also an input variable of H, because by definition of safe head generalization H and $H\{Y/t\}$ have the same input variables. If X is a local variable of $t_1 \neq t_2$ in C, then X is a local variable of $t_1 \neq t_2$ in $GenC$, because X does not occur in $Y = t$.

Case 7: P_{k+1} is derived by a safe application of the case split rule (see Definition 6) to a clause C in P_k. Let us consider the following two clauses in P_{k+1}:

C_1. $(H \leftarrow Body)\{X/t\}$

C_2. $H \leftarrow X \neq t, Body$.

derived by safe case split from C. Let us now show that C_1 and C_2 are safe. Let us consider clause C_1 and let Y be a variable occurring in a disequation $t_1 \neq t_2$ in $Body\{X/t\}$. $t_1 \neq t_2$ is of the form $(u_1 \neq u_2)\{X/t\}$ where $u_1 \neq u_2$ occurs in $Body$. We consider two cases.

Case A: $Y \in vars(t)$. By the definition of safe case split, either Y is an input variable of H or Y does not occur in C. If Y is an input variable of H, then Y is an input variable of $H\{X/t\}$, and if Y does not occur in C, then Y is a local variable of $(u_1 \neq u_2)\{X/t\}$ in C_1.

Case B: $Y \notin vars(t)$. We have that Y occurs in $u_1 \neq u_2$, and thus, from the inductive hypothesis that C is safe, it follows that Y is either an input variable of H or a local variable of $u_1 \neq u_2$ in C. If Y is an input variable of H, then Y is either an input variable of $H\{X/t\}$, and if Y a local variable of $u_1 \neq u_2$ in C, then it is a local variable of $(u_1 \neq u_2)\{X/t\}$ in C_1.

Thus, C_1 is a safe clause.

Let us now consider clause C_2 and let Y be a variable occurring in a disequation $t_1 \neq t_2$ in $X \neq t, Body$. If $t_1 \neq t_2$ occurs in $Body$ then from the inductive hypothesis that C is safe, it follows that Y is either an input variable of H or a local variable of $t_1 \neq t_2$ in C_2. If $t_1 \neq t_2$ is $X \neq t$, then by the definition of safe case split (i) X is an input variable of H, and (ii) for every variable $Y \in vars(t)$, either (ii.1) Y is an input variable of H or (ii.2) Y does not occur in $H, Body$, and thus, Y is a local variable of $X \neq t$ in C_2.

Thus, C_2 is a safe clause.

Case 8: P_{k+1} is derived by applying the equation elimination rule to a clause C_1 in P_k of the form: $H \leftarrow G_1,\ t_1 = t_2,\ G_2$. We consider two cases:

Case A: t_1 and t_2 are unifiable via the most general unifier ϑ. We derive the clause: C_2. $(H \leftarrow G_1, G_2)\vartheta$. We can show that clause C_2 is safe similarly to Case 3 (A).

Case B: t_1 and t_2 are not unifiable. In this case P_{k+1} is safe because P_{k+1} is $P_k - \{C_1\}$ and, by inductive hypothesis all clauses in P_k are safe.

Case 9: P_{k+1} is derived by applying the disequation replacement rule to clause C in P_k. Let us consider the cases 9.1–9.5 of Rule 9. Cases 9.1 and 9.3–9.5 are straightforward, because they consist in the deletion of a disequation in $bd(C)$ or in the deletion of clause C. Thus, in these cases the safety of program P_{k+1} derives directly from the safety of P_k.

Let us now consider case 9.2. Suppose that clause C is of the form: $H \leftarrow G_1,\ f(t_1,\ldots,t_m) \neq f(u_1,\ldots,u_m),\ G_2$, and it is replaced by the following m (≥ 0) clauses:

$C_1.\ H \leftarrow G_1, t_1 \neq u_1, G_2$

...

$C_m.\ H \leftarrow G_1, t_m \neq u_m, G_2$

We now prove that, for $j = 0,\ldots,m$, C_j is safe. Indeed, for $j = 0 \ldots m$, if we consider a variable X occurring in $t_j \neq u_j$ then, by the inductive hypothesis, either (i) X is an input variable of H or (ii) X is a local variable of $f(t_1,\ldots,t_m) \neq f(u_1,\ldots,u_m)$ in C, and thus, X is a local variable of $t_j \neq u_j$ in C_j.

In the case where X occurs in a disequation in G_1 or G_2, it follows directly from the inductive hypothesis that X is either an input variable of H or a local variable of that disequation in C_j.

Thus, C_j is safe. □

Preservation of Modes

Now we show that, if the program $P_0 \cup Defs_n$ satisfies a mode M and we apply our transformation rules according to the restrictions indicated in Theorem 4.1, then the derived program P_n satisfies M.

In this section and in the rest of the paper, we will use the following notation and terminology. Let us consider two non-basic atoms A_1 and A_2 of the form $p(t_1,\ldots,t_m)$ and $p(u_1,\ldots,u_m)$, respectively. By $A_1 = A_2$ we denote the conjunction of equations: $t_1 = u_1,\ldots,t_m = u_m$. By $mgu(A_1, A_2)$ we denote a relevant mgu of two unifiable non-basic atoms A_1 and A_2. Similarly, by $mgu(t_1, t_2)$ we denote a relevant mgu of two unifiable terms t_1 and t_2. The *length* of the derivation $G_0 \mapsto_P G_1 \mapsto_P \ldots \mapsto_P G_n$ is n. Given a program P and a mode M for P, we say that a derivation $G_0 \mapsto_P G_1 \mapsto_P \ldots \mapsto_P G_n$ is *consistent with* M iff for $i = 0,\ldots,n-1$, if the leftmost atom of G_i is a non-basic atom A then A satisfies M.

The following properties of the operational semantics can be proved by induction on the length of the derivations.

Lemma 7.2. Let P be a program and G_1 a goal. G_1 succeeds in P iff there exists a substitution ϑ, called an *answer substitution* for G_1, such that for all goals G_2,

$(G_1, G_2) \ \mapsto_P^* \ G_2\vartheta$

Lemma 7.3. Let P be a safe program w.r.t. mode M, let Eqs be a conjunction of equations, and let G_1 be a goal without occurrences of disequations. For all goals G_2, if there exists a goal (A', G') such that A' is a non-basic atom which does not satisfy M and

$(Eqs,\ G_1,\ G_2) \ \mapsto_P^* \ (A', G')$

then there exists a goal (A'', G'') such that A'' is a non-basic atom which does not satisfy M and

$(G_1,\ Eqs,\ G_2) \ \mapsto_P^* \ (A'', G'')$

Lemma 7.4. Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9. Let M be a mode for $P_0 \cup \mathit{Defs}_n$ such that: (i) $P_0, \cup \mathit{Defs}_n$ is safe w.r.t. M, (ii) $P_0 \cup \mathit{Defs}_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are safe w.r.t. M. Then, for $k = 0, \ldots, n$, for all goals G, if all derivations from G using $P_0 \cup \mathit{Defs}_n$ are consistent with M, then all derivations from G using P_k are consistent with M.

Proof: By Proposition 7.1 we have that, for $k = 0, \ldots, n$, the program P_k is safe w.r.t. M.
The proof proceeds by induction on k.
The base case ($k = 0$) follows from the fact that all derivations from G using P_0 are also derivations using $P_0 \cup \mathit{Defs}_n$.
In order to prove the step case, we prove the following counterpositive statement:

for all goals (A_0, G_0), if there exists a goal (A_s, G_s) such that $(A_0, G_0) \mapsto^*_{P_{k+1}} (A_s, G_s)$ and (A_s, G_s) does not satisfy M, then there exists a goal (A_t, G_t) such that $(A_0, G_0) \mapsto^*_{P_k} (A_t, G_t)$ and A_t does not satisfy M.

We proceed by induction on the length s of derivation of (A_s, G_s) from (A_0, G_0) using P_{k+1}.
As an inductive hypothesis we assume that, for all $r < s$ and for all goals $\hat{G}$, if there exists a derivation $\hat{G} \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} (A_r, G_r)$ of length r, such that A_r does not satisfy M, then there exists (A', G') such that $\hat{G} \mapsto^*_{P_k} (A', G')$ and A' does not satisfy M.
Let us consider the derivation $(A_0, G_0) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} (A_s, G_s)$ of length s, such that A_s does not satisfy M.
If $s = 0$ then G is (A_s, G_s) and $(A_0, G_0) \mapsto^*_{P_k} (A_s, G_s)$ where A_s does not satisfy M.
If $s > 0$ then we may assume $A_0 \neq \mathit{true}$, and we have the following cases.

Case 1: A_0 is the equation $t_1 = t_2$. Thus, by Point (1) of the operational semantics of Section 3.3, the derivation from (A_0, G_0) to (A_s, G_s) using P_{k+1} is of the form:

$(A_0, G_0) \mapsto_{P_{k+1}} G_0\, \mathit{mgu}(t_1, t_2) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} (A_s, G_s)$

By the inductive hypothesis there exists (A', G') such that $G_0\, \mathit{mgu}(t_1, t_2) \mapsto^*_{P_k} (A', G')$ and A' does not satisfy M. Thus, $(A_0, G_0) \mapsto^*_{P_k} (A', G')$.

Case 2: A_0 is the disequation $t_1 \neq t_2$. The proof proceeds as in Case 1, by using Point (2) of the operational semantics and the inductive hypothesis.

Case 3: A_0 is a non-basic atom which satisfies M. (The case where A_0 does not satisfy M is subsumed by the case $s = 0$.) By Point (3) of the operational semantics, the derivation from (A_0, G_0) to (A_s, G_s) using P_{k+1} is of the form:

$(A_0, G_0) \mapsto_{P_{k+1}} (\mathit{bd}(E), G_0)\mathit{mgu}(A_0, \mathit{hd}(E)) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} (A_s, G_s)$

where E is a renamed apart clause in P_{k+1}.
If $E \in P_k$ then $(A_0, G_0) \mapsto_{P_k} (\mathit{bd}(E), G_0)\mathit{mgu}(A_0, \mathit{hd}(E))$ and the thesis follows directly from the inductive hypothesis.
Otherwise, if $E \in (P_{k+1} - P_k)$, we prove that:

there exists a goal (A_t, G_t) such that $(A_0, G_0) \mapsto^*_{P_k} (A_t, G_t)$ and A_t does not satisfy M (†)

by considering the following cases, corresponding to the rule which is applied to derive E.

Case 3.1: E is derived by applying the definition introduction rule. Thus, $E \in \mathit{Defs}_n$ and (†) follows from the inductive hypothesis and the hypothesis that $P_0 \cup \mathit{Defs}_n$ satisfies M.

Case 3.2: E is derived by unfolding a clause C in P_k of the form $H \leftarrow D, G_1, A, G_2$, where D is a conjunction of disequations, w.r.t. the non-basic atom A. By Proposition 4.2 we may assume that no disequation occurs in G_1, A, G_2. Let $C_1, \ldots, C_m$, with $m \geq 0$, be the clauses of P_k such that, for all $i \in \{1, \ldots, m\}$ A is unifiable with the head of C_i via the mgu ϑ_i.

Thus, E is of the form $(H \leftarrow D, G_1, bd(C_i), G_2)\vartheta_i$, for some $i \in \{1, \ldots, m\}$, and the derivation from (A_0, G_0) to (A_s, G_s) using P_{k+1} is of the form:

$$(A_0, G_0) \mapsto_{P_{k+1}} ((D, G_1, bd(C_i), G_2)\vartheta_i, G_0)\eta_i \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} (A_s, G_s)$$

where η_i is an mgu of A_0 and $H\vartheta_i$. By the inductive hypothesis there exists (A', G') such that A' does not satisfy M and:

$$((D, G_1, bd(C_i), G_2)\vartheta_i, G_0)\eta_i \mapsto^*_{P_k} (A', G')$$

Since ϑ_i is $mgu(A, hd(C_i))$, ϑ_i is relevant, and $vars(G_0) \cap vars((A, hd(C_i))) = \emptyset$, we have that:

$$(D, G_1, bd(C_i), G_2, G_0)\vartheta_i\eta_i \mapsto^*_{P_k} (A', G')$$

and thus, by the definition of the operational semantics (Point 1), we have that:

$$(A = hd(C_i), A_0 = H, D, G_1, bd(C_i), G_2, G_0) \mapsto^*_{P_k} (A', G')$$

Then, by properties of mgu's, we have that:

$$(A_0 = H, A = hd(C_i), D, G_1, bd(C_i), G_2, G_0) \mapsto^*_{P_k} (A', G')$$

Since A_0 satisfies M, C is safe, and C_i is renamed apart, we have that $vars(D\ mgu(A_0, H)) \cap vars(A, hd(C_i)) = \emptyset$. Therefore, we have that $(D\ mgu(A_0, H)\ mgu(A\ mgu(A_0, H), hd(C_i)))$ $= (D\ mgu(A_0, H))$ and, thus:

$$(A_0 = H, D, A = hd(C_i), G_1, bd(C_i), G_2, G_0) \mapsto^*_{P_k} (A', G')$$

Now, by Lemma 7.3, there exists a goal (A'', G'') such that:

$$(A_0 = H, D, G_1, A = hd(C_i), bd(C_i), G_2, G_0) \mapsto^*_{P_k} (A'', G'')$$

where A'' is a non-basic atom which does not satisfy M. There are two cases:

Case A. $(A_0 = H, D, G_1) \mapsto^*_{P_k} (A'', G''')$ for some goal G'''. In this case, by using clause $C \in P_k$, we have that:

$$(A_0, G_0) \mapsto_{P_k} (D, G_1, A, G_2, G_0)\, mgu(A_0, H) \mapsto^*_{P_k} (A'', G'''')$$

for some goal G''''.

Case B. There is no (A''', G''') such that $(A_0 = H, D, G_1) \mapsto^*_{P_k} (A''', G''')$ and A''' does not satisfy M. In this case $(A_0 = H, D, G_1, A = hd(C_i))$ succeeds in P_k. It follows that, for some substitution ϑ,

$$\begin{aligned}
&(A_0 = H, D, G_1, A = hd(C_i), bd(C_i), G_2, G_0) \\
&\qquad \mapsto^*_{P_k} (A = hd(C_i), bd(C_i), G_2, G_0)\vartheta && \text{(by Lemma 7.2)} \\
&\qquad \mapsto_{P_k} (bd(C_i), G_2, G_0)\vartheta\ mgu(A\vartheta, hd(C_i)) \\
&\qquad\qquad && \text{(because mgu's are relevant and } C_i \text{ is renamed apart)} \\
&\qquad \mapsto^*_{P_k} (A'', G'''')
\end{aligned}$$

for some goal G''''. Thus,

$$\begin{aligned}(A_0 = H, D, G_1, A, G_2, G_0) &\\ &\mapsto^*_{P_k} (A, G_2, G_0)\vartheta\\ &\mapsto_{P_k} (bd(C_i), G_2, G_0)\vartheta\ mgu(A\vartheta, hd(C_i))\\ &\mapsto^*_{P_k} (A'', G'''')\end{aligned}$$

and therefore, by using clause $C \in P_k$,

$$(A_0, G_0) \ \mapsto^*_{P_k} \ (A'', G'''')$$

where A'' is a non-basic atom which does not satisfy M. Thus, (†) holds.

Case 3.3: E is derived by a safe application of the folding rule (see Definition 4). In particular, suppose that from the following clauses in P_k:

$$\begin{cases} C_1.\ H \leftarrow\ G_1, (A_1, K_1)\vartheta, G_2\\ \cdots\\ C_m.\ H \leftarrow\ G_1, (A_m, K_m)\vartheta, G_2 \end{cases}$$

and the following definition clauses in $Defs_k$:

$$\begin{cases} D_1.\ newp(X_1, \ldots, X_h) \leftarrow A_1, K_1\\ \cdots\\ D_m.\ newp(X_1, \ldots, X_h) \leftarrow A_m, K_m \end{cases}$$

we have derived the clause E of the form:

$$E.\ H \leftarrow G_1, newp(X_1, \ldots, X_h)\vartheta, G_2$$

where Property Σ of Definition 4 holds, that is, each input variable of $newp(X_1, \ldots, X_h)\vartheta$, is also an input variable of at least one of the non-basic atoms occurring in $(H, G_1, A_1\vartheta, \ldots, A_m\vartheta)$.

Thus, the derivation from (A_0, G_0) to (A_s, G_s) using P_{k+1} is of the form:

$$(A_0, G_0) \mapsto_{P_{k+1}} (G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)mgu(A_0, H) \mapsto^*_{P_{k+1}} (A_s, G_s)$$

By the inductive hypothesis, there exists a goal (A', G') such that A' does not satisfy M and the following holds:

$$(G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)mgu(A_0, H) \mapsto^*_{P_k} (A', G')$$

There are two cases:

Case A: $G_1 mgu(A_0, H) \mapsto^*_{P_k} (A', G'')$ for some goal G''. In this case we have that, for some $i \in \{1, \ldots, m\}$, and for some goal G''',

$$\begin{aligned}(A_0, G_0) \ &\mapsto_{P_k} (G_1, (A_i, K_i)\vartheta, G_2, G_0)mgu(A_0, H) && \text{(by using clause } C_i \text{ in } P_k)\\ &\mapsto^*_{P_k} (A', G''')\end{aligned}$$

Thus, (†) holds.

Case B: There is no (A'', G'') such that $G_1 mgu(A_0, H) \mapsto^*_{P_k} (A'', G'')$ and A'' does not satisfy M. In this case $G_1 mgu(A_0, H)$ succeeds in P_k, and thus, for some substitution α,

$$(A_0, G_0) \mapsto^*_{P_k} (newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)\alpha \mapsto^*_{P_k} (A', G')$$

By Property Σ, we have that $newp(X_1, \ldots, X_h)\vartheta\alpha$ satisfies M.

It can be shown the following fact. Let us consider the set of all definition clauses with head predicate $newp$ in $Defs_k$, for any $k \in \{0, \ldots, n\}$:

$$\begin{cases} newp(X_1, \ldots, X_h) \leftarrow Body_1 \\ \quad \ldots \\ newp(X_1, \ldots, X_h) \leftarrow Body_m \end{cases}$$

If for a substitution β and a goal G, the atom $newp(X_1, \ldots, X_h)\beta$ satisfies M and $newp(X_1, \ldots, X_h)\beta, G \mapsto^*_{P_k} (A', G')$, where A' is a non-basic atom which does not satisfy M, then for some $i \in \{1, \ldots, m\}$ we have that there exists a goal (A_t, G_t) such that $Body_i\beta, G \mapsto^*_{P_k} (A_t, G_t)$, where A_t is a non-basic atom which does not satisfy M.

By using this fact, we have that, for some $i \in \{1, \ldots, m\}$,

$$(A_0, G_0) \mapsto^*_{P_k} ((A_i, K_i)\vartheta, G_2, G_0)\alpha \mapsto^*_{P_{k+1}} (A_t, G_t)$$

where A_t is a non-basic atom which does not satisfy M and thus, (†) holds.

Case 3.4: E is derived by applying the head generalization rule. In this case (†) follows from the inductive hypothesis and from the definition of the operational semantics (Point 1).

Case 3.5: E is derived by safe case split (see Definition 6) from a clause C in P_k. By Proposition 4.2, we may assume that C is of the form: $H \leftarrow D, B$, where D is a conjunction of disequations and in B there are no occurrences of disequations. Thus, E is of one of the following two forms:

C_1. $(H \leftarrow D, B)\{X/t\}$

C_2. $H \leftarrow X \neq t, D, B$

where X is an input variable of H, X does not occur in t, and for all variables $Y \in vars(t)$, *either* Y is an input variable of H *or* Y does not occur in C.

Case A: E is C_1. Thus, the derivation from (A_0, G_0) to (A_s, G_s) using P_{k+1} takes the form:

$$(A_0, G_0) \mapsto_{P_{k+1}} ((D, B)\{X/t\}, G_0)\, mgu(A_0, H\{X/t\}) \mapsto^*_{P_{k+1}} (A_s, G_s)$$

By the inductive hypothesis, there exists a goal (A', G') such that A' does not satisfy M and the following holds:

$$((D, B)\{X/t\}, G_0)\, mgu(A_0, H\{X/t\}) \mapsto^*_{P_k} (A', G')$$

By properties of mgu's and Point (1) of the operational semantics, we have that:

$$A_0 = H,\ X = t,\ D,\ B,\ G_0 \mapsto^*_{P_k} (A', G')$$

By the conditions for safe case split, we have that:

$$vars((X = t)\, mgu(A_0, H)) \cap vars((D,\ B,\ G_0)\, mgu(A_0, H)) = \emptyset$$

and therefore:

$$A_0 = H,\ D,\ B,\ G_0 \mapsto^*_{P_k} (A', G')$$

Thus, by using clause $C \in P_k$,

$$(A_0, G_0) \mapsto_{P_k} (D,\ B,\ G_0) mgu(A_0, H) \mapsto^*_{P_k} (A', G')$$

and (†) holds.

Case B: E is C_2. Thus, the derivation from (A_0, G_0) to (A_s, G_s) using P_{k+1} takes the form:

$(A_0, G_0) \mapsto_{P_{k+1}} (X \neq t, D, B, G_0)\, mgu(A_0, H) \mapsto^*_{P_{k+1}} (A_s, G_s)$

By the inductive hypothesis, there exists a goal (A', G') such that A' does not satisfy M and:

$(X \neq t, D, B, G_0)\, mgu(A_0, H) \mapsto^*_{P_k} (A', G')$

Since the answer substitution for any successful disequation is the identity substitution, we have that:

$(D, B, G_0)\, mgu(A_0, H) \mapsto^*_{P_k} (A', G')$

Thus, by using clause $C \in P_k$, we have that

$(A_0, G_0) \mapsto^*_{P_k} (A', G')$

and (†) holds.

Case 3.6: E is derived by applying the equation elimination rule. In this case (†) is a consequence of the inductive hypothesis, Point (1) of the operational semantics, the safety of P_k, and Lemma 7.3.

Case 3.7: E is derived by applying the disequation replacement rule. In this case (†) is a consequence of the inductive hypothesis, Point (2) of the operational semantics, and the properties of unification. □

From Lemma 7.4 and Definition 1 we have the following proposition.

Proposition 7.5 (Preservation of Modes) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9. Let M be a mode for $P_0 \cup Defs_n$ such that: (i) $P_0 \cup Defs_n$ is safe w.r.t. M, (ii) $P_0 \cup Defs_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are safe w.r.t. M. Then, for $k = 0, \ldots, n$, the program P_k satisfies M.

Partial Correctness

For proving the partial correctness of the transformation rules w.r.t. the operational semantics (that is, Proposition 7.8), we will use the following two lemmata.

Lemma 7.6. Let P be a safe program w.r.t. mode M, let Eqs be a conjunction of equations, and let G_1 be a goal without occurrences of disequations. For all goals G_2, if

$(Eqs,\ G_1,\ G_2) \mapsto^*_P G_2\vartheta$

then either

$(G_1,\ Eqs,\ G_2) \mapsto^*_P G_2\vartheta$

or there exists a goal (A', G') such that A' is a non-basic atom which does not satisfy M and

$G_1 \mapsto^*_P (A', G')$

Lemma 7.7. Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9. Let M be a mode for $P_0 \cup Defs_n$ such that: (i) $P_0 \cup Defs_n$ is safe w.r.t. M, (ii) $P_0 \cup Defs_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are all safe w.r.t. M.

Then, for $k = 0, \ldots, n-1$, for each goal G, if there exists a derivation $G \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$ which is consistent with M, then $G \mapsto^*_{P_k \cup \mathit{Defs}_n} \mathit{true}$, that is, G succeeds in $P_k \cup \mathit{Defs}_n$.

Proof: By hypotheses (i–iii), and Propositions 7.1 and 7.5, for $k = 0, \ldots, n$, program P_k is safe and satisfies M. Let G be a goal of the form (A_0, G_0), such that there exists a derivation

$\delta: \quad (A_0, G_0) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is consistent with M. We will prove that:

$(A_0, G_0) \mapsto^*_{P_k \cup \mathit{Defs}_n} \mathit{true}$

The proof proceeds by induction on the length s of the derivation δ.
Base Case. For $s = 0$, the goal (A_0, G_0) is *true* and the thesis follows from the fact that *true* succeeds in all programs.

Step Case. Let us now assume the following

Inductive Hypothesis: for all $r < s$ and for all goals G, if there exists a derivation $G \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$ of length r which is consistent with M, then $G \mapsto^*_{P_k \cup \mathit{Defs}_n} \mathit{true}$.

There are the following three cases.

Case 1: A_0 is the equation $t_1 = t_2$. By Point (1) of the operational semantics of Section 3.3, the derivation δ is of the form:

$(t_1 = t_2, G_0) \mapsto_{P_{k+1}} G_0\, \mathit{mgu}(t_1, t_2) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

Thus, the derivation $G_0\, \mathit{mgu}(t_1, t_2) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$ has length $s-1$ and it is consistent with M. By the inductive hypothesis there exists a derivation $G_0\, \mathit{mgu}(t_1, t_2) \mapsto^*_{P_k} \mathit{true}$. Thus, $(A_0, G_0) \mapsto^*_{P_k} \mathit{true}$ and (A_0, G_0) succeeds in $P_k \cup \mathit{Defs}_n$.

Case 2: A_0 is the disequation $t_1 \neq t_2$. The proof proceeds as in Case 1, by using Point (2) of the operational semantics and the inductive hypothesis.

Case 3: A_0 is a non-basic atom which satisfies M (otherwise there is no derivation starting from (A_0, G_0) which is consistent with M). By Point (3) of the operational semantics, the derivation δ is of the form:

$(A_0, G_0) \mapsto_{P_{k+1}} (\mathit{bd}(E), G_0)\mathit{mgu}(A_0, \mathit{hd}(E)) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

where E is a renamed apart clause in P_{k+1}.
If $E \in P_k$ then $(A_0, G_0) \mapsto_{P_k} (\mathit{bd}(E), G_0)\mathit{mgu}(A_0, \mathit{hd}(E))$ and the thesis follows directly from the inductive hypothesis.
Otherwise, if $E \in (P_{k+1} - P_k)$, we prove that (A_0, G_0) succeeds in $P_k \cup \mathit{Defs}_n$ by considering the following cases, which correspond to the rules applied for deriving E.

Case 3.1: E is derived by applying the definition introduction rule. Thus, E is a clause in Defs_n of the form: $\mathit{newp}(X_1, \ldots, X_h) \leftarrow B$ and the derivation δ is of the form:

$(\mathit{newp}(t_1, \ldots, t_h), G_0) \mapsto_{\mathit{Defs}_n} (B\{X_1/t_1, \ldots, X_h/t_h\}, G_0) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

By the inductive hypothesis, we have that:

$(B\{X_1/t_1, \ldots, X_h/t_h\}, G_0) \mapsto^*_{P_k} \mathit{true}$

and thus,

$(\mathit{newp}(t_1, \ldots, t_h), G_0) \mapsto^*_{P_k \cup \mathit{Defs}_n} \mathit{true}$

Case 3.2: E is derived by unfolding a clause C in P_k of the form $H \leftarrow D, G_1, A, G_2$, where D is a conjunction of disequations, w.r.t. the non-basic atom A. By Proposition 4.2 we may assume that no disequation occurs in G_1, A, G_2. Let $C_1, \ldots, C_m$, with $m \geq 0$, be the clauses of P_k such that, for all $i \in \{1, \ldots, m\}$ A is unifiable with the head of C_i via the mgu ϑ_i.

Thus, E is of the form $(H \leftarrow D, G_1, bd(C_i), G_2)\vartheta_i$, for some $i \in \{1, \ldots, m\}$, and the derivation δ is of the form:

$$(A_0, G_0) \mapsto_{P_{k+1}} ((D, G_1, bd(C_i), G_2)\vartheta_i, G_0)\eta_i \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$$

where η_i is an mgu of A_0 and $H\vartheta_i$. By the inductive hypothesis we have that:

$$((D, G_1, bd(C_i), G_2)\vartheta_i, G_0)\eta_i \mapsto^*_{P_k} \mathit{true}$$

Since ϑ_i is $mgu(A, hd(C_i))$, ϑ_i is relevant, and $vars(G_0) \cap vars((A, hd(C_i))) = \emptyset$, we have that:

$$(D, G_1, bd(C_i), G_2, G_0)\vartheta_i\eta_i \mapsto^*_{P_k} \mathit{true}$$

and thus, by the definition of the operational semantics (Point 1), we have that:

$$(A = hd(C_i), A_0 = H, D, G_1, bd(C_i), G_2, G_0) \mapsto^*_{P_k} \mathit{true}$$

Then, by properties of mgu's, we have that:

$$(A_0 = H, A = hd(C_i), D, G_1, bd(C_i), G_2, G_0) \mapsto^*_{P_k} \mathit{true}$$

Since A_0 satisfies M, C is safe, and C_i is renamed apart, we have that $vars(D\, mgu(A_0, H)) \cap vars(A, hd(C_i)) = \emptyset$. Therefore, we have that $(D\ mgu(A_0, H)\ mgu(A\ mgu(A_0, H), hd(C_i))) = (D\ mgu(A_0, H))$ and, thus:

$$(A_0 = H, D, A = hd(C_i), G_1, bd(C_i), G_2, G_0) \mapsto^*_{P_k} \mathit{true}$$

Now, by Lemma 7.6, there are the following two cases.

Case A. $(A_0 = H, D, G_1, A = hd(C_i), bd(C_i), G_2, G_0) \mapsto^*_{P_k} \mathit{true}$

In this case, by Points (1) and (3) of the operational semantics we have that:

$$(A_0 = H, D, G_1, A, G_2, G_0) \mapsto^*_{P_k} \mathit{true}$$

and thus, by using clause C in P_k,

$$(A_0, G_0) \mapsto^*_{P_k} \mathit{true}$$

Case B. There exists a goal (A', G') such that:

$$(A_0 = H, D, G_1) \mapsto^*_{P_k} (A', G')$$

where A' is a non-basic atom which does not satisfy the mode M. In this case we have that, for some goal G'',

$$A_0 \mapsto^*_{P_k} (A', G'')$$

which is impossible because A_0 and P_k satisfy M.

Case 3.3: E is derived by a safe application of the folding rule (see Definition 4). In particular, suppose that from the following clauses in P_k:

$$\begin{cases} C_1.\ H \leftarrow\ G_1, (A_1, K_1)\vartheta, G_2 \\ \cdots \\ C_m.\ H \leftarrow\ G_1, (A_m, K_m)\vartheta, G_2 \end{cases}$$

and the following definition clauses in $Defs_k$:

$$\begin{cases} D_1.\ newp(X_1, \ldots, X_h) \leftarrow A_1, K_1 \\ \cdots \\ D_m.\ newp(X_1, \ldots, X_h) \leftarrow A_m, K_m \end{cases}$$

we have derived the clause E of the form:

$$E.\ H \leftarrow G_1, newp(X_1, \ldots, X_h)\vartheta, G_2$$

where Property Σ of Definition 4 holds, that is, each input variable of $newp(X_1, \ldots, X_h)\vartheta$, is also an input variable of at least one of the non-basic atoms occurring in $\{H, G_1, A_1\vartheta, \ldots, A_m\vartheta\}$.

Thus, the derivation δ is of the form:

$$(A_0, G_0) \mapsto_{P_{k+1}} (G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)mgu(A_0, H) \mapsto^*_{P_{k+1}} true$$

By the inductive hypothesis, the following holds:

$$(G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)mgu(A_0, H) \mapsto^*_{P_k} true$$

and therefore, for some substitution α,

$$(A_0, G_0) \mapsto^*_{P_k} (newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)\alpha \mapsto^*_{P_k} true$$

By Property Σ, we have that $newp(X_1, \ldots, X_h)\vartheta\alpha$ satisfies M.

It can be shown the following fact. Let us consider the set of all definition clauses with head predicate $newp$ in $Defs_k$, for any $k \in \{0, \ldots, n\}$:

$$\begin{cases} newp(X_1, \ldots, X_h) \leftarrow Body_1 \\ \qquad \cdots \\ newp(X_1, \ldots, X_h) \leftarrow Body_m \end{cases}$$

If for a substitution β for a goal G, the atom $newp(X_1, \ldots, X_h)\beta, G$ satisfies M and we have that $newp(X_1, \ldots, X_h)\beta, G \mapsto^*_{P_k} true$, then for some $i \in \{1, \ldots, m\}$ we have that $Body_i\beta \mapsto^*_{P_k} true$.

By using this fact, we have that, for some $i \in \{1, \ldots, m\}$,

$$(A_0, G_0) \mapsto^*_{P_k} ((A_i, K_i)\vartheta, G_2, G_0)\alpha \mapsto^*_{P_k} true$$

Case 3.4: E is derived by applying the head generalization rule. In this case $(A_0, G_0) \mapsto^*_{P_k} true$ follows from the inductive hypothesis and from the definition of the operational semantics (Point 1).

Case 3.5: E is derived by safe case split (see Definition 6) from a clause C in P_k. By Proposition 4.2, we may assume that C is of the form: $H \leftarrow D, B$, where D is a conjunction of disequations and in B there are no occurrences of disequations. Thus, E is of one of the following two forms:

$$C_1.\ (H \leftarrow D, B)\{X/t\}$$

C_2. $H \leftarrow X \neq t, D, B$

where X is an input variable of H, X does not occur in t, and for all variables $Y \in vars(t)$, *either* Y is an input variable of H *or* Y does not occur in C.

Case A: E is C_1. Thus, the derivation δ takes the form:

$(A_0, G_0) \mapsto_{P_{k+1}} ((D, B)\{X/t\}, G_0)\, mgu(A_0, H\{X/t\}) \mapsto^*_{P_{k+1}} true$

By the inductive hypothesis, we have that:

$((D, B)\{X/t\}, G_0)\, mgu(A_0, H\{X/t\}) \mapsto^*_{P_k} true$

By properties of mgu's and Point (1) of the operational semantics, we have that:

$(A_0 = H,\ X = t,\ D,\ B,\ G_0) \mapsto^*_{P_k} true$

By the conditions for safe case split, we have that:

$vars((X = t)\, mgu(A_0, H)) \cap vars((D,\ B,\ G_0)\, mgu(A_0, H)) = \emptyset$

and therefore:

$(A_0 = H,\ D,\ B,\ G_0) \mapsto^*_{P_k} true$

Thus, by using clause $C \in P_k$,

$(A_0, G_0) \mapsto_{P_k} (D,\ B,\ G_0) mgu(A_0, H) \mapsto^*_{P_k} true$

Case B: E is C_2. Thus, the derivation δ takes the form:

$(A_0, G_0) \mapsto_{P_{k+1}} (X \neq t, D, B, G_0)\, mgu(A_0, H) \mapsto^*_{P_{k+1}} true$

By the inductive hypothesis, we have that:

$(X \neq t, D, B, G_0)\, mgu(A_0, H) \mapsto^*_{P_k} true$

Since the answer substitution for any successful disequation is the identity substitution, we have that:

$(D, B, G_0)\, mgu(A_0, H) \mapsto^*_{P_k} true$

Thus, by using clause $C \in P_k$,

$(A_0, G_0) \mapsto^*_{P_k} true$

Case 3.6: E is derived by applying the equation elimination rule. In this case $(A_0, G_0) \mapsto^*_{P_k} true$ is a consequence of the inductive hypothesis, Point (1) of the operational semantics, the fact that P_k is safe and satisfies M, and Lemma 7.6.

Case 3.7: E is derived by applying the disequation replacement rule. In this case $(A_0, G_0) \mapsto^*_{P_k} true$ is a consequence of the inductive hypothesis, Point (2) of the operational semantics, and the properties of unification. □

Proposition 7.8 (Partial Correctness) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9. Let M be a mode for $P_0 \cup Defs_n$ such that: (i) $P_0 \cup Defs_n$ is safe w.r.t. M, (ii) $P_0 \cup Defs_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are all safe w.r.t. M.
Then, for $k = 0, \ldots, n$, for each non-basic atom A which satisfies mode M, if A succeeds in P_k then A succeeds in $P_0 \cup Defs_k$.

Proof: By Proposition 7.5, P_k is safe and therefore, if a non-basic atom A satisfies mode M and succeeds in P_k, then every successful derivation from A is consistent with M. Thus, the thesis follows from Lemma 7.7. □

Completeness

For the proofs of Propositions 7.1 (Preservation of Safety), 7.5 (Preservation of Modes), and 7.8 (Partial Correctness), we have proceeded by induction on the length of the derivations and by cases on the rule used to derive program P_{k+1} from program P_k. For the proof of Proposition 7.18 below (Completeness), we will proceed by induction w.r.t. more sophisticated well-founded orderings. This proof technique is a suitable modification of the one based on *weight consistent* proof trees [11, 37].

The following definition introduces some well-founded orders and other notions which are needed for the proofs presented in this section.

Definition 13. (i) Given a derivation δ of the form $G_0 \mapsto_P G_1 \mapsto_P \ldots \mapsto_P G_z$, we denote by $\lambda(\delta)$ the number of goals G_i in δ such that G_i is of the form (A, K) where A is a non-basic atom.
(ii) We define the following functions μ and ν which given a program and a goal return either a non-negative integer or ∞ (we assume that, for all non-negative integers n, $\infty > n$):

$$\mu(P, G) = \begin{cases} \min\{\lambda(\delta) \mid \delta \text{ is a successful derivation of } G \text{ in } P\} & \text{if } G \text{ succeeds in } P \\ \infty & \text{otherwise} \end{cases}$$

$$\nu(P, G) = \begin{cases} \min\{n \mid n \text{ is the length of a successful derivation of } G \text{ in } P\} & \text{if } G \text{ succeeds in } P \\ \infty & \text{otherwise} \end{cases}$$

(iii) Given a program P and two goals G_1 and G_2, we write $G_1 \succ_P G_2$ iff $\mu(P, G_1) > \mu(P, G_2)$. Similarly, we write $G_1 \succeq_P G_2$ iff $\mu(P, G_1) \geq \mu(P, G_2)$.

(iv) Given two programs P and Q, we say that a derivation $G_0 \mapsto_P G_1 \mapsto_P \ldots \mapsto_P G_z$ is *quasi-decreasing* w.r.t. $\succ_Q$ iff for $i = 0, \ldots, z-1$, either (1) $G_i \succ_Q G_{i+1}$ or (2) the leftmost atom of G_i is a basic atom and $G_i \succeq_Q G_{i+1}$.

(v) Let P a program and G_1, G_2 be goals. If there exists a derivation δ from G_1 to G_2 such that $\lambda(\delta) = s$, then we write $G_1 \mapsto_P^s G_2$.

For any program P the relation $\succ_P$ is a well-founded order and, for all goals G_1, G_2, and G_3, we have that $G_1 \succ_P G_2$ and $G_2 \succeq_P G_3$ implies $G_1 \succ_P G_3$.

Lemma 7.9. Let P be a program and G be a goal. If G succeeds in P then G has a derivation which is quasi-decreasing w.r.t. $\succ_P$.

Proof: The derivation δ from G using P such that $\lambda(\delta) \leq \lambda(\delta')$ for all successful derivations δ' from G, is quasi-decreasing w.r.t. $\succ_P$. □

Lemma 7.10. Let M be a mode for program P, such that P is safe w.r.t. M and P satisfies M. Let *Eqs* be a conjunction of equations, and G_0, G_1, G_2 be goals. Suppose also that no disequation occurs in G_1 and all derivations from the goal (G_0, G_1) are consistent with M. Then:

(i) $(G_0,\ G_1,\ Eqs,\ G_2) \mapsto_P^* true$ iff $(G_0,\ Eqs,\ G_1,\ G_2) \mapsto_P^* true$

(ii) $\mu(P,\ (G_0,\ G_1,\ Eqs,\ G_2)) = \mu(P,\ (G_0,\ Eqs,\ G_1,\ G_2))$

(iii) $\nu(P,\ (G_0,\ G_1,\ Eqs,\ G_2)) = \nu(P,\ (G_0,\ Eqs,\ G_1,\ G_2))$

Proof: By induction on the length of the derivations. □

Lemma 7.11. Let M be a mode for program P, such that P is safe w.r.t. M and P satisfies M. Let ϑ be a substitution and G_0, G_1, G_2 be goals. Suppose also that no disequation occurs in G_2 and all derivations from the goal (G_0, G_2) are consistent with M. Then:

(i) if $(G_0,\ G_1,\ G_2)\vartheta \mapsto_P^* \mathit{true}$ then $(G_0,\ G_2) \mapsto_P^* \mathit{true}$

(ii) $\mu(P,\ (G_0, G_1, G_2)\vartheta) \geq \mu(P,\ (G_0, G_2))$

(iii) $\nu(P,\ (G_0, G_1, G_2)\vartheta) \geq \nu(P,\ (G_0, G_2))$

Proof: By induction on the length of the derivations. □

Lemma 7.12. Let M be a mode for program P, such that P is safe w.r.t. M and P satisfies M. Let *Diseqs* be a conjunction of disequations and G be a goal. Suppose also that $\mathit{vars}(\mathit{Diseqs}) \cap \mathit{vars}(G) = \emptyset$. Then:

(i) $(G,\ \mathit{Diseqs}) \mapsto_P^* \mathit{true}$ iff $(\mathit{Diseqs},\ G) \mapsto_P^* \mathit{true}$

(ii) $\mu(P,\ (G,\ \mathit{Diseqs})) = \mu(P,\ (\mathit{Diseqs},\ G))$

(iii) $\nu(P,\ (G,\ \mathit{Diseqs})) = \nu(P,\ (\mathit{Diseqs},\ G))$

Proof: The proof proceeds by induction on the length of the derivations. □

Let us consider a transformation sequence $P_0, \ldots, P_n$ constructed by using the transformation rules 1–9 according to the hypothesis of Theorem 4.1. For reasons of simplicity we assume that each definition clause is used for folding, and thus, by Condition 1 of Theorem 4.1, it is unfolded during the construction of $P_0, \ldots, P_n$. We can rearrange the sequence $P_0, \ldots, P_n$ into a new sequence $P_0, \ldots, P_0 \cup \mathit{Defs}_n, \ldots, P_j, \ldots, P_l, \ldots, P_n$ such that: (1) $P_0, \ldots, P_0 \cup \mathit{Defs}_n$ is constructed by applications of the definition introduction rule, (2) $P_0 \cup \mathit{Defs}_n, \ldots, P_j$ is constructed by unfolding every clause in Defs_n, (3) $P_j, \ldots, P_l$ is constructed by applications of rules 3–9, and (4) either $l = n$ or $l = n - 1$ and P_n is derived from P_{n-1} by an application of the definition elimination rule w.r.t. predicate p.

Throughout the rest of this section we will refer to the transformation sequence $P_0, \ldots, P_0 \cup \mathit{Defs}_n, \ldots, P_j, \ldots, P_n$ constructed as indicated above. We also assume that M is a mode for $P_0 \cup \mathit{Defs}_n$ such that: (i) $P_0 \cup \mathit{Defs}_n$ is safe w.r.t. M, (ii) $P_0 \cup \mathit{Defs}_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are all safe w.r.t. M.

Thus, by Propositions 7.1 and 7.5, for $k = 0, \ldots, n$, program P_k is safe and satisfies M.

Lemma 7.13. Let us consider the transformation sequence $P_0, \ldots, P_0 \cup \mathit{Defs}_n, \ldots, P_j$ constructed as indicated above. Then the following properties hold.
(i) For all clauses $\mathit{newp}(X_1, \ldots, X_h) \leftarrow \mathit{Body}$ in Defs_n, for all substitutions ϑ, and for all goals G_1, G_2, such that all derivations from $(G_1,\ \mathit{Body}\vartheta,\ G_2)$ using P_j are consistent with M, we have that:
(i.1) $(G_1,\ \mathit{Body}\,\vartheta,\ G_2) \succeq_{P_j} (G_1,\ \mathit{newp}(X_1, \ldots, X_h)\vartheta,\ G_2)$;
(i.2) all derivations starting from $(G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta,\ G_2)$ using P_j are consistent with M;
(ii) for all non-basic atoms A satisfying M, if A succeeds in $P_0 \cup \mathit{Defs}_n$ then A succeeds in P_j.

Notice that, by Point (i.1), if $(G_1,\ \mathit{Body}\,\vartheta,\ G_2)$ succeeds in P_j then $(G_1,\ \mathit{newp}(X_1, \ldots, X_h)\vartheta,\ G_2)$ succeeds in P_j.

Proof: By induction on the length of the derivations. □

For the proof of the following Lemma 7.15 we will use the following property.

Lemma 7.14. Let us consider the transformation sequence $P_j, \ldots, P_l$ and the mode M for $P_0 \cup \mathit{Defs}_n$ as indicated above. For $k = j, \ldots, l$ and for all goals G_1 and G_2 such that there exists a derivation $G_1 \mapsto_{P_k} \ldots \mapsto_{P_k} G_2$, if all derivations from G_1 using P_j are consistent with M then all derivations from G_2 using P_j are consistent with M.

Proof: The proof proceeds by induction on k and on the length of the derivation $G_1 \mapsto_{P_k} \ldots \mapsto_{P_k} G_2$. We omit the details. □

Lemma 7.15. Let us consider the transformation sequence $P_j, \ldots, P_l$ and the mode M for $P_0 \cup \mathit{Defs}_n$ as indicated above. Let G be a goal such that (i) no disequation occurs in G and (ii) all derivations from G using P_j are consistent with M. For $k = j, \ldots, l$, if G has a successful derivation in P_j, then G has a successful derivation in P_k which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Proof: Let us consider the following ordering on goals:

$G_1 \rhd G_2$ iff *either* $G_1 \succ_{P_j} G_2$ *or* $G_1 \succeq_{P_j} G_2$ and $\nu(P_j, G_1) > \nu(P_j, G_2)$.

$\rhd$ is a well-founded order.

The proof proceeds by induction on k.

Base Case. The case $k = j$ follows from Lemma 7.9.

Step Case. For $k \geq j$ we assume the following:

Inductive Hypothesis (I1). For each goal G' such that no disequation occurs in G' and all derivations from G' using P_j are consistent with M, if G' has a successful derivation in P_j, then G' has a successful derivation in P_k which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Let us now consider a goal G of the form (A_0, G_0) such that no disequation occurs in (A_0, G_0) and all derivations from (A_0, G_0) using P_j are consistent with M. Let us assume that there exists a derivation of the form:

$\delta: \quad (A_0, G_0) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

We wish to show that there exists a derivation of the form:

$\delta': \quad (A_0, G_0) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. We prove the existence of such a derivation δ' by induction on the well-founded order $\rhd$.

We assume the following:

Inductive Hypothesis (I2). For each goal $\hat{G}$ such that no disequation occurs in $\hat{G}$ and all derivations from $\hat{G}$ using P_j are consistent with M and $(A_0, G_0) \rhd \hat{G}$, if there exists a derivation of the form:

$\hat{G} \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$, then there exists a derivation of the form:

$\hat{G} \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Now we proceed by cases.

Case 1: A_0 is the equation $t_1 = t_2$. By Point (1) of the operational semantics of Section 3.3, the derivation δ is of the form:

$(t_1 = t_2, G_0) \mapsto_{P_k} G_0\ mgu(t_1, t_2) \mapsto_{P_k} \ldots \mapsto_{P_k} true$

Let us consider the derivation:

$G_0\ mgu(t_1, t_2) \mapsto_{P_k} \ldots \mapsto_{P_k} true$

By Proposition 7.8, we have that both $(t_1 = t_2, G_0)$ and $G_0\ mgu(t_1, t_2)$ succeed in P_j. Moreover, by Point (1) of the operational semantics $\nu(P_j,\ (t_1 = t_2, G_0)) > \nu(P_j,\ G_0\ mgu(t_1, t_2))$. Thus, $(t_1 = t_2, G_0) \rhd G_0\ mgu(t_1, t_2)$ and, by the inductive hypothesis (I2), there exists a successful derivation of the form:

$G_0\ mgu(t_1, t_2) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} true$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Since $(t_1 = t_2, G_0) \succeq_{P_j} G_0\ mgu(t_1, t_2)$, the following derivation:

$(t_1 = t_2, G_0) \mapsto_{P_{k+1}} G_0\ mgu(t_1, t_2) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} true$

is quasi-decreasing w.r.t. $\succ_{P_j}$.

Case 2: A_0 is a non-basic atom which satisfies M (otherwise there is no derivation starting from (A_0, G_0) which is consistent with M). By Point (3) of the operational semantics, in P_k there exists a renamed apart clause C, such that the derivation δ is of the form:

$(A_0, G_0) \mapsto_{P_k} (bd(C), G_0)mgu(A_0, hd(C)) \mapsto_{P_k} \ldots \mapsto_{P_k} true$

By Proposition 4.2 we may assume that clause C is of the form $H \leftarrow Diseqs, B$, where $Diseqs$ is a conjunction of disequations and B is a goal without occurrences of disequations. Thus, $Diseqs\ mgu(A_0, H)$ succeeds and δ is of the form:

$(A_0, G_0) \mapsto_{P_k} (Diseqs, B, G_0)mgu(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} (B, G_0)mgu(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} true$

If $C \in P_{k+1}$ then there exists a derivation:

$(A_0, G_0) \mapsto_{P_{k+1}} (Diseqs, B, G_0)mgu(A_0, H) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} (B, G_0)mgu(A_0, H)$

and the thesis follows from the inductive hypothesis (I2), because we have that $(A_0, G_0) \succ_{P_j} (B, G_0)mgu(A_0, H)$ (recall that δ is quasi-decreasing w.r.t. $\succ_{P_j}$).

Otherwise, if $C \in (P_k - P_{k+1})$, we construct the derivation δ' by considering the following cases, which correspond to the rules applied for deriving P_{k+1} from P_k.

Case 2.1: P_{k+1} is derived by unfolding clause C in P_k w.r.t. a non-basic atom, say A. Thus, clause C is of the form $H \leftarrow Diseqs, G_1, A, G_2$. Let $C_1, \ldots, C_m$, with $m \geq 0$, be the clauses of P_k such that, for $i = 1, \ldots, m$, A is unifiable with the head of C_i. Thus, $P_{k+1} = (P_k - \{C\}) \cup \{D_1, \ldots, D_m\}$, where for $i = 1, \ldots, m$, D_i is the clause $(H \leftarrow Diseqs, G_1, bd(C_i), G_2)\ mgu(A, hd(C_i))$. For reasons of simplicity we assume that for $i = 1, \ldots, m$, no disequation occurs in $bd(C_i)$. In the general case where, for some $i \in \{1, \ldots, m\}$, $bd(C_i)$ has occurrences of disequations, the proof proceeds in a very similar way, by using Proposition 4.2, Lemma 7.12, and the hypothesis that all applications of the unfolding rule are safe (see Definition 3).

The derivation δ is of the form:

$(A_0, G_0) \mapsto_{P_k} (Diseqs, G_1, A, G_2, G_0)mgu(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} true$

From the fact that δ is quasi-decreasing w.r.t. $\succ_{P_j}$, from Point (1) of the operational semantics, and from the definition of $\succ_{P_j}$, we have that:

$$(A_0, G_0) \succ_{P_j} (A_0 = H, \mathit{Diseqs}, G_1, A, G_2, G_0)$$

and the derivation

$$(A_0 = H, \mathit{Diseqs}, G_1, A, G_2, G_0) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$$

is quasi-decreasing w.r.t. $\succ_{P_j}$.

Thus, by Points (1) and (3) of the operational semantics, there exists a clause in P_k, say C_i, such that the derivation

$$(A_0 = H, \mathit{Diseqs}, G_1, A = hd(C_i), bd(C_i), G_2, G_0) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$$

is quasi-decreasing w.r.t. $\succ_{P_j}$. Moreover, we have that:

$$(A_0, G_0) \succ_{P_j} (A_0 = H, \mathit{Diseqs}, G_1, A = hd(C_i), bd(C_i), G_2, G_0).$$

Since all derivations from (A_0, G_0) using P_j are consistent with M, we have that all derivations from $(A_0 = H, \mathit{Diseqs}, G_1)$ using P_j are consistent with M, and therefore, by Lemma 7.4, all derivations from $(A_0 = H, G_1)$ using P_k are consistent with M. Then, since no disequation occurs in G_1, by Lemma 7.10, there exists a derivation

$$(A_0 = H, \mathit{Diseqs}, A = hd(C_i), G_1, bd(C_i), G_2, G_0) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Moreover, we have that:

$$(A_0, G_0) \succ_{P_j} (A_0 = H, \mathit{Diseqs}, A = hd(C_i), G_1, bd(C_i), G_2, G_0).$$

Now, since by Lemma 7.1 all clauses in P_k are safe, we have that:

$$\mathit{vars}(\mathit{Diseqs}\, mgu(A_0, H)) \cap \mathit{vars}((A = hd(C_i))mgu(A_0, H)) = \emptyset$$

and therefore, by using properties of mgu's, there exists a derivation

$$(A = hd(C_i), A_0 = H, \mathit{Diseqs}, G_1, bd(C_i), G_2, G_0) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Let ϑ_i be $mgu(A, hd(C_i))$ and η_i be $mgu(A_0, H\,\vartheta_i))$. By Points (1) and (2) of the operational semantics, we have that $\mathit{Diseqs}\,\vartheta_i\,\eta_i$ succeeds and there exists a derivation of the form

$$((G_1, bd(C_i), G_2)\,\vartheta_i, G_0)\,\eta_i \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$$

Moreover, we have that:

$$(A_0, G_0) \succ_{P_j} ((G_1, bd(C_i), G_2)\,\vartheta_i, G_0)\,\eta_i \qquad (*)$$

and thus, by the inductive hypothesis (I2), there exists a derivation of the form

$$((G_1, bd(C_i), G_2)\,\vartheta_i, G_0)\,\eta_i \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Since $\mathit{Diseqs}\,\vartheta_i\,\eta_i$ succeeds, by using clause D_i in P_{k+1} for the first step, we can construct the following derivation:

$$(A_0, G_0) \mapsto_{P_{k+1}} ((\mathit{Diseqs}, G_1, bd(C_i), G_2)\,\vartheta_i, G_0)\,\eta_i \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$$

which, by property (*), is quasi-decreasing w.r.t. $\succ_{P_j}$.

Case 2.2: P_{k+1} is derived from P_k by a safe application of the folding rule (see Definition 4). In particular, suppose that clause C is one of the following clauses occurring in P_k:

$$\begin{cases} C_1.\ H \leftarrow\ \mathit{Diseqs}, G_1, (A_1, K_1)\vartheta, G_2 \\ \cdots \\ C_m.\ H \leftarrow\ \mathit{Diseqs}, G_1, (A_m, K_m)\vartheta, G_2 \end{cases}$$

where *Diseqs* is a conjunction of disequations and no disequation occurs in (G_1, G_2). We also suppose that the following definition clauses occur in Defs_k:

$$\begin{cases} D_1.\ \mathit{newp}(X_1, \ldots, X_h) \leftarrow A_1, K_1 \\ \cdots \\ D_m.\ \mathit{newp}(X_1, \ldots, X_h) \leftarrow A_m, K_m \end{cases}$$

and we have derived a clause E of the form:

$$E.\ H \leftarrow \mathit{Diseqs}, G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta, G_2$$

where Property Σ of Definition 4 holds, that is, each input variable of $\mathit{newp}(X_1, \ldots, X_h)\vartheta$, is also an input variable of at least one of the non-basic atoms occurring in $(H, G_1, A_1\vartheta, \ldots, A_m\vartheta)$.

Thus, $P_{k+1} = (P_k - \{C_1, \ldots, C_m\}) \cup \{E\}$.

We may assume, without loss of generality, that clause C is C_1, and the derivation δ is of the form:

$$(A_0, G_0) \mapsto_{P_k} (\mathit{Diseqs}, G_1, (A_1, K_1)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$$

Thus, *Diseqs* $\mathit{mgu}(A_0, H)$ succeeds and, since δ is consistent with M, by Lemma 7.7, we have that $(G_1, (A_1, K_1)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H)$ succeeds in P_j.

Moreover, by Lemma 7.14, all derivations from $(G_1, (A_1, K_1)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H)$ using P_j are consistent with M.

Thus, by Lemmata 7.9 and 7.13, all derivations from $(G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H)$ using P_j are consistent with M and there exists a derivation of the form:

$$(G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H) \mapsto_{P_j} \ldots \mapsto_{P_j} \mathit{true}$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

No disequation occurs in $(G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H)$, and thus, by the inductive hypothesis (I1), there exists a derivation of the form:

$$(G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Since δ is quasi-decreasing w.r.t. $\succ_{P_j}$, by Lemma 7.13, we also have that:

$$(A_0, G_0) \rhd (G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H)$$

Thus, by the Inductive hypothesis (I2), there exists a derivation

$$(G_1, \mathit{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0)\mathit{mgu}(A_0, H) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$$

which is quasi decreasing w.r.t. $\succ_{P_j}$.

Since $Diseqs\ mgu(A_0, H)$ succeeds, by using clause $E \in P_{k+1}$, we can construct the following derivation

$$(A_0, G_0) \mapsto_{P_{k+1}} (Diseqs, G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)mgu(A_0, H) \mapsto_{P_{k+1}} \ldots$$
$$\mapsto_{P_{k+1}} true$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Indeed,

$$(A_0, G_0) \succ_{P_j} (Diseqs, G_1, (A_1, K_1)\vartheta, G_2, G_0)mgu(A_0, H)$$
(because δ is quasi-decreasing)
$$\succeq_{P_j} (Diseqs, G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)mgu(A_0, H)$$
(by Lemma 7.13)

Case 2.3: P_{k+1} is derived by deleting clause C from P_k by applying the subsumption rule. Thus, clause C is of the form $(H \leftarrow Diseqs, G_1, G_2)\vartheta$ and there exists a clause D in P_k of the form $H \leftarrow Diseqs, G_1$. By Proposition 4.2 we may assume that no disequation occurs in G_1.

Thus, the derivation (δ) is of the form:

$$(A_0, G_0) \mapsto_{P_k} ((Diseqs, G_1, G_2)\vartheta, G_0)mgu(A_0, H\vartheta) \mapsto_{P_k} \ldots \mapsto_{P_k} true$$

Since all derivations starting from (A_0, G_0) using P_k are consistent with M and, by using clause D, $(A_0, G_0) \mapsto_{P_k} (Diseqs, G_1, G_0)mgu(A_0, H)$, we have that all derivations starting from $(Diseqs, G_1, G_0)mgu(A_0, H)$ using P_k are consistent with M. Moreover, no disequation occurs in G_0 and therefore, by Lemma 7.11, there exists a derivation

$$(A_0, G_0) \mapsto_{P_k} (Diseqs, G_1, G_0)mgu(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} true$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Thus, $(Diseqs\ mgu(A_0, H))$ succeeds and there exists a derivation

$$(G_1, G_0)mgu(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} true$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Since $(A_0, G_0) \rhd (G_1, G_0)mgu(A_0, H)$, by the inductive hypothesis (I2), there exists a derivation

$$(G_1, G_0)mgu(A_0, H) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} true$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Since D belongs to P_{k+1} and $(Diseqs\ mgu(A_0, H))$ succeeds, there exists a derivation

$$(A_0, G_0) \mapsto_{P_{k+1}} (Diseqs, G_1, G_0)mgu(A_0, H) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} true$$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Case 2.4: P_{k+1} is derived from P_k by applying the head generalization rule to clause C. Thus, C is of the form $H\{X/t\} \leftarrow Body$ and $P_{k+1} = (P_k - \{C\}) \cup \{GenC\}$, where clause $GenC$ is of the form $H \leftarrow X{=}t, Body$.

In this case we can show that we can construct the derivation δ' which is quasi-decreasing w.r.t. $\succ_{P_j}$, by using (i) Point (1) of the operational semantics, (ii) the inductive hypothesis (I2) and (iii) the fact that, for all goals of the form $(t_1{=}t_2, G)$, where t_1 and t_2 are unifiable terms, and for all programs P, $\mu(P, (t_1{=}t_2, G)){=}\mu(P, Gmgu(t_1, t_2))$.

Case 2.5: P_{k+1} is derived from P_k by applying the safe case split rule (see Definition 6) to clause C. By Proposition 4.2, we may assume that C is a clause of the form $H \leftarrow \mathit{Diseqs}, B$, where Diseqs is a conjunction of disequations and B is a goal without occurrences of disequations. We also assume that from C we have derived two clauses of the form:

C_1. $(H \leftarrow \mathit{Diseqs}, B)\{X/t\}$

C_2. $H \leftarrow X \neq t, \mathit{Diseqs}, B$

where X is an input variable of H, X does not occur in t, and for all variables $Y \in \mathit{vars}(t)$, *either* Y is an input variable of H *or* Y does not occur in C.

We have that $P_{k+1} = (P_k - \{C\}) \cup \{C_1, C_2\}$. The derivation δ is of the form:

$(A_0, G_0) \mapsto_{P_k} (\mathit{Diseqs}, B, G_0)\, \mathit{mgu}(A_0, H) \mapsto_{P_k} \ldots \mapsto_{P_k} \mathit{true}$

Thus, $(\mathit{Diseqs}\, \mathit{mgu}(A_0, H))$ succeeds and, since δ is quasi-decreasing, we have that $(A_0, G_0) \rhd (B, G_0)\, \mathit{mgu}(A_0, H)$. The goal $(B, G_0)\, \mathit{mgu}(A_0, H)$ has no occurrences of disequations and, by the inductive hypothesis (I2), there exists a derivation

$(B, G_0)\, \mathit{mgu}(A_0, H) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$. Since $(\mathit{Diseqs}\, \mathit{mgu}(A_0, H))$ succeeds, there exists a derivation

$(\mathit{Diseqs}, B, G_0)\, \mathit{mgu}(A_0, H) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Since X is an input variable of H, there exists a binding X/u in $\mathit{mgu}(A_0, H)$ where u is a ground term. We consider the following two cases.

Case A: t and u are unifiable, and thus, u is an instance of t. In this case A_0 and $H\{X/t\}$ are unifiable and, by the hypotheses on X/t, we have that:

$(\mathit{Diseqs}, B, G_0)\, \mathit{mgu}(A_0, H) = ((\mathit{Diseqs}, B)\{X/t\}, G_0)\, \mathit{mgu}(A_0, H\{X/t\})$

Thus, we can construct a derivation of the form:

$(A_0, G_0) \mapsto_{P_{k+1}} ((\mathit{Diseqs}, B)\{X/t\}, G_0)\, \mathit{mgu}(A_0, H\{X/t\}) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$.

Case B: t and u are not unifiable. Thus, $(X \neq t)\mathit{mgu}(A_0, H)$ succeeds and the following derivation is quasi-decreasing w.r.t. $\succ_{P_j}$.

$$\begin{aligned}(A_0, G_0) \;&\mapsto_{P_{k+1}} (X \neq t, \mathit{Diseqs}, B, G_0)\, \mathit{mgu}(A_0, H) \\ &\mapsto_{P_{k+1}} (\mathit{Diseqs}, B, G_0)\, \mathit{mgu}(A_0, H) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}\end{aligned}$$

Case 2.6: P_{k+1} is derived from P_k by applying the equation elimination rule to clause C. In this case the existence of a derivation

$(A_0, G_0) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$, can be proved by using (i) the inductive hypothesis (I2), (ii) Point (1) of the operational semantics, (iii) the fact that P_k is safe and satisfies M, and (iv) Lemma 7.10.

Case 2.7: P_{k+1} is derived from P_k by applying the disequation replacement rule to clause C. In this case the existence of a derivation

$(A_0, G_0) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mathit{true}$

which is quasi-decreasing w.r.t. $\succ_{P_j}$, can be proved by using (i) the inductive hypothesis (I2), (ii) Point (2) of the operational semantics, and (iii) the properties of unification. □

Lemma 7.16. Let us consider the transformation sequence $P_j, \ldots, P_l$ and the mode M for $P_0 \cup \mathit{Defs}_n$ as indicated above. For $k = j, \ldots, l$, for each non-basic atom A which satisfies mode M, if A succeeds in P_j then A succeeds in P_k.

Proof: It follows from Lemma 7.15, because if an atom A satisfies M and succeeds in P_j, then A has a successful derivation in P_j which is consistent with M and quasi-decreasing w.r.t. $\succ_{P_j}$. Indeed, by Proposition 7.5, P_j satisfies M, and thus, all derivations starting from A are consistent with M. □

Lemma 7.17. If program P_n is derived from program P_{n-1} by an application of the definition elimination rule w.r.t. a non-basic predicate p, then for each atom A which has predicate p, if A succeeds in $P_0 \cup \mathit{Defs}_n$ then A succeeds in P_n.

Proof: If A has predicate p then p depends on all clauses which are used for any derivation starting from A. Thus, every derivation from A using $P_0 \cup \mathit{Defs}_n$ is also a derivation using P_n. □

Proposition 7.18 (Completeness) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules 1–9 and let p be a non-basic predicate in P_n. Let M be a mode for $P_0 \cup \mathit{Defs}_n$ such that: (i) $P_0 \cup \mathit{Defs}_n$ is safe w.r.t. M, (ii) $P_0 \cup \mathit{Defs}_n$ satisfies M, and (iii) the applications of the unfolding, folding, head generalization, and case split rules during the construction of $P_0, \ldots, P_n$ are all safe w.r.t. M. Suppose also that:

1. *if* the folding rule is applied for the derivation of a clause C in program P_{k+1} from clauses $C_1, \ldots, C_m$ in program P_k using clauses $D_1, \ldots, D_m$ in Defs_k, with $0 \leq k < n$,
   then for every $i \in \{1, \ldots, m\}$ there exists $j \in \{1, \ldots, n-1\}$ such that D_i occurs in P_j and P_{j+1} is derived from P_j by unfolding D_i.

2. during the transformation sequence $P_0, \ldots, P_n$ the definition elimination rule *either* is never applied *or* it is applied w.r.t. predicate p once only, when deriving P_n from P_{n-1}.

Then for each atom A which has predicate p and satisfies mode M, if A succeeds in $P_0 \cup \mathit{Defs}_n$ then A succeeds in P_n.

Proof: Let us consider a transformation sequence $P_0, \ldots, P_n$ constructed by using the transformation rules 1–9 according to conditions 1 and 2.

As already mentioned, we can rearrange the sequence $P_0, \ldots, P_n$ into a new sequence $P_0, \ldots, P_0 \cup \mathit{Defs}_n, \ldots, P_j, \ldots, P_l, \ldots, P_n$ such that: (1) $P_0, \ldots, P_0 \cup \mathit{Defs}_n$ is constructed by applications of the definition introduction rule, (2) $P_0 \cup \mathit{Defs}_n, \ldots, P_j$ is constructed by unfolding every clause in Defs_n, (3) $P_j, \ldots, P_l$ is constructed by applications of rules 3–9, and (4) either $l = n$ or $l = n - 1$ and P_n is derived from P_{n-1} by an application of the definition elimination rule w.r.t. predicate p.

Thus, Proposition 7.18 follows from Lemmata 7.13, 7.16, and 7.17. □

70.

References

[1] K. R. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–576. Elsevier, 1990.

[2] R. S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. *Science of Computer Programming*, 12:93–104, 1989.

[3] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.

[4] O. Danvy, R. Glück, and P. Thiemann, editors. *Partial Evaluation. International Seminar, Dagstuhl Castle, Germany, February 1996*, volume 1110 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[5] D. De Schreye, R. Glück, J. Jørgensen, M. Leuschel, B. Martens, and M. H. Sørensen. Conjunctive partial deduction: Foundations, control, algorithms, and experiments. *Journal of Logic Programming*, 41(2–3):231–277, 1999.

[6] S. K. Debray and D. S. Warren. Automatic mode inference for logic programs. *Journal of Logic Programming*, 5:207–229, 1988.

[7] Y. Deville. *Logic Programming: Systematic Program Development*. Addison-Wesley, 1990.

[8] H. Fujita. An algorithm for partial evaluation with constraints. Technical Memorandum TM-0367, ICOT, Tokyo, Japan, 1987.

[9] Y. Futamura, K. Nogi, and A. Takano. Essence of generalized partial computation. *Theoretical Computer Science*, 90:61–79, 1991.

[10] J. P. Gallagher. Tutorial on specialization of logic programs. In *Proceedings of ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '93, Copenhagen, Denmark*, pages 88–98. ACM Press, 1993.

[11] M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In M. Hermenegildo and J. Penjam, editors, *Proceedings Sixth International Symposium on Programming Language Implementation and Logic Programming (PLILP '94)*, Lecture Notes in Computer Science 844, pages 340–354. Springer-Verlag, 1994.

[12] R. Glück and A.V. Klimov. Occam's razor in metacomputation: the notion of a perfect process tree. In P. Cousot, M. Falaschi, G. Filé, and A. Rauzy, editors, *3rd International Workshop on Static Analysis, Padova, Italy, September 1993*, Lecture Notes in Computer Science 724, pages 112–123. Springer-Verlag, 1993.

[13] R. Glück and M. H. Sørensen. A roadmap to metacomputation by supercompilation. In O. Danvy, R. Gück, and P. Thiemann, editors, *Partial Evaluation*, Lecture Notes in Computer Science 1110, pages 137–160. Springer, 1996.

[14] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF*. Lecture Notes in Computer Science 78. Springer-Verlag, 1979.

[15] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.

[16] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[17] M. Leuschel. On the power of homeomorphic embedding for online termination. In G. Levi, editor, *Proceedings of the Fifth Static Analysis Symposium, SAS '98, Pisa, Italy*, Lecture Notes in Computer Science 1503, pages 230–245. Springer-Verlag, 1998.

[18] M. Leuschel and B. Martens. Global control for partial deduction through characteristic atoms and global trees. Report CW 220, K.U. Leuven, Belgium, 1995.

[19] M. Leuschel, B. Martens, and D. De Schreye. Controlling generalization and polyvariance in partial deduction of normal logic programs. *ACM Transactions on Programming Languages and Systems*, 20(1):208–258, 1998.

[20] M. Leuschel, B. Martens, and D. de Schreye. Some achievements and prospects in partial deduction. *ACM Computing Surveys*, 30 (Electronic Section)(3es):4, 1998.

[21] Y. A. Liu. Efficiency by incrementalization: An introduction. *Higher-Order and Symbolic Computation*, 13(4):289–313, 2000.

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

[23] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.

[24] B. Martens, D. De Schreye, and T. Horváth. Sound and complete partial deduction with unfolding based on well-founded measures. *Theoretical Computer Science*, 122:97–117, 1994.

[25] R. Paige and S. Koenig. Finite differencing of computable expressions. *ACM Transactions on Programming Languages and Systems*, 4(3):402–454, 1982.

[26] A. Pettorossi. Transformation of programs and use of tupling strategy. In *Proceedings Informatica 77, Bled, Yugoslavia*, pages 1–6, 1977.

[27] A. Pettorossi, M. Proietti, and S. Renault. Enhancing partial deduction via unfold/fold rules. In J. Gallagher, editor, *Logic Program Synthesis and Transformation, Proceedings of LoPSTr '96, Stockholm, Sweden, August 28-30, 1996*, Lecture Notes in Computer Science 1207, pages 147–168. Springer-Verlag, 1997.

[28] A. Pettorossi, M. Proietti, and S. Renault. Reducing nondeterminism while specializing logic programs. In *Proc. 24-th ACM Symposium on Principles of Programming Languages, Paris, France*, pages 414–427. ACM Press, 1997.

[29] S. Prestwich. Online partial deduction of large programs. In *ACM Sigplan Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM '93, Copenhagen, Denmark*, pages 111–118. ACM Press, 1993.

[30] S. Prestwich. An unfold rule for full Prolog. In K.-K. Lau and T. Clement, editors, *Logic Program Synthesis and Transformation, Proceedings LoPSTr '92, Manchester, U.K.*, Workshops in Computing, pages 199–213. Springer-Verlag, 1993.

[31] M. Proietti and A. Pettorossi. Unfolding-definition-folding, in this order, for avoiding unnecessary variables in logic programs. *Theoretical Computer Science*, 142(1):89–124, 1995.

[32] S. Renault. A system for transforming logic programs. R 97–04, Department of Computer Science, University of Rome Tor Vergata, Rome, Italy, 1997.

[33] A. Roychoudhury, K. Narayan Kumar, C.R. Ramakrishnan, and I.V. Ramakrishnan. A parameterized unfold/fold transformation framework for definite logic programs. In *Proceedings of Principles and Practice of Declarative Programming (PPDP)*, Lecture Notes in Computer Science 1702, pages 396–413. Springer-Verlag, 1999.

[34] D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12:7–51, 1993.

[35] D. A. Smith. Partial evaluation of pattern matching in constraint logic programming languages. In *Proceedings ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '91, New Haven, CT, USA*, SIGPLAN Notices, 26, 9, pages 62–71. ACM Press, 1991.

[36] M. H. Sørensen, R. Glück, and N. D. Jones. Towards unifying partial evaluation, deforestation, supercompilation, and GPC. In D. Sannella, editor, *Fifth European Symposium on Programming Languages and Systems, ESOP '94*, Lecture Notes in Computer Science 788, pages 485–500. Springer-Verlag, 1994.

[37] H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, editor, *Proceedings of the Second International Conference on Logic Programming, Uppsala, Sweden*, pages 127–138. Uppsala University, 1984.

[38] V. F. Turchin. The concept of a supercompiler. *ACM TOPLAS*, 8(3):292–325, 1986.

[39] P. L. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.

[40] D. H. D. Warren. Implementing Prolog – compiling predicate logic programs. Research Report 39 & 40, Department of Artificial Intelligence, University of Edinburgh, 1977.