Derivation of Efficient Logic Programs by Specialization and Reduction of Nondeterminism*

Alberto Pettorossi  
DISP, University of Roma Tor Vergata, Roma, Italy  
adp@iasi.rm.cnr.it

Maurizio Proietti  
IASI-CNR, Roma, Italy  
proietti@iasi.rm.cnr.it

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

February 1, 2008

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 matching programs and parsers for regular languages. The derivations we have performed could not have been done by previously known partial evaluation techniques.

1 Introduction
The goal of program specialization [21] is the adaptation of a generic program to a specific context of use. Partial evaluation [7, 21] 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 [13, 26, 29], where it is also called partial deduction. In this paper we follow a rule-based approach to the specialization of logic programs [4, 36, 37, 41]. In particular, we consider definite logic programs [28] and we propose new program specialization techniques based on unfold/fold transformation rules [6, 46]. 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 derived, specialized program, and every program of the sequence is obtained from the previous one by applying a transformation rule.

* 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.
As shown in [36, 41], partial deduction can be viewed as a particular rule-based program transformation technique using the definition, unfolding, and folding rules [46] 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 [16], partial deduction 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 3 we will see that the use of these restricted transformation rules makes it easier to automate the partial deduction process, but it may limit the program improvements which can be achieved during program specialization. In particular, when we perform partial deduction 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 deduction is demonstrated in Section 3.3 where we revisit the familiar problem of looking for occurrences of a pattern in a string. It has been shown in [11, 13, 15] that by partial deduction of a string matching program, we may derive a deterministic finite automaton (DFA, for short), similarly to what is done by the Knuth-Morris-Pratt algorithm [22]. However, in [11, 13, 15] the string matching program to which partial deduction is applied, is deterministic. We show that by applying partial deduction to 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 DFA.

**Conjunctive partial deduction** [8] is a program specialization technique which extends partial deduction by allowing the specialization of logic programs w.r.t. conjunctions of atoms, instead of a single atom. Conjunctive partial deduction can be realized by the definition, unfolding, and folding rules where each new predicate introduced by the definition rule is defined by precisely one non-recursive clause whose body is a conjunction of atoms (in this sense conjunctive partial deduction is said to be polygenetic).

Conjunctive partial deduction may sometimes reduce nondeterminism. In particular, it may transform generate-and-test programs into programs where the generation phase and the test phase are interleaved. However, as shown in Section 3.3, conjunctive partial deduction is not capable to derive from the nondeterministic version of the matching program a new program which corresponds to a DFA.

In our paper, we propose a specialization technique which enhances both partial deduction and conjunctive partial deduction by making use of more powerful transformation rules. In particular, in Section 4 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 conjunctions of atoms, and we allow folding steps which use these predicate definitions consisting of several clauses. We also consider the following extra rules: head generalization, case split, equation elimination, and disequation replacement. These rules may introduce, replace, and eliminate equations and negated equations between terms.

Similarly to [14, 46, 40], our extended set of program transformation rules preserves the least Herbrand model semantics. For the logic language with equations and negated equations considered in this paper, we adopt the usual Prolog operational semantics with the left-to-right selection rule, where equations are evaluated by using unification. Unfortunately, the unrestricted use of the extended set of transformation rules may not preserve the Prolog operational semantics. To overcome this problem, we consider: (i) the class of safe programs and (ii) suitably restricted transformation rules, called safe transformation rules. Through some examples we show that the class of safe programs and the safe transformation rules are general enough to allow significant program specializations.

Our notions of safe programs and transformation rules, and also the notion of determinism are based on the modes which are associated with predicate calls [32, 19]. We describe these notions in Section 5 where we also prove that the application of safe transformation rules preserve the operational semantics of safe programs.

Then, in Section 6 we introduce a strategy, called Determinization, for applying our safe trans-
2 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. The declarative semantics we consider, is a straightforward extension of the usual least Herbrand model of definite logic programs. The operational semantics essentially is SLD-resolution as implemented by most Prolog systems: atoms are selected from left to right, and equations are evaluated by using unification. This operational semantics is sound w.r.t. the declarative semantics (see Theorem 2 below). However, since non-ground disequations can be selected, a goal evaluated according to our operational semantics can fail, even if it is true according to the declarative semantics. In this sense, the operational semantics is not complete w.r.t. the declarative semantics.

For the notions of substitution, composition of substitutions, identity substitution, domain of a substitution, restriction of a substitution, instance, most general unifier (abbreviated as mgu), ground expression, ground substitution, renaming substitution, variant, and for other notions not defined here, we refer to [28].

2.1 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 $true, =, \neq$ are said to be basic, and the other predicate symbols are said to be non-basic. Predicate symbols will also be called predicates, for short.

Now we introduce 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**: $B ::= true \mid t_1 = t_2 \mid t_1 \neq t_2$
- **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 $(G, true)$.

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

- $C ::= A \leftarrow G$

Given a clause $C$ of the form: $A \leftarrow G$, the non-basic atom $A$ 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 $A \leftarrow G$ where $G$ is a basic goal, is called a unit clause. We write a unit clause of the form: $A \leftarrow true$ also as: $A \leftarrow$. We say that $C$ is a clause for a predicate $p$ iff $C$ is a clause of the form $p(\ldots) \leftarrow G$. 
In what follows we will feel free to use different meta-variables to denote our syntactic expressions, and in particular, we will also denote non-basic atoms by $H, H_1, \ldots$, and goals by $K, K_1, Body, Body_1, \ldots$

Given a program $P$, we consider the relation $\delta_P$ over pairs of predicates such that $\delta_P(p, q)$ holds iff there exists in $P$ a clause for $p$ whose body contains an occurrence of $q$. Let $\delta_P^+$ be the transitive closure of $\delta_P$. We say that $p$ depends on $q$ in $P$ iff $\delta_P^+(p, q)$ holds. We say that a predicate $p$ depends on a clause $C$ in a program $P$ iff either $C$ is a clause for $p$ or $C$ is a clause for a predicate $q$ and $p$ depends on $q$ in $P$.

Terms, atoms, goals, clauses, and programs are collectively called expressions, ranged over by $e, e_1, e_2, \ldots$. By $\text{vars}(e)$ we denote the set of variables occurring in an expression $e$. We say that $X$ is a local variable of a goal $G$ in a clause $C : H \leftarrow G_1, G, G_2$ iff $X \in \text{vars}(G) - \text{vars}(H, G_1, G_2)$.

The application of a renaming substitution to an expression is also called a renaming of variables. A renaming of variables can be applied to a clause whenever needed, because it 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 a renaming of variables, so that each variable of $C'$ is a fresh new variable. (For a formal definition of this concept, see the definition of standardized apart clause in [2, 28])

For any two unifiable terms $t_1$ and $t_2$, there exists at least one mgu $\vartheta$ which is relevant (that is, each variable occurring in $\vartheta$ also occurs in $\text{vars}(t_1) \cup \text{vars}(t_2)$) and idempotent (that is, $\vartheta \vartheta = \vartheta$) [2]. Without loss of generality, we assume that all mgu’s considered in this paper are relevant and idempotent.

### 2.2 Declarative Semantics

In this section we extend the definition of least Herbrand model of definite logic programs [2, 28] to logic programs with equations and disequations between terms. We follow the approach usually taken when defining the least $\mathcal{D}$-model of CLP programs (see, for instance, [20]). According to this approach, we consider a class of Herbrand models, called $\mathcal{H}$-models, where the predicates $\text{true}$, $=$, and $\neq$ have a fixed interpretation. In particular, the predicate $=$ is interpreted as the identity relation over the Herbrand universe and the predicate $\neq$ is interpreted as the complement of the identity relation. Then we define the least Herbrand model of a logic program with equations and disequations between terms as the least $\mathcal{H}$-model of the program.

The Herbrand base $\mathcal{HB}$ is the set of all ground non-basic atoms. An $\mathcal{H}$-interpretation is a subset of $\mathcal{HB}$. Given an $\mathcal{H}$-interpretation $I$ and a ground goal, or ground clause, or program $\varphi$, the relation $I \models \varphi$, read as $\varphi$ is true in $I$, is inductively defined as follows (as usual, by $I \not\models \varphi$ we indicate that $I \models \varphi$ does not hold):

(i) $I \models \text{true}$

(ii) for every ground term $t$, $I \models t = t$

(iii) for every pair of distinct ground terms $t_1$ and $t_2$, $I \models t_1 \neq t_2$

(iv) for every non-basic ground atom $A$, $I \models A$ iff $A \in I$

(v) for every pair of ground goals $G_1$ and $G_2$, $I \models G_1, G_2$ iff $I \models G_1$ and $I \models G_2$

(vi) for every ground clause $C$, $I \models C$ iff either $I \models \text{hd}(C)$ or $I \not\models \text{bd}(C)$

(vii) for every program $P$, $I \models P$ iff for every ground instance $C$ of a clause in $P$, $I \models C$.

As a consequence of the above definition, a ground basic goal is true in an $\mathcal{H}$-interpretation iff it is true in all $\mathcal{H}$-interpretations. We say that a ground basic goal holds iff it is true in all $\mathcal{H}$-interpretations.

An $\mathcal{H}$-interpretation $I$ is said to be an $\mathcal{H}$-model of a program $P$ iff $I \models P$. Since the model intersection property holds for $\mathcal{H}$-models, similarly to [20, 28], we can prove the following important result.

**Theorem 1** For any program $P$ there exists an $\mathcal{H}$-model of $P$ which is the least (w.r.t. set inclusion) $\mathcal{H}$-model.

The least Herbrand model of a program $P$ is defined as the least $\mathcal{H}$-model of $P$ and is denoted by $M(P)$. 


We define the operational semantics of our programs by introducing, for each program $P$, a relation $G_1 \xrightarrow{\vartheta} P G_2$, where $G_1$ and $G_2$ are goals and $\vartheta$ is a substitution, defined as follows:

1. $(t_1 = t_2, G) \xrightarrow{\vartheta} P G\vartheta$ iff $t_1$ and $t_2$ are unifiable via an mgu $\vartheta$.
2. $(t_1 \neq t_2, G) \xrightarrow{\vartheta} P G$ iff $t_1$ and $t_2$ are not unifiable and $\varepsilon$ is the identity substitution.
3. $(A, G) \xrightarrow{\vartheta} P (bd(C), G)\vartheta$ iff (i) $A$ is a non-basic atom, (ii) $C$ is a renamed apart clause in $P$, and (iii) $A$ and $hd(C)$ are unifiable via an mgu $\vartheta$.

A sequence $G_0 \xrightarrow{\vartheta_1} \ldots \xrightarrow{\vartheta_n} P G_n$, with $n \geq 0$, is called a derivation using $P$. If $G_n$ is true then the derivation is said to be successful. If there exists a successful derivation $G_0 \xrightarrow{\vartheta_1} \ldots \xrightarrow{\vartheta_n} P$ true and $\vartheta$ is the substitution obtained by restricting the composition $\vartheta_1 \ldots \vartheta_n$ to the variables of $G_0$, then we say that the goal $G_0$ succeeds in $P$ with answer substitution $\vartheta$.

When denoting derivations, we will feel free to omit their associated substitutions. In particular, given two goals $G_1$ and $G_2$, we write $G_1 \leftarrow P G_2$ iff there exists a substitution $\vartheta$ such that $G_1 \xrightarrow{\vartheta} P G_2$.

We say that $G_2$ is derived in one step from $G_1$ (using $P$) iff $G_1 \leftarrow P G_2$ holds. In particular, if $G_2$ is derived in one step from $G_1$ according to Point (3) of the operational semantics by using a clause $C$, then we say that $G_2$ is derived in one step from $G_1$ using $C$. The relation $\leftarrow P$ is the reflexive and transitive closure of $\leftarrow P$. Given two goals $G_1$ and $G_2$ such that $G_1 \leftarrow P G_2$ holds, we say that $G_2$ is derived from $G_1$ (using $P$). We will feel free to omit the reference to program $P$ when it is understood from the context.

The operational semantics presented above 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 $G_1 \leftarrow P G_2$, 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 [28], we have the following relationship between the declarative and the operational semantics.

**Theorem 2** For any program $P$ and ground goal $G$, if $G$ succeeds in $P$ then $M(P) \models G$.

The converse of Theorem 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.

### 2.4 Deterministic Programs

Various notions of determinism have been proposed for logic programs in the literature (see, for instance, [10] [13] [31] [33]). They capture various properties such as: “the program succeeds at most once”, or “the program succeeds exactly once”, or “the program will never backtrack to find alternative solutions”.

Let us now present the definition of deterministic program used in this paper. This definition is based on the operational semantics described in Section 2.3.

We first need the following notation. Given a program $P$, a clause $C \in P$, and two goals $(A_0, G_0)$ and $(A_n, G_n)$, where $A_0$ is a non-basic atom, we write $(A_0, G_0) \Rightarrow_C (A_n, G_n)$ iff there exists a derivation $(A_0, G_0) \leftarrow P \ldots \leftarrow P (A_n, G_n)$, such that: (i) $n > 0$, (ii) $(A_1, G_1)$ is derived in one step from $(A_0, G_0)$ using $C$, (iii) for $i = 1, \ldots , n - 1$, $A_i$ is a basic atom, and (iv) either $A_n$ is a non-basic atom or $(A_n, G_n)$ is the basic atom true. We write $G_0 \Rightarrow_C^* P G_n$ iff there exists clauses $C_1, \ldots , C_n$ in $P$ such that $G_0 \Rightarrow_C C_1 \ldots \Rightarrow_C C_n G_n$.

**Definition 1 (Determinism)** A program $P$ is deterministic for a non-basic atom $A$ iff for each goal $G$ such that $A \Rightarrow_P G$, there exists at most one clause $C$ such that $G \Rightarrow_C G'$ for some goal $G'$.
We say that a program $P$ is nondeterministic for a non-basic atom $A$ if it is not the case that $P$ is deterministic for $A$, that is, there exists a goal $G$ derivable from $A$, and there exist at least two goals $G_1$ and $G_2$, and two distinct clauses $C_1$ and $C_2$ in $P$, such that $G \Rightarrow C_1, G_1$ and $G \Rightarrow C_2, G_2$.

According to Definition 11, the following program is deterministic for any atom of the form non_zero($X_s, Y_s$) where $X_s$ is a ground list.

1. non_zero([], []) ←
2. non_zero([0,$X_s$], $Y_s$) ← non_zero($X_s, Y_s$)
3. non_zero([$X|X_s$], [$X|Y_s$]) ← $X \neq 0$, non_zero($X_s, Y_s$)

Notice that the above definition of a deterministic program for a non-basic atom $A$ allows some search during the construction of a derivation starting from $A$. Indeed, there may be a goal $G$ derived from $A$ such that from $G$ we can derive in one step two or more new goals using distinct clauses. However, if the program is deterministic for $A$, after evaluating the basic atoms occurring at leftmost positions in these new goals, at most one derivation can be continued and at most one successful derivation can be constructed. For instance, from the goal non_zero([0,0,1], $Y_s$) we can derive in one step two distinct goals: (i) non_zero([0,1], $Y_s$) (using clause 2), and (ii) $0 \neq 0$, non_zero([0,1], $Y_s$') (using clause 3). However, there exists only one clause $C$ (that is, clause 2) such that non_zero([0,0,1], $Y_s$) $\Rightarrow C$ $G'$ for some goal $G'$ (that is, non_zero([0,1], $Y_s$')).

3 Partial Deduction via Unfold/Fold Transformations

In this section we recall the rule-based approach to partial deduction. We also point out some limitations of partial deduction and conjunctive partial deduction. These limitations motivate the introduction of the new, enhanced rules and strategies for program specialization presented in Sections 4, 5 and 6.

3.1 Transformation Rules and Strategies for Partial Deduction

In the rule-based approach, partial deduction can be viewed as the construction of a sequence $P_0, \ldots, P_n$ of programs, called a transformation sequence, where $P_0$ is the initial program to be specialized, $P_n$ is the final, specialized program, and for $k = 0, \ldots, n - 1$, program $P_{k+1}$ is derived from program $P_k$ by by applying one of the following transformation rules PD1–PD4.

**Rule PD1 (Atomic Definition Introduction)** We introduce a clause $D$, called atomic definition clause, of the form

\[
\text{newp}(X_1, \ldots, X_h) \leftarrow A
\]

where (i) newp is a non-basic predicate symbol not occurring in $P_0, \ldots, P_k$, (ii) $A$ is a non-basic atom whose predicate occurs in program $P_0$, and (iii) $\{X_1, \ldots, X_h\} = \text{vars}(A)$.

Program $P_{k+1}$ is the program $P_k \cup \{D\}$.

We denote by $\text{Defs}_k$ the set of atomic definition clauses which have been introduced by the definition introduction rule during the construction of the transformation sequence $P_0, \ldots, P_k$. Thus, in particular, we have that $\text{Defs}_0 = \emptyset$.

**Rule PD2 (Definition Elimination).** Let $p$ be a predicate symbol. By definition elimination w.r.t. $p$ we derive the program $P_{k+1} = \{C \in P_k \mid p \text{ depends on } C\}$.

**Rule PD3 (Unfolding).** Let $C$ be a renamed apart clause of $P_k$ 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 $i = 1, \ldots, m$, $A$ is unifiable with the head of $C_i$ via the mgu $\vartheta_i$. By unfolding $C$ w.r.t. $A$, for $i = 1, \ldots, m$, we derive the clause $D_i : (H \leftarrow G_1, \text{bd}(C_i), G_2)\vartheta_i$.

Program $P_{k+1}$ is the program $(P_k - \{C\}) \cup \{D_1, \ldots, D_m\}$.

**Rule PD4 (Atomic Folding).** Let $C$ be a renamed apart clause of $P_k$ of the form: $H \leftarrow G_1, A\vartheta, G_2$, where: (i) $A$ is a non-basic atom, and (ii) $\vartheta$ is a substitution, and let $D$ be an atomic definition clause in $\text{Defs}_k$ of the form: $N \leftarrow A$. By folding $C$ w.r.t. $A\vartheta$ using $D$ we derive the non-basic atom $N\vartheta$ and we derive the clause $E : H \leftarrow G_1, N\vartheta, G_2$. 

6
Partial Deduction Strategy

**Input:** A program $P$ and a non-basic atom $p(t_1, \ldots, t_h)$ w.r.t. which we want to specialize $P$.

**Output:** A program $P_{pd}$ and a non-basic atom $p_{pd}(X_1, \ldots, X_r)$, such that: (i) $\{X_1, \ldots, X_r\} = \text{vars}(p(t_1, \ldots, t_h))$, and (ii) for every ground substitution $\vartheta = \{X_1/u_1, \ldots, X_r/u_r\}$,

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

**Initialize:** Let $S$ be the clause $p_{pd}(X_1, \ldots, X_r) \leftarrow p(t_1, \ldots, t_h)$. Let $\text{Ancestors}(S)$ be the empty sequence of clauses.

**Trans:** $P := P$; $\text{Defs} := \{S\}$; $\text{Cls} := \{S\}$;

**while** $\text{Cls} \neq \emptyset$ **do**

(1) **Unfold:**

**while** there exists a clause $C \in \text{Cls}$ with $\text{Select}(\text{Ancestors}(C), C) \neq \text{halt}$ **do**

Let $\text{Unf}(C) = \{E \mid E \text{ is derived by unfolding } C \text{ w.r.t. } \text{Select}(\text{Ancestors}(C), C)\}$.

$\text{Cls} := (\text{Cls} - \{C\}) \cup \text{Unf}(C)$;

for each $E \in \text{Unf}(C)$ let $\text{Ancestors}(E)$ be the sequence $\text{Ancestors}(C)$ followed by $C$

**end-while**;

(2) **Define-Fold:**

$\text{NewDefs} := \emptyset$;

**while** there exists a clause $C \in \text{Cls}$ and there exists a non-basic atom $A \in \text{bd}(C)$ which has not been derived by folding **do**

Let $G$ be the atomic definition clause $\text{Gen}(\text{Defs}, A)$ and $F$ be the clause derived by folding $C$ w.r.t. $A$ using $G$.

**end-while**;
\[
Cla := (Cla - \{C\}) \cup \{F\};
\]
\[
\text{if } G \not\in Defs \text{ then } (Defs := Defs \cup \{G\}; \ NewDef := NewDef \cup \{G\})
\]
end-while;

TransfP := TransfP \cup Cla; \ Cla := NewDef
end-while;

We derive the final program \(P_{pd}\) by applying the definition elimination rule and keeping only the clauses of \(TransfP\) on which \(p_{pd}\) depends.

A given unfolding selection function \(Select\) is said to be \textit{progressive} iff for the empty sequence () of clauses and for any clause \(C\) whose body contains at least one non-basic atom, we have that \(Select((), C) \neq halt\).

We have the following correctness result which is a straightforward corollary of Theorem 5 of Section 4.2.

**Theorem 3 (Correctness of Partial Deduction w.r.t. the Declarative Semantics)**

Let \(Select\) be a progressive unfolding selection function. Given a program \(P\) and a non-basic atom \(p(t_1, \ldots, t_h)\), if the partial deduction strategy using \(Select\) terminates with output program \(P_{pd}\) and output atom \(p_{pd}(X_1, \ldots, X_r)\), then for every ground substitution \(\vartheta = \{X_1/u_1, \ldots, X_r/u_r\}\),

\[
M(P) \models p(t_1, \ldots, t_h)\vartheta \text{ iff } M(P_{pd}) \models p_{pd}(X_1, \ldots, X_r)\vartheta.
\]

We say that an unfolding selection function \(Select\) is \textit{halting} iff for any infinite sequence \(C_1, C_2, \ldots\) of clauses, there exists \(n \geq 0\) such that \(Select((C_1, C_2, \ldots, C_n), C_{n+1}) = halt\).

We have the following theorem whose proof is similar to the one in [25].

**Theorem 4 (Termination of Partial Deduction)**

Let \(Select\) be a halting unfolding selection function and \(Gen\) be a stabilizing generalization function. Then for any input program \(P\) and non-basic atom \(p(t_1, \ldots, t_h)\), the partial deduction strategy using \(Select\) and \(Gen\) terminates.

The following example shows that the unfolding rule (and thus, the partial deduction strategy) is not correct w.r.t. the operational semantics.

**Example 1** Let us consider the following program \(P_1\):

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

By unfolding clause 1 w.r.t. \(q(X)\) we derive the following program \(P_2\):

3. \(p \leftarrow b \neq a\)
4. \(q(b) \leftarrow\)

We have that the goal \(p\) does not succeed in \(P_1\), while it succeeds in \(P_2\).

We will address this correctness issue in detail in Section 5, where we will present a set of transformation rules which are correct w.r.t. the operational semantics for the class of \textit{safe} programs (see Theorem 5).
3.2 An Example of Partial Deduction: String Matching

In this section we illustrate the partial deduction strategy by means of a well-known program specialization example which consists in specializing a general string matching program w.r.t. a given pattern (see \[11, 13, 44\] for a similar example). 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 program, called \( \text{Match} \), for searching a pattern \( P \) in a string \( S \) in \( \{a, b\}^* \). Strings in \( \{a, b\}^* \) are denoted by lists of \( a \)'s and \( b \)'s. This program is deterministic for atoms of the form \( \text{match}(P, S) \), where \( P \) and \( S \) are ground lists.

The program \( \text{Match} \) is the initial, deterministic program:

\[
\begin{align*}
\text{match}(P, S) & \leftarrow \text{match1}(P, S, P, S) \\
\text{match1}([], S, Y, Z) & \leftarrow \\
\text{match1}(\left[C|P\right], [C|S], Y, Z) & \leftarrow \text{match1}(P, S, Y, Z) \\
\text{match1}(\left[a|P\right], [b|S], Y, \left[C|Z\right]) & \leftarrow \text{match1}(Y, Z, Y, Z) \\
\text{match1}(\left[b|P\right], [a|S], Y, \left[C|Z\right]) & \leftarrow \text{match1}(Y, Z, Y, Z)
\end{align*}
\]

Let us assume that we want to specialize this program \( \text{Match} \) w.r.t. the goal \( \text{match}([a, a, b], S) \), 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 deduction strategy using the following unfolding selection function \( \text{DetU} \) and generalization function \( \text{Variant} \).

(1) The function \( \text{DetU} : \text{Clauses}^* \times \text{Clauses} \rightarrow \text{NBAtons} \cup \{\text{halt}\} \) is defined as follows:

(i) \( \text{DetU}([], C) = A \) if \( A \) is the leftmost non-basic atom in the body of clause \( C \),

(ii) \( \text{DetU}([C_1, C_2, \ldots, C_n], C) = A \) if \( n \geq 1 \) and \( A \) is the leftmost non-basic 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) \( \text{DetU}([C_1, C_2, \ldots, C_n], C) = \text{halt} \) if there exists no non-basic atom in the body of \( C \) which is unifiable with at most one clause head in the program to be partially evaluated.

(2) The function \( \text{Variant} : \mathcal{P}(\text{Clauses}) \times \text{NBAtons} \rightarrow \text{Clauses} \) is defined as follows:

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

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

The function \( \text{DetU} \) corresponds to the determinate unfolding rule considered in \[13\]. We have that \( \text{DetU} \) is not halting and \( \text{Variant} \) is not stabilizing. Nevertheless, in our example, as the reader may verify, the partial deduction strategy using \( \text{DetU} \) and \( \text{Variant} \) terminates and generates the following specialized program:

The program \( \text{Match}_{\text{pd}} \) is deterministic for atoms of the form \( \text{match}_{\text{pd}}(S) \), where \( S \) is a ground list, and it corresponds to a DFA 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) by clause 6 we have that \( \text{new1} \) is the initial state for goals of the form \( \text{match}_{\text{pd}}(w) \), where \( w \) is any ground list representing
a word in \( \{a, b\}^* \), and (v) clause 11 corresponds to a transition, labeled by \( b \), to an unnamed final state where any remaining portion of the input word is accepted.

Thus, via partial deduction we may derive a DFA from a deterministic string matching program, and the derived program corresponds to the Knuth-Morris-Pratt string matching algorithm \[22\].

### 3.3 Some Limitations of Partial Deduction

The fact that the partial deduction strategy derives a DFA is a consequence of the fact that the initial string matching program \textit{Match} is rather sophisticated and, indeed, the correctness proof of the program \textit{Match} is not straightforward. Actually, the partial deduction strategy does not derive a DFA if we consider, instead of the program \textit{Match}, the following naive initial program for string matching:

| Program \textit{Naive\_Match} (initial, nondeterministic) |
|-----------------------------------------------------------|
| 1. \( \text{naive\_match}(P, S) \leftarrow \text{append}(X, R, S), \text{append}(L, P, X) \) |
| 2. \( \text{append}([], Y, Y) \leftarrow \) |
| 3. \( \text{append}([A|X], Y, [A|Z]) \leftarrow \text{append}(X, Y, Z) \) |

This program is nondeterministic for atoms of the form \( \text{naive\_match}(P, S) \), where \( P \) and \( S \) are ground lists. The correctness of this naive program is straightforward because for a given pattern \( P \) and a string \( S \), \textit{Naive\_Match} tests whether or not \( P \) 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 the partial deduction strategy does not derive a DFA when starting from the program \textit{Naive\_Match}. Indeed, if we specialize \textit{Naive\_Match} w.r.t. the goal \( \text{naive\_match}([a, a, b], S) \) by applying the partial deduction strategy using the unfolding selection function \textit{DetU} and the generalization function \textit{Variant}, then we derive the following program \textit{Naive\_Match\_pd} which does not correspond to a DFA and it is nondeterministic:

| Program \textit{Naive\_Match\_pd} (specialized by partial deduction, nondeterministic) |
|-------------------------------------------------------------|
| 4. \( \text{naive\_match\_pd}(S) \leftarrow \text{new}(X, R, S), \text{new}(L, X) \) |
| 5. \( \text{new}([], Y, Y) \leftarrow \) |
| 6. \( \text{new}([A|X], Y, [A|Z]) \leftarrow \text{new}(X, Y, Z) \) |
| 7. \( \text{new}([], [a, a, b]) \leftarrow \) |
| 8. \( \text{new}([A|X], [A|Z]) \leftarrow \text{new}(X, Z) \) |

Indeed, this \textit{Naive\_Match\_pd} program 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 of that position.

From the program \textit{Naive\_Match} we can derive a specialized program which is much more efficient than \textit{Naive\_Match\_pd} by applying conjunctive partial deduction, instead of partial deduction. Conjunctive partial deduction, viewed as a sequence of applications of transformation rules, enhances partial deduction because: (i) one may introduce a definition clause whose body is a conjunction of atoms, instead of one atom only (see rule PD1), and (ii) one may fold a clause w.r.t. a conjunction of atoms in its body, instead of one atom only (see rule PD4). By applying conjunctive partial deduction one may avoid intermediate data structures, such as the list \( X \) constructed by using clause 1 of program \textit{Naive\_Match}. Indeed, by using the ECCE system for conjunctive partial deduction \[24\], from the \textit{Naive\_Match} program we derive the following specialized program:

| Program \textit{Naive\_Match\_cpd} (specialized by conjunctive partial deduction, nondeterministic) |
|-------------------------------------------------------------|
| 9. \( \text{naive\_match\_cpd}([X, Y, Z]|S) \leftarrow \text{new}(X, Y, Z, S) \) |
| 10. \( \text{new}(a, a, b, S) \leftarrow \) |
| 11. \( \text{new}(X, Y, Z, [C]|S) \leftarrow \text{new}(Y, Z, C, S) \) |

This \textit{Naive\_Match\_cpd} program searches for the pattern \([a, a, b]\) in the input string by looking at the first three elements of that string. If they are \( a, a, \) and \( b \), in this order, then the search succeeds,
otherwise the search for the pattern continues in the tail of the string. Although this Naive\_Match\_cpd program is much more efficient than the initial Naive\_Match program, it does not correspond to a DFA because, when searching for the pattern $[a, a, b]$, it looks at a prefix of length 3 of the input string, instead of one symbol only.

The failure of partial deduction and conjunctive partial deduction to derive a DFA when starting from the Naive\_Match program, is due to some limitations which can be overcome by using the enhanced transformation rules 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 rules for partial deduction or conjunctive partial deduction, a new predicate can be defined by introducing one clause only. By folding using definition clauses of the enhanced form, we can derive specialized programs where nondeterminism is reduced and intermediate data structures are avoided. Among our enhanced rules we also have the so called case split rule which, given a clause, produces two mutually exclusive instances of that clause by introducing negated equations. The application of this rule allows subsequent folding steps which reduce nondeterminism.

By applying the enhanced transformation rules according to the Determinization Strategy we will present in Section 6, one can automatically specialize the nondeterministic program Naive\_Match w.r.t. the goal naive\_match\_cut\(([a, a, b], S)\) thereby deriving the following deterministic program (this derivation is not presented here and it is similar to the one presented in Section 7.1):

| Program | Naive\_Match\_cut | (specialized by Determinization, deterministic) |
|---------|------------------|-------------------------------------------------|
| 12.     | naive\_match\_cut\((S) \leftarrow \text{new1}(S)\) |
| 13.     | \text{new1}([a|S]) \leftarrow \text{new2}(S) |
| 14.     | \text{new1}([C|S]) \leftarrow C \neq a, \text{new1}(S) |
| 15.     | \text{new2}([a|S]) \leftarrow \text{new3}(S) |
| 16.     | \text{new2}([C|S]) \leftarrow C \neq a, \text{new1}(S) |
| 17.     | \text{new3}([b|S]) \leftarrow \text{new4}(S) |
| 18.     | \text{new3}([a|S]) \leftarrow \text{new3}(S) |
| 19.     | \text{new3}([C|S]) \leftarrow C \neq b, C \neq a, \text{new1}(S) |
| 20.     | \text{new4}(S) \leftarrow |

The program Naive\_Match\_cut corresponds in a straightforward way to a DFA. Moreover, since the clauses of Naive\_Match\_cut are pairwise mutually exclusive, the disequations in their bodies can be dropped in favor of cuts (or equivalently, if-then-else constructs) as follows:

| Program | Naive\_Match\_cut | (specialized, with cuts) |
|---------|------------------|-------------------------|
| 21.     | naive\_match\_cut\((S) \leftarrow \text{new1}(S)\) |
| 22.     | \text{new1}([a|S]) \leftarrow !, \text{new2}(S) |
| 23.     | \text{new1}([C|S]) \leftarrow \text{new1}(S) |
| 24.     | \text{new2}([a|S]) \leftarrow !, \text{new3}(S) |
| 25.     | \text{new2}([C|S]) \leftarrow \text{new1}(S) |
| 26.     | \text{new3}([b|S]) \leftarrow !, \text{new4}(S) |
| 27.     | \text{new3}([a|S]) \leftarrow !, \text{new3}(S) |
| 28.     | \text{new3}([C|S]) \leftarrow \text{new1}(S) |
| 29.     | \text{new4}(S) \leftarrow |

Computer experiments confirm that the final Naive\_Match\_cut program is indeed more efficient than the Naive\_Match, Naive\_Match\_pd, and Naive\_Match\_cpd programs. In Section 7 we will present more experimental results which demonstrate that the specialized programs derived by our technique are more efficient than those derived by partial deduction or conjunctive partial deduction.
4 Transformation Rules for Logic Programs with Equations and Disequations between Terms

In this section we present the program transformation rules which we use for program specialization. These rules extend the unfold/fold rules considered in [14, 40, 46] to logic programs with atoms which denote equations and disequations between terms. The transformation rules we present in this section enhance in several respects the rules PD1-PD4 for partial deduction which we have considered in Section 3. In particular, we consider a definition introduction rule (see Rule 1) which allows the introduction of new predicates defined by several clauses whose bodies are non-atomic goals, while by rule PD1 a new predicate can be defined by introducing one clause whose body is an atomic goal.

We also consider a folding rule (see Rule 4) by which we can fold several clauses at a time, while by rule PD4 we can fold one clause only. 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 5 below.

4.1 Transformation Rules

Similarly to Section 3 the process of program transformation is viewed as a transformation sequence constructed by applying some transformation rules. However, as already mentioned, in this section we consider an enhanced set of transformation rules. A transformation sequence $P_0, \ldots, P_n$ is constructed from a given initial program $P_0$ by applications of the transformation rules 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 $\gamma_1$ of clauses of $P_k$, (ii) deriving a set $\gamma_2$ of clauses by applying a transformation rule to $\gamma_1$, and (iii) replacing $\gamma_1$ by $\gamma_2$ in $P_k$.

Notice that rules 2 and 3 are in fact equal to rules PD2 and PD3, respectively. However, we rewrite them below for the reader’s convenience.

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

\[
\begin{align*}
D_1 & : \text{newp}(X_1, \ldots, X_h) \leftarrow \text{Body}_1 \\
& \quad \vdots \\
D_m & : \text{newp}(X_1, \ldots, X_h) \leftarrow \text{Body}_m
\end{align*}
\]

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 $\text{Body}_j$, (iii) for all $j \in \{1, \ldots, m\}$, every non-basic predicate occurring in $\text{Body}_j$ also occurs in $P_0$, and (iv) for all $j \in \{1, \ldots, m\}$, there exists at least one non-basic atom in $\text{Body}_j$.

Program $P_{k+1}$ is the program $P_k \cup \{D_1, \ldots, D_m\}$.

As in Section 3 we denote by $\text{Defs}_{k}$ the set of definition clauses introduced by the definition introduction rule during the construction of the transformation sequence $P_0, \ldots, P_k$. In particular, we have that $\text{Defs}_0 = \emptyset$.

Rule 2 (Definition Elimination) Let $p$ be a predicate symbol. By definition elimination w.r.t. $p$ we derive the program $P_{k+1} = \{C \in P_k \mid p \text{ depends on } C\}$.

Rule 3 (Unfolding) Let $C$ be a renamed apart clause of $P_k$ 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 $i = 1, \ldots, m$, $A$ is unifiable with the head of $C_i$ via the mgu $\vartheta_i$. By unfolding $C$ w.r.t. $A$, for $i = 1, \ldots, m$, we derive the clause $D_i : (H \leftarrow G_1, \text{bd}(C_i), G_2)\vartheta_i$.

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
\[
\begin{align*}
C_1. & \quad H \leftarrow G_1, \text{Body}_1 \vartheta, G_2 \\
\vdots \\
C_m. & \quad H \leftarrow G_1, \text{Body}_m \vartheta, G_2
\end{align*}
\]
be renamed clauses of \( P_k \), for a suitable substitution \( \vartheta \), and let
\[
\begin{align*}
D_1. & \quad \text{newp}(X_1, \ldots, X_h) \leftarrow \text{Body}_1 \\
\vdots \\
D_m. & \quad \text{newp}(X_1, \ldots, X_h) \leftarrow \text{Body}_m
\end{align*}
\]
be all clauses in \( \text{Defs}_k \) which have \( \text{newp} \) as head predicate. Suppose that for \( i = 1, \ldots, m \), the following condition holds: for every variable \( X \) occurring in the goal \( \text{Body}_i \) 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 \( \text{Body}_i \) and different from \( X \). By folding \( C_1, \ldots, C_m \) using \( D_1, \ldots, D_m \) we derive the single clause \( \text{E}: H \leftarrow G_1, \text{newp}(X_1, \ldots, X_h) \vartheta, G_2 \).

Program \( P_{k+1} \) is the program \( (P_k - \{C_1, \ldots, C_m\}) \cup \{\text{E}\} \).

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 (by considering the substitution \( \vartheta = \{U/X, V/Y\} \)) using the two definition 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 \( \text{E}: p(X) \leftarrow a(X, Y), r(Y) \).

Rule 5 (Subsumption) (i) Given a substitution \( \vartheta \), we say that a clause \( H \leftarrow G_1, G_2 \vartheta \) 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: \( H\{X/t\} \leftarrow \text{Body} \) 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 \( \text{GenC}: H \leftarrow X=t, \text{Body} \).

Program \( P_{k+1} \) is the program \( (P_k - \{C\}) \cup \{\text{GenC}\} \).

Rule 8 is a particular case of the rule of \( \text{generalization} + \text{equality introduction} \) considered, for instance, in [35].

Rule 7 (Case Split) Let \( C \) be a clause in \( P_k \) of the form: \( H \leftarrow \text{Body} \). By case split of \( C \) w.r.t. \( t \), the binding \( X/t \) where \( X \) does not occur in \( t \), we derive the following two clauses:
\[
\begin{align*}
C_1. & \quad (H \leftarrow \text{Body})\{X/t\} \\
C_2. & \quad H \leftarrow X \neq t, \text{Body}.
\end{align*}
\]

Program \( P_{k+1} \) is the program \( (P_k - \{C\}) \cup \{C_1, C_2\} \).

In this Rule, we do not assume that \( X \) occurs in \( C \). However, in the Determinization Strategy of Section 4, we will always apply the case split rule to a clause \( C : H \leftarrow \text{Body} \) w.r.t. a binding \( X/t \) where \( X \) occurs in \( H \). This use of the case split rule will be sufficient to derive mutually exclusive clauses. Indeed, according to our operational semantics, if \( G \leftarrow P_{k+1} \) \( C_1 \) and \( X \) occurs in \( H \), then no \( G_2 \) exists such that \( G \leftarrow P_{k+1} \) \( C_2 \) using clause \( C_2 \). The same holds by interchanging \( C_1 \) and \( C_2 \). We will return to this property in Definitions 8 (Semideterminism) and 12 (Mutual Exclusion) below.

Rule 8 (Equation Elimination) Let \( C_1 \) be a clause in \( P_k \) of the form:
\[
C_1. \quad H \leftarrow G_1, t_1=t_2, G_2
\]
If \( t_1 \) and \( t_2 \) are unifiable via the most general unifier \( \vartheta \), then by equation elimination we derive the following clause:
\[
C_2. \quad (H \leftarrow G_1, G_2) \vartheta
\]
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 \( P_k - \{C_1\} \).
By rule 5 we may delete clause 3 which is subsumed by clause 2 and we derive a new program:

\[ H \leftarrow G_1, t_1 \neq t_2, G_2 \]

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 \( m \) (\( \geq 0 \)) clauses: \( H \leftarrow G_1, t_1 \neq u_1, G_2, \ldots, H \leftarrow G_1, t_m \neq u_m, G_2 \)

3 if C is of the form: \( H \leftarrow G_1, X \neq X, G_2 \), then C is removed from \( P_k \)

4 if C is of the form: \( H \leftarrow G_1, X \neq X, G_2 \), then C is replaced by \( H \leftarrow G_1, X \neq t, G_2 \)

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 \( \rho \) which is a bijective mapping from the set of the local variables of \( X \neq t_1 \) in C onto the set of the local variables of \( X \neq t_2 \) 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, by Rule 5, if a disequation has occurs twice in the body of a clause, then we can remove the rightmost occurrence.

4.2 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 2.2. In that sense we also say that our transformation rules are correct w.r.t. the given declarative semantics. The following correctness theorem extends similar results holding for logic programs [14 40 46] to the case of logic programs with equations and disequations.

Theorem 5 (Correctness of the Rules w.r.t. the Declarative 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 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 \( \text{Defs}_k \), 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 \( p \), we have that \( M(P_0 \cup \text{Defs}_n) \models A \) iff \( M(P_n) \models A \).

Proof: It is a simple extension of a similar result presented in [14] for the case where we use the unfolding, folding, and generalization + equality introduction rules. The proof technique used in [14] can be adapted to prove also the correctness of our extended set of rules. \( \square \)

In Example 1 of Section 3 we have shown that the unfolding rule may not preserve the operational semantics. The following examples show that also other transformation rules may not preserve the operational semantics.

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 \( P_6 \).

Finally, let us consider the following two operations on the body of a clause: (i) removal of a duplicate atom, and (ii) reordering of atoms. The following examples show that these two operations, which preserve the declarative semantics, may not preserve the operational semantics. Notice, however, that the removal of a duplicate atom and the reordering of atoms cannot be accomplished by the transformation rules listed in Section 4, except for special case considered at Point 9.5 of the disequation replacement rule.

Example 5 Let us consider the program \( P_7 \):

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 \( P_8 \) obtained from \( P_7 \) by replacing clause 1 by the following clause:

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

The goal \( p \) succeeds in \( P_7 \), while it does not succeed in \( P_8 \). Indeed, (i) for program \( P_7 \) we have that:

\[ p \mapsto_{P_7} q(X, Y), q(X, Y), X \neq Y \mapsto_{P_7} q(X, b), X \neq b \mapsto_{P_7} a \neq b \mapsto_{P_7} true, \text{ and (ii) for program } P_8 \text{ we have that: either } p \mapsto_{P_8} X \neq b \text{ or } p \mapsto_{P_8} a \neq Y. \]

In Case (ii), since \( X \) and \( Y \) are unifiable with \( b \) and \( a \), respectively, we have that \( p \mapsto_{P_8}^* true \) does not hold.

Example 6 Let us consider the program \( P_9 \):

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

and the program \( P_{10} \) obtained from \( P_9 \) by replacing clause 1 by the following clause:

4. \( p \leftarrow r(X), q(X) \)

The goal \( p \) succeeds in \( P_9 \), while it does not succeed in \( P_{10} \).

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 each 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, in order 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.
5 Program Transformations based on Modes

Modes provide information about the directionality of predicates, by specifying whether an argument should be used as input or output (see, for instance, [32, 49]). Mode information is very useful for specifying and verifying logic programs [2, 10] and it is used in existing compilers, such as Ciao and Mercury, to generate very efficient code [19, 45]. Mode information has also been used in the context of program transformation to provide sufficient conditions which ensure that reorderings of atoms in the body of a clause preserve program termination [8].

In this paper we use mode information for: (i) specifying classes of programs and goals w.r.t. which the transformation rules we have presented in Section 4.1 preserve the operational semantics (see Section 2.3), and (ii) designing our strategy for specializing programs and reducing nondeterminism.

5.1 Modes

A mode for a non-basic predicate $p$ of arity $h$ ($\geq 0$) 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). In particular, if $h = 0$, then $p$ has a unique mode which is $p$ itself. Given an atom $p(t_1, \ldots, t_h)$ and a mode $p(m_1, \ldots, m_h)$,

(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 which do not occur in $P$. Thus, if $M$ is a mode for a program $P_1$ and, by applying a transformation rule, from $P_1$ we derive a new program $P_2$ where all occurrences of a predicate have been eliminated, then $M$ is a mode also for $P_2$. The following rules may eliminate occurrences of predicates: definition elimination, unfolding, folding, subsumption, disequation replacement (case 9.5). Clearly, if from $P_1$ we derive $P_2$ by applying the definition introduction rule, then in order to obtain a mode for $P_2$ we should add to $M$ a mode for the newly introduced predicate (unless it is already in $M$).

Example 7 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 a different mode for $P$.

Definition 2 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 $A_0 \leftarrow A \cdot \leftarrow G$, we have that $A$ satisfies $M$.

With reference to Example 7 above, program $P$ satisfies mode $M_1$, but it does not satisfy mode $M_2$.

In general, the property that a program satisfies a mode is undecidable. Two approaches are usually followed for verifying this property: (i) the first one uses abstract interpretation methods (see, for instance, [9, 32]) which always terminate, but may return a don’t know answer, and (ii) the second one checks suitable syntactic properties of the program at hand, such as well-modedness [2], which imply that the mode is satisfied.

Our technique is independent of any specific method used for verifying that a program satisfies a mode. However, as the reader may verify, all programs presented in the examples of Section 7 are well-moded and, thus, they satisfy the given modes.
Now we introduce a class of programs, called safe programs, and we prove that if the transformation rules are applied to a safe program and suitable restrictions hold, then the given program and the derived program are equivalent w.r.t. the operational semantics.

**Definition 3 (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 understood from the context.

In order to get our desired correctness result (see Theorem 6 below), we need to restrict the use of our transformation rules as indicated in Definitions 4-7 below. In particular, these restrictions ensure that, by applying the transformation rules, program safety and mode satisfaction are preserved (see Propositions 3 and 4 in Appendix A).

**Definition 4 (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 4.1) whereby from the following clause of $P_k$: 

\[
H \leftarrow G_1, A, G_2
\]

we derive the clauses:

\[
\begin{align*}
D_1. & \quad (H \leftarrow G_1, bd(C_1), G_2) \vartheta_1 \\
\vdots & \\
D_m. & \quad (H \leftarrow G_1, bd(C_m), G_2) \vartheta_m
\end{align*}
\]

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 $\vartheta_i$.

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 $dd_1$, we have that either $X$ is an input variable of $H \vartheta_i$ or $X$ is a local variable of $d$ in $C_i$.

To see that unrestricted applications of the unfolding rule may not preserve safety, let us consider the following program:

1. $p \leftarrow q(X), \ r(X)$
2. $q(1) \leftarrow$
3. $r(X) \leftarrow X \neq 0$

and the mode $M = \{p, q(?), r(+)\}$ for it. By unfolding clause 1 w.r.t. the atom $r(X)$ we derive the clause:

4. $p \leftarrow q(X), \ X \neq 0$

This clause is not safe w.r.t. $M$ because $X$ does not occur in its head.

**Definition 5 (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 4.1) whereby from the following clauses in $P_k$:

\[
\begin{align*}
C_1. & \quad H \leftarrow G_1, (A_1, K_1) \vartheta, G_2 \\
\vdots & \\
C_m. & \quad H \leftarrow G_1, (A_m, K_m) \vartheta, G_2
\end{align*}
\]

and the following definition clauses in $Defs_k$:

\[
\begin{align*}
D_1. & \quad newp(X_1, \ldots, X_h) \leftarrow A_1, K_1 \\
\vdots & \\
D_m. & \quad newp(X_1, \ldots, X_h) \leftarrow A_m, K_m
\end{align*}
\]

17
we derive the new clause:

\[ H \leftarrow G_1, \text{new}(X_1, \ldots, X_h) \theta, G_2 \]

We say that this application of the folding rule is safe w.r.t. mode \( M \) iff the following Property \( \Sigma \) holds:

\[ \text{(Property } \Sigma \text{)} \quad \text{Each input variable of } \text{new}(X_1, \ldots, X_h) \theta \text{ is also an input variable of at least one of the non-basic atoms occurring in } (H, G_1, A_1 \theta, \ldots, A_m \theta). \]

Unrestricted applications of the folding rule may not preserve modes. Indeed, let us consider the following initial program:

1. \( p \leftarrow q(X) \)
2. \( q(1) \leftarrow \)

Suppose that first we introduce the definition clause:

3. \( \text{new}(X) \leftarrow q(X) \)

and then we apply the clause split rule, thereby deriving:

4. \( \text{new}(0) \leftarrow q(0) \)
5. \( \text{new}(X) \leftarrow X \neq 0, q(X) \)

The program made out of clauses 1, 2, 4, and 5 satisfies the mode \( M = \{p, q(?) \text{, } \text{new}(+)\} \). By folding clause 1 using clause 3 we derive:

6. \( p \leftarrow \text{new}(X) \)

This application of the folding rule is not safe and the program we have derived, consisting of clauses 2, 4, 5, and 6, does not satisfy \( M \).

**Definition 6 (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 4.1) to a clause of \( P_k \) is safe iff \( X \) is not an input variable w.r.t. \( M \).

The restrictions considered in Definition 6 are needed to preserve safety. For instance, the clause \( p(t(X)) \leftarrow X \neq 0 \) is safe w.r.t. the mode \( M = \{p(+)\} \), while \( p(Y) \leftarrow Y = t(X), X \neq 0 \) is not safe.

**Definition 7 (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 4.1) whereby from a clause \( C \) in \( P_k \) of the form: \( H \leftarrow \text{Body} \) we derive the following two clauses:

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

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 \( Y \in \text{vars}(t) \), 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 \( G \leftarrow G_1 \) using clause \( C_1 \) and \( G \leftarrow G_2 \) using clause \( C_2 \). In Definition 12 below, we will formalize this property by saying that the clauses \( C_1 \) and \( C_2 \) are mutually exclusive.

Similarly to the unfolding and head generalization rules, the unrestricted use of the case split rule may not preserve safety. For instance, from the clause \( p(X) \leftarrow \) which is safe w.r.t. the mode \( M = \{p(?)\} \), we may derive the two clauses \( p(0) \leftarrow \) and \( p(X) \leftarrow X \neq 0 \), and this last clause is not safe w.r.t. \( M \).

We have shown in Section 4.1 (see Example 6), that the reordering of atoms in the body of a clause may not preserve the operational semantics. Now we prove that a particular reordering of atoms, called disequation promotion, which consists in moving to the left the disequations occurring in the body of a safe clause, preserves the operational semantics. Disequation promotion (not included, for reason of simplicity, among the transformation rules) allows us to rewrite the body of a safe clause
so that every disequation occurs to the left of every atom different from a disequation thereby deriving
the normal form of that clause (see Section 5). The use of normal forms will simplify the proof of
Theorem 6 below and the presentation of the Determinization Strategy in Section 6.

**Proposition 1 (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 \( q(+) \).

In Theorem 6 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 6 (Correctness of the Rules w.r.t. the Operational Semantics)** Let \( P_0, \ldots , P_n \) be
a transformation sequence constructed by using the transformation rules \[\text{and let } p \text{ be a non-basic}
predicate in } P_n. \text{ Let } M \text{ be a mode for } P_0 \cup \text{Defs}_n \text{ such that: (i) } P_0 \cup \text{Defs}_n \text{ is safe w.r.t. } M, \text{ (ii) } P_0 \cup \text{Defs}_n \text{ satisfies } M, \text{ and (iii) the applications of the unfolding, folding, head generalization, and case split rules}
during the construction of } P_0, \ldots , P_n \text{ are all safe w.r.t. } M. \text{ Suppose also that Conditions } 1 \text{ and } 2 \text{ of
Theorem 5 hold. Then: (i) } P_n \text{ is safe w.r.t. } M, \text{ (ii) } P_n \text{ satisfies } M, \text{ and (iii) for each atom } A \text{ which has predicate } p \text{ and satisfies mode } M, \text{ } A \text{ succeeds in } P_0 \cup \text{Defs}_n \text{ iff } A \text{ succeeds in } P_n.

**Proof**: See Appendix A.

\[\square\]

### 5.3 Semideterministic Programs

In this section we introduce the concept of semideterminism which characterizes the class of programs
which can be obtained by using the Determinization Strategy of Section 5. (The reader should not
confuse the notion of semideterminism presented here with the one considered in \[\text{.}

We have already noticed that if a program \( P \) is deterministic for an atom \( A \) according to Definition 1
then there is at most one successful derivation starting from \( A \), and \( A \) succeeds in \( P \) with at
most one answer substitution. Thus, if an atom succeeds in a program with more than one answer
substitution, and none of these substitutions is more general than another, then there is no chance to
transform that program into a new program which is deterministic for that atom.

For instance, let us consider the following generalization of the problem of Sections 3.2 and 3.3:

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 \( N \). The following program Match.Pos computes \( N \) for any given \( P \) and \( S \):
The \textit{Match\_Pos} program is nondeterministic for atoms of the form \textit{match\_pos}(P, S, N) where \(P\) and \(S\) are ground lists, and it computes one answer substitution for each occurrence of \(P\) in \(S\).

Suppose that we want to specialize \textit{Match\_Pos} w.r.t. the atom \textit{match\_pos}([a, a, b], S, N). Thus, we want to derive a new, specialized program \textit{Match\_Pos}_s and a new binary predicate \textit{match\_pos}_s. This new program should be able to compute multiple answer substitutions for a goal. For instance, for the atom \textit{match\_pos}_s([a, a, b, a, a, b], N) the program \textit{Match\_Pos}_s should compute the two substitutions \(\{N/0\}\) and \(\{N/s(s(0))\}\) and, thus, \textit{Match\_Pos}_s cannot be deterministic for the atom \textit{match\_pos}_s([a, a, b, a, a, b], N).

Now, in order to deal with programs which may return multiple answer substitutions, we introduce the notion of semideterminism, which is weaker than that of determinism. Informally, we may say that a semideterministic program has the minimum amount of nondeterminism which is needed to compute multiple answer substitutions. In Section 6, we will prove that the Determinization Strategy, if it terminates, derives a semideterministic program.

\textbf{Definition 8 (Semideterminism)} A program \(P\) is \textit{semideterministic} for a non-basic atom \(A\) iff for each goal \(G\) such that \(A \Rightarrow^*_P G\), there exists at most one clause \(C\) such that \(G \Rightarrow_C G'\) for some goal \(G'\) different from \textit{true}.

Given a mode \(M\) for a program \(P\), we say that \(P\) is \textit{semideterministic} w.r.t. \(M\) iff \(P\) is semideterministic for each non-basic atom which satisfies \(M\).

We will show in Section 7.1 that by applying the Determinization Strategy, from \textit{Match\_Pos}_s we derive the following specialized program \textit{Match\_Pos}_s which is semideterministic for atoms of the form \textit{match\_pos}_s(S, N), where \(S\) is a ground list.

\begin{center}
\begin{tabular}{ll}
\hline
\textbf{Program} & \textbf{Match\_Pos}_s \\
\hline
9. & \textit{match\_pos}_s(S, N) \leftarrow \textit{new\_1}(S, N) \\
20. & \textit{new\_1}([a|S], M) \leftarrow \textit{new\_2}(S, M) \\
21. & \textit{new\_2}([C|S], s(N)) \leftarrow C \neq a, \textit{new\_1}(S, N) \\
32. & \textit{new\_3}([a|S], M) \leftarrow \textit{new\_3}(S, M) \\
33. & \textit{new\_3}([C|S], s(s(N))) \leftarrow C \neq a, \textit{new\_1}(S, N) \\
46. & \textit{new\_3}([a|S], s(M)) \leftarrow \textit{new\_3}(R, S) \\
47. & \textit{new\_3}([b|S], M) \leftarrow \textit{new\_4}(R, S) \\
48. & \textit{new\_3}([C|S], s(s(s(N)))) \leftarrow C \neq a, C \neq b, \textit{new\_1}(S, N) \\
49. & \textit{new\_4}(S, 0) \leftarrow \\
55. & \textit{new\_4}([a|S], s(s(s(M)))) \leftarrow \textit{new\_2}(S, M) \\
56. & \textit{new\_4}([C|S], s(s(s(s(N))))) \leftarrow C \neq a, \textit{new\_1}(S, N) \\
\hline
\end{tabular}
\end{center}

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

\textbf{Definition 9 (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 \textit{satisfiable} w.r.t. \(V\) iff there exists a ground substitution \(\sigma\) with domain \(V\), such that every ground instance of \(D\sigma\) holds (see Section 2.2). 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:
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 2.1).

**Definition 10 (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 11 (Guard of a Clause)** The guard of a clause $C$, denoted $\text{grd}(C)$, is $\text{bd}(C)$ if all atoms in $\text{bd}(C)$ are disequations, otherwise $\text{grd}(C)$ is the (possibly empty) conjunction of the disequations occurring in $\text{bd}(C)$ to the left of the leftmost atom which is not a disequation.

**Definition 12 (Mutually Exclusive Clauses)** Let us consider a mode $M$ for the following two, renamed apart clauses:

\[
C_1. \quad p(t_1, u_1) \leftarrow G_1 \\
C_2. \quad p(t_2, u_2) \leftarrow G_2
\]

where: (i) $p$ is a predicate of arity $k (\geq 0)$ 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 $\vartheta$ and $(\text{grd}(C_1), \text{grd}(C_2))\vartheta$ is not satisfiable w.r.t. $\text{vars}(t_1, t_2)$.

If $h = 0$ we stipulate that the empty tuples $t_1$ and $t_2$ are unifiable via an mgu which is the identity substitution.

The following proposition is useful for proving that a program is semideterministic.

**Proposition 2 (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**: See Appendix B. \qed

In Section 6 we will present a strategy for deriving specialized programs which satisfies the hypotheses (i)–(iv) of the above Proposition 2 and thus, these derived programs are semideterministic.

The following examples show that in Proposition 2 no hypothesis on program $P$ can be discarded.

**Example 8** 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 $p \leftarrow p, (q, q)$, and there exist two non-basic goals, namely $q$ and $(q, q)$, such that $(q, q) \Rightarrow p, q$ and $(q, q) \Rightarrow p, (q, q)$. 

21
**Example 9** Consider the following program $Q$ and the mode $M = \{p(\cdot), q_1, q_2\}$ 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 $p(1) \not\Rightarrow Q p(1)$, 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 10** 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 $p \not\Rightarrow_R r(X)$ 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 $p \not\Rightarrow^*_{R} r(X)$ 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 11** 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 $p \not\Rightarrow^*_S p$, and there exist two non-basic goals, namely $r_1$ and $r_2$, such that $p \Rightarrow_S r_1$ and $p \Rightarrow_S r_2$.

We conclude this section by observing that when a program consists of mutually exclusive clauses and, thus, it is semideterministic, it may be executed very efficiently on standard Prolog systems by inserting cuts in a suitable way. We will return to this point in Section 6.1 when we discuss the speedups obtained by our specialization technique.

### 6 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 4.1. Our strategy pursues the following objectives. (1) The specialization of a program w.r.t. a particular goal. This is similar to what partial deduction does. (2) The elimination of multiple or intermediate data structures. This is similar to what the strategies for eliminating unnecessary variables and conjunctive partial deduction do. (3) The reduction of nondeterminism. This is accomplished by deriving programs whose non-unit clauses are mutually exclusive w.r.t. a given mode, that is, by Proposition 2 semideterministic programs.

The 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. For reasons of clarity, during the presentation of the Determinization Strategy we use high-level descriptions of the subsidiary strategies. These descriptions are used to establish the correctness of Determinization (see Theorem 7). Full details of the subsidiary strategies will be given in Sections 6.2, 6.3, and 6.4 respectively.

#### 6.1 The Determinization Strategy

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 \(j = 1, \ldots, r\), \(m_j + \text{if} \ X_j\) is an input variable of \(p(t_1, \ldots, t_h)\) according to the mode \(M\). We assume that the program \(P\) is safe \wrt\ \(M\). Thus, also program \(P \cup \{S\}\) is safe \wrt\ \(M \cup \{p_s(m_1, \ldots, m_r)\}\). 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) \(\text{TransfP}\), which is the set of clauses from which we will construct the specialized program, (2) \(\text{Defs}\), which is the set of clauses introduced by the definition introduction rule, and (3) \(\text{Cls}\), which is the set of clauses to be transformed during the current iteration. Initially, \(\text{Cls}\) 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.

The Determinization Strategy starts off each iteration by applying the Unfold-Simplify subsidiary strategy to the set \(\text{Cls}\), thereby deriving a new set of clauses called \(\text{UnfoldedCls}\). The Unfold-Simplify strategy first unfolds the clauses in \(\text{Cls}\), and then it simplifies the derived set of clauses by applying the equation elimination, disequation replacement, and subsumption rules.

Then the set \(\text{UnfoldedCls}\) is divided into two sets: (i) \(\text{UnitCls}\), which is the set of unit clauses, and (ii) \(\text{NonunitCls}\), which is the set of non-unit clauses. The Determinization Strategy proceeds by applying the Partition subsidiary strategy to \(\text{NonunitCls}\), thereby deriving a new set of clauses called \(\text{PartitionedCls}\). The Partition strategy consists of suitable applications of the case split, equation elimination, disequation replacement, and head generalization rules such that the set \(\text{PartitionedCls}\) has the following property: it can be partitioned into sets of clauses, called \textit{packets}, such that two clauses taken from different packets are mutually exclusive \wrt\ a suitable mode.

The Determinization Strategy continues by applying the Define-Fold subsidiary strategy to the clauses in \(\text{PartitionedCls}\), thereby deriving a new, semideterministic set of clauses called \(\text{FoldedCls}\). The Define-Fold subsidiary strategy introduces a (possibly empty) set \(\text{NewDefs}\) of definition clauses such that each packet can be folded into a single clause by using a set of definition clauses in \(\text{Defs} \cup \text{NewDefs}\). We have that clauses derived by folding different packets are mutually exclusive and, thus, \(\text{UnitCls} \cup \text{FoldedCls}\) is semideterministic.

At the end of each iteration, \(\text{UnitCls} \cup \text{FoldedCls}\) is added to \(\text{TransfP}\), \(\text{NewDefs}\) is added to \(\text{Defs}\), and the value of the set \(\text{Cls}\) is updated to \(\text{NewDefs}\).

The Determinization Strategy terminates when \(\text{Cls} = \emptyset\), that is, no new predicate is introduced during the current iteration.

### Determinization Strategy

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

**Output:** A specialized program \(P_s\), and an atom \(p_s(X_1, \ldots, X_r) = \text{vars}(p(t_1, \ldots, t_h))\) 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 for \(p_s(X_1, \ldots, X_r)\sigma\).

**Initialize:** Let \(S\) be the clause \(p_s(X_1, \ldots, X_r) \leftarrow p(t_1, \ldots, t_h)\).

\(\text{TransfP} := P; \ \text{Defs} := \{\{\}\}; \ \text{Cls} := \{\{\}\}; \ M_s := M \cup \{p_s(m_1, \ldots, m_r)\}\), where for any \(j = 1, \ldots, r\), \(m_j + \text{if} \ X_j\) is an input variable of \(p(t_1, \ldots, t_h)\) according to the mode \(M\);

**while** \(\text{Cls} \neq \emptyset\) **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 6.2 below, and from \(\text{Cls}\) we derive a new set of clauses \(\text{UnfoldedCls}\).
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 6.3 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 renaming of variables):

\[
\begin{aligned}
H &\leftarrow \text{Diseqs}, G_1 \\
&\quad \cdots \\
H &\leftarrow \text{Diseqs}, G_m
\end{aligned}
\]

where $\text{Diseqs}$ 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 6.4 below. According to that strategy, we introduce a (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) from each packet in $PartitionedCls$ we derive a single clause of the form:

\[
H \leftarrow \text{Diseqs}, \text{newp}(\ldots)
\]

by an application of the folding rule, which is safe w.r.t. $M_{new}$, using the clauses in $Defs \cup NewDefs$.

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

(4) $\text{TransfP} := \text{TransfP} \cup UnitCls \cup FoldedCls$; $Defs := Defs \cup NewDefs$; $Cls := NewDefs$;

$M_s := M_s \cup M_{new}$

end-while

We derive the specialized program $P_s$ by applying the definition elimination rule and keeping only the clauses of $\text{TransfP}$ on which $p_s$ depends.

The Determinization Strategy may fail to terminate for two reasons: (i) the Unfold-Simplify subsidiary strategy may not terminate, because it may perform infinitely many unfolding steps, and (ii) the condition $Cls \neq \emptyset$ for exiting the while-do loop may always be false, because at each iteration the Define-Fold subsidiary strategy may introduce new definition clauses. We will discuss these issues in more detail in Section 9.

Now we 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 for $p_s(X_1, \ldots, X_r)\sigma$ as indicated by the following theorem.

**Theorem 7 (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 $\{X_1, \ldots, X_r\} = \text{vars}(p(t_1, \ldots, t_h))$, 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 for $p_s(X_1, \ldots, X_r)\sigma$.

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 $P \cupDefs$ and $\{X_1, \ldots, X_r\} = \text{vars}(p(t_1, \ldots, t_h))$, 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 $M(P \cupDefs) \models p_s(X_1, \ldots, X_r)\vartheta$. By the correctness of the transformation rules w.r.t. the least Herbrand model (see Theorem 5), we have that $M(P \cupDefs) \models p_s(X_1, \ldots, X_r)\vartheta$ iff $M(P_s) \models p_s(X_1, \ldots, X_r)\vartheta$.

Point (ii.1) follows from Theorem 6 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 $\sigma$, 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 $\text{TransfP}_w - P$ is semideterministic w.r.t. $M_s$, where $\text{TransfP}_w$ is the set of clauses which is the value of the variable $\text{TransfP}$ at the end of the while-do statement of the Determinization Strategy. Indeed, $P_s$ is equal to $\text{TransfP}_w - P$ 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 $\text{TransfP}_w$.

By Proposition 2, it is enough to prove that: (a) $\text{TransfP}_w - P$ is linear, (b) $\text{TransfP}_w - P$ is safe w.r.t. $M_s$, (c) $\text{TransfP}_w - P$ satisfies $M_s$, and (d) the non-unit clauses of $\text{TransfP}_w - P$ 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: $H \leftarrow \text{Diseqs, neup}()$, where a single non-basic atom occurs in the body. All other clauses in $\text{TransfP}_w - P$ are unit clauses.

Properties (b) and (c) follow from Theorem 6 recalling that the application of the unfolding, folding, head generalization, and case split rules are all safe.

Property (d) can be proved by showing that, during the execution of the Determinization Strategy, the following Property (I) holds: all the non-unit clauses of $\text{TransfP} - P$ are pairwise mutually exclusive w.r.t. $M_s$. Indeed, initially $\text{TransfP} - P$ is empty and thus, Property (I) holds. Furthermore, Property (I) is an invariant of the while-do loop. Indeed, 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 $\text{TransfP}$ are the elements of the set $\text{FoldedCls}$ 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 $\text{FoldedCls}$ are pairwise mutually exclusive w.r.t. $M_{new}$, and their head predicates do not occur in $\text{TransfP}$. Thus, the clauses of $\text{TransfP} \cup \text{UnitCls} \cup \text{FoldedCls}$ 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) $\text{TransfP} := \text{TransfP} \cup \text{UnitCls} \cup \text{FoldedCls}$ and $M_s := M_s \cup M_{new}$, we have that Property (I) holds. □

Now we 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 7.

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 $Y$,
Definition 13 (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 $\vartheta$, 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 $\text{Simplify}(S)$ 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.

\begin{procedure}
\text{Unfold-Simplify}(\text{Cls}, \text{UnfoldedCls}).
\begin{definition}
\text{Input}: A set $\text{Cls}$ of clauses in a program $P$ and a mode $M_s$ for $P$. $P$ is safe w.r.t. $M_s$ and for each $C \in \text{Cls}$, the input variables of the leftmost non-basic atom in the body of $C$ are input variables of the head of $C$.
\end{definition}
\begin{definition}
\text{Output}: A new set $\text{UnfoldedCls}$ of clauses which are derived from $\text{Cls}$ by applying the safe unfolding, equation elimination, disequation replacement, and subsumption rules. The clauses in $\text{UnfoldedCls}$ are safe w.r.t. $M_s$.
\end{definition}
\begin{notation}
(1) Unfold w.r.t. Leftmost Non-basic Atom:
\text{UnfoldedCls} := \{ E \mid \text{there exists a clause } C \in \text{Cls} \text{ and clause } E \text{ is derived by unfolding } C \text{ w.r.t. the leftmost non-basic atom in its body} \};
\text{UnfoldedCls} := \text{Simplify}(\text{UnfoldedCls})
\end{notation}
\begin{notation}
(2) Unfold w.r.t. Leftmost Consumer Atom:
\text{while} there exists a clause $C \in \text{UnfoldedCls}$ whose body has a leftmost consumer atom, say $A$, such that the unfolding of $C$ w.r.t. $A$ is safe \textbf{do}
\text{UnfoldedCls} := (\text{UnfoldedCls} - \{C\}) \cup \{ E \mid \text{E is derived by unfolding } C \text{ w.r.t. } A \};
\text{UnfoldedCls} := \text{Simplify}(\text{UnfoldedCls})
\textbf{end-while}
\end{notation}
\end{procedure}

Notice that our assumptions on the input program $P$ and clauses $\text{Cls}$ ensure that the first unfolding step performed by the Unfold-Simplify procedure is safe.

Notice also that our Unfold-Simplify strategy may fail to terminate. We will briefly return to this issue in Section 9.

Our Unfold-Simplify strategy differs from usual unfolding strategies for (conjunctive) partial deduction (see, for instance, \cite{8, 13, 33, 41}), because mode information is used. We have found this strategy very effective on several examples as shown in the following Section 7.
6.3 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 \( k \) (\( \geq 0 \)), 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 renaming of variables):
\[
\begin{align*}
H & \leftarrow \text{Diseqs}, G_1 \\
& \ldots \\
& H \leftarrow \text{Diseqs}, G_m
\end{align*}
\]
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:
\[
\begin{align*}
C_1. \quad & p(t_1, u_1) \leftarrow \text{Body}_1 \\
C_2. \quad & p(t_2, u_2) \leftarrow \text{Body}_2
\end{align*}
\]
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 \( \vartheta \) such that \( t_1 \vartheta = t_2 \), and for any substitution \( \rho \) which is a bijective mapping from the set of local variables of \( \text{grd}(C_1\vartheta) \) in \( C_1\vartheta \) onto the set of local variables of \( \text{grd}(C_2) \) in \( C_2 \), \( \text{grd}(C_1\vartheta\rho) \) cannot be made syntactically equal to \( \text{grd}(C_2) \) 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 \( \vartheta \) of \( t_i \) and \( t_j \) and a binding \( Y/t_a \) in \( \vartheta \) 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 \( \vartheta \). Now every safe clause whose normal form has a disequation of the form \( X \neq t \), where \( X \) is a local variable of that disequation in that clause, is mutually exclusive w.r.t. any other safe clause. This is the case because, for any substitution \( \sigma \) which does not bind \( X \), \( t_\sigma \) is unifiable with \( X \) and, thus, \( X \neq t_\sigma \) is not satisfiable. Thus, for some \( i, j \in \{1, 2\} \), with \( i \neq j \), there exists a disequation \( (Y \neq t_a)\vartheta \) in \( \text{grd}(C_i\vartheta) \) where \( Y \vartheta \) is an input variable of \( \text{hd}(C_i\vartheta) \), such that for any substitution \( \rho \) which is a bijective mapping from the set of local variables of \( \text{grd}(C_i\vartheta) \) in \( C_i\vartheta \) onto the set of local variables of \( \text{grd}(C_j\vartheta) \) in \( C_j\vartheta \) and for every disequation \( (Z \neq t_b)\vartheta \) in \( \text{grd}(C_j\vartheta) \), 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 \( \text{hd}(C_j\vartheta) \). 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:
\[
\begin{align*}
C_{21}. \quad & p(t_2, u_2) \leftarrow \text{Body}_2\{X/r\} \\
C_{22}. \quad & p(t_2, u_2) \leftarrow X \neq r, \text{Body}_2
\end{align*}
\]

We update the value of NonunitCls as follows:
\[
\begin{align*}
\text{NonunitCls} := (\text{NonunitCls} - \{C_2\}) \cup \{C_{21}, C_{22}\} \\
\text{NonunitCls} := \text{Simplify}(\text{NonunitCls}).
\end{align*}
\]
Now the set NonunitCls is partitioned into subsets of clauses and after suitable renaming of variables and disequation promotion, each subset is of the form:

\[
\begin{align*}
  &\{ \ p(t, u_1) \leftarrow \text{Diseqs, Goal}_1 \\
  &\ldots \\
  &\ p(t, u_m) \leftarrow \text{Diseqs, Goal}_m 
\end{align*}
\]

where \( \text{Diseqs} \) is a conjunction of disequations and for \( k = 1, \ldots, m \), no disequation occurs in \( \text{Goal}_k \), 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:

\[
\begin{align*}
  &\{ \ p(t, u) \leftarrow \text{Eqs}_1, \text{Diseqs}, \text{Goal}_1 \\
  &\ldots \\
  &\ p(t, u) \leftarrow \text{Eqs}_m, \text{Diseqs}, \text{Goal}_m 
\end{align*}
\]

where \( u \) is the most specific common generalization of the terms \( u_1, \ldots, u_m \) and, for \( k = 1, \ldots, m \), the goal \( \text{Eqs}_k \) 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.

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

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

**Theorem 8** The Partition procedure terminates.

*Proof:* See Appendix C.

When the Partition procedure terminates, it returns a set \( \text{PartitionedCls} \) 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.

### 6.4 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 \( \text{Defs} \) of definition clauses. PartitionedCls is the union of the disjoint packets of clauses computed by the Partition subsidiary strategy.

**Output:** (i) A (possibly empty) set \( \text{NewDefs} \) of definition clauses, together with a mode \( M_{\text{new}} \) consisting of exactly one mode for each distinct head predicate in \( \text{NewDefs} \). For each \( C \in \text{NewDefs} \), 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 \( \text{FoldedCls} \) of folded clauses.

\( \text{NewDefs} := \emptyset; \ M_{\text{new}} := \emptyset; \ \text{FoldedCls} := \emptyset; \)

**while** there exists in PartitionedCls a packet \( Q \) of the form:
\[
\begin{align*}
H & \leftarrow \text{Diseqs}, \ G_1 \\
\vdots \\
H & \leftarrow \text{Diseqs}, \ G_m
\end{align*}
\]

where \text{Diseqs} is a conjunction of disequations and for \( k = 1, \ldots, m \), no disequation occurs in \( G_k \).

\[\text{do} \quad \text{PartitionedCls} := \text{PartitionedCls} - Q \quad \text{and apply the definition and safe folding rules as follows.}\]

(Case \( \alpha \)) Let us suppose that the set \( \text{Defs} \) of the available definition clauses contains a subset of clauses of the form:

\[
\begin{align*}
\text{newq} (X_1, \ldots, X_h) & \leftarrow G_1 \\
\vdots \\
\text{newq} (X_1, \ldots, X_h) & \leftarrow G_m
\end{align*}
\]

such that: (i) they are all the clauses in \( \text{Defs} \) for predicate \( \text{newq} \), (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 \( H, \text{Diseqs} \) (this property is needed for the correctness of folding, see Section 4.1), and (iii) for \( i = 1, \ldots, h \), if \( X_i \) is an input argument of \( \text{newq} \) 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:

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

(Case \( \beta \)) If in \( \text{Defs} \) there is no set of definition clauses satisfying the conditions described in Case \( \alpha \), then we add to \( \text{NewDefs} \) the following clauses for a new predicate \( \text{newr} \):

\[
\begin{align*}
\text{newr} (X_1, \ldots, X_h) & \leftarrow G_1 \\
\vdots \\
\text{newr} (X_1, \ldots, X_h) & \leftarrow G_m
\end{align*}
\]

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 \( H, \text{Diseqs} \), 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 \( M_{\text{new}} \) the mode \( \text{newr} (m_1, \ldots, m_h) \) 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:

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

end-while

Notice that the post-conditions on the set \( \text{NewDefs} \) which is derived by the Define-Fold procedure (see Point (i) of the Output of the procedure), ensure the satisfaction of the pre-conditions on the set \( \text{Cls} \) which is an input of the Unfold-Simplify procedure. Indeed, recall that the set \( \text{Cls} \) is constructed during the Determinization Strategy by the assignment \( \text{Cls} := \text{NewDefs} \). 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 5). This fact is implied in Case \( \alpha \) by Condition (iii), and in Case \( \beta \) by the definition of the mode for \( \text{newr} \).

Finally, notice that the Define-Fold procedure terminates. However, this procedure does not guarantee the termination of the specialization process, because at each iteration of the while-do loop of the Determinization Strategy, the Define-Fold procedure may introduce a nonempty set of new definition clauses. We will briefly discuss this issue in Section 9.

### 7 Examples of Application of the Determinization 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.
The body of clause 7 has no consumer atoms (notice that, for instance, the mgu of \(W\)). We consider again the program \(\text{Match}\). Thus, the first iteration of the Determinization Strategy terminates with \(\text{Unfold-Simplify}\). The unfolding of clause 7 amounts to its deletion because the atom \(\text{match}\_\text{pos}(\_\_\_, \_\_\_, S)\) is not unifiable with any head in program \(\text{Match}\). We leave it to the reader to verify that \(\text{Match}\_\text{Pos}\) satisfies \(M\).

The derivation we will perform using the Determinization Strategy is more challenging than the ones presented in the literature (see, for instance, [11, 12, 13, 15, 44]) because an occurrence of the pattern \(P\) in the string \(S\) is specified in the initial program (see clause 1) in a nondeterministic way by stipulating the existence of two substrings \(L\) and \(R\) such that \(S\) is the concatenation of \(L\), \(P\), and \(R\).

We want to specialize the \(\text{Match}\_\text{Pos}\) program w.r.t. the atom \(\text{match}\_\text{pos}([a, a, b], S, N)\). Thus, we first introduce the definition clause:

6. \(\text{match}\_\text{pos}_s(S, N) \leftarrow \text{match}\_\text{pos}([a, a, b], S, N)\)

The mode of the new predicate is \(\text{match}\_\text{pos}_s(\_\_\_, ?)\) because \(S\) is an input argument of \(\text{match}\_\text{pos}\) and \(N\) is not an input argument. Our transformation strategy starts off with the following initial values: \(\text{Defs} = \text{Cls} = \{6\}\), \(\text{TransfP} = \text{Match}\_\text{Pos}\), and \(M_s = M \cup \{\text{match}\_\text{pos}_s(\_\_\_, ?)\}\).

First iteration

\(\text{Unfold-Simplify}\). By unfolding clause 6 w.r.t. the leftmost atom in its body we derive:

7. \(\text{match}\_\text{pos}_s(S, N) \leftarrow \text{append}(Y, R, S), \text{append}(L, [a, a, b], Y), \text{length}(L, N)\)

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

\(\text{Partition}\). \(\text{NonunitCls}\) is made out of clause 7 only, and thus, the \(\text{Partition}\) subsidiary strategy immediately terminates and produces a set \(\text{PartitionedCls}\) which consists of a single packet made out of clause 7.

\(\text{Define-Fold}\). In order to fold clause 7 in \(\text{PartitionedCls}\), the \(\text{Define-Fold}\) subsidiary strategy introduces the following definition clause:

8. \(\text{new}_1(S, N) \leftarrow \text{append}(Y, R, S), \text{append}(L, [a, a, b], Y), \text{length}(L, N)\)

The mode of \(\text{new}_1\) is \(\text{new}_1(\_\_\_, ?)\). By folding clause 7 using clause 8 we derive:

9. \(\text{match}\_\text{pos}_s(S, N) \leftarrow \text{new}_1(S, N)\)

Thus, the first iteration of the Determinization Strategy terminates with \(\text{Defs} = \{6, 8\}\), \(\text{Cls} = \{8\}\), \(\text{TransfP} = \text{Match}\_\text{Pos} \cup \{9\}\), and \(M_s = M \cup \{\text{match}\_\text{pos}_s(\_\_\_, ?), \text{new}_1(\_\_\_, ?)\}\).

Second iteration

\(\text{Unfold-Simplify}\). We follow the subsidiary strategy described in Section 5.3 and we first unfold clause 8 in \(\text{Cls}\) w.r.t. the leftmost atom in its body. We get:

10. \(\text{new}_1(S, N) \leftarrow \text{append}(L, [a, a, b], [\_\_\_]), \text{length}(L, N)\)

11. \(\text{new}_1([C|S], N) \leftarrow \text{append}(Y, R, S), \text{append}(L, [a, a, b], [C|Y]), \text{length}(L, N)\)

Now we unfold clauses 10 and 11 w.r.t. the leftmost consumer atom of their bodies (see the underlined atoms). The unfolding of clause 10 amounts to its deletion because the atom \(\text{append}(L, [a, a, b], [\_\_\_])\) is not unifiable with any head in program \(\text{Match}\_\text{Pos}\). The unfolding of clause 11 yields two new clauses that are further unfolded according to the \(\text{Unfold-Simplify}\) subsidiary strategy. After some unfolding steps, we derive the following clauses:

12. \(\text{new}_1([a|S], 0) \leftarrow \text{append}(\_\_\_, a, b, R, S)\)

13. \(\text{new}_1([C|S], s(N)) \leftarrow \text{append}(Y, R, S), \text{append}(L, [a, a, b], Y), \text{length}(L, N)\)

\(\text{Partition}\). 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:
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:

\[
\begin{align*}
\text{new1}([a|S], M) \leftarrow & \quad M = 0, \quad \text{append}([a, b], R, S) \\
\text{new1}([a|S], M) \leftarrow & \quad M = s(N), \quad \text{append}(Y, R, S), \quad \text{append}(L, [a, a, b], Y), \quad length(L, N)
\end{align*}
\]

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 \((\alpha)\) of the Define-Fold subsidiary strategy. Thus, we introduce a new predicate new2 as follows:

\[
\begin{align*}
\text{new2}(S, M) & \leftarrow M = 0, \quad \text{append}([a, b], R, S) \\
\text{new2}(S, M) & \leftarrow M = s(N), \quad \text{append}(Y, R, S), \quad \text{append}(L, [a, a, b], Y), \quad length(L, N)
\end{align*}
\]

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:

\[
\begin{align*}
\text{new1}([a|S], M) & \leftarrow \text{new2}(S, M)
\end{align*}
\]

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 \((\beta)\) of the Define-Fold subsidiary strategy. By folding clause 15 we derive the following clause:

\[
\begin{align*}
\text{new1}([C|S], s(N)) & \leftarrow C \neq a, \quad \text{new1}(S, N)
\end{align*}
\]

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\}; \(M_s := M_s \cup \{\text{new2}(+)?\}\).

**Third iteration**

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

\[
\begin{align*}
\text{new2}([a|S], 0) & \leftarrow \text{append}([b], R, S) \\
\text{new2}([a|S], s(0)) & \leftarrow \text{append}([a, b], R, S) \\
\text{new2}([C|S], s(s(N))) & \leftarrow \text{append}(Y, R, S), \quad \text{append}(L, [a, a, b], Y), \quad length(L, N)
\end{align*}
\]

**Partition.** The set NonunitCls is identical to UnfoldedCls. From NonunitCls we derive the set PartitionedCls which is the union of two packets. The first packet consists of the following clauses:

\[
\begin{align*}
\text{new2}([a|S], M) & \leftarrow M = 0, \quad \text{append}([b], R, S) \\
\text{new2}([a|S], M) & \leftarrow M = s(0), \quad \text{append}([a, b], R, S) \\
\text{new2}([a|S], M) & \leftarrow M = s(s(N)), \quad \text{append}(Y, R, S), \quad \text{append}(L, [a, a, b], Y), \quad length(L, N)
\end{align*}
\]

The second packet consists of the following clause only:

\[
\begin{align*}
\text{new2}([C|S], s(s(N))) & \leftarrow C \neq a, \quad \text{append}(Y, R, S), \quad \text{append}(L, [a, a, b], Y), \quad length(L, N)
\end{align*}
\]

**Define-Fold.** We introduce the following definition clauses:

\[
\begin{align*}
\text{new3}(S, M) & \leftarrow M = 0, \quad \text{append}([b], R, S) \\
\text{new3}(S, M) & \leftarrow M = s(0), \quad \text{append}([a, b], R, S) \\
\text{new3}(S, M) & \leftarrow M = s(s(N)), \quad \text{append}(Y, R, S), \quad \text{append}(L, [a, a, b], Y), \quad length(L, N)
\end{align*}
\]
where the mode for new3 is new3(+, ?). By folding, from PartitionedCls we derive the following two clauses:

32. new2([a|S], M) ← new3(S, M)
33. new2([C|S], s(s(N))) ← C ≠ a, new1(S, N)

which constitute the set FoldedCls.

The third iteration of the Determinization Strategy terminates with the following assignments: 

Defs := Defs ∪ \{29, 30, 31\}; Cls := \{29, 30, 31\}; TransfP := TransfP ∪ \{32, 33\}; Ms := Ms ∪ \{new3(+, ?)\}.

Fourth iteration

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

34. new3([b|S], 0) ← append([], R, S)
35. new3([a|S], s(0)) ← append([b], R, S)
36. new3([a|S], s(s(0))) ← append([a, b], R, S)
37. new3([C|S], s(s(N))) ← 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)) ← M = 0, append([b], R, S)
39. new3([a|S], s(M)) ← M = s(0), append([a, b], R, S)
40. new3([a|S], s(M)) ← M = s(s(N)), append(Y, R, S), append(L, [a, a, b], Y), length(L, N)
41. new3([b|S], M) ← M = 0, append([], R, S)
42. new3([b|S], M) ← M = s(s(N)), append(Y, R, S), append(L, [a, a, b], Y), length(L, N)
43. new3([C|S], s(s(N))) ← C ≠ a, C ≠ b, append(Y, R, S), append(L, [a, a, b], Y), length(L, N)

PartitionedCls consists of three packets: \{38, 39, 40\}, \{41, 42\}, and \{43\}.

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

44. new4(S, M) ← M = 0, append([], R, S)
45. new4(S, M) ← M = 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)) ← new3(R, S)
47. new3([b|S], M) ← new4(R, S)
48. new3([C|S], s(s(N))) ← C ≠ a, C ≠ b, new1(S, N)

The fourth iteration terminates with the following assignments: 

Defs := Defs ∪ \{44, 45\}; Cls := \{44, 45\}; TransfP := TransfP ∪ \{46, 47, 48\}; Ms := Ms ∪ \{new4(+, ?)\}.

Fifth iteration

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

49. new4(S, 0) ←
50. new4([a|S], s(s(0))) ← append([a, b], R, S)
51. new4([C|S], s(s(s(N)))) ← 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(M))) ← M = 0, append([a, b], R, S)
53. new4([a|S], s(s(M))) ← M = s(N), append(Y, R, S), append(L, [a, a, b], Y), length(L, N)
54. new4([C|S], s(s(s(N)))) ← C ≠ a, append(Y, R, S), append(L, [a, a, b], Y), length(L, N)

PartitionedCls consists of two packets: \{52, 53\} and \{54\}.

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, (i) we fold clauses 52 and 53 using
clauses 18 and 19, and (ii) we fold clause 54 using clause 8. Since no new definition is introduced, the set $Cls$ is empty and the transformation strategy terminates. Our final specialized program is the program $Match\_Pos_s$ shown in Section 5.3.

The $Match\_Pos_s$ 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 $match\_pos\_s(S, N)$, where $S$ is bound to a list of characters, and by clause 1 $match\_pos\_s(S, N)$ 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).

7.2 Multiple Pattern Matching

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 the list $Ps$. For any given $Ps$ and $S$ the following program computes $N$ in a nondeterministic way:

Program $Mmatch$

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

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 7.1, 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. $mmatch_s(S, N) \leftarrow mmatch([[a, a, a], [a, a, b]], S, N)$

The mode of the new predicate is $mmatch_s(+, ?)$ because $S$ is an input argument of $mmatch$ and $N$ is not an input argument. Thus, our Determinization Strategy starts off with the following initial values: $Defs = Cls = \{3\}$, $TransfP = Mmatch$, and $M_0 = M \cup \{mmatch_s(+, ?)\}$.
We have that \( \text{in}_\text{language}(E, S) \) holds iff \( S \) is a string in \( \{a, b\}^* \) and \( S \) belongs to the language denoted by the regular expression \( E \). In this \( \text{Reg\_Expr} \) program we have used the predicate \( \text{ne}\_\text{append}(S_1, S_2, S) \) which holds iff the non-empty string \( S \) is the concatenation of the nonempty string \( S_1 \) and the string \( S_2 \). The use of the atom \( \text{ne}\_\text{append}(S_1, S_2, S) \) in clause 10 ensures that we have a terminating program, that is, a program for which we cannot have an infinite derivation when starting from a ground goal. Indeed, if in clause 10 we replace \( \text{ne}\_\text{append}(S_1, S_2, S) \) by \( \text{append}(S_1, S_2, S) \), then we may construct
an infinite derivation because from a goal of the form \(\text{accepts}(E^*, S)\) we can derive a new goal of the form \((\text{accepts}(E, L), \text{accepts}(E^*, S))\).

We consider the following mode for the program \(\text{Reg} \_ \text{Expr}\):
\[
\{\text{in} \_ \text{language}(+ , +), \text{string}(+), \text{accepts}(+ , +), \text{symbol}(+), \text{ne} \_ \text{append}(?, ?, +), \text{append}(?, ?, +)\}.
\]

We use our Determinization Strategy to specialize the program \(\text{Reg} \_ \text{Expr}\) w.r.t. the atom \(\text{in} \_ \text{language}((aa^*(b+bb))^*, S)\). Thus, we begin by introducing the definition clause:

15. \(\text{in} \_ \text{language}_s(S) \leftarrow \text{in} \_ \text{language}((aa^*(b+bb))^*, S)\)

The mode for this new predicate is \(\text{in} \_ \text{language}_s(+ )\) because \(S\) is an input argument of \(\text{in} \_ \text{language}\). The output of the Determinization Strategy is the following specialized program \(\text{Reg} \_ \text{Expr}_s\):

| Program \(\text{Reg} \_ \text{Expr}_s\) (specialized, semideterministic) |
|---|
| 16. \(\text{in} \_ \text{language}_s(S) \leftarrow \text{new}1(S)\) |
| 17. \(\text{new}1([ ] ) \leftarrow \) |
| 18. \(\text{new}1([a|S]) \leftarrow \text{new}2(S)\) |
| 19. \(\text{new}2([a|S]) \leftarrow \text{new}3(S)\) |
| 20. \(\text{new}2([b|S]) \leftarrow \text{new}4(S)\) |
| 21. \(\text{new}3([a|S]) \leftarrow \text{new}3(S)\) |
| 22. \(\text{new}3([b|S]) \leftarrow \text{new}4(S)\) |
| 23. \(\text{new}4([ ] ) \leftarrow \) |
| 24. \(\text{new}4([a|S]) \leftarrow \text{new}2(S)\) |
| 25. \(\text{new}4([b|S]) \leftarrow \text{new}1(S)\) |

This specialized program corresponds to a deterministic finite automaton.

### 7.4 Matching Regular Expressions

The following nondeterministic program defines a relation \(\text{re} \_ \text{match}(E, S)\), where \(E\) is a regular expression and \(S\) is a string, which holds iff there exists a substring \(P\) of \(S\) such that \(P\) belongs to the language denoted by \(E\):

| Program \(\text{Reg} \_ \text{Expr} \_ \text{Match}\) (initial, nondeterministic) |
|---|
| 1. \(\text{re} \_ \text{match}(E, S) \leftarrow \text{append}(Y, R, S), \text{append}(L, P, Y), \text{accepts}(E, P)\) |

The predicates \(\text{append}\) and \(\text{accepts}\) are defined as in the programs \(\text{Naive} \_ \text{Match}\) (see Section 3.3) and \(\text{Reg} \_ \text{Expr}\) (see Section 7.3), respectively, and their clauses are not listed here. We consider the following mode for the program \(\text{Reg} \_ \text{Expr} \_ \text{Match}\): \(\{\text{append}(?, ?, +), \text{accept}(+, +), \text{re} \_ \text{match}(+, +)\}\).

We want to specialize the program \(\text{Reg} \_ \text{Expr} \_ \text{Match}\) w.r.t. the regular expression \(aa^*b\). Thus, we introduce the following definition clause:

2. \(\text{re} \_ \text{match}_s(S) \leftarrow \text{re} \_ \text{match}(aa^*b, S)\)

The mode of this new predicate is \(\text{re} \_ \text{match}_s(+)\) because \(S\) is an input argument of \(\text{re} \_ \text{match}\). The output of the Determinization Strategy is the following program:

| Program \(\text{Reg} \_ \text{Expr} \_ \text{Match}_s\) (specialized, semideterministic) |
|---|
| 3. \(\text{re} \_ \text{match}_s(S) \leftarrow \text{new}1(S)\) |
| 4. \(\text{new}1([a|S]) \leftarrow \text{new}2(S)\) |
| 5. \(\text{new}1([C|S]) \leftarrow \neq a, \text{new}1(S)\) |
| 6. \(\text{new}2([a|S]) \leftarrow \text{new}3(S)\) |
| 7. \(\text{new}2([C|S]) \leftarrow \neq a, \text{new}1(S)\) |
| 8. \(\text{new}3([a|S]) \leftarrow \text{new}4(S)\) |
| 9. \(\text{new}3([b|S]) \leftarrow \text{new}3(S)\) |
| 10. \(\text{new}3([C|S]) \leftarrow \neq a, \neq b, \text{new}1(S)\) |
| 11. \(\text{new}4(S) \leftarrow \) |

Similarly to the single-pattern string matching example of Section 3.3, this specialized, semideterministic program corresponds to a deterministic finite automaton.
7.5 Specializing Context-free Parsers to Regular Grammars

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

| Program $CF_{\text{Parser}}$ |  (initial, nondeterministic) |
|-----------------------------|-------------------------------|
| 1. $\text{string\_parse}(G, A, W) \leftarrow \text{string}(W), \text{parse}(G, A, W)$ |
| 2. $\text{string}([],[]) \leftarrow$ |
| 3. $\text{string}([0|W]) \leftarrow \text{string}(W)$ |
| 4. $\text{string}([1|W]) \leftarrow \text{string}(W)$ |
| 5. $\text{parse}(G, [\cdot,\cdot,\cdot]) \leftarrow$ |
| 6. $\text{parse}(G, [A|X], [A|Y]) \leftarrow \text{terminal}(A), \text{parse}(G, X, Y)$ |
| 7. $\text{parse}(G, [A|X], Y) \leftarrow \text{nonterminal}(A), \text{member}(A \rightarrow B, G)$, $\text{append}(B, X, Z), \text{parse}(G, Z, Y)$ |
| 8. $\text{member}(A, [A|X]) \leftarrow$ |
| 9. $\text{member}(A, [B|X]) \leftarrow \text{member}(A, X)$ |

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

10. $\text{string\_parse}_s(W) \leftarrow \text{parse}([], [s \rightarrow 0u], [s \rightarrow 0v], [s \rightarrow 0w], [u \rightarrow 0], [u \rightarrow 0u], [u \rightarrow 0v], [v \rightarrow 0], [v \rightarrow 0v], [v \rightarrow 0u], [w \rightarrow 1], [w \rightarrow 0w], [s], [W])$

The output of the Determinization Strategy is the following specialized program $CF_{\text{Parser}}_s$:  

| Program $CF_{\text{Parser}}_s$ |  (specialized, semideterministic) |
|-----------------------------|-------------------------------|
| 11. $\text{string\_parse}_s(W) \leftarrow \text{new}1(W)$ |
| 12. $\text{new}1([0|W]) \leftarrow \text{new}2(W)$ |
| 13. $\text{new}2([0|W]) \leftarrow \text{new}3(W)$ |
| 14. $\text{new}2([1|W]) \leftarrow \text{new}4(W)$ |
| 15. $\text{new}3([\cdot]) \leftarrow$ |
| 16. $\text{new}3([0|W]) \leftarrow \text{new}5(W)$ |
| 17. $\text{new}3([1|W]) \leftarrow \text{new}4(W)$ |
| 18. $\text{new}4([\cdot]) \leftarrow$ |
| 19. $\text{new}5([\cdot]) \leftarrow$ |
| 20. $\text{new}5([0|W]) \leftarrow \text{new}3(W)$ |
| 21. $\text{new}5([1|W]) \leftarrow \text{new}4(W)$ |

This program corresponds to a deterministic finite automaton.

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
The ability of our Determinization Strategy of putting together the computations performed by the initial program in different branches of the computation tree, so that common repeated subcomputations are avoided, is based on the ideas which motivate the tupling strategy \[^{23}\] first proposed as a transformation technique for functional languages.

8 Experimental Evaluation

The Determinization Strategy has been implemented in the MAP program transformation system \[^{39}\]. All program specialization examples presented in Sections \[^{33}\] and \[^{7}\] have been worked out in a fully automatic way by the MAP system. We have compared the specialization times and the speedups obtained by the MAP system with those obtained by ECCE, a system for (conjunctive) partial deduction \[^{24}\]. All experimental results reported in this section have been obtained by using...
\[
\text{string\_parse}_s(g, [s], [0^n 1]) \quad (n \geq 2)
\]

\[
\begin{align*}
&\text{new}_1([0^n 1]) \\
&\text{new}_2([0^{n-1} 1]) \\
&\text{new}_3([0^{n-2} 1]) \\
&\vdots \\
&\text{true}
\end{align*}
\]

Figure 3: Derivation tree \(T_2\) for \text{string\_parse}_s([0^n 1]).

SICStus Prolog 3.8.5 running on a Pentium II under Linux.

In Table 1 we consider the examples of Sections 3.3, 5.3, and 7, and we show the times taken (i) for performing partial deduction by using the ECCE system, (ii) for performing conjunctive partial deduction by using the ECCE system, and (iii) for applying the Determinization Strategy by using the MAP system. The static input shown in Column 2 of Table 1 is the goal w.r.t. which we have specialized the programs of Column 1. For running the ECCE system suitable choices among the available unfolding strategies and generalization strategies should be made. We have used the choices suggested by the system itself for partial deduction and conjunctive partial deduction, and we made some changes only when specialization was not performed within a reasonable amount of time. For running the MAP system the only information to be provided by the user is the mode for the program to be specialized. The system assumes that the program satisfies this mode and no mode analysis is performed.

| Program         | Static Input                        | ECCE (PD) | ECCE (CPD) | MAP (Det) |
|-----------------|-------------------------------------|-----------|------------|-----------|
| Naive\_Match    | naive\_match([aab], S)             | 360       | 370        | 70        |
| Naive\_Match    | naive\_match([aaaaaaaaaab], S)     | 420       | 2120       | 480       |
| Match\_Pos      | match\_pos([aab], S, N)            | 540       | 360        | 100       |
| Match\_Pos      | match\_pos([aaaaaaaaaab], S, N)    | 650       | 910        | 500       |
| Mmatch           | mmatch([[aaa], [aab]], S, N)       | 1150      | 1400       | 280       |
| Mmatch           | mmatch([[[aa], [aaa]], [aab]], S, N) | 1740  | 2040       | 220       |
| Reg\_Expr       | in\_language((a\^n(b+b))\^*, S)   | 6260      | 138900     | 420       |
| Reg\_Expr       | in\_language(a\^n(b+bb+bbb), S)   | 3460      | 5430       | 230       |
| Reg\_Expr\_Match| re\_match(aa\^n(b+b), S)           | 970       | 5290       | 210       |
| Reg\_Expr\_Match| re\_match(a\^n(b+b), S)           | 1970      | 11200      | 300       |
| CF\_Parser      | string\_parse\_re\_match(g, [s], W)| 23400     | 32700      | 1620      |
| CF\_Parser      | string\_parse\_re\_match(g, [s], W)| 31200     | 31800      | 2000      |

Table 1: Specialization Times (in milliseconds).

The experimental results of Table 1 show that the MAP implementation of the Determinization Strategy is much faster than the ECCE implementation of both partial deduction and conjunctive partial deduction. We believe that, essentially, this is due to the fact that ECCE employs very sophisticated techniques, such as those based on homeomorphic embeddings, for controlling the unfolding and the generalization steps, and ensuring the termination of the specialization process. For a fair comparison, however, we should recall that Determinization may not terminate on examples different from those considered in this paper.

We have already mentioned in Section 3.3 that the performance of the programs derived by the Determinization Strategy may be further improved by applying post-processing transformations which
exploit the semideterminism of the programs. In particular, we may: (i) reorder the clauses so that unit clauses appear before non-unit clauses, and (ii) remove disequations by introducing cuts instead. The reader may verify that these transformations preserve the operational semantics. For a systematic treatment of cut introduction, the reader may refer to [10, 13]. As an example we now show the program obtained from \( \text{Match}_\text{Pos} \) (see Section 3.3) after the above post-processing transformations have been performed.

\[
\begin{align*}
\text{match}_\text{pos}(S, N) & \leftarrow \text{new}1(S, N) \\
\text{new}1([a]S, M) & \leftarrow !, \ \text{new}2(S, M) \\
\text{new}1([C]S, s(N)) & \leftarrow \text{new}1(S, N) \\
\text{new}2([a]S, M) & \leftarrow !, \ \text{new}3(S, M) \\
\text{new}2([C]S, s(s(N))) & \leftarrow \text{new}1(S, N) \\
\text{new}3([a]S, s(M)) & \leftarrow !, \ \text{new}3(R, S) \\
\text{new}3([b]S, M) & \leftarrow !, \ \text{new}4(R, S) \\
\text{new}3([C]S, s(s(s(N)))) & \leftarrow \text{new}1(S, N) \\
\text{new}4(S, 0) & \leftarrow \\
\text{new}4([a]S, s(s(s(M)))) & \leftarrow !, \ \text{new}2(S, M) \\
\text{new}4([C]S, s(s(s(N)))) & \leftarrow \text{new}1(S, N)
\end{align*}
\]

In Table 2 below we report the speedups obtained by partial deduction, conjunctive partial deduction, Determinization, and Determinization followed by disequation removal and cut introduction. Every speedup is computed as the ratio between the timing of the initial program and the timing of the specialized program. These timings were obtained by running the various programs several times (up to 10,000) on significantly large input lists (up to 4,000 items).

| Program       | Static Input                                      | Speedup (PD) | Speedup (CPD) | Speedup (Det) | Speedup (Det & Cut) |
|---------------|---------------------------------------------------|--------------|---------------|---------------|---------------------|
| \textit{Naive\_Match} | \textit{naive\_match([aab], S)}                  | 3.1          | 5.8\times 10^4 | 3.0\times 10^4 | 6.8\times 10^4     |
| \textit{Naive\_Match} | \textit{naive\_match([aaaaaaab], S)}             | 3.3          | 6.9\times 10^4 | 5.8\times 10^4 | 12.4\times 10^4    |
| \textit{Match\_Pos}   | \textit{match\_pos([aab], S, N)}                 | 1.6          | 3.6\times 10^3 | 1.8\times 10^3 | 4.0\times 10^3     |
| \textit{Match\_Pos}   | \textit{match\_pos([aaaaaaab], S, N)}            | 2.1          | 5.3\times 10^3 | 2.9\times 10^3 | 8.1\times 10^3     |
| \textit{Mmatch}       | \textit{mmatch([aaa], [aab], S, N)}              | 1.7          | 4.5\times 10^3 | 3.5\times 10^3 | 6.2\times 10^3     |
| \textit{Mmatch}       | \textit{mmatch([aa], [aaa], [aab], S, N)}       | 1.6          | 2.5\times 10^3 | 3.9\times 10^3 | 5.4\times 10^3     |
| \textit{Reg\_Expr}    | \textit{in\_language(aa\ast(b+bb))}, S          | 29.8         | 6.2\times 10^7 | 2.3\times 10^9 | 3.9\times 10^9     |
| \textit{Reg\_Expr}    | \textit{in\_language(a\ast(b+bb+bbb)), S}       | 1.3\times 10^4 | 3.3\times 10^4 | 4.6\times 10^4 | 5.7\times 10^4     |
| \textit{Reg\_Expr\_Match} | \textit{re\_match(aa\ast(b)), S}               | 5.7\times 10^4 | 2.7\times 10^4 | 1.5\times 10^6 | 3.0\times 10^6     |
| \textit{Reg\_Expr\_Match} | \textit{re\_match(a\ast(b + bb)), S}        | 2.1\times 10^2 | 3.4\times 10^3 | 2.5\times 10^9 | 4.1\times 10^9     |
| \textit{CF\_Parser}   | \textit{string\_parse(g, s), W}                 | 1.5          | 1.5           | 87.1          | 87.1               |
| \textit{CF\_Parser}   | \textit{string\_parse(g1, s), W}                | 1.1          | 1.1           | 61.3          | 61.3               |

Table 2: Speedups.

To clarify the content of Table 2 let us remark that:

Column 1 shows the names of the initial programs with reference to Sections 3.3, 5.3, and 7.

Column 2 shows the static input. 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 \textit{string\_parse} atom denotes the regular grammar considered in Example 7.5. The argument \(g_1\) of the last \textit{string\_parse} atom denotes the regular grammar:

\[\{s \rightarrow u, \ s \rightarrow 1, \ u \rightarrow 0, \ u \rightarrow 0v, \ u \rightarrow 0w, \ v \rightarrow 1, \ v \rightarrow 0v, \ v \rightarrow 1u, \ w \rightarrow 1, \ w \rightarrow 1w\}\]

Column 3, called Speedup (PD), shows the speedups we have obtained after the application of partial deduction.
Column 4, called Speedup (CPD), shows the speedups we have obtained after the application of conjunctive partial deduction.

Column 5, called Speedup (Det), shows the speedups we have obtained after the application of the Determinization Strategy.

Column 6, called Speedup (Det & Cut), shows the speedups we have obtained after the application of the Determinization Strategy followed by the removal of disequations and the introduction of cuts.

Let us now discuss our experimental results of Table 2. In all examples the best speedups are those obtained after the application of the Determinization Strategy followed by the removal of disequations and the introduction of cuts (see column Det & Cut).

As expected, conjunctive partial deduction gives higher speedups than partial deduction.

In some cases, conjunctive partial deduction gives better results than Determinization (see the first 5 rows of columns CPD and Det). This happens in examples where most nondeterminism is avoided by eliminating intermediate lists (see, for instance, the example of Section 3.3). In those examples, in fact, the Determinization Strategy may be less advantageous than conjunctive partial deduction because it introduces disequations which may be costly to check at runtime. However, as already mentioned, all disequations may be eliminated by introducing cuts (or, equivalently, if-then-else constructs) and the programs derived after disequation removal and cut introduction are indeed more efficient than those derived by conjunctive partial deduction (see column Det & Cut).

For some programs (see, for instance, the entries for Reg.Expr and CF.Parser) the speedups of the (Det) column are equal to the speedups of the (Det & Cut) column. The reason for this fact is the absence of disequations in the specialized program, so that the introduction of cuts does not improve efficiency.

We would like to notice that further post-processing techniques are applicable. For instance, similarly to the familiar case of finite automata, we may eliminate clauses corresponding to \(\varepsilon\)-transitions where no input symbols are consumed (such as clause 9 in program Match.Pos), and we may also 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 because they are outside the scope of the paper.

In summary, the experimental results of Table 2 confirm that in the examples we have considered, the Determinization Strategy followed by the removal of disequations in favour of cuts, achieves greater speedups than (conjunctive) partial deduction. However, it should be noticed that, as already mentioned, Determinization does not guarantee termination, while (conjunctive) partial deduction does, and in order to terminate in all cases, (conjunctive) partial deduction employs generalization techniques that may reduce speedups. In the next section we further discuss the issue of devising a generalization technique that ensures the termination of the Determinization Strategy.

9 Concluding Remarks and Related Work

We have proposed a specialization technique for logic programs based on an automatic strategy, called Determinization 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 several clauses are replaced by a single clause, 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 very important for reducing
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. To further increase program efficiency, in a post-processing phase these disequalities may be removed in favour of cuts.

We assume that the initial program to be specialized is associated with a mode of use for its predicates. Our Determinization Strategy makes use of this mode information for directing the various transformation steps, and in particular, the applications of the unfolding and case split rules. Moreover, if our strategy terminates, it derives specialized programs which are semideterministic w.r.t. the given mode. This notion has been formally defined in Section 5.3. Although semideterminism is not in itself a guarantee for efficiency improvement, it is often the case that efficiency is increased because nondeterminism is reduced and redundant computations are avoided.

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 [14, 40, 46].

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, related to the modes of the predicates, both on the programs to be specialized and on the applicability of the transformation rules.

In Section 3 we have extensively discussed the fact that our specialization technique is more powerful than partial deduction [21, 29]. The main reason of the greater power of our technique is that it uses more powerful transformation rules. In particular, partial deduction 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). In this respect, our technique improves over conjunctive partial deduction [8], which is a specialization technique where new predicates are defined in terms of conjunctions of atoms.

We have implemented the Determinization Strategy in the MAP transformation system [39] and we have tested this implementation by performing several specializations of string matching and parsing programs. We have also compared the results obtained by using the MAP system with those obtained by using the ECCE system for (conjunctive) partial deduction [24]. Our computer experiments confirm that the Determinization Strategy pays off w.r.t. both partial deduction and conjunctive partial deduction.

Our transformation technique works for programs where the only negative literals which are allowed in the body of a clause, are disequations between terms. The extension of the Determinization Strategy to normal logic programs would require an extension of the transformation rules and, in particular, it would be necessary to use a negative unfolding rule, that is, a rule for unfolding a clause w.r.t. a (possibly nonground) negative literal different from a disequation. The correctness of unfold/fold transformation systems which use the negative unfolding rule has been studied in contexts rather different from the one considered here (see, for instance, the work on transformation of first order programs [42]) and its use within the Determinization Strategy requires further work.

The Determinization Strategy may fail to terminate for two reasons: (i) the Unfold-Simplify subsidiary strategy may apply the unfolding rule infinitely often, and (ii) the while-do loop of the Determinization Strategy may not terminate, because at each iteration the Define-Fold subsidiary strategy may introduce new predicates.

The termination of the Unfold-Simplify strategy can be guaranteed by applying the techniques for finite unfolding already developed for (conjunctive) partial deduction (see, for instance, [8, 23, 30]). Indeed, the unfolding rule used in this paper is similar to the unfolding rule used in partial deduction.

The introduction of an infinite number of new predicates can be avoided by extending various
methods based on generalization, such as those used in (conjunctive) partial deduction \cite{8, 13, 25, 37}. Recall that in conjunctive partial deduction we may generalize a predicate definition essentially by means of two techniques: (i) the replacement of a term by a variable, which is then taken as an argument of a new predicate definition, and (ii) the splitting of a conjunction of literals into subconjunctions (together with the introduction of a new predicate for each subconjunction). It has been shown that the use of (i) and (ii) in a suitably controlled way, allows conjunctive partial deduction to terminate in all cases. However, termination is guaranteed at the expense of a possibly incomplete specialization or a possibly incomplete elimination of the intermediate data structures.

In order to avoid the introduction of an infinite number of new predicate definitions while applying the Determinization Strategy, we may follow an approach similar to the one used in the case of conjunctive partial deduction. However, besides the generalization techniques (i) and (ii) mentioned above, we may also need (iii) the splitting of the set of clauses defining a predicate into subsets (together with the introduction of a new predicate for each subset). Similarly to the case of conjunctive partial deduction, it can be shown that suitably controlled applications of the generalization techniques (i), (ii), and (iii) guarantee the termination of the Determinization Strategy at the expense of deriving programs which may fail to be semideterministic.

We leave it for further research the issue of controlling generalization, so that we achieve the termination of the specialization process and at the same time we maximize the reduction of nondeterminism.

In the string matching examples we have worked out, our strategy is able to automatically derive programs which behave like Knuth-Morris-Pratt algorithm, in the sense that they generate a finite automaton from any given pattern and a general pattern matcher. 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 deduction techniques \cite{11, 13, 44} consists in the fact that we have started from naive, nondeterministic initial programs, while the corresponding derivations by partial deduction described in the literature, use initial programs which are deterministic. Our derivations also improve over the derivations performed by using supercompilation with perfect driving \cite{15, 47} and generalized partial computation \cite{12}, which start from initial functional programs which already incorporate some ingenuity.

A formal derivation of the Knuth-Morris-Pratt algorithm for pattern matching has also been presented in \cite{3}. 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 \cite{3.3}), 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.

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 \cite{17}). Forms of reasoning by cases have been incorporated in program specialization techniques such as the already mentioned supercompilation with perfect driving \cite{15, 47} and generalized partial computation \cite{12}. 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 \ref{sec:examples} 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 \cite{48} for the case of functional programs and the strategy for eliminating unnecessary variables \cite{38} for the case of logic programs). Moreover, our strategy derives specialized programs which avoid repeated subcomputations (see the Context-free Parsing example of Section \ref{sec:context-free}). In this respect our strategy is similar to the tupling strategy for functional programs \cite{34}.

Finally, our specialization strategy is related to the program derivation techniques called finite
differencing \cite{33} and incrementalization \cite{27}. These techniques use program invariants to avoid costly, repeated calculations of function calls. Our specialization strategy implicitly discovers and exploits program invariants when 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 program derivation methods based on program invariants. These methods, in fact, are presented in a very different framework.

This paper is an improved version of \cite{35}. 
Appendix A. Proof of Theorem 6

For the reader’s convenience, we rewrite the statement of Theorem 6.

Theorem 6 (Correctness of the Rules 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 \( P_0 \cup \text{Defs}_n \) such that: (i) \( P_0 \cup \text{Defs}_n \) is safe w.r.t. \( M \), (ii) \( P_0 \cup \text{Defs}_n \) 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 \( \text{Defs}_k \), 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 \( P_0 \cup \text{Defs}_n \) iff \( A \) succeeds in \( P_n \).

The proof of Theorem 6 will be divided in four parts, corresponding to Propositions 3, 4, 5, and 6 presented below.

Proposition 3 (Preservation of Safety) shows that program \( P_n \) derived according to the hypotheses of Theorem 6 is safe w.r.t. mode \( M \) (that is, Point (i) of the thesis of Theorem 6). Proposition 4 (Preservation of Modes) shows that \( P_n \) satisfies \( M \) (that is, Point (ii) of the thesis of Theorem 6). Propositions 5 (Partial Correctness) and 6 (Completeness) show the if part and the only-if part, respectively, of Point (iii) of the thesis of Theorem 6. For proving these propositions we will use various notions and lemmata which we introduce below.

A1. Preservation of Safety

In this section we prove that, if the transformation rules are applied according to the restrictions indicated in Theorem 6, 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 3 (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 \( P_0 \cup \text{Defs}_n \) such that: (i) \( P_0 \cup \text{Defs}_n \) 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 \( P_0 \cup \text{Defs}_n \) 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 \( \text{Defs}_n \) 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 4). 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
Case 5: P

Case 4: P

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 $h d(D_i)$ 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 $Y \in \text{vars}(u_1 \neq u_2)$ such that $X \in \text{vars}(Y \vartheta)$. By the inductive hypothesis, $C$ is safe and thus, $Y$ is either an input variable of $h d(C)$ or a local variable of $u_1 \neq u_2$ in $C$. We have that: (i) if $Y$ is an input variable of $h d(C)$ then $X$ is an input variable of $h d(D_i)$, 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: (B.1) an input variable of $H \vartheta_i$ or (B.2) a local variable of $u_1 \neq u_2$ in $C_i$. In case (B.1) $X$ is an input variable of $h d(D_i)$, which is equal to $H \vartheta_i$. In case (B.2) $X$ does not occur in $\vartheta_i$ and, since $\text{vars}(C) \cap \text{vars}(C_i) = \emptyset$, $X$ is a local variable of $(u_1 \neq u_2) \vartheta_i$, which is equal to $t_1 \neq t_2$, 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:

$$C. H \leftarrow G_1, \text{newp}(X_1, \ldots, X_h) \vartheta_i, G_2$$

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

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

using the following definition clauses in $\text{Defs}_k$:

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

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 $\text{newp}(X_1, \ldots, X_h) \vartheta_i$.

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 6). Let $\text{GenC}$ be a clause in $P_{k+1}$ of the form:

$$H \leftarrow Y = t, \text{Body}$$

derived from a clause $C$ in $P_k$ of the form:

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

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

Let us now prove that $\text{GenC}$ is safe. Let $X$ be a variable occurring in a disequation $t_1 \neq t_2$ in $\text{Body}$. 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 from the definition of safe head generalization it follows that $H$ 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 $\text{GenC}$, 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 \( \text{Case 7} \)) to a clause \( C \) in \( P_k \). Let us consider the following two clauses in \( P_{k+1} \):

\[
\begin{align*}
C_1 & : (H \leftarrow \text{Body})\{X/t\} \\
C_2 & : H \leftarrow X \neq t, \text{Body}.
\end{align*}
\]

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 \( \text{Body}\{X/t\} \). \( t_1 \neq t_2 \) is of the form \( (u_1 \neq u_2)\{X/t\} \) where \( u_1 \neq u_2 \) occurs in \( \text{Body} \). We consider two cases.

Case A: \( Y \in \text{vars}(t) \). 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: \( Y \notin \text{vars}(t) \). 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 \( X \neq t, \text{Body} \). If \( t_1 \neq t_2 \) occurs in \( \text{Body} \) 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 \( Y \in \text{vars}(t) \), either (ii.1) \( Y \) is an input variable of \( H \) or (ii.2) \( Y \) does not occur in \( H, \text{Body} \), 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 \( \vartheta \). 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 \( P_k \setminus \{C_1\} \) 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 \( m \geq 0 \) clauses:

\[
C_1, H \leftarrow G_1, t_1 \neq u_1, G_2
\]

\[
\vdots
\]

\[
C_m, H \leftarrow G_1, t_m \neq u_m, G_2
\]

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. \( \square \)
A.2. Preservation of Modes

Here we show that, if the program $P_0 \cup \text{Defs}_n$ satisfies a mode $M$ and we apply our transformation rules according to the restrictions indicated in Theorem 6, 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 $\text{mgu}(A_1, A_2)$ we denote a relevant mgu of two unifiable non-basic atoms $A_1$ and $A_2$. Similarly, by $\text{mgu}(t_1, t_2)$ we denote a relevant mgu of two unifiable terms $t_1$ and $t_2$. The length of the derivation $G_0 \quad \rightarrow \quad G_1 \quad \rightarrow \quad \ldots \quad \rightarrow \quad G_n$ is $n$. Given a program $P$ and a mode $M$ for $P$, we say that a derivation $G_0 \quad \rightarrow \quad G_1 \quad \rightarrow \quad \ldots \quad \rightarrow \quad G_n$ 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 1** Let $P$ be a program and $G_1$ a goal. If $G_1$ succeeds in $P$ with answer substitution $\vartheta$, then for all goals $G_2$, $(G_1, G_2) \rightarrow_P G_2 \vartheta$.

**Lemma 2** Let $P$ be a safe program w.r.t. mode $M$, let $\text{Eqs}$ 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

$$(\text{Eqs}, G_1, G_2) \quad \rightarrow^*_P (A', G')$$

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

$$(G_1, \text{Eqs}, G_2) \quad \rightarrow^*_P (A'', G'').$$

**Lemma 3** Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules $\text{R1}$-$\text{R3}$. Let $M$ be a mode for $P_0 \cup \text{Defs}_n$ such that: (i) $P_0 \cup \text{Defs}_n$ is safe w.r.t. $M$, (ii) $P_0 \cup \text{Defs}_n$ 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 $P_0 \cup \text{Defs}_n$ are consistent with $M$, then all derivations from $G$ using $P_k$ are consistent with $M$.

**Proof:** By Proposition 9 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 $P_0 \cup \text{Defs}_n$.

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 $(A_0, G_0) \quad \rightarrow^*_P (A_s, G_s)$ and $(A_s, G_s)$ does not satisfy $M$, then there exists a goal $(A_t, G_t)$ such that $(A_0, G_0) \quad \rightarrow^*_P (A_t, G_t)$ 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 $G$, if there exists a derivation $G \quad \rightarrow^*_P (A_r, G_r)$ of length $r$, such that $A_r$ does not satisfy $M$, then there exists $(A', G')$ such that $G \quad \rightarrow^*_P (A', G')$ and $A'$ does not satisfy $M$.

Let us consider the derivation $(A_0, G_0) \quad \rightarrow^*_P (A_s, G_s)$ of length $s$, such that $A_s$ does not satisfy $M$.

If $s = 0$ then $G$ is $(A_s, G_s)$ and $(A_0, G_0) \quad \rightarrow^*_P (A_s, G_s)$ where $A_s$ does not satisfy $M$.

If $s > 0$ then we may assume $A_0 \neq true$, 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 2.3, the derivation from $(A_0, G_0)$ to $(A_s, G_s)$ using $P_{k+1}$ is of the form:

$$(A_0, G_0) \quad \rightarrow^*_P (A', G') \quad \rightarrow^*_P (A_s, G_s)$$

By the inductive hypothesis there exists $(A', G')$ such that $(A_0, G_0) \quad \rightarrow^*_P (A', G')$ and $A'$ does not satisfy $M$. Thus, $(A_0, G_0) \quad \rightarrow^*_P (A', G')$. 

47
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:

$$(A_0, G_0) \xrightarrow{P_{k+1}} (bd(E), G_0)\text{mgue}(A_0, hd(E)) \xrightarrow{P_{k+1}} \ldots \xrightarrow{P_{k+1}} (A_s, G_s)$$

where $E$ is a renamed apart clause in $P_{k+1}$.

If $E \in P_k$ then $(A_0, G_0) \xrightarrow{P_k} (bd(E), G_0)\text{mgue}(A_0, hd(E))$ 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 $(A_0, G_0) \xrightarrow{P_k} (A_t, G_t)$ 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, $E \in \text{Defs}_n$ and (†) follows from the inductive hypothesis and the hypothesis that $P_0 \cup \text{Defs}_n$ 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 1 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 mgue $\vartheta_i$.

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:

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

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

$$( (D, G_1, bd(C_i), G_2)\vartheta_i, G_0)\eta_i \xrightarrow{P_k} (A', G')$$

Since $\vartheta_i$ is mgue($A, hd(C_i)$), $\vartheta_i$ is relevant, and $\text{vars}(G_0) \cap \text{vars}((A, hd(C_i))) = \emptyset$, we have that:

$$(D, G_1, bd(C_i), G_2, G_0)\vartheta_i\eta_i \xrightarrow{P_k} (A', G')$$

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

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

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

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

Since $A_0$ satisfies $M$, $C$ is safe, and $C_i$ is renamed apart, we have that $\text{vars}(D \text{mgue}(A_0, H)) \cap \text{vars}(A, hd(C_i)) = \emptyset$. Thus, $(D \text{mgue}(A_0, H) \text{mgue}(A \text{mgue}(A_0, H), hd(C_i))) = (D \text{mgue}(A_0, H))$ and we have that:

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

Now, by Lemma 2 there exists a goal $(A'', G'')$ such that:

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

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

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

$$(A_0, G_0) \xrightarrow{P_k} (D, G_1, A, G_2, G_0) \text{mgue}(A_0, H) \xrightarrow{P_k} (A'', G''')$$

for some goal $G'''$. 48
Case 3.3: We have derived the clause

\[
E \quad \text{following holds:}
\]

Thus, the derivation from \((A_0 = H, D, G_1, A = hd(C_i))\) succeeds in \(P_k\). It follows that, for some substitution \(\vartheta\),

\[
(A_0 = H, D, G_1, A = hd(C_i), bd(C_i), G_2, G_0) \quad \text{(by Lemma 1)}
\]

for some goal \(G''\). Thus,

\[
(A_0 = H, D, G_1, A, G_2, G_0)
\]

and therefore, by using clause \(C \in P_k\),

\[
(A_0, G_0) \quad \text{(by using clause } C \in P_k)\]

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 3). In particular, suppose that from the following clauses in \(P_k\):

\[
\begin{align*}
C_1. \ H & \leftarrow G_1, (A_1, K_1)\vartheta, G_2 \\
\vdots \\
C_m. \ H & \leftarrow G_1, (A_m, K_m)\vartheta, G_2
\end{align*}
\]

and the following definition clauses in \(Defs_k\):

\[
\begin{align*}
D_1. \ newp(X_1, \ldots, X_h) & \leftarrow A_1, K_1 \\
\vdots \\
D_m. \ newp(X_1, \ldots, X_h) & \leftarrow A_m, K_m
\end{align*}
\]

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 \(\Sigma\) of Definition 3 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:

\[
(A_0, G_0) \rightarrow P_{k+1} \quad (G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)\ mgu(A_0, H) \rightarrow P_{k+1} \quad (A_s, G_s)
\]

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

\[
(G_1, newp(X_1, \ldots, X_h)\vartheta, G_2, G_0)\ mgu(A_0, H) \rightarrow P_k \quad (A', G')
\]

There are two cases:

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

\[
(A_0, G_0) \quad \text{(by using clause } C_i \text{ in } P_k)
\]

Thus, (‡) holds.

Case B: There is no \((A'', G'')\) such that \(G_1\ mgu(A_0, H) \rightarrow P_k \quad (A'', G'')\) and \(A''\) does not satisfy \(M\). In this case \(G_1\ mgu(A_0, H)\) succeeds in \(P_k\), and thus, for some substitution \(\alpha\),
By Property $\Sigma$, we have that $\text{newp}(X_1,\ldots,X_h)\vartheta\alpha$ satisfies $M$. It can be shown the following fact. Let us consider the set of all definition clauses with head predicate $\text{newp}$ in $\text{Defs}_k$, for any $k \in \{0,\ldots,n\}$:

$$\begin{cases}
\text{newp}(X_1,\ldots,X_h) \leftarrow \text{Body}_1 \\
\vdots \\
\text{newp}(X_1,\ldots,X_h) \leftarrow \text{Body}_m
\end{cases}$$

If for a substitution $\beta$ and a goal $G$, the atom $\text{newp}(X_1,\ldots,X_h)\beta$ satisfies $M$ and $\text{newp}(X_1,\ldots,X_h)\beta,G \longrightarrow^s_{P_k} (A',G')$, 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 $(A_i,G_i)$ such that $\text{Body}_i,\beta,G \longrightarrow^s_{P_k} (A_i,G_i)$, where $A_i$ is a non-basic atom which does not satisfy $M$.

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

$$(A_0,G_0) \longrightarrow^s_{P_k} (A_i,K_i)\vartheta,G_2,G_0)\alpha \longrightarrow^s_{P_{k+1}} (A_i,G_i)$$

where $A_i$ is a non-basic atom which does not satisfy $M$ and thus, $(\dagger)$ holds.

**Case 3.4:** $E$ is derived by applying the head generalization rule. In this case $(\dagger)$ 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 1) from a clause $C$ in $P_k$. By Proposition 11 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:

$$\begin{align*}
C_1. & \quad (H \leftarrow D,B)\{X/t\} \\
C_2. & \quad H \leftarrow X \neq t,D,B
\end{align*}$$

where $X$ is an input variable of $H$, $X$ does not occur in $t$, and for all variables $Y \in \text{vars}(t)$, 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:

$$(A_0,G_0) \longrightarrow^s_{P_{k+1}} ((D,B)\{X/t\},G_0)\text{mgu}(A_0,H\{X/t\}) \longrightarrow^s_{P_{k+1}} (A_s,G_s)$$

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

$$((D,B)\{X/t\},G_0)\text{mgu}(A_0,H\{X/t\}) \longrightarrow^s_{P_k} (A',G')$$

By properties of $\text{mgu}$'s and Point (1) of the operational semantics, we have that:

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

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

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

and therefore:

$$A_0 = H, D, B, G_0 \longrightarrow^s_{P_k} (A',G')$$

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

$$(A_0,G_0) \longrightarrow_{P_k} (D,B,G_0)\text{mgu}(A_0,H) \longrightarrow^s_{P_k} (A',G')$$

and $(\dagger)$ 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:

$$(A_0,G_0) \longrightarrow^s_{P_{k+1}} (X \neq t,D,B,G_0)\text{mgu}(A_0,H) \longrightarrow^s_{P_{k+1}} (A_s,G_s)$$

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

$$(X \neq t,D,B,G_0)\text{mgu}(A_0,H) \longrightarrow^s_{P_k} (A',G')$$

Since the answer substitution for any successful disequation is the identity substitution, we have that:
Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules. Let $M$ be a mode for $P_0 \cup \text{Defs}_n$ such that: (i) $P_0 \cup \text{Defs}_n$ is safe w.r.t. $M$, (ii) $P_0 \cup \text{Defs}_n$ 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 $M$.

From Lemma 3 and Definition 2 we have the following proposition.

**Proposition 4 (Preservation of Modes)** Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules. Let $M$ be a mode for $P_0 \cup \text{Defs}_n$ such that: (i) $P_0 \cup \text{Defs}_n$ is safe w.r.t. $M$, (ii) $P_0 \cup \text{Defs}_n$ 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 $M$.

**A3. Partial Correctness**

For proving the partial correctness of the transformation rules w.r.t. the operational semantics (that is, Proposition 5), we will use the following two lemmata.

**Lemma 4** Let $P$ be a safe program w.r.t. mode $M$, let $\text{Eqs}$ be a conjunction of equations, and let $G_1$ be a goal without occurrences of disequations. For all goals $G_2$, if
\[
(\text{Eqs}, G_1, G_2) \implies^r_p G_2 \vartheta
\]
then either
\[
(G_1, \text{Eqs}, G_2) \implies^r_p G_2 \vartheta
\]
or there exists a goal $(A', G')$ such that $A'$ is a non-basic atom which does not satisfy $M$ and
\[
G_1 \implies^r_p (A', G').
\]

**Lemma 5** Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules. Let $M$ be a mode for $P_0 \cup \text{Defs}_n$ such that: (i) $P_0 \cup \text{Defs}_n$ is safe w.r.t. $M$, (ii) $P_0 \cup \text{Defs}_n$ 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 $G \implies^*_{P_k \cup \text{Defs}_n} \text{true}$ which is consistent with $M$, then $G \implies^*_{P_k \cup \text{Defs}_n} \text{true}$, that is, $G$ succeeds in $P_k \cup \text{Defs}_n$.

**Proof:** By hypotheses (i–iii), and Propositions 3 and 4 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
\[
\delta : (A_0, G_0) \implies^*_{P_k \cup \text{Defs}_n} \text{true}
\]
which is consistent with $M$. We will prove that:
\[
(A_0, G_0) \implies^*_{P_k \cup \text{Defs}_n} \text{true}
\]
The proof proceeds by induction on the length $s$ of the derivation $\delta$.

**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 $G \implies^{r + 1}_{P_k \cup \text{Defs}_n} \text{true}$ of length $r$ which is consistent with $M$, then $G \implies^{r + 1}_{P_k \cup \text{Defs}_n} \text{true}$.

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 2.3, the derivation \( \delta \) is of the form:

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

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

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 \( \delta \) is of the form:

\[
(A_0, G_0) \mapsto_{P_{k+1}} (bd(E), G_0) \mbox{mgu}(A_0, h_d(E)) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mbox{true}
\]

where \( E \) is a renamed apart clause in \( P_{k+1} \).

If \( E \in P_k \) then \( (A_0, G_0) \mapsto_{P_k} (bd(E), G_0) \mbox{mgu}(A_0, h_d(E)) \) 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 \( P_k \cup \mbox{Defs}_n \) 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 \( \mbox{Defs}_n \) of the form: \( \text{newp}(X_1, \ldots, X_h) \leftarrow B \) and the derivation \( \delta \) is of the form:

\[
(\text{newp}(t_1, \ldots, t_h), G_0) \mapsto_{\mbox{Defs}_n} (B\{X_1/t_1, \ldots, X_h/t_h\}, G_0) \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mbox{true}
\]

By the inductive hypothesis, we have that:

\[
(B\{X_1/t_1, \ldots, X_h/t_h\}, G_0) \mapsto_{P_k} \mbox{true}
\]

and thus,

\[
(\text{newp}(t_1, \ldots, t_h), G_0) \mapsto_{P_k \cup \mbox{Defs}_n} \mbox{true}
\]

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 11 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 \( \vartheta_i \).

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 \( \delta \) is of the form:

\[
(A_0, G_0) \mapsto_{P_{k+1}} ((D, G_1, bd(C_i), G_2) \vartheta_i, G_0) \vartheta_i \mapsto_{P_{k+1}} \ldots \mapsto_{P_{k+1}} \mbox{true}
\]

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

\[
((D, G_1, bd(C_i), G_2) \vartheta_i, G_0) \vartheta_i \mapsto_{P_k} \mbox{true}
\]

Since \( \vartheta_i \) is \( \mbox{mgu}(A, h_d(C_i)) \), \( \vartheta_i \) is relevant, and \( \mbox{vars}(G_0) \cap \mbox{vars}((A, h_d(C_i))) = \emptyset \), we have that:

\[
(D, G_1, bd(C_i), G_2, G_0) \vartheta_i \vartheta_i \mapsto_{P_k} \mbox{true}
\]

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

\[
(A = h_d(C_i), A_0 = H, D, G_1, bd(C_i), G_2, G_0) \mapsto_{P_k} \mbox{true}
\]

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

\[
(A_0 = H, A = h_d(C_i), D, G_1, bd(C_i), G_2, G_0) \mapsto_{P_k} \mbox{true}
\]

Since \( A_0 \) satisfies \( M \), \( C \) is safe, and \( C_i \) is renamed apart, we have that \( \mbox{vars}(D \mbox{mgu}(A_0, H)) \cap \mbox{vars}(A, h_d(C_i)) = \emptyset \). Thus, \( (D \mbox{mgu}(A_0, H) \mbox{mgu}(A \mbox{mgu}(A_0, H), h_d(C_i))) = (D \mbox{mgu}(A_0, H)) \) and we have that:

\[
(A_0 = H, D, A = h_d(C_i), G_1, bd(C_i), G_2, G_0) \mapsto_{P_k} \mbox{true}
\]
Now, by Lemma 4 there are the following two cases.

Case A. \((A_0 = H, D, G_1, A = \text{hd}(C_i), bd(C_i), G_2, G_0) \rightarrow P_k \text{ true}\)

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

\((A_0 = H, D, G_1, A, G_2, G_0) \rightarrow P_k \text{ true}\)

and thus, by using clause \(C\) in \(P_k\),

\((A_0, G_0) \rightarrow P_k \text{ true}\)

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

\((A_0 = H, D, G_1) \rightarrow P_k (A', G')\)

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

\(A_0 \rightarrow P_k (A', G'')\)

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 5). In particular, suppose that from the following clauses in \(P_k\):

\[
\begin{align*}
C_1. H & \leftarrow G_1, (A_1, K_1)\vartheta, G_2 \\
\cdots \\
C_m. H & \leftarrow G_1, (A_m, K_m)\vartheta, G_2
\end{align*}
\]

and the following definition clauses in \(\text{Defs}_k\):

\[
\begin{align*}
D_1. \text{newp}(X_1, \ldots, X_h) & \leftarrow A_1, K_1 \\
\cdots \\
D_m. \text{newp}(X_1, \ldots, X_h) & \leftarrow A_m, K_m
\end{align*}
\]

we have derived the clause \(E\) of the form:

\(E. H \leftarrow G_1, \text{newp}(X_1, \ldots, X_h)\vartheta, G_2\)

where Property \(\Sigma\) of Definition 5 holds, that is, each input variable of \(\text{newp}(X_1, \ldots, X_h)\vartheta\), 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 \(\delta\) is of the form:

\((A_0, G_0) \rightarrow P_{k+1} (G_1, \text{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0) \text{mgu}(A_0, H) \rightarrow P_{k+1} \text{ true}\)

By the inductive hypothesis, the following holds:

\((G_1, \text{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0) \text{mgu}(A_0, H) \rightarrow P_k \text{ true}\)

and therefore, for some substitution \(\alpha\),

\((A_0, G_0) \rightarrow P_k (\text{newp}(X_1, \ldots, X_h)\vartheta, G_2, G_0)\alpha \rightarrow P_k \text{ true}\)

By Property \(\Sigma\), we have that \(\text{newp}(X_1, \ldots, X_h)\vartheta\alpha\) satisfies \(M\).

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

\[
\begin{align*}
\text{newp}(X_1, \ldots, X_h) & \leftarrow \text{Body}_1 \\
\cdots \\
\text{newp}(X_1, \ldots, X_h) & \leftarrow \text{Body}_m
\end{align*}
\]

If for a substitution \(\beta\) for a goal \(G\), the atom \(\text{newp}(X_1, \ldots, X_h)\beta, G\) satisfies \(M\) and we have that \(\text{newp}(X_1, \ldots, X_h)\beta, G \rightarrow P_k \text{ true}\), then for some \(i \in \{1, \ldots, m\}\) we have that \(\text{Body}_i\beta \rightarrow P_k \text{ true}\).

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

\((A_0, G_0) \rightarrow P_k ((A_i, K_i)\vartheta, G_2, G_0)\alpha \rightarrow P_k \text{ true}\)
Case 3.4: $E$ is derived by applying the head generalization rule. In this case $(A_0, G_0) \mapsto_P^* true$ 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 7) from a clause $C$ in $P_k$. By Proposition 11 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 $\delta$ takes the form:

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

By the inductive hypothesis, we have that:

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

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

$$(A_0=H, X=t, D, B, G_0) \mapsto_P^* true$$

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:

$$(A_0=H, D, B, G_0) \mapsto_P^* true$$

Thus, by using clause $C \in P_{k}$,

$$(A_0, G_0) \mapsto_{P_{k}} (D, B, G_0)mgu(A_0, H) \mapsto_P^* true$$

Case B: $E$ is $C_2$. Thus, the derivation $\delta$ takes the form:

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

By the inductive hypothesis, we have that:

$$(X \neq t, D, B, G_0) mgu(A_0, H) \mapsto_P^* true$$

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

$$(D, B, G_0) mgu(A_0, H) \mapsto_P^* true$$

Thus, by using clause $C \in P_{k}$,

$$(A_0, G_0) \mapsto_P^* true$$

Case 3.6: $E$ is derived by applying the equation elimination rule. In this case $(A_0, G_0) \mapsto_P^* true$ 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 4.

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

$\square$

Proposition 5 (Partial Correctness) Let $P_0, \ldots, P_n$ be a transformation sequence constructed by using the transformation rules. Let $M$ be a mode for $P_0 \cupDefs_n$ such that: (i) $P_0 \cupDefs_n$ is safe w.r.t. $M$, (ii) $P_0 \cupDefs_n$ 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 $P_0 \cupDefs_k$. 

54
Proof: Suppose that a non-basic atom \( A \) which satisfies \( M \) has a successful derivation using \( P_k \). By Proposition 3, \( P_k \) satisfies \( M \) and, therefore, \( A \) has a successful derivation using \( P_k \) which is consistent with \( M \). Thus, the thesis follows from Lemma 5. \( \square \)

A4. Completeness

For the proofs of Propositions 3 (Preservation of Safety), 4 (Preservation of Modes), and 5 (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 6 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 [14, 46].

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

**Definition 14** (i) Given a derivation \( \delta \) of the form \( G_0 \rightarrow p G_1 \rightarrow p \ldots \rightarrow p G_z \), we denote by \( \lambda(\delta) \) the number of goals \( G_i \) in \( \delta \) such that \( G_i \) is of the form \((A, K)\) where \( A \) is a non-basic atom. (ii) We define the following functions \( \mu \) and \( \nu \) which given a program and a goal return either a non-negative integer or \( \infty \) (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 \( G_0 \rightarrow_P G_1 \rightarrow_P \ldots \rightarrow_P G_z \) 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 \( \delta \) from \( G_1 \) to \( G_2 \) such that \( \lambda(\delta) = s \), then we write \( G_1 \rightarrow_P^s G_2 \).

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 6** 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 \( \delta \) from \( G \) using \( P \) such that \( \lambda(\delta) \leq \lambda(\delta') \) for all successful derivations \( \delta' \) from \( G \), is quasi-decreasing w.r.t. \( \succ_P \). \( \square \)

**Lemma 7** 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) \( (G_0, G_1, Eqs, G_2) \rightarrow_P^* \text{ true} \iff (G_0, Eqs, G_1, G_2) \rightarrow_P^* \text{ true} \)

(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. \( \square \)

**Lemma 8** Let \( M \) be a mode for program \( P \), such that \( P \) is safe w.r.t. \( M \) and \( P \) satisfies \( M \). Let \( \vartheta \) 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:
Proof: By induction on the length of the derivations. \hfill \Box

Lemma 9 Let $M$ be a mode for program $P$, such that $P$ is safe w.r.t. $M$ and $P$ satisfies $M$. Let $Diseq$ be a conjunction of disequations and $G$ be a goal. Suppose also that $\text{vars}(Diseq) \cap \text{vars}(G) = \emptyset$. Then:

(i) $(G, Diseq) \rightarrow_p^* \text{ true}$ \iff $(Diseq, G) \rightarrow_p^* \text{ true}$

(ii) $\mu(P, (G, Diseq)) = \mu(P, (Diseq, G))$

(iii) $\nu(P, (G, Diseq)) = \nu(P, (Diseq, G))$

Proof: The proof proceeds by induction on the length of the derivations. \hfill \Box

Let us consider a transformation sequence $P_0, \ldots, P_n$ constructed by using the transformation rules 11, 9 according to the hypothesis of Theorem 6. For reasons of simplicity we assume that each definition clause is used for folding, and thus, by Condition 1 of Theorem 6, 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 $P_0, \ldots, P_0 \cup \text{Defs}_n, \ldots, P_j, \ldots, P_l, \ldots, P_n$ such that: (1) $P_0, \ldots, P_0 \cup \text{Defs}_n$ is constructed by applications of the definition introduction rule, (2) $P_0 \cup \text{Defs}_n, \ldots, P_j$ is constructed by unfolding every clause in $\text{Defs}_n$, (3) $P_j, \ldots, P_l$ is constructed by applications of rules 3 and 4, 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 $P_0, \ldots, P_0 \cup \text{Defs}_n, \ldots, P_j, \ldots, P_l, \ldots, P_n$ constructed as indicated above. We also assume that $M$ is a mode for $P_0 \cup \text{Defs}_n$ such that: (i) $P_0 \cup \text{Defs}_n$ is safe w.r.t. $M$, (ii) $P_0 \cup \text{Defs}_n$ 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 2 and 4 for $k = 0, \ldots, n$, program $P_k$ is safe and satisfies $M$.

Lemma 10 Let us consider the transformation sequence $P_0, \ldots, P_0 \cup \text{Defs}_n, \ldots, P_j$ constructed as indicated above. Then the following properties hold.

(i) For all clauses $\text{newp}(X_1, \ldots, X_h) \leftarrow \text{Body}$ in $\text{Defs}_n$, for all substitutions $\vartheta$, and for all goals $G_1, G_2$, such that all derivations from $(G_1, \text{Body} \vartheta \vartheta, G_2)$ using $P_j$ are consistent with $M$, we have that:

(i.1) $(G_1, \text{Body} \vartheta \vartheta, G_2) \succeq_{P_j} (G_1, \text{newp}(X_1, \ldots, X_h) \vartheta, G_2)$;

(i.2) all derivations starting from $(G_1, \text{newp}(X_1, \ldots, X_h) \vartheta, G_2)$ using $P_j$ are consistent with $M$;

(ii) for all non-basic atoms $A$ satisfying $M$, if $A$ succeeds in $P_0 \cup \text{Defs}_n$ then $A$ succeeds in $P_j$.

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

Proof: By induction on the length of the derivations. \hfill \Box

For the proof of the following Lemma 12 we will use the following property.

Lemma 11 Let us consider the transformation sequence $P_j, \ldots, P_l$ and the mode $M$ for $P_0 \cup \text{Defs}_n$ as indicated above. For $k = j, \ldots, l$ and for all goals $G_1$ and $G_2$ such that there exists a derivation $G_1 \rightsquigarrow_{P_j} \ldots \rightsquigarrow_{P_k} G_2$, 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 $G_1 \rightsquigarrow_{P_k} \ldots \rightsquigarrow_{P_k} G_2$. We omit the details. \hfill \Box
Lemma 12 Let us consider the transformation sequence \( P_j, \ldots, P_l \) and the mode \( M \) for \( P_0 \cup \text{Defs}_n \) as indicated above. Let \( G \) be a goal such that (i) no disequation occurs in \( G \) and (ii) all derivations from \( P_j \) 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. \( \succ_p \).

Proof: Let us consider the following ordering on goals:
\[
G_1 \succ G_2 \text{ iff either } G_1 \succ_p G_2 \text{ or } G_1 \succeq_p G_2 \text{ and } \nu(P_j, G_1) > \nu(P_j, G_2).
\]
\( \succ \) is a well-founded order.
The proof proceeds by induction on \( k \).

Base Case. The case \( k = j \) follows from Lemma 6.

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. \( \succ_p \).

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:
\[
\delta : (A_0, G_0) \stackrel{P_k}{\longrightarrow} \ldots \stackrel{P_k}{\longrightarrow} \text{true}
\]
which is quasi-decreasing w.r.t. \( \succ_p \).
We wish to show that there exists a derivation of the form:
\[
\delta' : (A_0, G_0) \stackrel{P_{k+1}}{\longrightarrow} \ldots \stackrel{P_{k+1}}{\longrightarrow} \text{true}
\]
which is quasi-decreasing w.r.t. \( \succ_p \). We prove the existence of such a derivation \( \delta' \) by induction on the well-founded order \( \succ \).

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 \((A_0, G_0) \succ \hat{G} \), if there exists a derivation of the form:
\[
\hat{G} \longrightarrow_{P_k} \ldots \longrightarrow_{P_k} \text{true}
\]
which is quasi-decreasing w.r.t. \( \succ_p \), then there exists a derivation of the form:
\[
\hat{G} \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} \text{true}
\]
which is quasi-decreasing w.r.t. \( \succ_p \).

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 2.4 the derivation \( \delta \) is of the form:
\[
(t_1 = t_2, G_0) \longrightarrow_{P_k} G_0 \text{mgu}(t_1, t_2) \longrightarrow_{P_k} \ldots \longrightarrow_{P_k} \text{true}
\]
Let us consider the derivation:
\[
G_0 \text{mgu}(t_1, t_2) \longrightarrow_{P_k} \ldots \longrightarrow_{P_k} \text{true}
\]
By Proposition 5 we have that both \((t_1 = t_2, G_0)\) and \(G_0 \text{mgu}(t_1, t_2)\) succeed in \( P_j \). Moreover, by Point (1) of the operational semantics \( \nu(P_j, (t_1 = t_2, G_0)) > \nu(P_j, G_0 \text{mgu}(t_1, t_2)) \). Thus, \((t_1 = t_2, G_0) \succ G_0 \text{mgu}(t_1, t_2)\) and, by the inductive hypothesis (i2), there exists a successful derivation of the form:
\[
G_0 \text{mgu}(t_1, t_2) \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} \text{true}
\]
which is quasi-decreasing w.r.t. \( \succ_p \). Since \((t_1 = t_2, G_0) \succeq P_j G_0 \text{mgu}(t_1, t_2)\), the following derivation:
\[
(t_1 = t_2, G_0) \longrightarrow_{P_{k+1}} G_0 \text{mgu}(t_1, t_2) \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} \text{true}
\]
is quasi-decreasing w.r.t. \( \succ_p \).

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 \( \delta \) is of the form:
\[
(A_0, G_0) \longrightarrow_{P_k} (bd(C), G_0)\text{mgu}(A_0, bd(C)) \longrightarrow_{P_k} \ldots \longrightarrow_{P_k} \text{true}
\]
By Proposition 1 we may assume that clause $C$ is of the form $H \leftarrow \text{ Diseqs}, B$, where Diseqs is a conjunction of disequations and $B$ is a goal without occurrences of disequations. Thus, Diseqs $\text{ mgu}(A_0, H)$ succeeds and $\delta$ is of the form:

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

If $C \in P_{k+1}$ then $(A_0, G_0) \longrightarrow_{P_{k+1}} (\text{ Diseqs}, B, G_0) \text{ mgu}(A_0, H) \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} (B, G_0) \text{ mgu}(A_0, H)$ and the thesis follows from the inductive hypothesis (12), because $(A_0, G_0) \succ P_j (B, G_0) \text{ mgu}(A_0, H)$ (recall that $\delta$ is quasi-decreasing w.r.t. $\succ P_j$).

Otherwise, if $C \in (P_k - P_{k+1})$, we construct the derivation $\delta'$ 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 $H \leftarrow \text{ Diseqs}, G_1, A, G_2$. 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 $(H \leftarrow \text{ Diseqs}, G_1, \text{ bd}(C_i), G_2) \text{ mgu}(A, \text{ hd}(C_i))$. For reasons of simplicity we assume that for $i = 1, \ldots, m$, no disequation occurs in $\text{ bd}(C_i)$. In the general case where, for some $i \in \{1, \ldots, m\}$, $\text{ bd}(C_i)$ has occurrences of disequations, the proof proceeds in a very similar way, by using Proposition 1 Lemma 9 and the hypothesis that all applications of the unfolding rule are safe (see Definition 5).

The derivation $\delta'$ is of the form:

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

From the fact that $\delta$ 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:

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

and the derivation

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

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

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

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

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

Since all derivations from $(A_0, G_0)$ using $P_j$ are consistent with $M$, we have that all derivations from $(A_0 = H, \text{ Diseqs}, G_1)$ using $P_j$ are consistent with $M$, and therefore, by Lemma 9, 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 there exists a derivation

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

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

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

Now, since by Lemma 9 all clauses in $P_k$ are safe, we have that:

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

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

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

which is quasi-decreasing w.r.t. $\succ P_j$. Let $\vartheta_i$ be $\text{ mgu}(A, \text{ hd}(C_i))$ and $\eta_i$ be $\text{ mgu}(A_0, H \vartheta_i)$. By Points (1) and (2) of the operational semantics, we have that Diseqs $\vartheta_i, \eta_i$ succeeds and there exists a derivation of the form
\[(G_1, bd(C_i), G_2) \vartheta_i, G_0) \eta_i \rightarrow_{P_k} \cdots \rightarrow_{P_k} true\]

Moreover, we have that:
\[(A_0, G_0) \triangleright P_j ((G_1, bd(C_i), G_2) \vartheta_i, G_0) \eta_i \]

and thus, by the inductive hypothesis (I2), there exists a derivation of the form
\[((G_1, bd(C_i), G_2) \vartheta_i, G_0) \eta_i \rightarrow_{P_{k+1}} \cdots \rightarrow_{P_{k+1}} true\]

which is quasi-decreasing w.r.t. \(\triangleright P_j\).

Since \(\text{Diseqs} \vartheta_i \eta_i\) succeeds, by using clause \(D_i\) in \(P_{k+1}\) for the first step, we can construct the following derivation:
\[(A_0, G_0) \rightarrow_{P_{k+1}} ((\text{Diseqs}, G_1, bd(C_i), G_2) \vartheta_i, G_0) \eta_i \rightarrow_{P_{k+1}} \cdots \rightarrow_{P_{k+1}} true\]

which, by property (\(\ast\)), is quasi-decreasing w.r.t. \(\triangleright P_j\).

**Case 2.2:** \(P_{k+1}\) is derived from \(P_k\) by a safe application of the folding rule (see Definition 5). In particular, suppose that clause \(C\) is one of the following clauses occurring in \(P_k\):

\[
\begin{align*}
C_1. \ H & \leftarrow \text{Diseqs}, G_1, (A_1, K_1) \vartheta, G_2 \\
\vdots \\
C_m. \ H & \leftarrow \text{Diseqs}, G_1, (A_m, K_m) \vartheta, G_2
\end{align*}
\]

where \(\text{Diseqs}\) is a conjunction of disequations and no disequation occurs in \((G_1, G_2)\). We also suppose that the following definition clauses occur in \(Def_{S_k}\):

\[
\begin{align*}
D_1. \ newp(X_1, \ldots, X_h) & \leftarrow A_1, K_1 \\
\vdots \\
D_m. \ newp(X_1, \ldots, X_h) & \leftarrow A_m, K_m
\end{align*}
\]

and we have derived a clause \(E\) of the form:

\[E. \ H \leftarrow \text{Diseqs}, G_1, newp(X_1, \ldots, X_h) \vartheta, G_2\]

where Property \(\Sigma\) of Definition 5 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, \(P_{k+1} = (P_k - \{C_1, \ldots, C_m\}) \cup \{E\}\).

We may assume, without loss of generality, that clause \(C\) is \(C_1\), and the derivation \(\delta\) is of the form:

\[(A_0, G_0) \rightarrow_{P_k} (\text{Diseqs}, G_1, (A_1, K_1) \vartheta, G_2, G_0) mgu(A_0, H) \rightarrow_{P_k} \cdots \rightarrow_{P_k} true\]

Thus, \(\text{Diseqs} mgu(A_0, H)\) succeeds and, since \(\delta\) is consistent with \(M\), by Lemma 5 we have that \((G_1, (A_1, K_1) \vartheta, G_2, G_0) mgu(A_0, H)\) succeeds in \(P_j\).

Moreover, by Lemma 11 all derivations from \((G_1, (A_1, K_1) \vartheta, G_2, G_0) mgu(A_0, H)\) using \(P_j\) are consistent with \(M\).

Thus, by Lemmata 6 and 11 all derivations from \((G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) mgu(A_0, H)\) using \(P_j\) are consistent with \(M\) and there exists a derivation of the form:

\[(G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) mgu(A_0, H) \rightarrow_{P_j} \cdots \rightarrow_{P_j} true\]

which is quasi-decreasing w.r.t. \(\triangleright P_j\).

No disequation occurs in \((G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) mgu(A_0, H)\), and thus, by the inductive hypothesis (I1), there exists a derivation of the form:

\[(G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) mgu(A_0, H) \rightarrow_{P_k} \cdots \rightarrow_{P_k} true\]

which is quasi-decreasing w.r.t. \(\triangleright P_j\).

Since \(\delta\) is quasi-decreasing w.r.t. \(\triangleright P_j\), by Lemma 10 we also have that:
(A_0, G_0) \triangleright (G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) \text{mg}(A_0, H)

Thus, by the Inductive hypothesis (I2), there exists a derivation

(G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) \text{mg}(A_0, H) \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} \text{true}

which is quasi-decreasing w.r.t. \( \succ P_j \).

Since \( \text{Diseqs mg}(A_0, H) \) succeeds, by using clause \( E \in P_{k+1} \), we can construct the following derivation

(A_0, G_0) \longrightarrow_{P_{k+1}} (\text{Diseqs}, G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) \text{mg}(A_0, H) \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} \text{true}

which is quasi-decreasing w.r.t. \( \succ P_j \) because:

\[
(A_0, G_0) \succ P_j (\text{Diseqs}, G_1, (A_1, K_1) \vartheta, G_2, G_0) \text{mg}(A_0, H) \quad \text{(because \( \vartheta \) is quasi-decreasing)}
\]

\[
\succeq P_j (\text{Diseqs}, G_1, newp(X_1, \ldots, X_h) \vartheta, G_2, G_0) \text{mg}(A_0, H) \quad \text{(by Lemma 10)}
\]

**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 \( (H \leftarrow \text{Diseqs}, G_1, G_2) \vartheta \) and there exists a clause \( D \in P_k \) of the form \( H \leftarrow \text{Diseqs}, G_1 \). By Proposition 1 we may assume that no disequation occurs in \( G_1 \).

Thus, the derivation \( (\vartheta) \) is of the form:

\[
(A_0, G_0) \longrightarrow_{P_k} ((\text{Diseqs}, G_1, G_2) \vartheta, G_0) \text{mg}(A_0, H) \vartheta \longrightarrow_{P_k} \ldots \longrightarrow_{P_k} \text{true}
\]

Since all derivations starting from \( (A_0, G_0) \) using \( P_k \) are consistent with \( M \) and, by using clause \( D \), \( (A_0, G_0) \longrightarrow_{P_k} (\text{Diseqs}, G_1, G_0) \text{mg}(A_0, H) \), we have that all derivations starting from \( (\text{Diseqs}, G_1, G_0) \text{mg}(A_0, H) \) using \( P_k \) are consistent with \( M \). Moreover, no disequation occurs in \( G_0 \) and therefore, by Lemma 10 there exists a derivation

\[
(A_0, G_0) \longrightarrow_{P_k} (\text{Diseqs}, G_1, G_0) \text{mg}(A_0, H) \longrightarrow_{P_k} \ldots \longrightarrow_{P_k} \text{true}
\]

which is quasi-decreasing w.r.t. \( \succ P_j \). Thus, \( (\text{Diseqs mg}(A_0, H)) \) succeeds and there exists a derivation

\[
(G_1, G_0) \text{mg}(A_0, H) \longrightarrow_{P_k} \ldots \longrightarrow_{P_k} \text{true}
\]

which is quasi-decreasing w.r.t. \( \succ P_j \). Since \( (A_0, G_0) \triangleright (G_1, G_0) \text{mg}(A_0, H) \), by the inductive hypothesis (I2), there exists a derivation

\[
(G_1, G_0) \text{mg}(A_0, H) \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} \text{true}
\]

which is quasi-decreasing w.r.t. \( \succ P_j \). Since \( D \) belongs to \( P_{k+1} \) and \( (\text{Diseqs mg}(A_0, H)) \) succeeds, there exists a derivation

\[
(A_0, G_0) \longrightarrow_{P_{k+1}} (\text{Diseqs}, G_1, G_0) \text{mg}(A_0, H) \longrightarrow_{P_{k+1}} \ldots \longrightarrow_{P_{k+1}} \text{true}
\]

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 \( H \{X / t\} \leftarrow \text{Body} \) and \( P_{k+1} = (P_k - \{C\}) \cup \{\text{GenC}\} \), where clause \( \text{GenC} \) is of the form \( H \leftarrow X = t, \text{Body} \).

In this case we can show that we can construct the derivation \( \vartheta' \) 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 \( P, \mu(P, (t_1 = t_2, G)) = \mu(P, G \text{mg}(t_1, t_2)) \).

**Case 2.5:** \( P_{k+1} \) is derived from \( P_k \) by applying the safe case split rule (see Definition 7) to clause \( C \). By Proposition 1 we may assume that \( C \) is a clause of the form \( H \leftarrow \text{Diseqs}, B \), where \( \text{Diseqs} \) 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:

\[
C_1, \ (H \leftarrow \text{Diseqs}, B)\{X / t\}
\]

\[
C_2, \ H \leftarrow X \neq t, \text{Diseqs}, B
\]
Lemma 13 Let us consider the transformation sequence $P_j, \ldots, P_l$ and the mode $M$ for $P_0 \cup \text{Defs}_n$ 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 \[12\] 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 \[11\] $P_j$ satisfies $M$, and thus, all derivations starting from $A$ are consistent with $M$. □

Lemma 14 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 $P_0 \cup \text{Defs}_n$ then $A$ succeeds in $P_n$. 

61
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 \( P_0 \cup \text{Defs}_n \) is also a derivation using \( P_n \). \( \Box \)

**Proposition 6 (Completeness)** Let \( P_0, \ldots, P_n \) be a transformation sequence constructed by using the transformation rules and let \( p \) be a non-basic predicate in \( P_n \). Let \( M \) be a mode for \( P_0 \cup \text{Defs}_n \) such that: (i) \( P_0 \cup \text{Defs}_n \) is safe w.r.t. \( M \), (ii) \( P_0 \cup \text{Defs}_n \) 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 \( \text{Defs}_k \), 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 \( P_0 \cup \text{Defs}_n \) then \( A \) succeeds in \( P_n \).

**Proof:** Let us consider a transformation sequence \( P_0, \ldots, P_n \) constructed by using the transformation rules and according to conditions 1 and 2.

As already mentioned, we can rearrange the sequence \( P_0, \ldots, P_n \) into a new sequence \( P_0, \ldots, P_0 \cup \text{Defs}_n, \ldots, P_j, \ldots, P_1, \ldots, P_n \) such that: (1) \( P_0, \ldots, P_0 \cup \text{Defs}_n \) is constructed by applications of the definition introduction rule, (2) \( P_0 \cup \text{Defs}_n, \ldots, P_j \) is constructed by unfolding every clause in \( \text{Defs}_n \), (3) \( P_j, \ldots, P_1 \) is constructed by applications of rules and (4) either (4.1) \( l = n \) or (4.2) \( l = n - 1 \) and \( P_j \) is derived from \( P_{n-1} \) by an application of the definition elimination rule w.r.t. predicate \( p \).

Thus, Proposition follows from Lemmata and . \( \Box \)

**Appendix B. Proof of Proposition 2**

For the proof of Proposition 2 we need the following two lemmata.

**Lemma 15** 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 \( \sigma, r_i \sigma \neq s_i \sigma \) holds iff for \( i = 1, \ldots, k \), and for every ground substitution \( \sigma, r_i \sigma \) is a ground term different from \( s_i \sigma \) iff for \( i = 1, \ldots, k \), it does not exist a ground substitution \( \sigma \) such that for \( s_i \sigma \) 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 \( P \). \( \Box \)

**Lemma 16** 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 24), 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) \( (A_0, G_0) \Rightarrow_P (A_1, G_1) \), and (ii) the relation \( \Rightarrow_P \) 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 $\mu$, 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:

$$C_j: p(t_j, u_j) \leftarrow \text{grd}_j, K_j,$$

where: (i) $t_j$ is a tuple of terms denoting the input arguments of $p$ and (ii) the goal $\text{grd}_j$ 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 $\vartheta_j$. Since $A_0$ satisfies $M$, for $j=1, 2$, the input variables of $hd(C_j)$ are bound by $\vartheta_j$ 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 $\vartheta$ whose domain is a subset of $vars(t_1, t_2)$, and there exists a ground substitution $\sigma$ 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 $\alpha$) for $j=1, 2$, if we restrict $\vartheta\sigma$ to $vars(t_j)$ then $\vartheta_j = \vartheta\sigma$.

By hypothesis, both $\text{grd}_1\vartheta_1$ and $\text{grd}_2\vartheta_2$ succeed in $P$. Thus, by Lemma 15, every ground instance of $\text{grd}_1\vartheta_1$ and $\text{grd}_2\vartheta_2$ holds. (Recall that the goals $\text{grd}_1\vartheta_1$ and $\text{grd}_2\vartheta_2$ 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 $\text{grd}_j$ either occurs in $t_j$ or it is a local variable of that disequation in $C_j$. Thus, by Property ($\alpha$), $\text{grd}_1\vartheta_1 = \text{grd}_1\vartheta\sigma$ and $\text{grd}_2\vartheta_2 = \text{grd}_2\vartheta\sigma$. Since every ground instance of $\text{grd}_1\vartheta_1$ and $\text{grd}_2\vartheta_2$ holds, we have that every ground instance of $(\text{grd}_1\vartheta\sigma, \text{grd}_2\vartheta\sigma)$ holds. In other words, there exists a ground substitution $\sigma$ whose domain is $vars(t_1, t_2)$, such that every ground instance of $(\text{grd}_1\vartheta\sigma, \text{grd}_2\vartheta\sigma)$ holds. By definition, this means that $(\text{grd}_1, \text{grd}_2)\vartheta$ 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 $\mu$, 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: $(A_0, G_0) \mathcal{\rightarrow}_P (K, G_0)\mu$. 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 $(B, G_3) \mathcal{\rightarrow}_P G_4$, 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 $(Bs, G_2) \mathcal{\rightarrow}_P (A_1, G_1)$, 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 $(A_0, G_0) \mathcal{\rightarrow}_P G_5$, 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 the last part of Case (i). \hfill \Box

Now we give the proof of Proposition 7

Proof: Take a non-basic atom $A$ which satisfies $M$. Every non-basic atom $A_0$ such that $A \mathcal{\rightarrow}_P (A_0, G_0)$ for some goal $G_0$, satisfies $M$ because $P$ satisfies $M$. Since $P$ is linear, $G_0$ is a basic goal. By Lemma 16, there exists at most one goal $(A_1, G_1)$ where $A_1$ is a non-basic atom, such that $(A_0, G_0) \Rightarrow_\mathcal{P} (A_1, G_1)$. Thus, there exists at most one non-unit clause $C$ in $P$ such that $(A_0, G_0) \Rightarrow_\mathcal{C} (A_1, G_1)$. This means that $P$ is semideterministic w.r.t. $M$. \hfill \Box

Appendix C. Proof of Proposition 8

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$_{in}$. This is the case 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 $C$: $p(f(X), r(Y, Y), r(X, U)) \leftarrow \text{Body}$, with mode $p(+, +, ?)$ for $p$, we have that $C^{eq}$ is: $p(V, W, r(X)) \leftarrow V = f(X), W = r(Y, Y), \text{Body}$.

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

$$
\left\{
\begin{array}{l}
p(v_1, u_1) \leftarrow \text{Eqs}_1, \text{Diseqs}_1, \text{Body}_1 \\
\vdots \\
p(v_n, u_n) \leftarrow \text{Eqs}_n, \text{Diseqs}_n, \text{Body}_n 
\end{array}
\right.
$$

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) $\text{Eqs}_i$ denotes a conjunction of equations of the form $X = t$, which bind the variables in $v_i$, (4) $\text{Diseqs}_i$ denotes a conjunction of disequations, and (5) $\text{Body}_i$ 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 $\text{Body}_i$, 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 $\text{Eqs}_i$ or $\text{Diseqs}_i$ 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 $C^{eq}_2$, $C^{eq}_{21}$, and $C^{eq}_{22}$ of the clauses $C_2$, $C_{21}$, and $C_{22}$, respectively. We have that: (i) $bd(C^{eq}_{21})$ has one more equation of the form $X = r$ w.r.t. $bd(C^{eq}_2)$, and (ii) $bd(C^{eq}_{22})$ has one more disequation of the form $X \neq r$ w.r.t. $bd(C^{eq}_2)$. We also have that there exists only a finite number of pairs $(X, r)$, 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 $bd(C^{eq}_2)$. 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,

- in Case (1): (1.1) $X \neq r$ does not occur in $bd(C^{eq}_2)$ because $X/r$ is a binding of an mgu of the input arguments of $bd(C_1)$ and $bd(C_2)$, 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 $bd(C_2)$ 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 $bd(C^{eq}_2)$ 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 $bd(C^{eq}_2)$ because $X \neq r$ does not occur in $bd(C_2)$ (indeed, we choose $X \neq r$ precisely to satisfy this condition). \(\square\)
Acknowledgments

We would like to thank D. De Schreye, S. Etalle, J. Gallagher, R. Glück, N. D. Jones, M. Leuschel, B. Martens, and M. H. Sørensen for stimulating discussions about partial deduction and logic program specialization. We also acknowledge very constructive and useful comments by the anonymous referees. This work has been partially supported by the EC under the HCM Project ‘Logic Program Synthesis and Transformation’ and the Italian Ministry for Education, University, and Research.

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

[3] R. S. Bird, J. Gibbons, and G. Jones. Formal derivation of a pattern matching algorithm. Science of Computer Programming, 12:93–104, 1989.

[4] A. Bossi, N. Cocco, and S. Dulli. A method for specializing logic programs. ACM Transactions on Programming Languages and Systems, 12(2):253–302, April 1990.

[5] A. Bossi, N. Cocco, and S. Etalle. Transforming left-terminating programs. In A. Bossi, editor, Proceedings of the Ninth International Workshop on Logic-based Program Synthesis, LOPSTR’99, Venezia, Italy, September 22-24, 1999, Lecture Notes in Computer Science 1817, pages 156–175. Springer, 2000.

[6] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. Journal of the ACM, 24(1):44–67, January 1977.

[7] 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.

[8] 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.

[9] S. K. Debray and D. S. Warren. Automatic mode inference for logic programs. Journal of Logic Programming, 5:207–229, 1988.

[10] Y. Deville. Logic Programming: Systematic Program Development. Addison-Wesley, 1990.

[11] H. Fujita. An algorithm for partial evaluation with constraints. Technical Memorandum TM-0367, ICOT, Tokyo, Japan, 1987.

[12] Y. Futamura, K. Nogi, and A. Takano. Essence of generalized partial computation. Theoretical Computer Science, 90:61–79, 1991.

[13] J. P. Gallagher. Tutorial on specialisation 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.

[14] 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.
[15] 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.

[16] R. Glück and M. H. Sørensen. A roadmap to metacomputation by supercompilation. In O. Danvy, R. Glück, and P. Thiemann, editors, Partial Evaluation, Lecture Notes in Computer Science 1110, pages 137–160. Springer, 1996.

[17] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. Edinburgh LCF. Lecture Notes in Computer Science 78. Springer-Verlag, 1979.

[18] F. Henderson, Z. Somogyi, and T. Conway. Determinism analysis in the Mercury compiler. In Proceedings of the Australian Computer Science Conference, Melbourne, Australia, pages 337–346, 1996.

[19] M. V. Hermenegildo, F. Bueno, G. Puebla, and P. López. Program analysis, debugging, and optimization using the CIAO system preprocessor. In D. De Schreye, editor, Proceedings of the 1999 International Conference on Logic Programming, Las Cruces, NM, USA, Nov. 29 -Dec. 4, 1999, pages 52–66. MIT Press, 1999.

[20] J. Jaffar, M. Maher, K. Marriott, and P. Stuckey. The semantics of constraint logic programming. Journal of Logic Programming, 37:1–46, 1998.

[21] N. D. Jones, C. K. Gomard, and P. Sestoft. Partial Evaluation and Automatic Program Generation. Prentice Hall, 1993.

[22] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. SIAM Journal on Computing, 6(2):323–350, 1977.

[23] 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.

[24] M. Leuschel. The ECCE partial deduction system and the DPPD library of benchmarks, Release 3, Nov. 2000. Accessible via http://www.ecs.soton.ac.uk/~mal.

[25] 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.

[26] M. Leuschel, B. Martens, and D. de Schreye. Some achievements and prospects in partial deduction. ACM Computing Surveys, 30 (Electronic Section)(3es):4, 1998.

[27] Y. A. Liu. Efficiency by incrementalization: An introduction. Higher-Order and Symbolic Computation, 13(4):289–313, 2000.

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

[29] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. Journal of Logic Programming, 11:217–242, 1991.

[30] 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.

[31] C. S. Mellish. Some global optimizations for a Prolog compiler. Journal of Logic Programming, 2(1):43–66, 1985.

[32] C. S. Mellish. Abstract interpretation of Prolog programs. In S. Abramsky and C. Hankin, editors, Abstract Interpretation of Declarative Languages, chapter 8, pages 181–198. Ellis Horwood, 1987.
[33] R. Paige and S. Koenig. Finite differencing of computable expressions. *ACM Transactions on Programming Languages and Systems*, 4(3):402–454, 1982.

[34] A. Pettorossi. Transformation of programs and use of tupling strategy. In *Proceedings Informatica 77, Bled, Yugoslavia*, pages 1–6, 1977.

[35] 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.

[36] 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.

[37] M. Proietti and A. Pettorossi. The loop absorption and the generalization strategies for the development of logic programs and partial deduction. *Journal of Logic Programming*, 16(1–2):123–161, 1993.

[38] 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.

[39] S. Renault. A system for transforming logic programs. R 97–04, Department of Computer Science, University of Rome Tor Vergata, Rome, Italy, 1997.

[40] 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.

[41] D. Sahlin. Mixtus: An automatic partial evaluator for full Prolog. *New Generation Computing*, 12:7–51, 1993.

[42] T. Sato. An equivalence preserving first order unfold/fold transformation system. *Theoretical Computer Science*, 105:57–84, 1992.

[43] H. Sawamura and T. Takeshima. Recursive unsolvability of determinacy, solvable cases of determinacy and their application to Prolog optimization. In *Proceedings of the International Symposium on Logic Programming, Boston*, pages 200–207. IEEE Computer Society Press, 1985.

[44] 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.

[45] Z. Somogyi, F. Henderson, and T. Conway. The execution algorithm of Mercury: an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29(1–3):17–64, 1996.

[46] 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.

[47] V. F. Turchin. The concept of a supercompiler. *ACM TOPLAS*, 8(3):292–325, 1986.

[48] P. L. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.

[49] D. H. D. Warren. Implementing Prolog – compiling predicate logic programs. Research Report 39 & 40, Department of Artificial Intelligence, University of Edinburgh, 1977.