Preliminary version. Final version in ACM Transactions on Programming Languages and Systems (ACM Press), Vol. 20, No. 4, pp. 768-844, 1998

Partial Evaluation of Functional Logic Programs

MARÍA ALPUENTE
Universidad Politécnica de Valencia
MORENO FALASCHI
Università di Udine
and
GERMÁN VIDAL
Universidad Politécnica de Valencia

Languages that integrate functional and logic programming with a complete operational semantics are based on narrowing, a unification-based goal-solving mechanism which subsumes the reduction principle of functional languages and the resolution principle of logic languages. In this article, we present a partial evaluation scheme for functional logic languages based on an automatic unfolding algorithm which builds narrowing trees. The method is formalized within the theoretical framework established by Lloyd and Shepherdson for the partial deduction of logic programs, which we have generalized for dealing with functional computations. A generic specialization algorithm is proposed which does not depend on the eager or lazy nature of the narrower being used. To the best of our knowledge, this is the first generic algorithm for the specialization of functional logic programs. We study the semantic properties of the transformation and the conditions under which the technique terminates, is sound and complete, and is generally applicable to a wide class of programs. We also discuss the relation to work on partial evaluation in functional programming, term-rewriting systems, and logic programming. Finally, we present some experimental results with an implementation of the algorithm which show in practice that the narrowing-driven partial evaluator effectively combines the propagation of partial data structures (by means of logical variables and unification) with better opportunities for optimization (thanks to the functional dimension).

Categories and Subject Descriptors: D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming; D.1.6 [**Programming Techniques**]: Logic Programming; D.3.3 [**Language Classifications**]: Multiparadigm Languages; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program transformation*

General Terms: Languages, Performance, Theory

Additional Key Words and Phrases: Conditional term-rewriting systems, integration of functional and logic programming, narrowing strategies, partial evaluation

---

This work has been partially supported by CICYT TIC 95-0433-C03-03 and by HCM project CONSOLE.
A preliminary, short version of this article appeared in the *Proceedings of the European Symposium on Programming, ESOP'96* [Alpuente et al. 1996b].
Authors' addresses: M. Alpuente and G. Vidal, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain; email: {alpuente; gvidal}@dsic.upv.es; M. Falaschi, Dipartimento di Matematica e Informatica, Via delle Scienze 206, 33100 Udine, Italy; email: falaschi@dimi.uniud.it.
Permission to make digital/hard copy of all or part of this material without fee is granted provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery, Inc. (ACM). To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 1999 ACM 0164-0925/99/0100-0111 $00.75

1. INTRODUCTION

Partial evaluation (PE) is a semantics-preserving performance optimization technique for computer programs which consists of the specialization of the program with respect to parts of its input [Consel and Danvy 1993; Jones et al. 1993]. The resulting program may execute more efficiently than the initial one because, by using the partially known data, it is possible to avoid some run-time computations which are performed at specialization time once and for all. The main issues related to automatic PE (specialization) concern the preservation of the (operational) semantics, the termination of the process, and the effectiveness of the transformation, i.e., execution speedup for a large class of programs [Sørensen et al. 1996]. To perform reductions at specialization time, a partial evaluator normally includes an interpreter [Consel and Danvy 1993; Glück and Sørensen 1996]. This often implies that the transformation inherits the evaluation strategy from the underlying interpreter, and thus it is common to speak of *call-by-value* (*innermost*, eager) and *call-by-name* (outside-in, *lazy*) partial evaluation.

PE has been widely applied in the field of functional programming (FP) [Consel and Danvy 1993; Jones et al. 1993; Turchin 1986] and logic programming (LP) [Gallagher 1993; Komorowski 1982; Lloyd and Shepherdson 1991; Pettorossi and Proietti 1994], where it is usually called *partial deduction* (PD). Although the objectives are similar, the general methods are often different due to the distinct underlying computational models. Techniques in conventional partial evaluation of functional programs usually rely on the reduction of expressions and constant propagation, while transformation techniques for logic languages exploit unification-based parameter propagation [Glück and Sørensen 1994]. The folding and unfolding transformations, which were first introduced by Burstall and Darlington [1977] for functional programs, are exploited in different ways by the various PE techniques. Unfolding is essentially the replacement of a call by its definition, with appropriate substitutions. Folding is the inverse transformation, which is the replacement of some piece of code by an equivalent function call. In recent years, several correspondences have been established among the different techniques in particular cases [Glück and Sørensen 1994; 1996; Pettorossi and Proietti 1996; Sørensen et al. 1996].

Functional logic programming languages allow us to integrate some of the best features of the classical declarative paradigms, namely functional and logic programming. Modern functional logic languages offer features from functional programming (expressivity of functions and types, efficient reduction strategies, nested expressions) and logic programming (unification, logical variables, partial data structures, built-in search) [Dershowitz 1995; Hanus 1994b]. The operational semantics of integrated languages is usually based on *narrowing* [Slagle 1974], a combination of unification for parameter passing and reduction as an evaluation mechanism which subsumes rewriting and SLD-resolution. Narrowing is complete in the sense of functional programming (computation of normal forms) as well as logic programming (computation of answers).

In this article, we formalize the first generic algorithm for the specialization of functional logic (FL) languages. In contrast to the approach usually taken with pure functional languages, we use the unification-based computation mechanism of narrowing for the specialization of the program as well as for its execution. The

logic dimension of narrowing allows us to treat expressions containing partial information in a natural way, by means of logical variables and unification, while the functional dimension allows us to consider efficient evaluation strategies as well as to include a deterministic simplification phase which provides better opportunities for optimization and improves both, the overall specialization and the efficiency of the method. Narrowing-driven PE (NPE) is formalized within the theoretical framework established in Lloyd and Shepherdson [1991] and Martens and Gallagher [1995] for the partial deduction of logic programs, although a number of concepts have been generalized for dealing with functional features, such as user-defined functions, nested function calls, eager and lazy evaluation strategies, or deterministic reduction steps. Our method is parametric with respect to the narrowing strategy which is used for the automatic construction of (finite) narrowing trees. The different instances of our framework preserve program semantics under conditions ascertained by reusing the well-known methods and results developed for narrowing.

We are not aware of any formal antecedent of the narrowing-driven approach to PE in the literature, although the idea can be traced back to Darlington and Pull [1988]. A closer, automatic approach is that of positive supercompilation [Glück and Sørensen 1994; 1996; Sørensen et al. 1996], whose basic operation is *driving* [Turchin 1986], a unification-based transformation mechanism which is somewhat similar to (lazy) narrowing. Another related work is the recent framework of *conjunctive partial deduction* (CPD), which aims at achieving unfold/fold-like transformations within fully automated PD [Glück et al. 1996; Leuschel et al. 1996]. Our experiments with a prototypical implementation, as discussed in Section 7, demonstrate that NPE combines some good features of deforestation [Wadler 1990], partial evaluation [Consel and Danvy 1991; Jones et al. 1993], and PD [Lloyd and Shepherdson 1991; Martens and Gallagher 1995], similarly to conjunctive partial deduction [Glück et al. 1996; Leuschel et al. 1996] and supercompilation [Turchin 1986; Glück and Sørensen 1996]. Although our work shares many similarities with these works, it still has some notable differences which are discussed in this article. A different PE method for a rewriting-based, FL language is presented in Lafave and Gallagher [1997b].

A partial evaluator for a logic language is a mapping which takes a program P and a goal G and derives a more efficient, specialized program P' which gives the same answers for G (and any of its instances) as P does. Lloyd and Shepherdson [1991] put the foundations of partial deduction in a formal setting and identified the *closedness* and *independence* conditions which assure the strong correctness of the transformation (i.e., soundness and completeness with respect to computed answers).

As a start, we consider a naive adaptation of Lloyd and Shepherdson's closedness condition which guarantees that all calls that might occur during the execution of the transformed program are covered by some program clause. Assume that a term t is considered closed by the set S of partially evaluated calls (terms) if t is an instance of some term in S by a constructor substitution (i.e., a substitution that assigns constructor terms to each variable). This "plain" notion of closedness also works well in the narrowing-driven PE framework, and it allows us to prove the completeness of the transformation under the conditions for the completeness of the

narrower. However, it is too restrictive for functional (logic) programs, where nested defined function symbols appear quite frequently, and novel, specific techniques are needed for specializing them. Otherwise, a flattening of complex nested expressions into a conjunction of calls must be done, which gives worse operational behavior, as it diminishes the opportunities for deterministic simplification [Hanus 1992].

Therefore, we extend the Lloyd and Shepherdson's notion of closedness to recurse over the structure of the terms. For instance, the term $f(g(x))$ is considered closed with respect to the set of calls $\{f(y), g(x)\}$. This recursive closedness substantially enhances the specialization power of our method, since it supplies a notion of "similarity" which subsumes the standard, PD-like notion of closedness and the *perfect* ("α-identical") closedness test of Wadler [1990] and Sørensen et al. [1996], which folds only calls that are identical up to renaming. As a counterpart, some additional requirements are necessary for the completeness of the transformation.

Under the recursive closedness condition, it is also natural to expect that NPE inherits the correctness results of each narrowing strategy. This indeed holds for the result of soundness. However, we cannot prove in a way independent from the narrowing strategy the completeness of the transformation. These theorems are by their nature dependent on the particular strategy which is considered, as it is known that different narrowing strategies have quite different semantics properties [Hanus 1994b]. In spite of that, our proofs reveal an interesting connection between the completeness of NPE and the compositionality of the underlying narrowing semantics. Actually, recursive closedness essentially amounts to considering that the meaning of a complex expression $f(g(x))$ can be derived from the semantics of its "flattened" constituents $f(y)$ and $g(x)$. This can be thought of as a kind of implicit *splitting* of calls, which is comparable to the partitioning techniques of conjunctive partial deduction [Glück et al. 1996; Leuschel et al. 1996]. This is to say that NPE brings the same potential for specialization as the improvement which is achieved by conjunctive partial deduction through extending the (implicit) folding involved in the Lloyd and Shepherdson closedness condition by means of splitting.

With regard to *full conditional narrowing*, we prove the completeness of the transformation under the conditions for the completeness of *basic conditional narrowing*, a well-known (compositional) refinement of narrowing which ignores paths that reduce terms within what was a variable of the original goal. A requirement of linearity of the specialized calls is needed if the transformed program is required to compute exactly the same answers. We also ascertain the correct notion of independence which ensures that the *overlap* between different specialized definitions is not critical for the considered goals and which allows us to prove the strong soundness of the partially evaluated program. We start by considering the most general notion of narrowing (full conditional narrowing), since it brings to light some common problems which are caused by the basic mechanisms of NPE and which are not tied to any particular strategy. We use this notion also because the analysis which we present to determine the correctness conditions gives a kind of proof structure which may be of relevance for other similar studies. In fact, most of the notions and results we formalize for unrestricted narrowing can be easily particularized to more sophisticated narrowing strategies, such as innermost narrowing [Fribourg 1985] and lazy narrowing [Reddy 1985], in order to develop practical and efficient partial evaluators. For instance, in Section 6, we introduce a call-by-value

partial evaluator based on normalizing innermost narrowing, whose correctness is easily derived from that of the NPE method based on full conditional narrowing. In Alpuente et al. [1997], we studied the properties of a call-by-name instance of the framework for a nonstrict FL language with a lazy narrowing semantics.

Due to its basic strategy, a partial evaluator can loop in two ways: either by infinitely unfolding a function call or by creating infinitely many specialized definitions. We present an automatic NPE algorithm for functional logic programs which follows a structure similar to the framework developed by Martens and Gallagher [1995] for partial deduction, where a clear distinction between *local* and *global* control is done. Roughly speaking, we can say that local control concerns the construction of partial narrowing trees for single terms, while global control is devoted to ensuring the closedness of the partially evaluated program without risking nontermination. Our algorithm starts by partially evaluating the set of calls which appear in the initial goal, and then it recursively specializes the terms which are introduced dynamically during this process. We introduce appropriate unfolding and abstraction (generalization) operators which ensure (local as well as global) termination. Thus, our framework defines an automatic specialization method, which is independent from the narrowing strategy and which always terminates (for appropriate instances) and guarantees the closedness of the resulting transformed program. The recursive notion of closedness substantially enhances the specialization power of the method, since generalization needs to be applied in fewer cases than when a plain notion of closedness is used (unless special partitioning techniques such as those in Leuschel et al. [1996] were incorporated to the method, which would essentially require a similar compositionality proviso for ensuring correctness of the transformation). Using the terminology of Glück and Sørensen [1996], our global control strategy allows us to produce both *polyvariant* and *polygenetic* specializations, i.e., narrowing-driven partial evaluation can produce different specializations for the same function definition, and can combine distinct original function definitions into a comprehensive specialized function.

We also present an implementation of the NPE method which, by means of some experimental results, highlights the practical benefits of combining the propagation of partial information (by means of unification) with the deterministic simplification of goals (by term-rewriting). The system allows the user to select either a call-by-name or a call-by-value strategy, which emphasizes the generality of our approach.

This article is organized as follows. In Section 2, basic definitions of functional logic programming are given. In Section 3, we formalize by means of narrowing the notions of resultant and partial evaluation of functional logic programs and establish the computational equivalence of the original and the partially evaluated programs under appropriate closedness and independence conditions. Section 4 presents a general, narrowing-driven PE scheme and describe its properties. Partial correctness of the method is proved. In Section 5, we present our solution to the NPE termination problems and make use of a deterministic simplification process which brings up further possibilities for specialization. In Section 6, the choice of a normalizing innermost narrowing strategy allows us to formalize a call-by-value partial evaluator for functional logic programs. The practical applicability of the call-by-value partial evaluator is illustrated in Section 7 by means of a set of benchmarks. A short description of the system is given as well as an analysis

of the benchmark results. Section 8 discusses the relation to other research on partial evaluation in functional programming, term-rewriting systems, and logic programming. Finally, Section 9 concludes and presents some directions for future research. We include an appendix containing proofs of some technical results.

2. PRELIMINARIES

We briefly summarize some known results about rewrite systems and functional logic programming. We refer to Baader and Nipkow [1998], Dershowitz and Jouannaud [1990], Hanus [1994b], Hölldobler [1989], and Klop [1992] for extensive surveys. Definitions are given in the one-sorted case. The extension to many-sorted signatures is straightforward [Padawitz 1988].

A *signature* is a set Σ of *function symbols*. A natural number is associated with every $f \in \Sigma$ denoting its arity. The set $\tau(\Sigma \cup V)$ of *terms* built from a signature Σ and a countably infinite set of *variables* V is the smallest set such that $V \subset \tau(\Sigma \cup V)$ and if $f \in \Sigma$ has arity n and $t_1, \ldots, t_n \in \tau(\Sigma \cup V)$, then $f(t_1, \ldots, t_n) \in \tau(\Sigma \cup V)$. We let $\tau(\Sigma)$ denote the set of *ground* terms, i.e., terms containing no variables. We assume that the alphabet Σ contains some primitive symbols, including at least the nullary constructor *true* and a binary equality function symbol, say $=$, written in infix notation, which allows us to interpret *equations* $s = t$ as terms, with $s, t \in \tau(\Sigma \cup V)$. The term *true* is also considered an equation. $Var(s)$ is the set of distinct variables occurring in the term s. Identity of terms is denoted by $\equiv$.

Subterm occurrences are referred to by using the notion of *position*. The set $O(t)$ of positions in a term is inductively defined as follows:

$$O(t) = \begin{cases} \{\Lambda\} & \text{if } t \in V \\ \{\Lambda\} \cup \{i.u \mid 1 \leq i \leq n \ \wedge \ u \in O(t_i)\} & \text{if } t \equiv f(t_1, \ldots, t_n) \end{cases}$$

Then, positions are sequences of natural numbers, where Λ denotes the empty sequence. Positions are ordered by the *prefix* ordering: $u \leq v$, if there exists w such that $u.w = v$. Positions u, v are *disjoint*, denoted $u \perp v$, if neither $u \leq v$ nor $v \leq u$. $\overline{O}(t)$ denotes the set of *nonvariable positions* of the term t. For $u \in O(t)$, $t_{|u}$ is the *subterm* at position u of t. $t[s]_u$ is the term obtained from t by replacing the subterm at position u by s. These notions extend to sequences of equations in a natural way. For instance, the nonvariable position set of a sequence of equations $g \equiv (e_1, \ldots, e_n)$ can be defined as follows: $\overline{O}(g) = \{i.u \mid u \in \overline{O}(e_i), i = 1, \ldots, n\}$, where positions $1, 2, \ldots, n$ (addressing the equations in g) are called *root* positions.

A *substitution* is a mapping from V to $\tau(\Sigma \cup V)$ such that the set $Dom(\sigma) = \{x \in V \mid x\sigma \not\equiv x\}$, which is called the *domain* of σ, is finite. The *empty* substitution is denoted by ϵ. A substitution θ is *more general* than σ, in symbols $\theta \leq \sigma$, if there exists a substitution γ such that $\theta\gamma = \sigma$. This preorder induces a partial (pre-)ordering on terms given by $t \leq t'$ iff $\exists \gamma.\, t' \equiv t\gamma$. A *renaming* is a substitution ρ for which there exists the inverse ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \epsilon$. Two terms t and t' are *variants* (of each other), if there exists a renaming ρ such that $t\rho \equiv t'$. Given a substitution θ and a set of variables $W \subseteq V$, we denote by $\theta_{\restriction W}$ the substitution obtained from θ by restricting its domain to W. We write $\theta = \sigma\ [W]$ if $\theta_{\restriction W} = \sigma_{\restriction W}$, and $\theta \leq \sigma\ [W]$ denotes the existence of a substitution γ such that $\theta\gamma = \sigma\ [W]$. A set of equations E is unifiable, if there exists ϑ such that for all $s = t$ in E we have $s\vartheta \equiv t\vartheta$. ϑ is called a unifier of E. A *most general unifier* $\theta = mgu(E)$ of E always

exists (up to renaming), i.e., $\theta \leq \sigma$ for every other unifier σ of E (e.g., see Lassez et al. [1988]).

A *conditional term-rewriting system* (CTRS for short) is a pair $(\Sigma, \mathcal{R})$, where $\mathcal{R}$ is a finite set of reduction (or rewrite) rule schemes of the form $(\lambda \to \rho \Leftarrow C)$, λ, $\rho \in \tau(\Sigma \cup V)$, $\lambda \notin V$. The condition C is a (possibly empty) sequence $e_1, \ldots, e_n$, $n \geq 0$, of equations. Variables in C or ρ that do not occur in λ are called *extra variables*. We will often write just $\mathcal{R}$ instead of $(\Sigma, \mathcal{R})$. If a rewrite rule has no condition, we write $\lambda \to \rho$. Note that function symbols are allowed to be arbitrarily nested in left-hand sides. Following Middeldorp and Hamoen [1994], a CTRS whose rules $(\lambda \to \rho \Leftarrow C)$ do not contain extra variables, i.e., $Var(\rho) \cup Var(C) \subseteq Var(\lambda)$, is called a 1-CTRS. A CTRS with extra variables only in the conditions of their rewrite rules, i.e., $Var(\rho) \subseteq Var(\lambda)$, is called a 2-CTRS, while a 3-CTRS may also have extra variables in the rhs's provided these occur in the corresponding conditions, i.e., $Var(\rho) \subseteq (Var(\lambda) \cup Var(C))$. For CTRS $\mathcal{R}$, $r \ll \mathcal{R}$ denotes that r is a new variant of a rule in $\mathcal{R}$ such that r contains only *fresh* variables, i.e., contains no variable previously met during computation (standardized apart). See Hanus [1992; 1995] for semantics-preserving methods to translate logic programs into CTRS's which have at least the same efficiency and may have better control behavior.

A rewrite step is the application of a rewrite rule to an expression. A term s *conditionally rewrites* to a term t, written $s \to_{\mathcal{R}} t$, if there exist $u \in O(s)$, $(\lambda \to \rho \Leftarrow s_1 = t_1, \ldots, s_n = t_n) \in \mathcal{R}$, and substitution σ such that $s_{|u} \equiv \lambda\sigma$, $t \equiv s[\rho\sigma]_u$, and for all $i \in \{1, \ldots, n\}$ we have $s_i \downarrow_{\mathcal{R}} t_i$. Here $s_1 \downarrow_{\mathcal{R}} s_2$ holds if there exists a term t such that $s_1 \to^*_{\mathcal{R}} t$ and $s_2 \to^*_{\mathcal{R}} t$, where $\to^*_{\mathcal{R}}$ is the transitive and reflexive closure of $\to_{\mathcal{R}}$. The instantiated left-hand side $\lambda\sigma$ of a reduction rule $(\lambda \to \rho \Leftarrow C)$ is called a *redex* (*red*ucible *ex*pression) with *contractum* $\rho\sigma$. When no confusion can arise, we omit the subscript $\mathcal{R}$. A term s is a *normal form*, if there is no term t with $s \to_{\mathcal{R}} t$. We let $s\!\downarrow$ denote the normal form of s. A *normalized* substitution σ assigns normal forms to each variable, i.e., $x\sigma$ is a normal form for all $x \in Dom(\sigma)$. A CTRS $\mathcal{R}$ is *noetherian* if there are no infinite sequences of the form $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} t_3 \to_{\mathcal{R}} \ldots$. A CTRS $\mathcal{R}$ is *confluent* if, whenever a term s reduces to two terms t_1 and t_2, both t_1 and t_2 reduce to the same term; in symbols, $s \to^*_{\mathcal{R}} t_1$ and $s \to^*_{\mathcal{R}} t_2$ imply $t_1 \downarrow t_2$. For syntactic conditions guaranteeing confluence, we refer to Klop [1992] and Moreno-Navarro and Rodríguez-Artalejo [1992]. When $\mathcal{R}$ is noetherian and confluent it is called *canonical*.

An equational Horn theory $\mathcal{E}$ consists of a finite set of equational Horn clauses of the form $(\lambda = \rho) \Leftarrow C$. An equational goal is is a sequence of equations $\Leftarrow C$, i.e., an equational Horn clause with no head. We usually leave out the $\Leftarrow$ symbol when we write goals. A Horn equational theory $\mathcal{E}$, satisfying $\lambda \notin V$ for each clause $(\lambda = \rho) \Leftarrow C$, can be viewed as a CTRS $\mathcal{R}$, where the rules are the heads (implicitly oriented from left to right) and where the conditions are the respective bodies.

2.1 Functional Logic Programming

Functional logic languages are extensions of functional languages with principles derived from logic programming [Reddy 1985]. The computation mechanism of functional logic languages is based on *narrowing*, a generalization of term-rewriting where unification replaces matching: both the rewrite rule and the term to be

rewritten can be instantiated. Under the narrowing mechanism [Slagle 1974], functional programs behave like logic programs: narrowing solves equations by computing solutions with respect to a given CTRS, which is henceforth called the "program." Essentially, narrowing consists of the instantiation of goal variables, followed by a reduction step on the instantiated goal.

Definition 2.1 (*Narrowing*). Let $\mathcal{R}$ be a program and g be an equational goal. We say that g conditionally narrows into g' if there exists an occurrence $u \in \overline{O}(g)$, a standardized apart variant $r \equiv (\lambda \rightarrow \rho \Leftarrow C)$ of a rewrite rule in $\mathcal{R}$, and a substitution σ such that

—σ is the most general unifier of $g_{|u}$ and λ and

—$g' \equiv (C, g[\rho]_u)\sigma$.

We write $g \stackrel{[u,r,\sigma]}{\leadsto} g'$ or simply $g \stackrel{\sigma}{\leadsto} g'$. The relation $\leadsto$ is called (*unrestricted* or *full*) conditional narrowing.

Note that, if logical variables do not occur in the goal, the narrowing evaluation is akin to functional languages (term rewriting). However, strategies for functional reduction are usually deterministic, while narrowing is not. Analogously, evaluation is similar to logic languages if interpreted functions are not used [Dershowitz 1995; Hanus 1992; 1994b].

A term s is called a (narrowing) *redex* if and only if there exists a new variant $(\lambda \rightarrow \rho \Leftarrow C)$ of a rewrite rule in $\mathcal{R}$ and a substitution σ such that $s\sigma \equiv \lambda\sigma$. A narrowing *derivation* for g in $\mathcal{R}$ is defined by $g \stackrel{\theta}{\leadsto}^* g'$ if and only if $\exists\theta_1, \ldots, \exists\theta_n.\ g \stackrel{\theta_1}{\leadsto} \ldots \stackrel{\theta_n}{\leadsto} g'$ and $\theta = \theta_1 \ldots \theta_n$. We say that the derivation has length n. If $n = 0$, then $\theta = \epsilon$. The extension of a CTRS $\mathcal{R}$ with the rewrite rules for dealing with the equality is denoted by $\mathcal{R}_+$. In the case of unrestricted narrowing, $\mathcal{R}_+$ denotes $\mathcal{R} \cup \{x = x \rightarrow true\}$, $x \in V$.[1] This allows us to treat syntactical unification as a narrowing step, by using the rule $(x = x \rightarrow true)$ to compute *mgu*'s. Then $s = t \stackrel{\sigma}{\leadsto} true$ holds if and only if $\sigma = mgu(\{s = t\})$.

Example 2.2. Let us consider the program

$$\begin{array}{rcll} even(0) & \rightarrow & t & \\ even(s(x)) & \rightarrow & odd(x) & \\ odd(x) & \rightarrow & t & \Leftarrow\ even(x) = f \\ odd(x) & \rightarrow & f & \Leftarrow\ even(x) = t \end{array}$$

and the goal $even(s(y)) = t$. The following derivation shows that conditional

[1]Specialization in lazy functional languages requires $\mathcal{R}_+$ to be defined as $\mathcal{R}_+ = (\mathcal{R} \cup \text{StrEq})$, where StrEq are the rules for the *strict equality*, i.e., the rules which give to equality the weak meaning of identity of finite objects (e.g., see Moreno-Navarro and Rodríguez-Artalejo [1992]).

narrowing computes the answer $\{y/s(0)\}$ (selected redex underlined at each step):

$$\begin{array}{rl}\underline{even(s(y))} = t \overset{\epsilon}{\leadsto} & \underline{odd(y)} = t \\ \overset{\epsilon}{\leadsto} & \underline{even(y)} = f,\ t = t \\ \overset{\sigma_1}{\leadsto} & \underline{odd(x)} = f,\ t = t \\ \overset{\epsilon}{\leadsto} & \underline{even(x)} = t,\ f = f,\ t = t \\ \overset{\sigma_2}{\leadsto} & t = t,\ f = f,\ t = t \\ \overset{\epsilon}{\leadsto}{}^{*} & true,\ true,\ true\end{array}$$

Here $\sigma_1 = \{y/s(x)\}$ and $\sigma_2 = \{x/0\}$.

We use the symbol $\top$ as a generic notation for sequences of the form $true, \ldots, true$. A *successful* derivation (or refutation) for g in $\mathcal{R}_+$ is a narrowing derivation $g \overset{\theta}{\leadsto}{}^{*} \top$, where $\theta_{\lceil Var(g)}$ is the *computed answer substitution* (CAS). We define the *success set* operational semantics of an equational goal g in the program $\mathcal{R}$ as the set of the (normalized) computed answer substitutions corresponding to all successful narrowing derivations for g in $\mathcal{R}_+$:

$$\mathcal{O}_{\mathcal{R}}(g) = \{\theta_{\lceil Var(g)} \mid g \overset{\theta}{\leadsto}{}^{*} \top, \text{ and } \theta_{\lceil Var(g)} \text{ is normalized}\}$$

Considering normalized solutions is justified in a functional setting, since the "values" one is looking for are always constructor terms, hence normalized [Dershowitz 1995]. Evaluating a term s in the sense of functional programming corresponds to solving the equation $s = y$ [Baader and Nipkow 1998]. Narrowing derivations can be represented by a (possibly infinite) finitely branching tree. Following Lloyd and Shepherdson [1991], we adopt the convention in this work that any derivation is potentially incomplete (a branch thus can be failed, incomplete, successful, or infinite). A *failing leaf* is a goal which is not $\top$ and which cannot be further narrowed.

2.2 Completeness of Narrowing

Each equational Horn theory $\mathcal{E}$ generates a smallest congruence relation $=_{\mathcal{E}}$ called $\mathcal{E}$-*equality* on the set of terms (the least equational theory which contains all logic consequences of $\mathcal{E}$ under the entailment relation $\models$ obeying the axioms of equality for $\mathcal{E}$). Given two terms s and t, we say that they are $\mathcal{E}$-*unifiable* if there is a substitution σ such that $s\sigma =_{\mathcal{E}} t\sigma$. σ is called a $\mathcal{E}$-*unifier* of s and t. By abuse, it is often called *solution*. Let $=_{\mathcal{R}}$ be the reflexive, symmetric, and transitive closure of $\rightarrow_{\mathcal{R}}$. If $\mathcal{E}$ is the set of (conditional) equations corresponding to $\mathcal{R}$, then $=_{\mathcal{E}}$ and $=_{\mathcal{R}}$ coincide. Via this correspondence, the notion of $\mathcal{R}$-unification is implicitly defined. We say that $\theta \leq_R \sigma\ [W]$ if there is a substitution γ such that $\theta\gamma =_{\mathcal{R}} \sigma\ [W]$, i.e., $x\theta\gamma =_{\mathcal{R}} x\sigma$ for all $x \in W$.

A narrowing algorithm is *complete* for (a class of) CTRS's if it generates a solution at least as general as any that satisfies the query (it generates a complete set of $\mathcal{E}$-unifiers). Formally, if $\mathcal{E} \models g\sigma$ then there exists a refutation $g \overset{\theta}{\leadsto}{}^{*} \top$ in $\mathcal{R}$ such that $\theta \leq_{\mathcal{R}} \sigma\ [Var(g)]$. It is well known that the subscript $\mathcal{R}$ in $\theta \leq_{\mathcal{R}} \sigma$ can be dropped if we only consider completeness with respect to normalized substitutions. Conditional narrowing has been shown to be a complete algorithm for

CTRS's satisfying different restrictions [Hanus 1994b; Hölldobler 1989; Middeldorp and Hamoen 1994], e.g., for canonical 1-CTRS's (or confluent 1-CTRS's when normalized solutions are considered), for level-canonical (i.e., level-confluent and noetherian) 2-CTRS's, and for level-canonical 3-CTRS's. See Appendix A.4 for a precise definition of *level-confluence*, a property that ensures the confluence of the rewrite relation when it is decomposed into inductive layers. To simplify our notation, we let $I\!R_c$ denote the class of CTRS's satisfying these kinds of conditions. This class is quite general and contains most of the interesting programs. In this article, we always consider CTRS's of the class $I\!R_c$ as *programs*, unless we explicitly state differently.

Since unrestricted narrowing has quite a large search space, several strategies to control the selection of redexes have been devised to improve the efficiency of narrowing by getting rid of some useless derivations. A *narrowing strategy* (or *position constraint*) is any well-defined criterion which obtains a smaller search space by permitting narrowing to reduce only some chosen positions, e.g., *basic* [Hullot 1980], *innermost* [Fribourg 1985], *innermost basic* [Hölldobler 1989], *lazy* [Reddy 1985], or *needed* narrowing [Antoy et al. 1994]. A narrowing strategy φ is a mapping that assigns to every goal g (different from $\top$) a subset $\varphi(g)$ of $\overline{O}(g)$ such that for all $u \in \varphi(g)$ the goal g is narrowable at occurrence u. An important property of a narrowing strategy φ is completeness, meaning that the narrowing constrained by φ is still complete. There is an inherited trade off coming from FP, between the benefits of lazy evaluation of orthogonal, nonterminating rules and those of eager simplification with terminating, nonorthogonal rules [Dershowitz 1995]. A survey of results about the completeness of narrowing strategies can be found in Cheong and Fribourg [1992] and Echahed [1988].

We can further improve narrowing, without losing completeness, by normalizing the goal between narrowing steps [Hussman 1985], in the case of a confluent and *decreasing* CTRS $\mathcal{R}$. Decreasingness is a property which exactly captures the finiteness of the recursive evaluation of functions in conditional term-rewriting systems [Dershowitz 1995] (see Appendix A.4 for a formal definition). A *normalizing conditional narrowing step* with respect to $\mathcal{R}$, $g \overset{\sigma}{\leadsto} g'\!\downarrow$, is given by a narrowing step $g \overset{\sigma}{\leadsto} g'$ followed by a normalization $g' \rightarrow^*_{\mathcal{R}} g'\!\downarrow$. It is useful to apply rewriting between narrowing steps, whenever it is possible, since rewriting may cut an infinite search space to a finite one and can save a lot of time and space [Hanus 1991]. It also allows us to avoid some of the extra-logical control features of purely logic languages like the Prolog *cut* operator [Hanus et al. 1995].

The idea of exploiting deterministic computations by including normalization has been applied to almost all narrowing strategies, e.g., basic [Nutt et al. 1989; Réty 1987], innermost [Fribourg 1985], innermost basic [Hölldobler 1989], LSE [Bockmayr and Werner 1995], and lazy narrowing [Hanus 1994a].

3. PARTIAL EVALUATION OF FUNCTIONAL LOGIC PROGRAMS

In this section, we introduce a generic procedure for the partial evaluation of functional logic programs and show its correctness. Our construction follows mainly the theoretical framework established in Lloyd and Shepherdson [1991] and Martens and Gallagher [1995] for the partial deduction of logic programs. However, a number of concepts and results have been generalized in order to deal with (interpreted)

nested function calls. Since functional logic programs subsume functional and logic programs, some notions get more elaborate in comparison to pure functional or logic languages. What we gain in return is a clean and general framework which subsumes some of the basic developments in both fields. The conditions for the correctness of the transformation cover many practical cases and are easy to check, since they are mostly syntactical. For pure languages, these conditions essentially boil down to conditions for the correctness of the partial evaluation of pure functional or logic programs.

Given a program P and a set of atoms S, the aim of partial deduction [Lloyd and Shepherdson 1991] is to derive a new program P' which computes the same answers as P for any input goal which is an instance of an atom in S. The program P' is obtained by gathering together the set of *resultants*, which are constructed as follows: for each atom A of S, first construct a finite SLD-tree, $\tau(A)$, for $P \cup \{\Leftarrow A\}$,[2] then consider the leaves of the nonfailing branches of $\tau(A)$, say $G_1, \ldots, G_r$, and the computed substitutions along these branches, say $\theta_1, \ldots, \theta_r$, and finally construct the clauses: $A\theta_1 \Leftarrow G_1, \ldots, A\theta_r \Leftarrow G_r$. The restriction to specialize single atoms $A \in S$ is motivated by the fact that resultants are required to be Horn clauses. Correctness of the transformation is ensured whenever $P' \cup \{G\}$ is S-*closed*, i.e., every atom in $P' \cup \{G\}$ is an instance of an atom in S. An *independence* condition, which holds if no two atoms in S have a common instance, is needed to guarantee that the residual program P' does not produce additional answers.

First we formalize the basic notions involved in the partial evaluation of functional logic programs using conditional narrowing.

There are many choices to extend the notion of resultant to an integrated setting. Consider a narrowing derivation from the initial goal g to g' computing θ, where g is an equation. This would correspond to the restriction to perform partial deduction on single atoms used in Lloyd and Shepherdson [1991]. The associated PD-like resultant (i.e., the rule $g\theta \to true \Leftarrow g'$) is not useful for several reasons. Firstly, it does not specialize any user-defined function of the original program, but the equality symbol in the atomic equation g. Also, if we handle both equations and conjunctions of equations as boolean-valued terms and define resultants by $(g\theta \to g')$, then few opportunities to find *regularities* (i.e., to reach the closedness condition [Pettorossi and Proietti 1996]) are given, and very poor specialization can be achieved. This is mainly due to the abstraction operators which are commonly applied in order to guarantee termination of the PE process, which force the method to dramatically generalize calls, thus giving up the intended specialization (cf. Section 5.2). This can be avoided by introducing some particular techniques which are able to adequately reorder and split up complex expressions into smaller pieces before considering them for specialization. This has been proposed in conjunctive partial deduction [Glück et al. 1996; Leuschel et al. 1996] and recently applied to our framework in Albert et al. [1998]. A different solution would be to form the equational clause $g\theta \Leftarrow g'$ and then try to orientate the head (i.e., the equation $g\theta$) as a terminating rule, which is needed for the completeness of some narrowing strategies (and thus for the completeness of the PE methods based on

[2]We use $\Leftarrow$ instead of the more standard arrow $\leftarrow$ to denote logical implication, which makes the relation with CTRS's more explicit.

them). Unfortunately, this is not always possible, since it depends on the form of g (take for example the initial equation $x + y = y + x$ which is nonterminating as are each of its instances).

We start from the simpler but general idea of considering initial goals of the form $s = y$, where y is a fresh variable which does not appear in s. This way, we consider narrowing sequences for single terms s which (possibly) contain nested functions.

Definition 3.1 (*Resultant*). Let $\mathcal{R}$ be a CTRS, and let s be a term. Consider the equation $s = y$, with the variable $y \notin Var(s)$. Given the conditional narrowing derivation[3]

$$s = y \overset{\theta}{\rightsquigarrow}{}^{*} C, e$$

for the goal $s = y$ in $\mathcal{R}_+$, we define the resultant of this derivation as

$$((s \to y)\theta \Leftarrow C)\sigma$$

where $\sigma = mgu(\{e\})$.

Intuitively, from the definition of the narrowing relation, the equation e in the derived goal (C, e) either has the form $t = y$ or is $true$. By taking $\sigma = mgu(\{e\})$, we essentially "force" the free variable y to be instantiated in the derivation. Let us illustrate this definition through a few simple examples.

Example 3.2. Consider the following program rule:

$$f(x) \to x \Leftarrow a = g(x)$$

The resultants that correspond to the derivations

(1) $f(\underline{f(z)}) = y \overset{\{x/f(z)\}}{\rightsquigarrow} a = g(f(z)), \underline{f(z) = y} \overset{\{y/f(z)\}}{\rightsquigarrow} a = g(f(z)), true$

(2) $\underline{f(z)} = y \overset{\{z/x\}}{\rightsquigarrow} a = g(x), \underline{x = y} \overset{\{y/x\}}{\rightsquigarrow} a = g(x), true$

(3) $\underline{f(z)} = y \overset{\{z/x\}}{\rightsquigarrow} a = g(x), x = y$

are, respectively, the following program rules:

(1) $f(f(z)) \to f(z) \Leftarrow a = g(f(z))$
(2) $f(x) \to x \Leftarrow a = g(x)$
(3) $f(y) \to y \Leftarrow a = g(y)$

Note that, in the third case, if the mgu $\sigma = \{x/y\}$ of the last equation $x = y$ had not been computed and this equation had not been removed, we would have obtained the rule $(f(x) \to y \Leftarrow a = g(x), x = y)$, which contains an extra variable y in the rhs of the rule while the original program does not contain extra variables.

The following lemma gives a basic property of resultants. It shows the correspondence between following a derivation and using the simple resultant rule which corresponds to that derivation (the proof can be found in Appendix A.1).

[3]Here C denotes the equations brought by the conditions of the applied program rules, while e is the single equation which directly descends from the original equation $s = y$.

LEMMA 3.3. *Let $\mathcal{R}$ be a CTRS and $r' \equiv (s\theta \to t \Leftarrow C)$ be a resultant for s in $\mathcal{R}$, obtained from the following derivation:*

$$\mathcal{D}_s \equiv [s = y \overset{\theta}{\leadsto}{}^* C, t = y]$$

Let g be an equational goal such that there exists $u \in \overline{O}(g)$ with $g_{|u} \equiv s\gamma$. For every narrowing derivation for g in $\mathcal{R}$

$$g \overset{\sigma}{\leadsto}{}^* g'$$

which employs the same rules and in the same order as $\mathcal{D}_s$ does (at the corresponding positions), the narrowing step

$$g \overset{\upsilon}{\leadsto} g''$$

can be proven using the resultant r', where $g' = g''$ and $\sigma = \upsilon$ $[Var(g)]$.

The partial evaluation of a goal is defined by constructing incomplete search trees for the goal and extracting the specialized definition—the resultants—from the nonfailing root-to-leaf branches. We say that a resultant is *trivial* (and useless for specialization) if it has the form $s \to s$. Note that a trivial resultant is obtained from each narrowing tree, associated with the derivation $(s = y) \overset{\{y/s\}}{\leadsto} true$.

Definition 3.4 (*Narrowing-Driven Partial Evaluation*). Let $\mathcal{R}$ be a CTRS, s be a term, and $y \notin Var(s)$ be a fresh variable. Let τ be a finite (possibly incomplete) narrowing tree for the goal $s = y$ in the extended CTRS $\mathcal{R}_+$ containing at least one nonroot node.

Let $\{h_i \mid i = 1, \ldots, k\}$ be the nonfailing leaves of the branches of τ, and $\mathcal{R}' = \{r_i \mid i = 1, \ldots, m,\ m \leq k\}$ the nontrivial resultants associated with the derivations[4] $\{(s = y) \overset{\sigma_i}{\leadsto}{}^+ h_i \mid i = 1, \ldots, k\}$. Then, the set $\mathcal{R}'$ is called a *partial evaluation* of s in $\mathcal{R}$ (using τ).

If S is a finite set of terms (modulo variants), then a partial evaluation of S in $\mathcal{R}$ (or partial evaluation of $\mathcal{R}$ with respect to S) is the union of the partial evaluations in $\mathcal{R}$ of the elements of S. A partial evaluation of an equational goal $s_1 = t_1, \ldots, s_n = t_n$ in $\mathcal{R}$ is the partial evaluation in $\mathcal{R}$ of the set $\{s_1, \ldots, s_n, t_1, \ldots, t_n\}$.

From this definition, it may appear that some potential for specialization is lost due to the fact that the terms s_i and t_i of each equation $s_i = t_i$ of the goal are specialized independently. However, there is no real loss of generality in restricting our discussion to partial evaluation of terms, since the problem of solving an equation $s = t$ in the extended program $\mathcal{R}_+$ is equivalent to narrowing the "term" $s = t$ to *true*, and hence we can construct resultants for a goal of the form $(s = t) = y$. Moreover, if the conjunction operator "*and*" is defined by program rules [Moreno-Navarro and Rodríguez-Artalejo 1992], the problem of solving a conjunction like $(s_1 = t_1, \ldots, s_n = t_n)$ is equivalent to the problem of solving the equation $and(s_1 = t_1, and(\ldots, s_n = t_n)\ldots) = true$. A similar approach has been taken in Lafave and Gallagher [1997b], where nested function symbols are also

[4]We denote the transitive closure of $\leadsto$ by $\leadsto^+$, i.e., a narrowing derivation whose length is strictly greater than zero.

allowed and where the equality and the conjunction operators are defined in the program (and thus handled as ordinary function symbols). Unfortunately, the specialization of such complex goals (containing conjunctions of equations) in practice requires foreseeing refined control options which are able to properly handle (and take advantage of) the specific properties of these symbols, such as commutativity, associativity, etc., in order to find regularities, as we mentioned before. These refinements are very natural in the NPE framework (through the idea of "primitive function symbols"), but they are outside the scope of this work; we again refer to Albert et al. [1998] for details. Note that the assumption that the initial goal has the form $(s = y)$ not only simplifies the formal development of our framework but also guarantees that the rules which are the heads of the produced resultants are terminating whenever the original CTRS is noetherian, as guaranteed by the following result (the proof can be found in Appendix A.2).

PROPOSITION 3.5. *The n-CTRS obtained as the PE of a term in a noetherian n-CTRS is noetherian, for $n \in \{1, 2\}$.*

3.1 Basic Correctness of Narrowing-Driven PE

As shown by Middeldorp and Hamoen [1994], unrestricted conditional narrowing is complete for the class of programs $I\!R_c$. In this section, we show that the conditions that determine $I\!R_c$ suffice to ensure the soundness of the PE transformation, whereas a suitable closedness condition (similar to that of PD) is required for completeness. If we consider the straightforward embedding of logic programs into functional logic programming, by means of boolean-valued confluent functions (with constructor function calls as arguments) representing predicates, the results in this section are essentially equivalent to those for partial deduction of pure positive logic programs. In subsequent sections, we investigate the correctness of the transformation for a more general, recursive notion of closedness which arises naturally in a functional logic setting and which allows us to improve the specialization that can be obtained in our framework.

THEOREM 3.6. *Let $\mathcal{R}$ be a program, S a finite set of terms, and $\mathcal{R}'$ a partial evaluation of $\mathcal{R}$ with respect to S. Let g be a goal. Then, for every $\theta' \in \mathcal{O}_{\mathcal{R}'}(g)$ there exists $\theta \in \mathcal{O}_{\mathcal{R}}(g)$ such that $\theta \leq \theta'$ $[Var(g)]$.*

PROOF. Let us denote the equational theory presented by $\mathcal{R}$ as $\mathcal{E}_{\mathcal{R}}$. The soundness of conditional narrowing implies that, for each computed resultant $r \in \mathcal{R}'$, $\mathcal{E}_{\mathcal{R}} \models \mathcal{E}_{\{r\}}$. Hence, $\mathcal{E}_{\mathcal{R}} \models \mathcal{E}_{\mathcal{R}'}$. Also, if θ' is a computed answer substitution for g in $\mathcal{R}'$, from the soundness of conditional narrowing we have that $\mathcal{E}_{\mathcal{R}'} \models g\theta'$, and thus $\mathcal{E}_{\mathcal{R}} \models \mathcal{E}_{\mathcal{R}'} \models g\theta'$, which means that θ' is a unifier of g in $\mathcal{E}_{\mathcal{R}}$. Now, by the completeness of conditional narrowing for the considered programs, there exists a conditional narrowing derivation $g \overset{\theta}{\leadsto}^* \top$ in $\mathcal{R}$ such that $\theta \leq \theta'$ $[Var(g)]$. Hence $\theta \in \mathcal{O}_{\mathcal{R}}(g)$, which completes the proof. Note that the proof is immediately generalizable to any narrowing strategy, under the conditions for the completeness of the considered strategy. $\square$

However, we cannot prove completeness of the transformation for all programs. The following example motivates the need for the *closedness* condition.

Example 3.7. Let $\mathcal{R}$ be the canonical (as well as level-canonical) program:

$$\begin{array}{rcl} pred(s(0)) & \rightarrow & 0 \\ pred(s(s(x))) & \rightarrow & s(pred(s(x))) \end{array}$$

Let $S = \{pred(s(s(0)))\}$. Then, a PE $\mathcal{R}'$ of $\mathcal{R}$ with respect to S is

$$pred(s(s(0))) \rightarrow s(pred(s(0))).$$

(A depth-1 narrowing tree has been considered to extract the resultants.) Now, the goal $pred(s(s(0))) = s(0)$ is true in $\mathcal{R}$ whereas it does not hold in $\mathcal{R}'$. Note that the call $pred(s(0))$ (which appears in the rhs of the resultant rule in $\mathcal{R}'$) is not covered by the specialized call in S.

In logic programming, the partial deduction of a program with respect to S is required to be S-closed, and only S-closed goals are considered. Roughly speaking, we say that the notion of closedness guarantees that all calls which might occur during the execution of the resulting program are covered by some program rule. The following definitions are necessary for our notion of closedness. A function symbol $f \in \Sigma$ is *irreducible* if and only if there is no rule $(\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R}$ such that f occurs as the outermost function symbol in λ; otherwise it is a *defined* function symbol. In theories where the above distinction is made, the signature Σ is partitioned as $\Sigma = \mathcal{C} \biguplus \mathcal{F}$, where $\mathcal{C}$ is the set of irreducible function symbols and where $\mathcal{F}$ is the set of defined function symbols. The members of $\mathcal{C}$ are also called *constructors*. Terms in $\tau(\mathcal{C} \cup V)$ are called *constructor terms*. A substitution σ is a *(ground) constructor*, if $x\sigma$ is a (ground) constructor for all $x \in Dom(\sigma)$.

We let $\mathcal{R}_{calls}$ denote the set of outer calls appearing in the program $\mathcal{R}$, i.e., for each rule $(\lambda \rightarrow \rho \Leftarrow s_1 = t_1, \ldots, s_n = t_n)$ in $\mathcal{R}$, $\mathcal{R}_{calls}$ contains the terms $\rho, s_1, \ldots, s_n, t_1, \ldots, t_n$. Analogously, for $g \equiv (s_1 = t_1, \ldots, s_n = t_n)$, we let $g_{calls} = \{s_1, \ldots, s_n, t_1, \ldots, t_n\}$.

We start with a cautious, conservative extension of the closedness condition of Lloyd and Shepherdson [1991]. Roughly speaking, we say that a term t is S-closed if each outer subterm t' of t headed by a defined function symbol is a constructor instance of some term in S.

Definition 3.8 (*Basic Closedness*). Let S be a finite set of terms and t be a term. We say that t is S-closed if $closed^-(S, t)$ holds, where the predicate $closed^-$ is defined inductively as follows:

$$closed^-(S,t) \Leftrightarrow \begin{cases} true & \text{if } t \in V, \text{ or } t \in \mathcal{C} \\ closed^-(S,t_1) \wedge \ldots \wedge closed^-(S,t_n) & \text{if } t \equiv c(t_1,\ldots,t_n),\ c \in \mathcal{C} \\ \exists s \in S.\ s\theta = t \ \wedge\ \theta \text{ is constructor} & \text{if } t \equiv f(t_1,\ldots,t_n),\ f \in \mathcal{F} \end{cases}$$

We say that a set of terms T is S-closed, written $closed^-(S, T)$, if $closed^-(S, t)$ holds for all $t \in T$, and we say that a CTRS $\mathcal{R}$ is S-closed if $closed^-(S, \mathcal{R}_{calls})$. Similarly, a goal g is S-closed if $closed^-(S, g_{calls})$.

The following example illustrates this definition.

Example 3.9. Consider $S = \{f(x), g(x)\}$, $\mathcal{F} = \{f, g\}$, and $\mathcal{C} = \{a, c\}$. The term $c(f(a), c(g(a), f(y)))$ is S-closed, since the subterms $f(a)$, $g(a)$, and $f(y)$ are constructor instances of the terms in S. However, the term $f(g(a))$ is not S-closed, since it is not a constructor instance of any term in S.

Now, we are ready to present our first (strong) completeness result for the narrowing-based PE transformation. Similarly to standard partial deduction, closedness is the only extra requirement we need to prove that $\mathcal{R}'$ does not lose computed answers.

THEOREM 3.10. *Let $\mathcal{R}$ be a CTRS, g a goal, S a finite set of terms, and $\mathcal{R}'$ a partial evaluation of $\mathcal{R}$ with respect to S such that $\mathcal{R}' \cup \{g\}$ is S-closed. Then, for all $\theta \in \mathcal{O}_{\mathcal{R}}(g)$, there exists $\theta' \in \mathcal{O}_{\mathcal{R}'}(g)$ such that $\theta' = \theta\ [Var(g)]$.*

We prove this theorem by showing that narrowing chains in the original program can be mimicked by the application of appropriate resultants in the specialized program. The only previous result we use is the independence of selection functions which allows one to prove the *strong completeness* of conditional narrowing for programs in $I\!R_c$ [Hölldobler 1989; Middeldorp et al. 1996], since independence means that completeness is not lost when we restrict applications of the narrowing rule to a single equation in each goal, i.e., the choice of the equations is don't-care nondeterministic.

Definition 3.11 (*Selection Function*). A selection function is a mapping that assigns to every goal $g = (e_1, \ldots, e_n)$, $n > 0$, a natural number $i \in \{1, \ldots, n\}$ denoting an equation e_i in g different from $true$.

We say that a narrowing derivation $\mathcal{D}_{\mathcal{S}}$ *respects* a selection function $\mathcal{S}$ if the selected equation in every step $g \leadsto g'$ of $\mathcal{D}_{\mathcal{S}}$ coincides with $\mathcal{S}(g)$.

For unrestricted conditional narrowing with normalized CASes, completeness and strong completeness coincide, as formalized in the following lemma (see the proof in Appendix A.3). Middeldorp et al. [1996] prove a similar result for unconditional programs.

LEMMA 3.12. *Let $\mathcal{S}$ be an arbitrary selection function. For every conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^{*} \top]$ such that θ is normalized, there exists a conditional narrowing refutation $\mathcal{D}_{\mathcal{S}} \equiv [g \overset{\theta}{\leadsto}{}^{*} \top]$ which respects $\mathcal{S}$.*

A key point for the proof of the completeness theorem is the correct propagation of closedness through narrowing derivations. The following example reveals that the goal which derives from an S-closed goal might not be S-closed.

Example 3.13. Let us consider the program $\mathcal{R} = \{f(g(x)) \to x,\ g(0) \to 0\}$ and the set of calls $S = \{f(x), g(0)\}$. Then, a partial evaluation of $\mathcal{R}$ with respect to S is the specialized S-closed program $\mathcal{R}'$:

$$\mathcal{R}' = \{\ \begin{array}{rcl} f(g(x)) & \to & x \\ g(0) & \to & 0\ \ \} \end{array}$$

The following narrowing step can be proven for the goal $f(z) = z$ in $\mathcal{R}'$:

$$f(z) = z \overset{\{z/g(x)\}}{\leadsto} x = g(x)$$

Now, although $\mathcal{R}' \cup \{f(z) = z\}$ is S-closed, the derived goal $x = g(x)$ is not.

Fortunately, this does not cause a problem, since we need not ensure that all function-headed terms in a goal are S-closed, but only those which can be narrowed

in the considered refutation. For instance, in Example 3.13, it is clear that the term $g(x)$ cannot be narrowed in any refutation starting from $f(z) = z$ such that

$$f(z) = z \overset{\{z/g(x)\}}{\leadsto} x = g(x) \overset{\delta}{\leadsto}{}^{*} \top,$$

since the corresponding CAS $\{z/g(x)\}\delta$ would not be normalized.

Now we can proceed with the proof of Theorem 3.10.

PROOF. The following auxiliary notion is used to ease the proof. Given a derivation $\mathcal{D}$, a goal g is *S-closed in $\mathcal{D}$* if g' is S-closed, where g' is the goal obtained from g by replacing each defined function symbol heading a term which cannot be narrowed in $\mathcal{D}$ by a fresh constructor symbol with identical arity. Then we prove that, for any successful conditional narrowing derivation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^{*} \top]$ in $\mathcal{R}$ with θ normalized such that g is S-closed in $\mathcal{D}$, there exists a corresponding successful derivation $\mathcal{D}' \equiv [g \overset{\theta'}{\leadsto}{}^{*} \top]$ in $\mathcal{R}'$, such that $\theta' = \theta\ [Var(g)]$. We prove the claim by induction on the number n of steps in the former derivation.

If $n = 1$, then $(x = x \rightarrow true)$ is the only rule that has been applied in $\mathcal{D}$, and thus the same rule can be used in $\mathcal{R}'$ to build $\mathcal{D}'$.

Let us consider the inductive case $n > 1$. Assume $g \equiv (s_1 = t_1, \ldots, s_m = t_m)$, $m \geq 1$. Without loss of generality, we consider that s_1 is narrowed in $\mathcal{D}$. Since g is S-closed, there exists a term $s \in S$ such that $s_1 = s\gamma$ with γ constructor. By Lemma 3.12, we can assume a fixed left-to-right selection rule. Thus, we can consider $\mathcal{D}$ to be of the form

$$\begin{array}{rcl} (s\gamma = t_1, s_2 = t_2, \ldots, s_m = t_m) & \overset{[u_1,r_1,\delta_1]}{\leadsto} & (C^1, s^1 = t_1, s_2 = t_2, \ldots, s_m = t_m)\delta_1 \\ & \overset{[u_2,r_2,\delta_2]}{\leadsto} & \ldots \\ & \overset{[u_k,r_k,\delta_k]}{\leadsto} & (C^k, s^k = t_1, s_2 = t_2, \ldots, s_n = t_n)\delta \\ & \overset{\sigma}{\leadsto}{}^{*} & \top \end{array}$$

with $\theta = \delta\sigma$ and $\delta = \delta_1 \ldots \delta_k$, $k > 0$. Now, since γ is constructor, the derivation

$$[s = y \overset{[u_1,r_1,\vartheta_1]}{\leadsto} \ldots \overset{[u_k,r_k,\vartheta_k]}{\leadsto} (C', s' = y)], \quad \vartheta = \vartheta_1 \ldots \vartheta_k$$

can be built, which uses the same rules and in the same order as in the first k steps of $\mathcal{D}$ and produces the resultant $r' \equiv (s\vartheta \rightarrow s' \Leftarrow C') \in \mathcal{R}'$. Now, by Lemma 3.3, we have

$$(s\gamma = t_1, s_2 = t_2, \ldots, s_m = t_m) \overset{[1.1,r',\delta']}{\leadsto} (C', s' = t_1, s_2 = t_2, \ldots, s_m = t_m)\delta',$$

where $\delta' = mgu(\{s\gamma = s\vartheta\})$, $(C^k, s^k = t_1, s_2 = t_2, \ldots, s_m = t_m)\delta = (C', s' = t_1, s_2 = t_2, \ldots, s_m = t_m)\delta'$, and $\delta = \delta'\ [Var(g)]$. This implies the existence of the subderivation $\mathcal{D}'' \equiv [(C', s' = t_1, s_2 = t_2, \ldots, s_m = t_m)\delta' \overset{\sigma}{\leadsto}{}^{*} \top]$ in $\mathcal{R}$.

Finally, in order to apply the induction hypothesis on the subderivation $\mathcal{D}''$, we first need to prove that each narrowable subterm of $(C', s' = t_1, s_2 = t_2, \ldots, s_m = t_m)\delta'$ is S-closed in $\mathcal{D}$. Since $\mathcal{R}'$ is S-closed, then both s' and C' are S-closed. By definition of closedness, it is immediate that $(C', s' = t_1, s_2 = t_2, \ldots, s_m = t_m)$ is also S-closed. By construction, the substitution δ' can only introduce terms appearing in γ (which does not affect the closedness of the goal, since they are constructor) or terms appearing in ϑ (which cannot be reduced in $\mathcal{D}$, since $\theta_{\upharpoonright Var(g)}$

is normalized). Therefore, $(C', s' = t_1, s_2 = t_2, \ldots, s_m = t_m)\delta'$ is S-closed in $\mathcal{D}$, and the claim follows by the induction hypothesis. □

Theorems in this section show the relationship between the computational behaviors of the original program $\mathcal{R}$ and its specialization $\mathcal{R}'$. For successful goals, $\mathcal{R}'$ is in general weaker than $\mathcal{R}$: $\mathcal{R}'$ is able to compute the answers that $\mathcal{R}$ obtains only for goals satisfying the closedness condition. Also, if a goal succeeds in $\mathcal{R}'$, it also succeeds in $\mathcal{R}$, although $\mathcal{R}'$ can produce additional answers.[5]

In summary, the results in this section show that standard techniques for partial deduction transfer smoothly to functional logic programs, with conditions which are not more demanding than those for pure logic programs. That is, there is no penalty for the inclusion of functions into logic programs, if we restrict ourselves to a PD-like notion of closedness. However, we consider this notion to be too restrictive in a functional logic context, where terms containing nested function calls are most frequently found (this is different to partial deduction, where all function symbols are constructors and where nested predicate calls are not allowed). For instance, in Example 3.9, it would seem natural to expect that the call $f(g(x))$ be considered covered by the functions of the partially evaluated program which results from specializing the calls $f(x)$ and $g(x)$ of S. However, this call is not a constructor instance of any term in S, and hence no completeness result could be applied to the execution of a goal which includes this subterm.

3.2 Generalized Correctness of Narrowing-Based PE

In this section, we introduce a generalized notion of closedness which allows us to cope with nested function calls which cannot be obtained through instantiation of the specialized calls by constructor substitutions. This is interesting because we get more powerful correctness results which apply to many practical cases and that produce better specializations, as we will show in Sections 4 and 5, where we consider an automatic PE procedure and prove its termination. Intuitively, the use of a general notion of closedness implies a less frequent need for generalization, which diminishes the loss of information and improves the potential for specialization (cf. Section 4). Also, the recursive notion of closedness allows us to safely execute a wider class of input goals which are recursively covered by the set of specialized functions. The following example indicates that closedness cannot be coarsely generalized and motivates the recursive inspection of subterms to guarantee not losing completeness.

Example 3.14. Consider the following (canonical) program:

$$\mathcal{R} = \{ \quad \begin{array}{rcl} h(x) & \rightarrow & x \\ f(0) & \rightarrow & 0 \\ f(c(x)) & \rightarrow & h(f(x)) \ \} \end{array}$$

---

[5]By adding the condition that the specialized calls in S do not *overlap* (for an appropriate, term-rewriting notion of overlap) we are able to prove a stronger version of the soundness theorem. We do this in Section 3.3.

A PE of $\mathcal{R}$ with respect to $S = \{f(c(x)),\ h(x)\}$ is the specialized program

$$\mathcal{R}' = \{ \quad \begin{array}{rcl} h(x) & \rightarrow & x \\ f(c(x)) & \rightarrow & h(f(x)) \quad \}. \end{array}$$

Although each term appearing in $\mathcal{R}'$ is an instance of some term in S, the program $\mathcal{R}'$ should not be considered closed with respect to S since the call $f(x)$ occurring in the term $h(f(x))$ (which appears in the rhs of the second rule of $\mathcal{R}'$) is not covered sufficiently by the rules of $\mathcal{R}'$. Actually, the goal $f(c(0)) = 0$, which is closed with respect to S, succeeds in $\mathcal{R}$ with CAS ϵ whereas it fails in $\mathcal{R}'$.

The following definition formalizes a generalized closedness relation which recursively inspects all subterms in the rhs and the condition of each program rule.

Definition 3.15 (*Generalized Closedness*). Let S be a finite set of terms and t be a term. We say that t is S-closed if $closed^+(S,t)$ holds, where the predicate $closed^+$ is defined inductively as follows:

$$closed^+(S,t) \Leftrightarrow \begin{cases} true & \text{if } t \in V, \quad \text{or } t \in \mathcal{C} \\ closed^+(S,t_1) \wedge \ldots \wedge closed^+(S,t_n) & \text{if } t \equiv c(t_1,\ldots,t_n),\ c \in \mathcal{C} \\ \exists s \in S.\ s\theta = t \ \wedge \displaystyle\bigwedge_{x/t' \in \theta} closed^+(S,t') & \text{if } t \equiv f(t_1,\ldots,t_n),\ f \in \mathcal{F} \end{cases}$$

The notion of closedness extends to sets of terms and rules in the usual way.

Note that this relation is more general than the basic closedness in the sense that $closed^-(S,t)$ implies $closed^+(S,t)$, but not vice versa.

In the rest of the article, we refer to this general notion of closedness, unless we explicitly state otherwise. Roughly speaking, we say that a term t is considered closed by a set S, if t can be "flattened" (split) into a set of subterms, each of which is an instance of a term in S. Note that the corresponding substitution θ is not required to be constructor, but we impose instead that the terms in θ are recursively S-closed. The following example illustrates this definition.

Example 3.16. Consider again the set $S = \{f(x), g(x)\}$ of Example 3.9. The term $c(g(f(a)))$ is S-closed, since the term $g(f(a))$ is an instance of $g(x) \in S$ with $\theta = \{x/f(a)\}$, and the term $f(a)$ in θ is S-closed.

The recursive notion of closedness allows us to strengthen the correctness PE theorems. The new formulation basically exploits the fact that the answers computed by a given goal, say $f(g(y)) = a$, can be retrieved from the answers computed independently by its flat "constituents," i.e., $f(x) = a$ and $g(y) = x$. This essentially corresponds to a kind of formulation of the compositionality property of the semantics. Unfortunately, this property does not hold in general for unrestricted (conditional) narrowing [Alpuente et al. 1996a; Vidal 1996], as opposed to the case of SLD-resolution, which is compositional with respect to the AND operator. As a consequence, the (generalized) closedness does not suffice for completeness under the conditions required for the completeness of unrestricted conditional narrowing, as illustrated by the following example.

Example 3.17. Let us consider the following program $\mathcal{R}$ in $I\!R_c$, inspired by an example in Middeldorp and Hamoen [1994]

$$\begin{array}{rcl} f(x) & \to & g(x,x) \Leftarrow g(x,x) = c \\ a & \to & b \\ g(a,b) & \to & c \\ g(b,b) & \to & c \Leftarrow f(a) = c \end{array}$$

and the set of terms $S = \{f(x), a, g(x,y)\}$. A partial evaluation of S in $\mathcal{R}$ is the following S-closed program $\mathcal{R}'$:

$$\begin{array}{rcl} f(b) & \to & g(b,b) \Leftarrow f(a) = c, c = c \\ f(b) & \to & c \Leftarrow f(a) = c, g(b,b) = c \\ a & \to & b \\ g(a,b) & \to & c \\ g(b,b) & \to & c \Leftarrow f(a) = c \end{array}$$

Now, the S-closed goal $f(a) = c$ succeeds in $\mathcal{R}$ with (normalized) CAS ϵ,

$$\begin{array}{rl} \underline{f(a)} = c \leadsto & g(a, \underline{a}) = c,\ g(a,a) = c \\ \leadsto & g(a,b) = c,\ g(a,\underline{a}) = c \\ \leadsto & \underline{g(a,b)} = c,\ g(a,b) = c \\ \leadsto & c = c,\ \underline{g(a,b)} = c \\ \leadsto & c = c,\ c = c \\ \leadsto^* & \top, \end{array}$$

whereas it fails in $\mathcal{R}'$.

The problem with this example is that unrestricted narrowing is not compositional, and thus the answers for $f(a) = c$ in $\mathcal{R}$ cannot be obtained from the resultants in $\mathcal{R}'$, even if they derive from the decomposed, independent goals $f(x) = y$ and $a = x$ in $\mathcal{R}$.

Therefore, some restrictions need to be added in order to preserve the correctness of the narrowing-driven PE transformation under the generalized closedness condition. The following results show that there exists a close connection between the completeness of basic narrowing and the completeness of the PE transformation based on unrestricted narrowing. Basic conditional narrowing was proven to be AND-compositional whenever it is complete [Alpuente et al. 1996a]. This suggests to us the requirements that ensure the completeness of the PE transformation. Note that here we give a direct proof which does not require the compositionality results in Alpuente et al. [1996a].

Basic (conditional) narrowing is a restricted form of (conditional) narrowing where only terms at *basic* positions are considered to be narrowed [Hullot 1980; Middeldorp and Hamoen 1994]. Informally, a basic position is a position which is not created during unification, i.e., a nonvariable position of the original goal or one that was introduced into the goal by the nonvariable part of the right-hand side or the condition of a rule applied in a preceding narrowing step. The idea behind the concept of *basic* is to avoid narrowing steps on subterms that are introduced by instantiation. Basic conditional narrowing is complete for confluent and *decreasing* 1-CTRS's and for level-canonical 2-CTRS's [Middeldorp and Hamoen 1994]. To

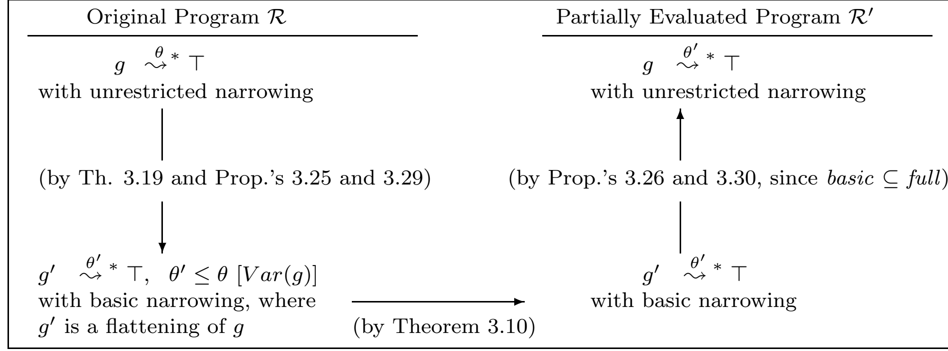

Fig. 1. Proof structure for Theorem 3.18.

simplify our formulation, let $I\!R_b$ denote the class of programs that fulfill the conditions for the completeness of basic conditional narrowing. The formal definition of basic narrowing can be found in Appendix A.4.

In the following, we show that the requirement of closedness suffices for the completeness of the narrowing-driven PE transformation for the class $I\!R_b$ of programs.

THEOREM 3.18. *Let $\mathcal{R}$ be a program in $I\!R_b$, g a goal, S a finite set of terms, and $\mathcal{R}'$ a partial evaluation of $\mathcal{R}$ with respect to S such that $\mathcal{R}' \cup \{g\}$ is S-closed. Then, for all $\theta \in \mathcal{O}_{\mathcal{R}}(g)$ there exists $\theta' \in \mathcal{O}_{\mathcal{R}'}(g)$ such that $\theta' \leq \theta\ [Var(g)]$.*

The remainder of this section is rather technical and is entirely devoted to proving Theorem 3.18. Essentially, we proceed in three successive steps, summarized in Figure 1:

(1) We first introduce a *flattening* transformation which preserves the answers computed by (basic) conditional narrowing. The idea behind this transformation is the following: given a set of terms S and a goal g such that $closed^+(S, g)$ holds, we want to transform g into a "flattened" goal g' such that $closed^-(S, g')$ holds.

(2) Then, for a successful basic conditional narrowing derivation $g' \overset{\sigma}{\leadsto}{}^* \top$ for g' in $\mathcal{R}$, we make use of Theorem 3.10 to ensure the existence of a corresponding successful basic conditional narrowing derivation for g' in the specialized program $\mathcal{R}'$ computing the same answer.

(3) Finally, we use the relationship between the answers computed by a goal and its flattened version in the opposite direction. Then, the fact that the basic narrowing relation is a subset of unrestricted narrowing concludes the proof.

We first recall the completeness of basic conditional narrowing, which can be formulated as follows.

THEOREM 3.19 [MIDDELDORP AND HAMOEN 1994]. *Let $\mathcal{R}$ be a program in $I\!R_b$ and g be a goal. For every normalized solution θ of g there exists a basic conditional narrowing refutation $g \overset{\theta'}{\leadsto}{}^* \top$ such that $\theta' \leq \theta\ [Var(g)]$.*

Basic conditional narrowing is strong complete whenever it is complete. The proof of this lemma is a straightforward adaptation of the proof of Lemma 3.12, and can be found in Appendix A.4.

LEMMA 3.20. *Let $\mathcal{R}$ be a program and $\mathcal{S}$ be an arbitrary selection function. For every basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^{*} \top]$ there exists a basic conditional narrowing refutation $\mathcal{D}_{\mathcal{S}} \equiv [g \overset{\theta}{\leadsto}{}^{*} \top]$ which respects $\mathcal{S}$.*

Now we introduce the flattening transformations which are essential for our proof of completeness. We first introduce the notion of elementary flattening. An *elementary flattening* of g is obtained by replacing some occurrences of a nonvariable term t in g by a fresh variable x and adding the equation $t = x$ to the resulting goal. We let $\breve{O}(g)$ denote the set of nonroot, nonvariable positions of the goal g. Given a goal g and a set of disjoint positions $\{u_1, \ldots, u_n\} \subseteq O(g)$, here we use the notation $g[s_1, \ldots, s_n]_{u_1,\ldots,u_n}$ to abbreviate the expression $(\ldots(g[s_1]_{u_1})\ldots[s_n]_{u_n})$.

Definition 3.21 (*Elementary Flattening*). Given a goal g (different from $\top$), we define the (set-valued) function $flat$ as follows:

$$flat(g) = \{(t = x, g[x, \ldots, x]_{u_1,\ldots,u_n}) \mid x \notin Var(g),\ \{u_1, \ldots, u_n\} \subseteq \breve{O}(g), \text{ and } g_{|u_i} \equiv t, \text{ for all } i = 1, \ldots, n\}$$

The elementary flattening is preparatory for a more elaborate flattening transformation which is required to satisfy the property that the transformed goal is S-closed with the basic notion of closedness whenever the original goal is S-closed with the generalized notion of closedness. Ensuring this property requires a mechanism to extract different duplicates of a given term, which might be necessary when the set S of specialized calls contains *nonlinear* terms (i.e., terms containing multiple occurrences of the same variable). The following example illustrates this point.

Example 3.22. Let us consider the set of terms $S = \{f(y, y), a, b\}$ and the S-closed goal $f(a, a) = b$. Now, if we perform the elementary flattening $(a = x, f(x, a) = b)$, the transformed goal is not S-closed (neither with the basic nor the generalized notion of closedness). Note that even applying two consecutive elementary flattening steps as follows $(a = x, a = y, f(x, y) = b)$, we obtain a goal which is not S-closed.

Only the elementary flattening which extracts the two occurrences of a in $f(a, a) = b$ produces the flattened goal $(a = x,\ f(x, x) = b)$, which achieves the desired property.

Given a finite set of terms S and an S-closed goal g, we now introduce the desired flattening transformation that we call *S-flattening.* This consists of a sequence of (elementary) flattenings which recursively extract a set of subterms from the goal g, and make them appear in a flat equation of the form $s = y$. The decomposition is done in such a way that the flattened goal g' is closed with respect to S according to the basic, nonrecursive notion of closedness.

Definition 3.23 (*S-Flattening*). Let S be a finite set of terms and g an S-closed goal. We say that g' is an *S-flattening* of g if and only if

(1) g' can be obtained from g by a finite sequence of elementary flattenings and

(2) g' is S-closed using the nonrecursive closedness (i.e., $closed^-(S, g')$ holds).

The proof of the following, easy lemma can be found in Appendix A.4.

LEMMA 3.24. *Let S be a finite set of terms and g be an S-closed goal. Then, an S-flattening of g always exists (although it might not be unique).*

The following propositions justify the use of an S-flattened goal in place of the original goal by establishing a precise correspondence between the answer substitutions computed by basic narrowing derivations (see the proofs in Appendix A.4).

PROPOSITION 3.25. *Let $\mathcal{R}$ be a program in $I\!R_b$ and S a finite set of terms. Let g be a goal and g' an S-flattening of g. Then, for each basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$, there exists a basic conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^* \top]$ such that $\theta' \leq \theta \; [Var(g)]$.*

PROPOSITION 3.26. *Let $\mathcal{R}$ be a CTRS and S a finite set of terms. Let g be a goal and g' an S-flattening of g. Then, for each basic conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^* \top]$, there exists a basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$ such that $\theta = \theta' \; [Var(g)]$.*

Observe that Proposition 3.25 only ensures the existence of a more general answer for the flattened goal. The following example shows that we cannot strengthen the ordering $\theta' \leq \theta$ to an identity $\theta' = \theta \; [Var(g)]$ between substitutions, due to the possible existence of nonlinear terms in the set of specialized calls S. In Section 3.3 we show that, if the terms in S are linear (or an implementation of narrowing with the *sharing* of common subterms is used), the flattening transformation preserves computed answers, which allows us to prove that the specialized program computes exactly the same answers as the original one.

Example 3.27. Consider the following program $\mathcal{R}$ in $I\!R_b$

$$\begin{array}{lcl} a & \to & b \\ f(a,b,c) & \to & c \\ f(b,b,w) & \to & c \end{array}$$

and the goal $g \equiv (f(a,a,z) = c)$. Now, there exists the following basic narrowing refutation

$$\begin{array}{rcl} f(a,\underline{a},z) = c & \overset{\epsilon}{\leadsto} & \underline{f(a,b,z)} = c \\ & \overset{\{z/c\}}{\leadsto} & \underline{c = c} \\ & \overset{\epsilon}{\leadsto} & \top, \end{array}$$

which yields the normalized CAS $\theta = \{z/c\}$ for g in $\mathcal{R}$. However, it is easy to see that basic narrowing only computes, for the flattened goal $g' = (a = x, f(x,x,z) = c)$ in $\mathcal{R}$, the more general answer (with respect to $Var(g)$) $\{z/w\}$, since the following

derivation is not basic:

$$\begin{array}{rcl} g' = (\underline{a = x}, f(x,x,z) = c) & \overset{\{x/a\}}{\leadsto} & f(a,\underline{a},z) = c \\ & \overset{\epsilon}{\leadsto} & \underline{f(a,b,z)} = c \\ & \overset{\{z/c\}}{\leadsto} & \underline{c = c} \\ & \overset{\epsilon}{\leadsto} & \top \end{array}$$

In order to make the application of Theorem 3.10 possible, we need to consider an intermediate CTRS which must also be S-closed, using the nonrecursive notion of closedness. We define the S-flattening of a CTRS as follows:

Definition 3.28. Let S be a finite set of terms and $\mathcal{R} \equiv \{r_1, \ldots, r_n\}$ be an S-closed CTRS. We say that the CTRS $\mathcal{R}_{flat} \equiv \{r'_1, \ldots, r'_n\}$ is an S-flattening of $\mathcal{R}$ if the following applies for all $i = 1, \ldots, n$: if $r_i \equiv (\lambda \to \rho \Leftarrow C)$, then the corresponding rule $r'_i \equiv (\lambda \to \rho' \Leftarrow C')$ in $\mathcal{R}_{flat}$ satisfies that $(C', \rho' = y)$ is an S-flattening of $(C, \rho = y)$, with $y \notin (Var(\rho) \cup Var(C))$.

Note that, as a trivial consequence of Lemma 3.24, the S-flattening of an S-closed CTRS always exists (although it might not be unique) and is S-closed using the nonrecursive notion of closedness.

The following results are easy consequences of Propositions 3.25 and 3.26, respectively, and establish the relation between the computational strengths of a program and one of its S-flattenings (the proofs can be found in Appendix A.4).

PROPOSITION 3.29. *Let S be a finite set of terms and $\mathcal{R}$ be an S-closed program in $I\!R_b$. Let $\mathcal{R}_{flat}$ be an S-flattening of $\mathcal{R}$ and g be a goal. Then, for each basic narrowing refutation $[g \overset{\theta}{\leadsto}{}^* \top]$ for g in $\mathcal{R}$, there exists a basic narrowing refutation $[g \overset{\theta'}{\leadsto}{}^* \top]$ for g in $\mathcal{R}_{flat}$ such that $\theta' \leq \theta$ $[Var(g)]$.*

PROPOSITION 3.30. *Let S be a finite set of terms and $\mathcal{R}$ be an S-closed CTRS. Let $\mathcal{R}_{flat}$ be an S-flattening of $\mathcal{R}$ and g be a goal. Then, for each basic narrowing refutation $[g \overset{\theta'}{\leadsto}{}^* \top]$ for g in $\mathcal{R}_{flat}$, there exists a basic narrowing refutation $[g \overset{\theta}{\leadsto}{}^* \top]$ for g in $\mathcal{R}$ such that $\theta' = \theta$ $[Var(g)]$.*

Finally, Theorem 3.18 follows easily from Theorem 3.10 and the results in this section (see the proof scheme in Figure 1).

PROOF OF THEOREM 3.18. First, we consider an answer substitution $\theta \in \mathcal{O}_{\mathcal{R}}(g)$ computed by unrestricted conditional narrowing. By the soundness of conditional narrowing, θ is a solution for g in $\mathcal{R}$. Since $\mathcal{R}$ satisfies the conditions for the completeness of basic conditional narrowing, by Theorem 3.19, there exists a basic conditional narrowing refutation $g \overset{\sigma}{\leadsto}{}^* \top$ for g in $\mathcal{R}$ such that $\sigma \leq \theta$ $[Var(g)]$. By Lemma 3.24, there exists an S-flattening g' of g as well as an S-flattening $\mathcal{R}_{flat}$ of $\mathcal{R}$ such that, by Propositions 3.25 and 3.29, $g' \overset{\sigma'}{\leadsto}{}^* \top$ in $\mathcal{R}_{flat}$, with $\sigma' \leq \sigma$ $[Var(g)]$. Now, we can apply Theorem 3.10, and we have that the basic narrowing refutation $g' \overset{\sigma'}{\leadsto}{}^* \top$ can also be performed with the rules of $\mathcal{R}'_{flat}$, where $\mathcal{R}'_{flat}$ is a partial evaluation of $\mathcal{R}_{flat}$ with respect to S, which is S-closed using the nonrecursive closedness and hence an S-flattening of $\mathcal{R}'$. Finally, by Propositions 3.26 and 3.30,

we have a basic conditional narrowing refutation $g \overset{\theta'}{\leadsto}{}^{*} \top$ for g in $\mathcal{R}'$ such that $\theta' = \sigma'\ [Var(g)]$, which concludes the proof (since the basic narrowing relation is a subset of conditional narrowing). ☐

In the following section, we introduce the conditions required to prove stronger soundness and completeness results for the transformation, i.e., the conditions under which the CASes of the original and the partially evaluated programs do coincide.

3.3 Strong Soundness and Completeness

First we introduce an *independence* condition that allows us to obtain a stronger version of the soundness theorem for the narrowing-driven PE. This condition guarantees that the derived program $\mathcal{R}'$ does not produce additional answers.

In the case of pure logic programming, only (pairwise) nonunifiability of the partially evaluated calls (atoms) in S is required for ensuring the independence of the resultant procedures, thus guaranteeing the strong correctness of the transformation. Here we need a more involved condition; namely, we have to consider *overlaps* among the specialized calls, as the following example illustrates.

Example 3.31. Consider the canonical program

$$\mathcal{R} = \{\ f(x) \to 0 \quad h(x) \to x\ \},$$

and let $S = \{f(h(x)), f(x), h(0)\}$. A partial evaluation of S in $\mathcal{R}$ is the transformed program

$$\mathcal{R}' = \{\ f(h(x)) \to f(x) \quad f(h(x)) \to 0 \quad f(x) \to 0 \quad h(0) \to 0\ \}.$$

Now consider the S-closed goal $f(h(z)) = 0$, which is able to compute the CAS $\{z/0\}$ in $\mathcal{R}'$, whereas it only computes the more general answer $\{z/x\}$ in $\mathcal{R}$. The problem is due to the overlap between the corresponding specialized rules, namely the superposition between the left-hand sides $h(0)$ and $f(h(x))$ of two resultants. This "interference" between the rules causes the narrowing reduction of the goal by using an inappropriate resultant (a resultant derived from a term that does not belong to the set of terms that one could use to prove that the goal is S-closed).

Definition 3.32 (*Overlap*). A term s overlaps a term t if there is a nonvariable subterm $s_{|u}$ of s such that $s_{|u}$ and t unify. If $s \equiv t$, we require that t be unifiable with a proper nonvariable subterm of s.

Definition 3.33 (*Independence*). A set of terms S is independent if there are no terms s and t in S such that s overlaps t.

The following lemma formalizes a fundamental property of independent sets of terms which allows us to prove that the specialized program does not compute additional solutions. This is essential for the proof of Theorem 3.35. The proofs can be found in Appendix A.5.

LEMMA 3.34. *Let S be an independent set of terms, t an S-closed term, and $t_{|u}$ a subterm of t such that $u \in \overline{O}(t)$. If $t_{|u}$ unifies with some term $s \in S$, then $s \leq t_{|u}$.*

The strong soundness theorem for partial evaluation based on unrestricted narrowing can now be given.

THEOREM 3.35. *Let $\mathcal{R}$ be a program, g a goal, S a finite and independent set of terms, and $\mathcal{R}'$ a partial evaluation of $\mathcal{R}$ with respect to S such that $\mathcal{R}' \cup \{g\}$ is S-closed. Then, for all $\theta' \in \mathcal{O}_{\mathcal{R}'}(g)$ there exists $\theta \in \mathcal{O}_{\mathcal{R}}(g)$ such that $\theta = \theta'$ $[Var(g)]$.*

Finally, we consider the strong completeness of narrowing-driven partial evaluation. In light of previous results in Section 3.2, preservation of CASes is not an easy task for unrestricted conditional narrowing. Nevertheless, we find it instructive to clarify the conditions which guarantee this result, since this can help to develop powerful partial evaluators based on more refined narrowing strategies. From the analysis of the proofs in Section 3.2 (see Figure 1), we observe that computed answers are lost at two different stages:

—when moving from an unrestricted narrowing derivation to a basic one and

—when S is not linear and the considered program and goal are flattened in order to obtain a transformed program and goal which are closed with respect to S using the nonrecursive notion of closedness.

Hence we conclude that the extra requirement for linearity of the specialized calls can give back the strong completeness of the PE transformation for the answers computed by basic conditional narrowing derivations, as implied by the following (see the proof in Appendix A.5).

PROPOSITION 3.36. *Let $\mathcal{R}$ be a program in $I\!R_b$ and S a finite and linear set of terms. Let g be a goal and g' an S-flattening of g. Then, for each basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$, there exists a basic conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^* \top]$ such that $\theta' = \theta$ $[Var(g)]$.*

The requirement for linearity of the specialized calls can be considered very severe in a logic programming context. We conjecture that a graph-based implementation of narrowing with the *sharing* of common subterms (in exchange for the linearity of S) would also yield strong completeness. This is justified by the fact that using a sharing-based implementation of narrowing prevents common subterms from being narrowed to different expressions, which is the source for the loss of the identity between the success set semantics of g and g' in Proposition 3.25.

The analysis of our proofs reveals that compositionality of the semantics is an essential ingredient for the completeness of the narrowing-driven PE transformation. Partial evaluation is complete whenever the underlying narrowing strategy is compositional (assuming also that the closedness is fulfilled). Hence, our results seemingly apply to other narrowing strategies, such as LSE narrowing [Bockmayr and Werner 1995] and innermost conditional narrowing [Fribourg 1985], whose derivations are basic (see Section 6 for a discussion on how the partial evaluation results transfer to innermost conditional narrowing). It would also be interesting to investigate the other variants of narrowing in the same systematic way. In Alpuente et al. [1997] we studied the case of lazy narrowing, where the requirement for orthogonal constructor-based programs, which is essential for the completeness of this strategy, guarantees the completeness of the transformation (under the closedness and independence conditions formalized in this article).

Table I. Summary of Requirements for Correctness

Semantic Property	Requirements Using BC	Requirements Using GC
soundness		
strong soundness	C^- I	C^+ I
completeness	C^-	C^+ $I\!R_b$
strong completeness	C^-	C^+ L $I\!R_b$

$I\!R_b$: conditions for the completeness of basic conditional narrowing
C^- : basic closedness of the specialized program and goal
C^+ : generalized closedness of the specialized program and goal
I : independence of the specialized terms in S
L : linearity of the specialized terms in S

We conclude with a summary of the main results in this section concerning correctness of the narrowing-driven partial evaluation for conditional narrowing (see Table I). We distinguish between the requirements which are necessary under the basic, nonrecursive notion of closedness (BC) or under the generalized notion of closedness (GC). Note that, when generalized closedness is considered, strong completeness only holds for the answers computed by basic conditional narrowing derivations (even under the assumptions of closedness and linearity of the specialized calls).

The PE theorems do not address the question of how the set S of terms should be computed to satisfy the required closedness (and independence) conditions, or how partial evaluation should actually be performed. Space limitations prevent us from considering the problem of the independence of the set of (to be) partially evaluated terms S, which should be obtained through some proper postprocessing renaming transformation which can be found in Alpuente et al. [1997] and is somehow similar to that of Benkerimi and Hill [1993], Benkerimi and Lloyd [1990], and Gallagher and Bruynooghe [1990]. In the following sections, we formalize a narrowing-based PE procedure which guarantees termination and achieves the closedness condition.

4. A GENERAL PARTIAL EVALUATION SCHEME

In the literature on program specialization, control problems have been tackled from two different approaches. *On-line* partial evaluators perform all the control decisions *during* the actual specialization phase, whereas *off-line* specializers perform an analysis phase that generates annotations on the program *before* the actual specialization begins [Jones et al. 1993]. Off-line methods can be faster than on-line methods, and they are easier to stop [Consel and Danvy 1993]. In order to achieve *self-application*, i.e., the possibility of specializing the partial evaluator itself [Jones et al. 1993], partial evaluation of functional programs has mainly stressed the off-line approach, while supercompilation and partial deduction have concentrated on the on-line approach, which usually provides more accurate specializations. In this work, we are concerned with the questions of precision and termination, but not self-application, and thus we elaborate on the on-line approach.

Now, we look at a simple on-line NPE method for functional logic programs which essentially proceeds as follows [Gallagher 1993]. For a given goal g and program $\mathcal{R}$, a partial evaluation for S in $\mathcal{R}$ is computed, with S initialized to the set of calls (terms) appearing in g. Then this process is repeated for any term which occurs in

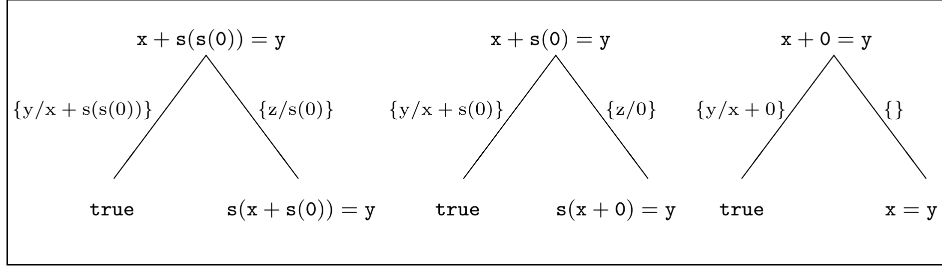

Fig. 2. Narrowing trees for the goals $x + s(s(0)) = y$, $x + s(0) = y$, and $x + 0 = y$.

the rhs's and in the conditions of the resulting rules and which is not closed with respect to the set of terms already evaluated. Assuming that it terminates, this procedure computes a set of partially evaluated terms S' and a set of resultants $\mathcal{R}'$ (the partial evaluation of S' in $\mathcal{R}$) such that the closedness condition for $\mathcal{R}' \cup \{g\}$ is satisfied. The following example illustrates the method.

Example 4.1. Consider the following program $\mathcal{R}$ which defines the addition on natural numbers which are constructed by 0 and s and the initial goal $x + s(s(0)) = y$

$$x + 0 \rightarrow x$$
$$x + s(z) \rightarrow s(x + z)$$

We consider simple narrowing trees constructed by one-step unfolding of goals. Starting with $S = \{x + s(s(0))\}$, and by using the procedure described above, we compute the trees depicted in Figure 2 for the set of terms $S' = \{x + s(s(0)), x + s(0), x + 0\}$. The partial evaluation of S' in $\mathcal{R}$ is the following residual program $\mathcal{R}'$:

$$x + s(s(0)) \rightarrow s(x + s(0))$$
$$x + s(0) \rightarrow s(x + 0)$$
$$x + 0 \rightarrow x$$

Note that $\mathcal{R}' \cup \{x + s(s(0)) = y\}$ is S'-closed.

There are two issues of correctness for a PE procedure: termination, i.e., given any input goal, execution should always reach a stage for which there is no way to continue; and (partial) correctness, i.e., (if execution terminates, then) the operational semantics of the goal with respect to the residual program and with respect to the original program should coincide.

As for termination, the PE procedure outlined above involves the two classical termination problems mentioned in the introduction. The first problem—the so-called *local* termination problem—is the termination of unfolding, or how to control and keep the expansion of the narrowing trees (which provide partial evaluations for individual calls) finite. The *global* level of control concerns the termination of recursive unfolding, or how to stop recursively constructing narrowing trees while still guaranteeing that the desired amount of specialization is retained and that the closedness condition is reached. As we mentioned before, the set of terms S appearing in the goal with which the specialization is performed usually needs to be

augmented in order to fulfill the closedness condition. This brings up the problem of how to keep this set finite throughout the PE process by means of some appropriate abstraction operator which guarantees termination. In the following, we establish a clear distinction between local and global control. This contrasts with Glück and Sørensen [1994], Sørensen and Glück [1995], and Turchin [1986], where these two issues are not (explicitly) distinguished, as only one-step unfolding is performed, and a large evaluation structure is built which comprises something similar to both our local narrowing trees and the global configurations of Martens and Gallagher [1995].

As for local termination, depth-bounds, loop-checks, and well-founded orderings are some commonly used tools for controlling the unfolding during the construction of the search trees [Bruynooghe et al. 1992]. As for global control, it is common to consider proper (well-founded) abstraction operators which are based on the notion of *msg*, *most specific generalization* (see Definition 5.10). As is well known, using the *msg* can induce a loss of precision. In the following section we will deal with the problem of finding appropriate operators to control termination, while still guaranteeing that all sensible unfolding, and therefore specialization, is obtained. Here instead we give a generic PE algorithm that is parametric by these operators.

The approach we follow originates from the framework for ensuring global termination of partial deduction given by Martens and Gallagher [1995], which we reformulate to deal with the use of functions in combination with logical variables. An important issue of partial evaluators is their *control restructuring* ability, which is concerned with the relationship between program points (function definitions) in the original and the residual program [Glück and Sørensen 1996]:

—*Monovariant:* any program point in the original program gives rise to zero or one program point in the residual program.
—*Polyvariant:* any program point in the original program can give rise to one or more program points in the residual program.
—*Monogenetic:* any program point in the residual program is produced from a single program point of the original program.
—*Polygenetic:* any program point in the residual program may be produced from one or more program points of the original program.

Narrowing-based PE is able to produce both polyvariant as well as polygenetic specializations. Polyvariance is achieved thanks to the use of the unification-based information propagation mechanism of narrowing, which allows us to compute a number of independent specializations for a given call with respect to different instantiations of its variables. The possibility of specializing complex, nested terms also allows the partial evaluator to produce polygenetic specializations, since these terms are treated as single units during the partial evaluation process (thus producing a single specialized definition constructed by using the definitions of all functions in the term).

In the following, we formalize a generic algorithm for partial evaluation of functional logic programs based on narrowing which ensures the closedness of the partially evaluated program. Our algorithm is generic with respect to

(1) the *narrowing relation* that constructs search trees,

(2) the *unfolding rule* which determines when and how to terminate the construction of the trees, and
(3) the *abstraction operator* used to guarantee that the set of terms obtained during partial evaluation is finite.

In the remainder of this section, we let $\leadsto_\varphi$ denote a generic (possibly normalizing) narrowing relation which uses the narrowing strategy φ. All notions concerning narrowing introduced so far can be extended to a narrower with strategy φ by replacing $\leadsto$ with $\leadsto_\varphi$ in the corresponding definition. Now, we formalize the notion of *unfolding rule* $U_{\leadsto_\varphi}$ (which we simply denote by U_φ when no confusion can arise), which constructs a (possibly incomplete) finite $\leadsto_\varphi$-narrowing tree and then extracts the resultants of the derivations of the tree.

Definition 4.2 (*Unfolding Rule*). An *unfolding rule* $U_\varphi(s, \mathcal{R})$ (or simply U_φ) is a mapping which returns a PE for the term s in the program $\mathcal{R}$ using the narrowing transition relation $\leadsto_\varphi$ (a set of resultants).

Given a set of terms S, by $U_\varphi(S, \mathcal{R})$ we denote the union of the sets $U_\varphi(s, \mathcal{R})$, for s in S.

We formulate our PE method by means of a transition system $(State, \longmapsto_\mathcal{P})$ whose transition relation $\longmapsto_\mathcal{P} \subseteq State \times State$ formalizes the computation steps. The set *State* of PE configurations (states) is a parameter of the definition. The notion of state has to be instantiated in each specialization of the framework. Here we let $c[S] \in State$ denote a generic configuration whose structure is left unspecified, as it depends on the specific partial evaluator, but which includes at least the set of (to be) partially evaluated terms S. When S is clear from the context, $c[S]$ will simply be denoted by c.

We now introduce an *abstraction operator* which is used to properly extend configurations when new terms are added by the PE algorithm.

Definition 4.3 (*Abstraction Operator*). Given a finite set of terms T and a PE configuration $c[S]$, an abstraction operator *abstract* is a mapping which returns a new PE configuration $c[S'] \equiv abstract(c[S], T)$ such that

(1) if $s \in S'$ then there exists $t \in (S \cup T)$ such that $t_{|p} = s\theta$ for some position p and substitution θ and
(2) for all $t \in (S \cup T)$, t is closed with respect to S'.

Roughly speaking, we say that condition (1) ensures that the abstraction operator does not "create" new function symbols (i.e., symbols not present in the input arguments), whereas condition (2) ensures the correct propagation of closedness information, which is essential for guaranteeing that the residual program indeed satisfies the closedness condition.

Now we introduce the transition relation $\longmapsto_\mathcal{P}$, which is the core mechanism of our PE algorithm.

Definition 4.4 (*PE Transition Relation*). We define the PE relation $\longmapsto_\mathcal{P}$ as the smallest relation satisfying

$$\frac{\mathcal{R}' = U_\varphi(S, \mathcal{R})}{c[S] \longmapsto_\mathcal{P} abstract(c[S], \mathcal{R}'_{calls})}.$$

The transition relation is computed in two steps. First $\mathcal{R}'$ is computed from S and $\mathcal{R}$. Then, by applying *abstract* to $c[S]$ and $\mathcal{R}'_{calls}$, we compute the next set of terms. Here $\mathcal{R}$ is a global parameter of the definition, and at each step an auxiliary system $\mathcal{R}'$ is computed. Similarly to Martens and Gallagher [1995], applying *abstract* in every iteration allows us to tune the control of polyvariance as much as needed. Also, it is within the *abstract* operation that the progress toward termination resides.

Now we are ready to formulate our generic algorithm for the narrowing-based partial evaluation of functional logic programs with respect to a given goal.

Definition 4.5 (*Narrowing-Driven PE Algorithm*). Let $\mathcal{R}$ be a program and g be a goal. We define the PE function $\mathcal{P}$ as follows:

$$\mathcal{P}(\mathcal{R}, g) = S \text{ if } abstract(c_0, g_{calls}) \longmapsto^*_{\mathcal{P}} c[S] \text{ and } c[S] \longmapsto_{\mathcal{P}} c[S],$$

where c_0 is the "empty" PE configuration.

The PE algorithm essentially proceeds as follows. First, it constructs the initial configuration containing the calls in the goal g. Then, it tries to derive new configurations until a "fixed point" is reached, i.e., until the application of $\longmapsto_{\mathcal{P}}$ to the current configuration returns the same state.

We note that the PE algorithm does not compute a partially evaluated program, but a set of partially evaluated terms S which unambiguously determines its associated partial evaluation $\mathcal{R}'$ in $\mathcal{R}$ (using U_φ). The following theorem is the main result in this section and establishes that the closedness condition is reached independently from the narrowing strategy, unfolding rule, and abstraction operator (see the proof in Appendix B.1).

THEOREM 4.6. *Let $\mathcal{R}$ be a program and g be a goal. If $\mathcal{P}(\mathcal{R}, g)$ terminates computing the set of terms S, then $\mathcal{R}' \cup \{g\}$ is S-closed, where the specialized program is given by $\mathcal{R}' = U_\varphi(S, \mathcal{R})$.*

As a corollary of the theorems in Sections 3.2 and 3.3, Theorem 4.6 establishes the partial correctness of the PE algorithm based on unrestricted conditional narrowing. This result also applies to innermost narrowing, as we show in Section 6.

The PE algorithm incorporates only the scheme of a complete narrowing-driven partial evaluator. The resulting partial evaluations might be further optimized by eliminating redundant functors and unnecessary repetition of variables, trying to adapt standard techniques presented in Benkerimi and Hill [1993], Benkerimi and Lloyd [1990], Gallagher [1993], and Gallagher and Bruynooghe [1990]. This is especially interesting in our setting, where functions appearing as arguments of calls are in no way "dead" structures (as in partial deduction), but can also generate new calls to function definitions. Moreover, since we allow the specialization of terms containing nested function symbols, the residual program might also contain nested function symbols in the lhs's of program rules, which prevents the program from being executed by using some narrowing strategies, such as lazy narrowing. A postprocessing renaming transformation is then necessary to restore the *constructor discipline* (cf. Section 6). The renaming phase may also serve to remove any remaining lack of independence in the set of specialized terms.

Since this renaming transformation can be treated as a postprocessing phase and is entirely independent of the methods described in this article, we do not include it here but refer to Alpuente et al. [1997].

In the following section we present a simple but useful solution to the termination problem by introducing some concrete kind of unfolding and abstraction operators.

5. ENSURING TERMINATION

The techniques we introduce to control termination of the NPE algorithm can be seen as an adaptation to our framework of existing techniques to control termination of partial deduction and (positive) supercompilation. At the local level, we mainly follow the approach of Sørensen and Glück [1995], which relies on *well-quasi orderings* to ensuring termination when constructing the so-called *partial process trees*. At the global level, our construction borrows some ideas from the method to ensure global termination of partial deduction of Martens and Gallagher [1995]. The main differences with respect to Martens and Gallagher [1995] are the following: (1) we also use at this level a well-quasi ordering instead of a well-founded order to control termination and (2) we do not register descendency relationships among expressions at the global level: we prefer to represent states by means of sequences of terms rather than adapting the tree-like structures of Martens and Gallagher [1995]. This simplifies our formulation and allows us to focus on the singularities of our method. The reader familiar with Martens and Gallagher [1995] and Sørensen and Glück [1995] can safely skip the remainder of this section and just refer to the examples.

A commonly used tool for proving termination properties is based on the intuitive notion of orderings in which a term that is "syntactically simpler" than another is smaller than the other. The following definition extends the homeomorphic embedding ("syntactically simpler") relation [Dershowitz and Jouannaud 1990] to nonground terms. Variants of this relation are used in termination proofs for term-rewriting systems [Dershowitz and Jouannaud 1991] and for ensuring local termination of partial deduction [Bol 1993]. Recently, after it was taken up in Sørensen and Glück [1995], several works have also adopted the use of (different variants) of the embedding ordering to avoid infinite computations during partial evaluation and supercompilation (e.g., see Alpuente et al. [1996a; 1997], Glück et al. [1996], Leuschel and Martens [1996], Leuschel et al. [1998], and Turchin [1996]).

Definition 5.1 (*Embedding Relation*). The homeomorphic embedding relation $\trianglelefteq$ on terms in $\tau(\Sigma \cup V)$ is defined as the smallest relation satisfying $x \trianglelefteq y$ for all $x, y \in V$, and $s \equiv f(s_1, \ldots, s_m) \trianglelefteq g(t_1, \ldots, t_n) \equiv t$, if and only if

(1) $f \equiv g$ (with $m = n$) and $s_i \trianglelefteq t_i$ for all $i = 1, \ldots, n$ or
(2) $s \trianglelefteq t_j$, for some j, $1 \leq j \leq n$.

Roughly speaking, we say that $s \trianglelefteq t$ if s may be obtained from t by deletion of operators. For example, $\surd\surd(u \times (u + v)) \trianglelefteq (w \times \surd\underline{\surd\surd}((\surd\underline{u} + \surd u)\underline{\times}(\surd\underline{u+}\surd\underline{v})))$. The following result is an easy consequence of Kruskal's Tree Theorem.

Theorem 5.2. *The embedding relation $\trianglelefteq$ is a well-quasi ordering of the set $\tau(\Sigma \cup V)$ for finite Σ, that is, $\trianglelefteq$ is a quasi-order (i.e., a transitive and reflexive binary relation) and, for any infinite sequence of terms $t_1, t_2, \ldots$ with a finite number*

of operators, there exist j, k with $j < k$ and $t_j \trianglelefteq t_k$ (i.e., the sequence is self-embedding).

Proof. The proof is standard (e.g., see Dershowitz and Jouannaud [1990] and Leuschel and Martens [1996]). We include our proof for completeness. Let $\sigma_\perp$ be the substitution mapping all variables to a fixed free constant symbol $\perp$. It is clear that $s \trianglelefteq t$ iff $s\sigma_\perp \preceq t\sigma_\perp$, where $\preceq$ is the standard homeomorphic embedding relation on the set of ground terms $\tau(\Sigma \cup \{\perp\})$. Now, since $\preceq$ is a well-quasi ordering, so is $\trianglelefteq$, and the result follows. □

The embedding relation $\trianglelefteq$ will be used in Section 5.1 to define an unfolding rule which guarantees local termination, i.e., a condition that ensures that narrowing trees are not expanded infinitely in depth. In Section 5.2, we define an abstraction operator that guarantees global termination of the selected instance of the NPE algorithm

5.1 Local Termination

In Section 4, the problem of obtaining (sensibly expanded) finite narrowing trees was shifted to that of defining sensible unfolding rules that somehow ensure that infinite unfolding is not performed. In this section, we introduce an unfolding rule which attempts to maximize unfolding while retaining termination. Our strategy is based on the use of the embedding ordering to stop narrowing derivations. This is simple but less crude than imposing ad hoc depth bounds and still guarantees finite unfolding in all cases.

The following criterion makes use of the embedding relation in a constructive way to produce finite narrowing trees. In order to avoid an infinite sequence of "diverging" calls, we compare each narrowing redex of the current goal with the selected redexes in the ancestor goals of the same derivation, and expand the narrowing tree under the constraints imposed by the comparison.[6] When the compared calls are in the embedding relation, we stop the derivation. We need the following notations.

Definition 5.3 (*Comparable Terms*). Let s and t be terms. We say that s and t are *comparable*, written $comparable(s, t)$, if and only if the outermost function symbol of s and t coincide.

Definition 5.4 (*Admissible Derivation*). Let $\mathcal{D} \equiv (g_0 \overset{[u_0,\theta_0]}{\leadsto_\varphi} \ldots \overset{[u_{n-1},\theta_{n-1}]}{\leadsto_\varphi} g_n)$ be a narrowing derivation for g_0 in $\mathcal{R}$. We say that $\mathcal{D}$ is *admissible* if and only if it does not contain a pair of comparable redexes included in the embedding relation $\trianglelefteq$. Formally,

$$admissible(\mathcal{D}) \Leftrightarrow \forall i = 1, \ldots, n,\ \forall u \in \varphi(g_i),\ \forall j = 0, \ldots, i-1. \\ (comparable(g_{j|u_j}, g_{i|u}) \Rightarrow g_{j|u_j} \not\trianglelefteq g_{i|u}).$$

To formulate the concrete unfolding rule, we also introduce the following preparatory definition.

[6] A slightly more liberal unfolding strategy may result from restricting attention to the so-called *covering ancestors*, i.e., the comparable ancestors of selected redexes. Here we prefer not to consider this optimization and refer to Albert et al. [1998] and Bruynooghe et al. [1992] for details.

Definition 5.5 (*Nonembedding Narrowing Tree*). Given a goal g_0 and a program $\mathcal{R}$, we define the *nonembedding narrowing tree* $\tau_{\varphi}^{\trianglelefteq}(g_0, \mathcal{R})$ for g_0 in $\mathcal{R}$ as follows:

$$\mathcal{D} \equiv [g_0 \overset{[u_0,\theta_0]}{\leadsto_{\varphi}} \dots \overset{[u_{n-1},\theta_{n-1}]}{\leadsto_{\varphi}} g_n \overset{[u_n,\theta_n]}{\leadsto_{\varphi}} g_{n+1}] \in \tau_{\varphi}^{\trianglelefteq}(g_0, \mathcal{R}),$$

if the following conditions hold:

(1) the derivation $(g_0 \overset{[u_0,\theta_0]}{\leadsto_{\varphi}} \dots \overset{[u_{n-1},\theta_{n-1}]}{\leadsto_{\varphi}} g_n)$ is admissible and

(2) (a) the leaf g_{n+1} is $\top$ ($\mathcal{D}$ is a successful derivation) or
  (b) the leaf g_{n+1} is failed ($\mathcal{D}$ is a failing derivation) or
  (c) there exists a position $u \in \varphi(g_{n+1})$ and a number $i \in \{1, \dots, n\}$ such that $g_{i|u_i}$ and $g_{n+1|u}$ are comparable, and $g_{i|u_i} \trianglelefteq g_{n+1|u}$ ($\mathcal{D}$ is incomplete and is cut off because there exists a risk of nontermination).

Hence, derivations are stopped when they either fail, succeed, or the considered redexes satisfy the embedding ordering. Before illustrating the previous definition by means of a simple example, we state the following result, which establishes the usefulness of nonembedding narrowing trees in ensuring local termination.

THEOREM 5.6. *For a program $\mathcal{R}$ and goal g_0, $\tau_{\varphi}^{\trianglelefteq}(g_0, \mathcal{R})$ is a finite (possibly incomplete) narrowing tree for $\mathcal{R} \cup \{g_0\}$ using $\leadsto_{\varphi}$.*

PROOF. The claim that $\tau_{\varphi}^{\trianglelefteq}(g_0, \mathcal{R})$ is a (possibly incomplete) narrowing tree for $\mathcal{R} \cup \{g_0\}$ using $\leadsto_{\varphi}$ follows directly from the fact that only and all (possibly incomplete) narrowing derivations for g_0 in $\mathcal{R}$ using $\leadsto_{\varphi}$ are collected. Now we prove that each $\mathcal{D} \in \tau_{\varphi}^{\trianglelefteq}(g_0, \mathcal{R})$ is finite. We prove the claim by contradiction. Let $\mathcal{D} \in \tau_{\varphi}^{\trianglelefteq}(g_0, \mathcal{R})$ and assume that $\mathcal{D}$ is infinite. This allows us to construct an infinite sequence $\mathcal{S}$ of terms $t_0, t_1, \dots$ consisting of the redexes $g_{0|u_0}, g_{1|u_1}, \dots$ selected for narrowing along the steps of $\mathcal{D}$. Since the number of rules in $\mathcal{R}$ is finite, then the number of differently defined function symbols $\mathcal{F}$ must also be finite. Let $\mathcal{F} = \{f_i\}_{i=1}^{n}$, $n > 0$. Then, we can split $\mathcal{S}$ in (at most) n (possibly infinite) subsequences $\mathcal{S}_i$ of terms, $1 \leq i \leq n$, each of which is formed by the terms whose outermost function symbol is f_i, and hence all terms in each $\mathcal{S}_i$ are comparable. Since $\mathcal{S}$ is infinite, then (at least) one of the subsequences, say $\mathcal{S}_m$ $(1 \leq m \leq n)$, is also infinite. By Theorem 5.2, $\mathcal{S}_m$ is self-embedding, which contradicts the fact that it is admissible. □

Example 5.7. Consider the well-known program `append` with initial query $\mathtt{append}(\mathtt{1}:\mathtt{2}:\mathtt{x_s}, \mathtt{y_s}) = \mathtt{y}$:[7]

$$\begin{array}{l} \mathtt{append}(\mathtt{nil}, \mathtt{y_s}) \to \mathtt{y_s} \\ \mathtt{append}(\mathtt{x} : \mathtt{x_s}, \mathtt{y_s}) \to \mathtt{x} : \mathtt{append}(\mathtt{x_s}, \mathtt{y_s}) \end{array}$$

There exists the following infinite branch in the (unrestricted) narrowing tree (for

[7] We use `nil` and `:` as constructors of lists.

simplicity, substitutions are restricted to goal variables):

$$\begin{array}{ll}\underline{\mathtt{append}(\mathtt{1}:\mathtt{2}:\mathtt{x_s},\mathtt{y_s})} = \mathtt{y} & \overset{\{\}}{\leadsto}\ \mathtt{1}:\underline{\mathtt{append}(\mathtt{2}:\mathtt{x_s},\mathtt{y_s})} = \mathtt{y} \\ & \overset{\{\}}{\leadsto}\ \mathtt{1}:\mathtt{2}:\underline{\mathtt{append}(\mathtt{x_s},\mathtt{y_s})} = \mathtt{y} \\ & \overset{\{\mathtt{x_s}/\mathtt{x}':\mathtt{x}'_s\}}{\leadsto}\ \mathtt{1}:\mathtt{2}:\mathtt{x}':\underline{\mathtt{append}(\mathtt{x}'_s,\mathtt{y_s})} = \mathtt{y} \\ & \overset{\{\mathtt{x}'_s/\mathtt{x}'':\mathtt{x}''_s\}}{\leadsto}\ \ldots \end{array}$$

According to the definition of nonembedding narrowing tree, the development of this branch is stopped at the fourth goal, since the derivation

$$\mathtt{append}(\mathtt{1}:\mathtt{2}:\mathtt{x_s},\mathtt{y_s}) = \mathtt{y} \overset{\{\}}{\leadsto} \mathtt{1}:\mathtt{append}(\mathtt{2}:\mathtt{x_s},\mathtt{y_s}) = \mathtt{y} \overset{\{\}}{\leadsto} \mathtt{1}:\mathtt{2}:\mathtt{append}(\mathtt{x_s},\mathtt{y_s}) = \mathtt{y}$$

is admissible, and the step:

$$\mathtt{1}:\mathtt{2}:\underline{\mathtt{append}(\mathtt{x_s},\mathtt{y_s})} = \mathtt{y} \overset{\{\mathtt{x_s}/\mathtt{x}':\mathtt{x}'_s\}}{\leadsto} \mathtt{1}:\mathtt{2}:\mathtt{x}':\underline{\mathtt{append}(\mathtt{x}'_s,\mathtt{y_s})} = \mathtt{y}$$

fulfills the ordering, because $\mathtt{append}(\mathtt{x_s},\mathtt{y_s}) \trianglelefteq \mathtt{append}(\mathtt{x}'_s,\mathtt{y_s})$.

Finally, we formalize the unfolding rule induced by our notion of nonembedding narrowing tree.

Definition 5.8 (*Nonembedding Unfolding Rule*). We define $U_{\varphi}^{\trianglelefteq}(s,\mathcal{R})$ as the set of resultants associated to the derivations in $\tau_{\varphi}^{\trianglelefteq}(s=y,\mathcal{R})$, with $y \notin Var(s)$.

Nontermination of the PE algorithm can be caused not only by the creation of an infinite narrowing tree but also by never reaching the closedness condition. In the following section we consider the issue of ensuring global termination.

5.2 Global Termination

In this section we show how the abstraction operator which is a parameter of the generic PE algorithm can be defined, by using a simple kind of structure (sequences of terms) that we manipulate in such a way that termination of the specialized algorithm is guaranteed and still provides the right amount of polyvariance which allows us not to lose too much precision despite the use of *msg*'s. For a more sophisticated and more expensive kind of tree-like structure which could improve the amount of specialization in some cases, see Martens and Gallagher [1995].

Definition 5.9 *(PE* Configuration)*. Let $State^* = \tau(\Sigma \cup V)^*$ be the standard free monoid over the set of terms, with the empty sequence of terms denoted by *nil* and the concatenation operation denoted by ",". We define a PE* *configuration* as a sequence of terms $(t_1,\ldots,t_n) \in State^*$. The *empty* PE* configuration is *nil*.

Upon each iteration of the algorithm, the current configuration $q \equiv (t_1,\ldots,t_n)$ is transformed in order to "cover" the terms which result from the partial evaluation of q in $\mathcal{R}$. This transformation is done by means of the following abstraction operation *abstract**, which makes use of the notion of "most specific generalization."

Definition 5.10 *(Most Specific Generalization).* A *generalization* of a nonempty set of terms $\{t_1,\ldots,t_n\}$ is a pair $\langle t,\{\theta_1,\ldots,\theta_n\}\rangle$ such that, for all $i=1,\ldots,n$, $t\theta_i =$

t_i. The pair $\langle t, \Theta\rangle$ is the *most specific generalization (msg)* of a set of terms S, written $msg(S)$, if (1) $\langle t, \Theta\rangle$ is a generalization of S and (2) for every other generalization $\langle t', \Theta'\rangle$ of S, t' is more general than t.

The *msg* of a set of terms is unique up to variable renaming [Lassez et al. 1988]. The following definition may seem technically complex; however, $abstract^*$ proceeds in a simple manner, and the reader may first look at the informal explanation reported after the definition.

Definition 5.11 (*Nonembedding Abstraction Operator*). Let q be a PE* configuration and t be a term. We define $abstract^*(q, t)$ inductively as follows:

$$abstract^*(q,t) \;=\; \begin{cases} q & \text{if } t \in V \\ abstract^*(q, \{t_1, \ldots, t_n\}) & \text{if } t \equiv c(t_1, \ldots, t_n),\ c \in \mathcal{C},\ n \geq 0 \\ abs_call(q,t) & \text{if } t \equiv f(t_1, \ldots, t_n),\ f \in \mathcal{F},\ n \geq 0 \end{cases}$$

where applying $abstract^*$ to a set of terms is simply defined as

$$abstract^*(q, \{t_1, \ldots, t_n\}) = abstract^*(\ldots abstract^*(q, t_1), \ldots, t_n)$$

with $n \geq 1$, and $abstract^*(q, \emptyset) = q$. The auxiliary function abs_call is defined as

—$abs_call(nil, t) = t$

—$abs_call((q_1, \ldots, q_n), t) =$

$$\begin{cases} (q_1, \ldots, q_n, t) & \text{if } \nexists i \in \{1, \ldots, n\}.\ (comparable(q_i, t) \wedge q_i \trianglelefteq t) \\ abstract^*((q_1, \ldots, q_n), S) & \text{if } i \text{ is the maximum } j \in \{1, \ldots, n\} \text{ such that } (comparable(q_j, t) \wedge q_j \trianglelefteq t\}), \text{ and } q_i\theta = t \text{ for some } \theta, \text{ where } S = \{s \mid x/s \in \theta\} \\ abstract^*(q', S) & \text{if } i \text{ is the maximum } j \in \{1, \ldots, n\} \text{ such that } (comparable(q_j, t) \wedge q_j \trianglelefteq t\}), \text{ and } q_i \not\leq t, \text{ where } q' \equiv (q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n),\ msg(q_i, t) = \langle w, \{\theta_1, \theta_2\}\rangle, \text{ and } S = \{w\} \cup \{s \mid x/s \in \theta_1 \text{ or } x/s \in \theta_2\}. \end{cases}$$

Let us informally explain this definition. Given a configuration q, in order to add a new term t (headed by a defined function symbol), the function $abstract^*$ proceeds as follows:

—If t does not embed any comparable term in q, then t is simply added to q.

—If t embeds several comparable terms of q, then it considers the rightmost one in the sequence, say t', and distinguishes two cases:
  —if t is an instance of t' with substitution θ, then it recursively attempts to add the terms in θ to q;
  —otherwise, the *msg* of t' and t is computed, say $\langle w, \theta_1, \theta_2\rangle$, and then it attempts to add w as well as the terms in θ_1 and θ_2 to the configuration resulting from removing t' from q.

We note that, in the latter case, if t' were not removed from q, the abstraction operator could run into an infinite loop, as shown in the following example.

Example 5.12. Let us consider the PE* configuration $q \equiv (f(x, x), g(x))$ and assume that abs_call is modified by not removing the rightmost term q_i of the

sequence q after computing the msg. Then, a call of the form $abs_call(q, f(y,z))$ would proceed as follows:

(1) the msg of $f(x,x)$ and $f(y,z)$ is computed, since they are comparable, $f(y,z)$ is not an instance of $f(x,x)$, and $f(y,z)$ embeds $f(x,x)$;
(2) then the call to $abstract^*(q, \{f(y,z), x\})$ is done, where $\langle f(y,z), \{\{y/x, z/x\}, \epsilon\}\rangle$ is the msg of $f(x,x)$ and $f(y,z)$;
(3) finally, the call to $abstract^*$ reproduces the original call: $abs_call(q, f(y,z))$, thus entering into an infinite loop.

A similar example can be found in Leuschel et al. [1998], where an alternative solution to this problem is proposed; namely, the so-called *strict* homeomorphic embedding $\trianglelefteq^*$ is used which requires s not to be a strict instance of s' for $s \trianglelefteq^* s'$. Thus $f(x,y)$ does not strictly embed $f(x,x)$, and then it is simply added to the structure which is used to store the specialized calls.

The loss of precision in the abstraction operator caused by the use of the msg is quite reasonable and is compensated by the simplicity of the resulting method. Also, although our list-like configurations can be considered poorer than the tree-like states of Martens and Gallagher [1995] and Sørensen and Glück [1995], they are commonly used and, in some cases, can avoid duplication of function definitions. The following example illustrates how our method achieves both termination and specialization. The positive supercompiler of Glück and Sørensen [1994] and Sørensen et al. [1994] does not terminate on this example, due to the infinite generation of "fresh" calls which, because of the growing accumulating parameter, are not an instance of any call that was obtained previously. The PD procedure of Benkerimi and Hill [1993] and Benkerimi and Lloyd [1990] results in the same nontermination pattern for a logic programming version of this program. Most current methods (e.g., Gallagher [1993], Glück et al. [1996], Leuschel and Martens [1996], Martens and Gallagher [1995], and Sørensen and Glück [1995]), would instead terminate on this example.

Example 5.13. Consider the following program $\mathcal{R}$, which checks whether a sequence is a palindrome by using a reversing function with accumulating parameter:

$$\begin{array}{l}
\texttt{palindrome(x)} \rightarrow \texttt{true} \quad \Leftarrow \ \texttt{reverse(x)} = \texttt{x} \\
\texttt{reverse(x)} \rightarrow \texttt{rev(x,nil)} \\
\texttt{rev(nil,}\texttt{y}_\texttt{s}\texttt{)} \rightarrow \texttt{y}_\texttt{s} \\
\texttt{rev(x:}\texttt{x}_\texttt{s}\texttt{,}\texttt{y}_\texttt{s}\texttt{)} \rightarrow \texttt{rev(}\texttt{x}_\texttt{s}\texttt{,x:}\texttt{y}_\texttt{s}\texttt{)}
\end{array}$$

and consider the goal $\texttt{palindrome(1:2:x)} = \texttt{y}$. We follow the generic PE algorithm with the nonembedding unfolding rule (Definition 5.8) and abstraction operator (Definition 5.11). The initial PE* configuration is

$$q_1 \ = \ abstract^*(nil, \texttt{palindrome(1:2:x}_\texttt{s}\texttt{)}) = \texttt{palindrome(1:2:x}_\texttt{s}\texttt{)}.$$

The partial evaluation of $\texttt{palindrome(1:2:x}_\texttt{s}\texttt{)}$ in $\mathcal{R}$ is the program $\mathcal{R}^1$:

$$\texttt{palindrome(1:2:x:x}_\texttt{s}\texttt{)} \rightarrow \texttt{true} \quad \Leftarrow \ \texttt{rev(x}_\texttt{s}\texttt{,x:2:1:nil)} = \texttt{1:2:x:x}_\texttt{s}$$

with $\mathcal{R}^1_{calls} = \{\texttt{true},\ \texttt{rev(x}_\texttt{s}\texttt{,x:2:1:nil)},\ \texttt{1:2:x:x}_\texttt{s}\}$. Then we obtain

$$\begin{aligned}
q_2 \ &= \ abstract^*(\texttt{palindrome(1:2:x}_\texttt{s}\texttt{)}, \mathcal{R}^1_{calls}) \\
&= \ (\texttt{palindrome(1:2:x}_\texttt{s}\texttt{)}, \texttt{rev(x}_\texttt{s}\texttt{,x:2:1:nil)}).
\end{aligned}$$

The partial evaluation of $\mathtt{rev}(\mathtt{x_s}, \mathtt{x{:}2{:}1{:}nil})$ in $\mathcal{R}$ is the program $\mathcal{R}^2$:

$$\begin{array}{l} \mathtt{rev}(\mathtt{nil}, \mathtt{x{:}2{:}1{:}nil}) \to \mathtt{x{:}2{:}1{:}nil} \\ \mathtt{rev}(\mathtt{x'{:}x'_s}, \mathtt{x{:}2{:}1{:}nil}) \to \mathtt{rev}(\mathtt{x'_s}, \mathtt{x'{:}x{:}2{:}1{:}nil}) \end{array}$$

with $\mathcal{R}^2_{calls} = \{\mathtt{x{:}2{:}1{:}nil},\ \mathtt{rev}(\mathtt{x'_s}, \mathtt{x'{:}x{:}2{:}1{:}nil})\}$. Then we obtain

$$\begin{array}{rcl} q_3 & = & abstract^*((\mathtt{palindrome}(\mathtt{1{:}2{:}x_s}), \mathtt{rev}(\mathtt{x_s}, \mathtt{x{:}2{:}1{:}nil})), \mathcal{R}^2_{calls}) \\ & = & (\mathtt{palindrome}(\mathtt{1{:}2{:}x_s}), \mathtt{rev}(\mathtt{x_s}, \mathtt{x_1{:}x_2{:}x_3{:}y_s})). \end{array}$$

The partial evaluation of $\mathtt{rev}(\mathtt{x_s}, \mathtt{x_1{:}x_2{:}x_3{:}y_s})$ in $\mathcal{R}$ is the program $\mathcal{R}^3$:

$$\begin{array}{l} \mathtt{rev}(\mathtt{nil}, \mathtt{x_1{:}x_2{:}x_3{:}y_s}) \to \mathtt{x_1{:}x_2{:}x_3{:}y_s} \\ \mathtt{rev}(\mathtt{x{:}x_s}, \mathtt{x_1{:}x_2{:}x_3{:}y_s}) \to \mathtt{rev}(\mathtt{x_s}, \mathtt{x{:}x_1{:}x_2{:}x_3{:}y_s}) \end{array}$$

with $\mathcal{R}^3_{calls} = \{\mathtt{x_1{:}x_2{:}x_3{:}y_s},\ \mathtt{rev}(\mathtt{x_s}, \mathtt{x{:}x_1{:}x_2{:}x_3{:}y_s})\}$. Then we obtain

$$\begin{array}{rcl} q_4 & = & abstract^*((\mathtt{palindrome}(\mathtt{1{:}2{:}x_s}), \mathtt{rev}(\mathtt{x_s}, \mathtt{x_1{:}x_2{:}x_3{:}y_s})), \mathcal{R}^3_{calls}) \\ & = & (\mathtt{palindrome}(\mathtt{1{:}2{:}x_s}), \mathtt{rev}(\mathtt{x_s}, \mathtt{x_1{:}x_2{:}x_3{:}y_s})) \equiv q_3. \end{array}$$

Thus, the specialized program $\mathcal{R}'$ resulting from the PE of $\mathcal{R}$ with respect to the set of terms $S' = \{\mathtt{palindrome}(\mathtt{1{:}2{:}x_s}), \mathtt{rev}(\mathtt{x_s}, \mathtt{y_s})\}$ is

$$\begin{array}{l} \mathtt{palindrome}(\mathtt{1{:}2{:}x{:}x_s}) \to \mathtt{true} \Leftarrow \mathtt{rev}(\mathtt{x_s}, \mathtt{x{:}2{:}1{:}nil}) = \mathtt{1{:}2{:}x{:}x_s} \\ \mathtt{rev}(\mathtt{nil}, \mathtt{x_1{:}x_2{:}x_3{:}y_s}) \to \mathtt{x_1{:}x_2{:}x_3{:}y_s} \\ \mathtt{rev}(\mathtt{x{:}x_s}, \mathtt{x_1{:}x_2{:}x_3{:}y_s}) \to \mathtt{rev}(\mathtt{x_s}, \mathtt{x{:}x_1{:}x_2{:}x_3{:}y_s}) \end{array}$$

where we have saved some infeasible branches which end with $fail$ at specialization time. Note that all computations on the partially static structure have been performed. In the new partially evaluated program, the known elements of the list in the argument of `palindrome` are "passed on" to the list in the second argument of `rev`. The logical inferences needed to perform this, whose number n is linear in the number of the known elements of the list, is done at partial evaluation time, which is thus more efficient than performing the n logical inferences at runtime.

The following lemma states that *abstract** is a correct instance of the generic notion of abstraction operator (the proofs of next lemma and theorem can be found in Appendix B.2).

LEMMA 5.14. *The function abstract** *is an abstraction operator.*

The following theorem establishes the (local as well as global) termination of the considered instance of the generic PE algorithm

THEOREM 5.15. *The narrowing-driven PE algorithm terminates for the domain State** *of* PE* *configurations and the nonembedding unfolding rule (Definition 5.8) and abstraction operator (Definition 5.11).*

As a consequence of theorems in Sections 3.2 and 3.3, Theorem 5.15 establishes the total correctness of the considered partial evaluation algorithm for unrestricted conditional narrowing. In Section 6.2, we present a PE theorem which gives the total correctness of a call-by-value partial evaluator based on innermost conditional narrowing.

The last example in this section illustrates the fact that our method can also eliminate intermediate data structures and turn multiple-pass programs into one-pass

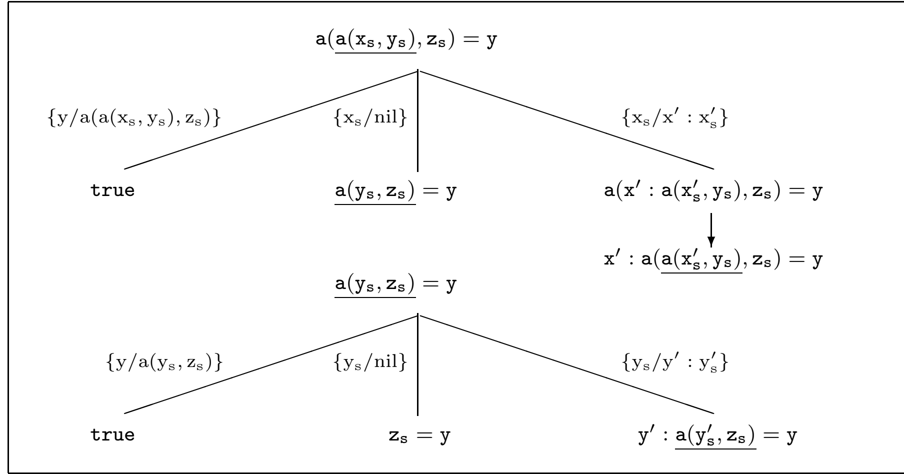

Fig. 3. Narrowing trees for the goals $\mathtt{a}(\mathtt{a}(\mathtt{x_s}, \mathtt{y_s}), \mathtt{z_s}) = \mathtt{y}$ and $\mathtt{a}(\mathtt{x_s}, \mathtt{y_s}) = \mathtt{y}$.

programs, as the deforestation method of Wadler [1990] and the positive supercompiler of Sørensen [1994] do. Standard partial evaluation does not succeed on this example [Glück and Sørensen 1996]. To achieve a similar effect in partial deduction, a special treatment of conjunctions as in CPD is needed (see Leuschel et al. [1996]).

Example 5.16. Consider again the program `append` of Example 5.7 with initial query $\mathtt{append}(\mathtt{append}(\mathtt{x_s}, \mathtt{y_s}), \mathtt{z_s}) = \mathtt{y}$. This goal appends three lists by appending the first two, yielding an intermediate list, and then appending the last one to that. We evaluate the goal by using normalizing conditional narrowing. Starting with the sequence $q = \mathtt{append}(\mathtt{append}(\mathtt{x_s}, \mathtt{y_s}), \mathtt{z_s})$, we compute the trees depicted in Figure 3 for the sequence of terms $q' = (\mathtt{append}(\mathtt{append}(\mathtt{x_s}, \mathtt{y_s}), \mathtt{z_s}), \mathtt{append}(\mathtt{x_s}, \mathtt{y_s}))$. Note that `append` has been abbreviated to `a` in the picture. Then we get the following residual program $\mathcal{R}'$:

$$
\begin{array}{l}
\mathtt{append}(\mathtt{append}(\mathtt{nil}, \mathtt{y_s}), \mathtt{z_s}) \to \mathtt{append}(\mathtt{y_s}, \mathtt{z_s}) \\
\mathtt{append}(\mathtt{append}(\mathtt{x} : \mathtt{x_s}, \mathtt{y_s}), \mathtt{z_s}) \to \mathtt{x} : \mathtt{append}(\mathtt{append}(\mathtt{x_s}, \mathtt{y_s}), \mathtt{z_s}) \\
\mathtt{append}(\mathtt{nil}, \mathtt{z_s}) \to \mathtt{z_s} \\
\mathtt{append}(\mathtt{y} : \mathtt{y_s}, \mathtt{z_s}) \to \mathtt{y} : \mathtt{append}(\mathtt{y_s}, \mathtt{z_s})
\end{array}
$$

which is able to append the three lists by passing over its input only once. This result has been obtained in our method by virtue of normalization. Without the normalization step, the ordering would have been satisfied too early in the rightmost branch of the top tree of Figure 3. Note that we did not adopt any specific strategy (like the call-by-name or the call-by-value ones) for executing the goal. Indeed the call-by-value partial evaluator described in Section 6 also achieves this specialization, thanks to the normalization of goals between narrowing steps. Thus a pure lazy evaluation strategy does not seem essential in this example contradicting a conjecture posed in Sørensen et al. [1994; 1996].

It is worth noting that, in the previous example, the resulting set of terms $\{\mathtt{append}(\mathtt{append}(\mathtt{x_s}, \mathtt{y_s}), \mathtt{z_s}), \mathtt{append}(\mathtt{x_s}, \mathtt{y_s})\}$ in q' is not independent. This exam-

ple illustrates the need for an extra renaming phase able to produce an independent set of terms such as $\{\mathtt{app_3}(\mathtt{x_s}, \mathtt{y_s}, \mathtt{z_s}), \mathtt{app_2}(\mathtt{x_s}, \mathtt{y_s})\}$ and associated specialized program

$$\begin{array}{l}
\mathtt{app_3}(\mathtt{nil}, \mathtt{y_s}, \mathtt{z_s}) \rightarrow \mathtt{app_2}(\mathtt{y_s}, \mathtt{z_s}) \\
\mathtt{app_3}(\mathtt{x} : \mathtt{x_s}, \mathtt{y_s}, \mathtt{z_s}) \rightarrow \mathtt{x} : \mathtt{app_3}(\mathtt{x_s}, \mathtt{y_s}, \mathtt{z_s}) \\
\mathtt{app_2}(\mathtt{nil}, \mathtt{z_s}) \rightarrow \mathtt{z_s} \\
\mathtt{app_2}(\mathtt{y} : \mathtt{y_s}, \mathtt{z_s}) \rightarrow \mathtt{y} : \mathtt{app_2}(\mathtt{y_s}, \mathtt{z_s})
\end{array}$$

which has the same computed answers as the original program `append` for the query $\mathtt{app_3}(\mathtt{x_s}, \mathtt{y_s}, \mathtt{z_s})$ (modulo the renaming transformation). Here we would like to observe that the postprocessing renaming in Alpuente et al. [1997] automatically obtains the desired optimal program.

The use of efficient forms of narrowing can significantly improve the accuracy of the specialization method and increase the efficiency of the resulting program, because runtime optimizations are also performed at specialization time. In the following section, we formalize a highly efficient instance of the generic PE algorithm which makes use of the simple mechanisms introduced so far to achieve (both local and global) termination.

6. A CALL-BY-VALUE PARTIAL EVALUATOR

We are now interested in an *innermost* strategy where narrowing steps are performed at (leftmost) innermost positions of the goals. This corresponds to the leftmost computation rule of Prolog and to eager evaluation in functional languages and allows us to formalize a call-by-value partial evaluator for functional logic programs. The following section briefly recalls the definition of *innermost conditional narrowing* [Fribourg 1985].

6.1 Innermost Conditional Narrowing

An *innermost* term is an operation applied to constructor terms, i.e., a term of the form $f(t_1, \ldots, t_k)$, where $f \in \mathcal{F}$ and, for all $i = 1, \ldots, k$, $t_i \in \tau(\mathcal{C} \cup V)$. A CTRS is *constructor-based* (CB) if the left-hand side of each rule is an innermost term. A CB program is similar to the set of function definitions with pattern matching in functional programming. It implies that there can be neither functional nestings on the lhs's of the rules nor axioms between constructors. Innermost terms are often called *patterns*. A function symbol is *completely defined* (everywhere defined) if it does not occur in any ground term in normal form, that is to say functions are reducible on all ground terms (of an appropriate sort). $\mathcal{R}$ is said to be completely defined (CD) if each defined function symbol in the signature is completely defined. In CD programs, the set of ground normal terms coincides with the set of ground constructor terms $\tau(\mathcal{C})$ over $\mathcal{C}$. In one-sorted theories, completely defined theories occur only rarely. However, they are common when using types, and each function is defined for all constructors of its argument types. Examples of functional logic languages following the CB-CD discipline are SLOG [Fribourg 1985] and LPG [Bert and Echahed 1986; 1995].

Let $\varphi_{\blacktriangleleft}(g)$ be the narrowing strategy which assigns the position p of the leftmost innermost subterm of g to the goal g. We formulate innermost conditional narrowing by means of a labeled transition system, whose transition relation $\leadsto_{\varphi_{\blacktriangleleft}}$

formalizes the computation steps. In the following, we will abbreviate $\leadsto_{\varphi_\blacktriangleleft}$ by $\leadsto_\blacktriangleleft$.

Definition 6.1. Let $\mathcal{R}$ be a program. We define the innermost conditional narrowing relation $\leadsto_\blacktriangleleft$ as the smallest relation satisfying

$$\frac{u = \varphi_\blacktriangleleft(g) \ \wedge \ (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}_+ \ \wedge \ \sigma = mgu(\{g_{|u} = \lambda\})}{g \overset{\sigma}{\leadsto}_\blacktriangleleft (C, g[\rho]_u)\sigma}.$$

Roughly speaking, we say that innermost narrowing selects an innermost subterm of the goal for the narrowing step, i.e., a function call is evaluated by narrowing only if all arguments were completely evaluated before.

The following theorem establishes the completeness of innermost conditional narrowing for constructor-based, completely defined canonical programs. Note that all answers computed by innermost conditional narrowing are normalized, since they are constructor [Fribourg 1985].

THEOREM 6.2 [FRIBOURG 1985]. *Let $\mathcal{R}$ be a CB-CD canonical program, g be a goal, and σ be a ground constructor solution of g such that $Var(g) \subseteq Dom(\sigma)$. Then, there is a computed answer substitution θ of $\mathcal{R} \cup \{g\}$ using $\leadsto_\blacktriangleleft$ such that $\theta \leq \sigma$ $[Var(g)]$.*

The condition $Var(g) \subseteq Dom(\sigma)$ in the premise of Theorem 6.2 guarantees that $g\sigma$ is ground. This condition cannot be dropped, contrary to what is generally assumed [Fribourg 1985; Hanus 1994b; Hölldobler 1989], as witnessed by the following

Example 6.3. Consider the following CB-CD canonical program:

$$\mathcal{R} = \{ \ f(0, y) \ \to \ y \qquad g(0) \ \to \ 0 \ \}$$

Then, the ground constructor substitution $\sigma = \{x/0\}$ is a solution of the equation $f(x, g(y)) = g(y)$. However, innermost conditional narrowing is not able to compute a more general answer.

It is easy to extend this strategy to incompletely defined functions, by just adding a so-called innermost *reflection* rule which skips an innermost function call that cannot be reduced [Hölldobler 1989]. For the sake of simplicity, here we assume that all functions are completely defined, i.e., innermost narrowing is sufficient to compute all answers.

In the following we consider the normalizing innermost conditional narrowing $\leadsto_\blacktriangleleft$, where the computation of the normal form between narrowing steps is performed by innermost conditional rewriting (i.e., a rewrite rule is applied to a term only if each of its subterms is in normal form). This loses no solutions and can be more efficiently implemented than Prolog's computation rule [Hanus 1990; 1991]. In order to ensure that the normal form of a goal uniquely exists and can be computed by any strategy of rewriting, only a decreasing subset of the program rules is used for normalization [Dershowitz and Jouannaud 1990]. Normalizing innermost narrowing is the foundation of several functional logic programming languages like SLOG [Fribourg 1985], LPG [Bert and Echahed 1986; 1995] and (a subset of) ALF [Hanus 1990]. Thus, the method that we propose can be used for the optimization of programs written in these languages. It has been shown that, since functions

allow more deterministic evaluation than predicates, normalizing innermost narrowing has the effect that functional logic programs are more efficiently executable than equivalent logic programs [Fribourg 1985; Hanus 1991]. Following a similar argument, we are able to show that functional logic programs are also more efficiently specializable than pure logic programs. For instance, Example 5.16 shows a typical optimization that standard PD *a la* Lloyd-Shepherdson fails to achieve, and an extension for approaching deforestation and supercompilation as in Glück et al. [1996] and Leuschel et al. [1996] is needed. Call-by-value NPE is able to achieve the optimization without any special tuning or extension simply because the calls are nested, and thus rewriting steps are able to replace a complex expression with something "smaller." This has the effect that the embedding relation which is used as a whistle to warn us that some part of the goal is possibly growing is blown later than in the corresponding SLD derivation, which gives better specialization in this example. Let us illustrate this point with a simple example where the whistle is not even blown, thanks to normalization.

Example 6.4. Consider the following logic program $\mathcal{P}$ and its naive translation to a CTRS $\mathcal{R}$:

$$\begin{array}{lll} \mathcal{P} : \mathtt{p(x)} \Leftarrow & \qquad & \mathcal{R} : \mathtt{p(x)} \rightarrow \mathtt{true} \Leftarrow \\ \quad\;\; \mathtt{p(x)} \Leftarrow \mathtt{p(c(x))} & & \quad\;\; \mathtt{p(x)} \rightarrow \mathtt{true} \Leftarrow \mathtt{p(c(x))} = \mathtt{true} \end{array}$$

Using the homeomorphic embedding relation on atoms and terms of Leuschel and Martens [1996] to prune SLD-trees, the partial evaluation of $\mathcal{P}$ with respect to the goal $\mathtt{p(x)}$ derives the same residual program $\mathcal{P}' = \mathcal{P}$, and no optimization is obtained. Instead, the partial evaluation of the functional logic program $\mathcal{R}$ with respect to the call $\mathtt{p(x)}$, using normalizing narrowing gives the residual program $\mathcal{R}'$

$$\mathtt{p(x)} \rightarrow \mathtt{true} \Leftarrow$$

which does not have recursion.

6.2 The Call-by-Value Partial Evaluator

In this section we consider an instance of the generic NPE algorithm which results from using

(1) the *normalizing innermost conditional narrowing* $\rightsquigarrow_{\blacktriangleleft}$ of Section 6.1 to construct search trees,
(2) the *nonembedding unfolding rule* $U^{\trianglelefteq}_{\blacktriangleleft}$ (i.e., $U^{\trianglelefteq}_{\rightsquigarrow_{\blacktriangleleft}}$) of Definition 5.8 to control the expansion of the trees,
(3) the domain $State^*$ for encoding PE* configurations, and
(4) the *nonembedding abstraction operator abstract** of Definition 5.11 which guarantees correctness and global termination.

By Theorem 4.6 and Theorem 5.15, the narrowing-driven PE algorithm always terminates, and the resulting partially evaluated programs are closed. As we pointed out in Section 3.2, this suffices to state the total correctness of the PE based on innermost narrowing, which is proven by easily reusing the results for unrestricted narrowing. Nevertheless, here we prefer to restrict ourselves to evaluate only innermost terms. This restriction is necessary for the residual program

to comply with the constructor based discipline, thus avoiding producing nondata (nonconstructor) answers (which, for the particular case of innermost narrowing, amounts to computing only the solutions produced by $\mathcal{R}$). This condition also guarantees that the program as well as the transformation inherit the advantages of the innermost strategy in terms of the efficiency of execution.

For simplicity, we consider in the following that programs are decreasing, and thus all program rules can be used for normalization. The following proposition is a straightforward extension of Proposition 3.5. The fact that the specialized program is constructor based follows trivially from the restriction to specialize only innermost terms (since computed answers are constructor).

PROPOSITION 6.5. *The PE of a decreasing CB program with respect to an innermost term is constructor based and decreasing.*

The following theorem states the (strong) soundness and completeness for partial evaluation based on innermost narrowing. We just sketch the main lines of the proof, since it is an easy consequence of previous results for unrestricted conditional narrowing. We denote the set of answer substitutions computed by innermost normalizing narrowing for g in $\mathcal{R}$ by $\mathcal{O}^{\blacktriangleleft}_{\mathcal{R}}(g)$.

THEOREM 6.6. *Let $\mathcal{R}$ be a confluent and decreasing CB-CD program, S a finite set of innermost terms, and g a goal. Let $\mathcal{R}'$ be a partial evaluation of $\mathcal{R}$ with respect to S such that $\mathcal{R}' \cup \{g\}$ is S-closed. Then,*

(*1*) $\mathcal{O}^{\blacktriangleleft}_{\mathcal{R}'}(g) \subseteq \mathcal{O}^{\blacktriangleleft}_{\mathcal{R}}(g)$, *if S is independent* (strong soundness).

(*2*) *If $\theta \in \mathcal{O}^{\blacktriangleleft}_{\mathcal{R}}(g)$, there is $\theta' \in \mathcal{O}^{\blacktriangleleft}_{\mathcal{R}'}(g)$ such that $\theta' \leq \theta$ $[Var(g)]$* (completeness).

PROOF. (sketch) As for strong soundness, the differences raised by the use of innermost normalizing narrowing instead of unrestricted narrowing do not override the proof of Theorem 3.35, if we consider the following facts:

—S is an independent set of innermost terms,

—following the proof of Theorem 3.35, the derivations which result from "expanding" the use of a resultant (of the residual program) by the corresponding sequence of narrowing steps necessary to build this resultant are also innermost derivations, and

—innermost rewriting steps can be seen as ordinary narrowing steps.

As for completeness, the result is a straightforward consequence of the fact that innermost conditional narrowing derivations are basic, and hence Theorem 3.18 directly applies. Even if the corresponding derivation in the specialized program is not innermost, the claim follows by the completeness of innermost normalizing narrowing for confluent and decreasing programs. We also note that strong completeness can be achieved by giving up the refinement of deterministic normalization. □

The following section reports on some experimental results from an implementation that confirm that our approach pays off in practice and gives significant speedups for standard benchmark examples.

7. THE CALL-BY-VALUE PARTIAL EVALUATOR IN PRACTICE

A narrowing-driven partial evaluator based on the ideas of this article has been implemented and used to conduct a number of experiments which demonstrate, as we hope to show, the practicality of the NPE approach.

The INDY system (Integrated Narrowing-Driven specialization system) is a rather concise implementation of the narrowing-driven partial evaluator. It is written in SICStus Prolog v3.6 and is publicly available [Albert et al. 1998]. The complete implementation consists of about 400 clauses (2000 lines of code). The partial evaluator is expressed by 95 clauses, the metainterpreter by 115 clauses (including the code needed to handle the *ground representation* [Bowen and Kowalski 1982]), the parser and other utilities by 95 clauses, the user interface by 50 clauses, and the postprocessing renaming by 45 clauses. The implementation only considers unconditional programs. Conditional programs are treated by using the predefined functions `and`, `if_then_else`, and `case_of`, which are reduced by standard defining rules (e.g., see Moreno-Navarro and Rodríguez-Artalejo [1992]). The specializer also performs a postprocessing renaming phase which allows us to automatically fulfill the independence condition as well as to avoid the restriction to specialize only innermost terms. This renaming technique can be found in Alpuente et al. [1997].

The user can adapt the following settings:

(1) *Unfolding rule*: one of the following three strategies can be selected to control the construction of the local trees: (a) the nonembedding unfolding rule, which expands derivations while new expressions do not embed previous expressions in the branch; (b) a depth-k bound strategy, which expands derivations down to the depth-k frontier of the tree; and (c) one-step derivations where further unfolding is only allowed on expressions whose outermost function symbol is predefined (e.g., `if_then_else`). The inspiration for this rule comes from the *transient reductions* of Sørensen and Glück [1995]. This strategy does not guarantee termination.

(2) *Narrowing strategy*: the default option is `innermost` (call-by-value) narrowing; otherwise, the user can select a `lazy` (call-by-name) narrowing strategy.

(3) *Normalization*: when it is switched on, normalization of the goal is performed before a narrowing step is attempted. There exists the possibility of using only a subset of the program rules for normalization. This is interesting when normalizing lazy narrowing is considered and programs contain terminating as well as non-terminating rules, since the use of (a subset of) terminating rules for normalization is safe in this context [Hanus 1994a].

In order to assess the practicality of our approach, we have benchmarked the speed and specialization achieved by the INDY system. We consider a set of benchmark programs which cover a range of (functional as well as logic) program transformation problems. The benchmarks used for the analysis are `ackermann`, the classical ackermann function; `allones`, which transforms all elements of a list into `1`; `applast`, which appends an element to the end of a given list and returns the last element of the resulting list; `double_app`, which concatenates three lists by performing two (nested) calls to append; `double_flip`, which flips a tree struc-

ture twice, then returning the original tree back; `fibonacci`, fibonacci's function; `kmp`, a seminaive string pattern matcher; `length_app`, which appends two lists and then computes the length of the resulting list; `max_length`, which returns the maximum and the length of a list; `palindrome`, a program to check whether a given list is a palindrome; `reverse`, the well-known reverse with accumulating parameter; `rev_acc_type`, equal to `reverse` with an extra type-check; `sumprod`, which obtains the sum and the product of the elements of a list. Some of the examples are typical PD benchmarks (see Lam and Kusalik [1991] and Leuschel et al. [1998]) adapted to a functional syntax, while others come from the literature of functional program transformations, such as positive supercompilation [Sørensen et al. 1996], fold/unfold transformations [Burstall and Darlington 1977; Darlington 1982], and deforestation [Wadler 1990]. We have used the homeomorphic embedding relation in all the experiments discussed below. The normalization of goals between narrowing steps was also enabled in all benchmarks.

Table II reproduces the source code of some of the benchmark programs as well as the specialized calls. Due to lack of space, we cannot include the code for all benchmarks and their resulting specialized versions. Further details can be found in Alpuente et al. [1998a].

Table III summarizes our timing results. Specialization times were measured on an HP 712/60 workstation, running under HP Unix v10.01 (column `MixTime`). Speedups were computed by running the original ($\mathcal{R}$) and specialized programs ($\mathcal{R}_{Mix}$) under the publicly available innermost FL system LPG[8] in order to be fair and have realistic evaluations. For technical reasons, execution times were measured on a SUN SparcStation IPX/40, running under SUN OS v5.2 (columns `Time` $\mathcal{R}$ and `Time` $\mathcal{R}_{Mix}$, respectively). Times are expressed in milliseconds and are not normalized. All times are the average of 10 executions. Runtime input goals were chosen to give a reasonably long overall time (goals can be found in Alpuente et al. [1998a]). The fourth column indicates the speedups. Unfortunately, we have not been able to compare the compiled code sizes, since programs are interpreted in LPG.

Let us briefly analyze our results. Some benchmarks are common examples used to illustrate the ability of a program transformer to perform *deforestation* [Wadler 1990]. This is a test that neither standard partial evaluation nor partial deduction can pass. Essentially, the aim of deforestation is the elimination of useless intermediate data structures, thus reducing the number of passes over data. For instance, in benchmarks `double_app` and `double_flip`, an intermediate data structure (a list and a tree, respectively) is created during the computation. After specialization, the program is completely deforested. The specialized code we obtain for `double_flip` is

```
double_flip1_1(leaf(A)) -> leaf(A)
double_flip1_1(tree(A,B,C)) -> tree(flip1_1(A),B,flip1_1(C))
flip1_1(leaf(A)) -> leaf(A)
flip1_1(tree(A,B,C)) -> tree(flip1_1(A),B,flip1_1(C))
```

which runs 1.57 times faster than the original program (a similar speedup has been

[8] LPG is a functional logic language implemented at LSR-IMAG, Grenoble, France, and is publicly available at `http://www-lsr.imag.fr/Les.Groupes/scop/f-logiciel.html`.

Table II. Benchmark Programs and Specialized Calls

double_app

```
append([],Y) -> Y
append([X|R],Y) -> [X|append(R,Y)]
```

call: `append(append(X,Y),Z)`

double_flip

```
double_flip(T) -> flip(flip(T))
flip(leaf(N)) -> leaf(N)
flip(tree(L,N,R)) -> tree(flip(R),N,flip(L))
```

call: `double_flip(T)`

allones

```
f(L) -> allones(length(L))
allones(0) -> []
allones(s(N)) -> [1,allones(N)]
length([]) -> 0
length([H|T]) -> sum(s(0),length(T))
sum(0,Y) -> Y
sum(s(X),Y) -> s(sum(X,Y))
```

call: `f(L)`

length_app

```
lengthapp(L1,L2) -> length(append(L1,L2))
length([]) -> 0
length([H|T]) -> sum(s(0),length(T))
sum(0,Y) -> Y
sum(s(X),Y) -> s(sum(X,Y))
append([],X) -> X
append([H|T],X) -> [H|append(T,X)]
```

call: `lengthapp(L1,L2)`

fibonacci

```
fib(0) -> s(0)
fib(s(0)) -> s(0)
fib(s(s(N))) -> sum(fib(s(N)),fib(N))
sum(0,Y) -> Y
sum(s(X),Y) -> s(sum(X,Y))
```

call: `fib(N)`

sumprod

```
sumprod(L) -> sum(sumlist(L),prodlist(L))
sumlist([]) -> 0
sumlist([H|T]) -> sum(H,sumlist(T))
prodlist([]) -> s(0)
prodlist([H|T]) -> prod(H,prodlist(T))
sum(0,Y) -> Y
sum(s(X),Y) -> s(sum(X,Y))
prod(0,Y) -> 0
prod(s(X),Y) -> sum(prod(X,Y),Y)
```

call: `sumprod(L)`

reverse

```
reverse(L) -> rev(L,[])
rev([],A) -> A
rev([H|T],A) -> rev(T,[H|A])
```

call: `reverse(L)`

rev_acc_type

```
rev([],A) -> A
rev([H|T],A) -> cond(islist(A),rev(T,[H|A]))
islist([]) -> true
islist([H|T]) -> islist(T)
cond(true,A) -> A
```

call: `rev(L,[])`

kmp

```
match(P,S) -> loop(P,S,P,S)
loop([],SS,OP,OS) -> true
loop([P|PP],[],OP,OS) -> false
loop([P|PP],[S|SS],OP,OS) -> if(eq(P,S),loop(PP,SS,OP,OS),next(OP,OS))
next(OP,[]) -> false
next(OP,[S|SS]) -> loop(OP,SS,OP,SS)
if(true,A,B) -> A
if(false,A,B) -> B
eq(0,0) -> true
eq(1,1) -> true
eq(0,1) -> false
eq(1,0) -> false
```

call: `match([0,0,1],S)`

achieved for double_app; see Table III). This seems to indicate that narrowing-driven PE is specially well suited for performing deforestation.

Deforestation algorithms do not recognize that an expression contains two or more functions that consume the same data structure. A further source of improvement can originate from the elimination of (unnecessary) multiple traversals of the same data structure. This is shown by the benchmarks allones, applast,

Table III. Runtimes and Speedups

Benchmark	MixTime	Time $\mathcal{R}$	Time $\mathcal{R}_{Mix}$	Speedup
ackermann	430	975	966	1.01
allones	40	275	217	1.27
applast	40	1015	416	2.44
double_app	30	708	464	1.53
double_flip	30	252	160	1.57
fibonacci	90	1464	1464	1
kmp	1090	934	14	66.71
length_app	110	287	168	1.71
max_length	1120	2432	365	6.66
palindrome	1630	746	541	1.38
reverse	50	893	825	1.08
rev_acc_type	90	720	1087	0.66
sumprod	1190	1498	1389	1.08

and `length_app`. For instance, the program `length_app` traverses the input lists twice, one for appending them and one for counting the number of elements they contain. INDY produces the following specialized program for `length_app`, which traverses the arguments only once and which runs 1.71 times faster than the original program:

```
lengthapp2_1([],[]) -> 0
lengthapp2_1([],[A|B]) -> s(length1_1(B))
lengthapp2_1([A|B],C) -> s(length2_1(B,C))

length1_1([]) -> 0
length1_1([A|B]) -> s(length1_1(B))

length2_1([],[]) -> 0
length2_1([],[A|B]) -> s(length1_1(B))
length2_1([A|B],C) -> s(length2_1(B,C))
```

The benchmarks `ackermann`, `fibonacci`, `max_length`, and `sumprod` are classical programs where *tupling* [Burstall and Darlington 1977; Chin 1993] can produce a significant improvement. The tupling transformation eliminates parallel traversals of identical data structures by merging loops together into a new recursive function defined on *tuples* (pairs), which traverses data structures only once. The transformation can also eliminate some unnecessary recursive calls. Table III shows that INDY is not able to perform all tupling automatically. The partial evaluation of fibonacci's function actually gives back the original program. Only `max_length` has been speeded up. Further investigation is needed to study how this kind of optimization can be achieved. Generally, tupling is very complicated, and automatic tupling algorithms either result in high runtime cost (which prevents them from being employed in a real system) or succeed only for a restricted class of programs. Leuschel et al. [1996] have recently shown that conjunctive partial deduction is able to achieve most of the tupling of the fold/unfold approach, with lower complexity and an easier understanding of control issues. Under a dynamic (fair) computation rule, and using the appropriate reordering of expressions which results from the idea of computing the *best matching terms* of Glück et al. [1996] and Leuschel

et al. [1996], within our framework we have been able to obtain the same kind of optimizations as in conjunctive partial deduction. (See Albert et al. [1998] for a preliminary description as well as a set of benchmark results.)

Benchmarks `reverse` and `rev_acc_type` are difficult due to the presence of an accumulating parameter and a checklist (in the case of `rev_acc_type`). This makes it difficult to produce a more efficient program without jeopardizing termination, and most PE systems either do not terminate or fail to achieve any specialization. For these benchmarks, INDY obtains a speedup factor of about 1.08 and a slowdown of 0.66, respectively. This is mainly due to the fact that some intermediate calls which are not covered by previous definitions appear. Namely, we obtain a sequence of calls of the form `rev(L,[])`, `rev(L',[X'])`, `rev(L'',[X'',X'])`, etc. To avoid nontermination, the partial evaluator is forced to generalize, thus losing all possible specialization. A similar situation has been given with `palindrome`, since this program makes use of the `reverse` function. More sophisticated techniques are required for satisfactory specialization.

Our last example is a kind of standard test in partial evaluation and similar techniques: the specialization of a seminaive pattern matcher for a fixed pattern into an efficient algorithm (sometimes called "the KMP-test" [Sørensen et al. 1996]). This example is particularly interesting because it provides a kind of optimization that neither (conventional) PE nor deforestation can achieve. The specialization of the program `kmp` to the pattern `[0,0,1]` is the almost optimal program:

```
match1_1([]) -> false
match1_1([0]) -> false
match1_1([1|A]) -> loop1_1(A)
match1_1([0,0]) -> false
match1_1([0,1|A]) -> loop1_1(A)
match1_1([0,0,0|A]) -> if1_1(A)
match1_1([0,0,1|A]) -> true
loop1_1([]) -> false
loop1_1([0]) -> false
loop1_1([1|A]) -> loop1_1(A)
loop1_1([0,0]) -> false
loop1_1([0,1|A]) -> loop1_1(A)
loop1_1([0,0,0|A]) -> if1_1(A)
loop1_1([0,0,1|A]) -> true
if1_1([]) -> false
if1_1([1|A]) -> true
if1_1([0|A]) -> if1_1(A)
```

which never backs up on the subject string when a mismatch is detected: whenever three consecutive `0`'s are found, the algorithm looks for one `1`, instead of attempting to match the whole pattern `[0,0,1]` from scratch. This is essentially a KMP-style pattern matcher which runs 66 times faster than the original program for the considered input string. However, the cost of unification with the lhs's of the rules is not irrelevant, and performance could be still improved. We have obtained the best specialization by using the unfolding rule which expands derivations while outermost function symbols are predefined (option c of setting 1): INDY gave the desired optimal KMP specialized matcher

```
match1_1(A) -> loop1_3(A)
```

```
loop1_3([]) -> false
loop1_3([0|A]) -> loop1_2(A)
loop1_3([1|A]) -> loop1_3(A)
loop1_2([]) -> false
loop1_2([0|A]) -> loop1_1(A)
loop1_2([1|A]) -> loop1_3(A)
loop1_1([]) -> false
loop1_1([0|A]) -> loop1_1(A)
loop1_1([1|A]) -> true
```

which is really a KMP-style pattern matcher for the pattern `[0,0,1]`. We note that the success of the specialization highly depends upon the use of bit strings, i.e., strings containing only `0`'s and `1`'s. Propagation of negative information could be helpful to achieve a similar effect for a general alphabet [Sørensen et al. 1996].

Our experiments show that the partial evaluator combines in a useful and effective way the propagation of partial data structures (by means of logical variables and unification) with better opportunities for optimization (thanks to the functional dimension). In general, the inclusion of deterministic simplification steps (normalization) has improved both the overall specialization and the efficiency of the method. We have also observed that normalization is essential for succeeding (with the call-by-value evaluation strategy) in benchmarks such as `double_app`, `double_flip`, and `length_app` where deforestation (or elimination of multiple traversals of the same data structure) is the main optimization to be done. INDY combines some good features of deforestation, partial evaluation, and partial deduction, similarly to conjunctive partial deduction and Turchin's supercompilation: the system passes the so-called "KMP-test" (which neither standard partial evaluation nor deforestation pass), and it is also able to do deforestation (which cannot be done by means of standard partial evaluation or partial deduction [Glück and Sørensen 1996]) without any ad hoc artifice.

We have compared our results with those of more sophisticated on-line partial evaluators, such as the ECCE conjunctive PD system [Jørgensen et al. 1996; Leuschel 1998; Leuschel et al. 1998], which is the most appropriate and recent referent (there is no public implementation of the positive supercompiler to our knowledge). In some cases, ECCE is clearly superior (e.g., for the benchmarks `rev_acc_type` and `max_length`). However, our results show that narrowing-driven partial evaluation can lead to significant performance improvements with respect to those of ECCE for some programs, particularly those which test deforestation (e.g., `double_app` and `double_flip`). INDY also gives reasonable speedups for the other benchmarks, which are close to those of ECCE and are sometimes better (e.g., for `double_app`, `double_flip`, and `kmp`, see Jørgensen et al. [1996]).

Considering the amount of specialization and the implementation costs of our modest implementation, figures are quite satisfactory and encouraging. Preliminary results in this section seem to point to the fact that narrowing-driven partial evaluation is better suited to performing deforestation (and elimination of multiple traversals of the same data structure) than the partial deductor ECCE. This is because these kinds of improvements highly benefit from both the functional and logic components of our system. However, because of our rough treatment of predefined functions (such as conjunction), tupling is not always achieved (without switching

on the finer control options of Albert et al. [1998]) and is better in ECCE. Actually, our approach was not expected to match the maturity level of (conjunctive) partial deductors such as ECCE. In fact, the system was not written with efficiency as a prime concern and significant optimizations are still to be implemented. In order to achieve a more competitive system, we plan to extend and refine INDY, e.g., by handling disequations and by providing a more sophisticated support for the treatment of predefined function symbols such as conjunction, where, and case expressions.

8. RELATED WORK

Very little work has been done in the area of functional logic program specialization. In the literature, we found only two noteworthy exceptions. Levi and Sirovich [1975] defined a PE procedure for the functional programming language TEL that uses a unification-based symbolic execution mechanism, which can be understood as (a form of lazy) narrowing. Darlington and Pull [1988] showed how unification can enable instantiation and unfolding steps to be combined to get the ability (of narrowing) to deal with logical variables. A partial evaluator for the functional language HOPE (extended with unification) was also outlined. No actual procedure was included, and no control issues were considered. The problems of ensuring termination and preserving semantics were not addressed in these papers.

Now we discuss the relation to some works of partial evaluation in functional programming, term-rewriting systems, and partial deduction of logic programs.

8.1 Supercompilation

The work on supercompilation [Turchin 1986] is the closest to our work. Supercompilation (supervised compilation) is a transformation technique for functional programs which consists of three core constituents: *driving*, *generalization*, and *generation of residual programs*. Driving can be understood as a unification-based function transformation mechanism, which uses some kind of evaluation machinery similar to (lazy) narrowing to build (possibly infinite) "trees of states" for a program with a given term. Turchin's papers use the following terminology [Romanenko 1991]: "generalized pattern matching" stands for unification; "contractions" stand for narrowing substitutions and, very often in the texts, "driving" stands for the narrowing itself, i.e., the operation of instantiation of a function call for all possible value cases of the arguments, followed by unfolding of the different branches. By virtue of driving, the supercompiler is able to get the same amount of (unification-based) information propagation as the partial deduction of logic programs [Glück and Sørensen 1994].

Turchin's papers describe the supercompiler for Refal (Recursive Function Algorithmic Language), a pattern-matching functional language with a nonstandard notion of patterns. The semantics of Refal is given in terms of a rewriting interpreter (with the inside-out order of evaluation), but driving is embedded into the supercompiler—an outside-in metaevaluator over Refal which subsumes deforestation [Wadler 1990], partial evaluation, and other standard transformations of functional programming languages [Glück and Sørensen 1996; Sørensen et al. 1994]. For example, supercompilation is able to support certain forms of theorem proving, program synthesis, and program inversion. Recently, Jones [1994] put Turchin's

driving methodology on a solid semantic foundation which is not tied to any particular programming language or data structure.

The positive supercompiler of Sørensen et al. [1996] is a reformulation of Turchin's supercompiler for a simpler call-by-name language with tree-structured patterns. In this language, all arguments of a function are input parameters, and there is only pattern matching for nonnested linear patterns. The formulation is generalizable to less restrictive languages at the cost of more complex driving algorithms. Local and global control are not (explicitly) distinguished, as only one-step unfolding is performed. A large evaluation structure is built which comprises something similar to both the local narrowing trees and the global configurations of Martens and Gallagher [1995]. Each call t in the driving process graph produces a residual function. The body of the definition of the new function is derived from the descendants of t in the graph.

The driving process does not preserve the semantics, as it can extend the domain of functions (as noted by Glück and Sørensen [1994], Jones et al. [1993], and Sørensen et al. [1994]). Techniques to ensure termination of driving are studied in Sørensen and Glück [1995] and Turchin [1988]. The idea of Turchin [1988] is to supervise the construction of the tree and, at certain moments, loop back, i.e., fold a configuration to one of the previous states, and in this way construct a finite graph. The *generalization* operation which makes it possible to loop back the current configuration is often necessary. In Sørensen and Glück [1995], termination is guaranteed following a method which is comparable to the Martens and Gallagher [1995] general approach for ensuring global termination of PD.

We note that our "root-to-leaf" notion of resultant (similar to partial deduction) gives rise to fewer rules than in the case of positive supercompilation, where rules are extracted from each single computation step, but has the disadvantage that there are less opportunities to find appropriate "regularities" (in the sense of Pettorossi and Proietti [1996]), just because fewer specialized functions are produced. On the other hand, our recursive notion of closedness substantially enhances the specialization power of the method, since it subsumes the *perfect* ("α-identical") closedness test of Sørensen et al. [1996]. Our recursive closedness is essentially equivalent to the folding operation introduced in Sørensen and Glück [1995], although our method allows us to also fold back a function call which is closed by a different branch of the process tree.

Glück and Sorensen [1994] focused on the correspondence between partial deduction and driving, stating the similarities between the driving of a functional program and the construction of an SLD-tree for a similar Prolog program. The authors did not exploit the close relationship between the driving and narrowing mechanisms. We think that exploiting this correspondence leads to a better understanding of how driving achieves its effects and makes it easier to answer many questions concerning correctness and termination of the transformation. Our results can also be seen as a new formulation of the essential principle of driving in simpler and more familiar terms to the logic programming community. Let us emphasize that we have established a precise correspondence between the answer substitutions computed in the original and the specialized programs for our PE algorithm. This is different from Scherlis [1981] and Sands [1995], where a specialization method for equations with complex lhs's is also defined (in the context of

strict and nonstrict functional languages, respectively), but they can only compare the ground equational consequences semantics of the programs.

8.2 Partial Evaluation for Term-Rewriting Systems

Earlier work on partial evaluation of term-rewriting systems has focused on self-application rather than on the termination or the correctness of the transformation [Bondorf 1988; 1989]. In Bondorf [1989], a self-applicable, call-by-value partial evaluator for (an intermediate language for) TRS's is presented, called TreeMix. During unfolding, variables are "backward" instantiated by a mechanism, introduced in Bondorf [1988], which involves unification and is very similar to driving. However, overlapping lhs's make this mechanism of instantiation followed by unfolding problematic [Bondorf 1988]. To solve this problem, rather than specializing a program in the form of a TRS, programs are first translated by TreeMix into intermediate code which is then specialized, and the residual program is finally translated back into a TRS. Bondorf's TreeMix works off-line, guided by program annotations generated before transformation. The intermediate language, called Tree, is able to express pattern matching at a level of abstraction which makes it feasible to achieve self-application. No correctness result for the PE transformation is given. Regarding termination, it is up to the user to decide, by manual annotation, when to unfold or when to generalize, in order to achieve accurate specialization. There is no guarantee against infinite specialization (or infinite unfolding). A severe restriction of the system is its fixed innermost reduction strategy: Tree is a nontyped, first-order, strict (innermost reduction order) functional language, and TreeMix heavily depends on the operational properties of the language. According to Bondorf [1989], call-by-name evaluation would require a complete revision of the partial evaluator.

The work by Miniussi and Sherman [1996] aims to improve the efficiency of a term-rewriting implementation of equational programs by tackling the problems associated to the partial evaluation of an imperative, intermediate code that the implementation uses for the equations. The method avoids some intermediate rewrite steps and achieves the fusing of intermediate data structures (in the style of Wadler's deforestation) without risking nontermination.

In Bonacina [1988], the author describes a (not much operational) PE algorithm based on (Knuth-Bendix) equational completion[9] (also called superposition). Completion is a unification-based rule commonly used in automatic reasoning to generate critical pairs (which are equational consequences of a set of rules coming from overlapping lhs's) as a means to derive canonical programs. Termination of the transformation is not guaranteed. Another related approach is that of Dershowitz and Reddy [1993], which also formulates a completion-based technique for the synthesis of functional programs, but requires a kind of heuristic ("eureka" rule to be discovered), and thus it is closer to fold/unfold transformation than to partial evaluation. Another *partial completion* procedure is defined by Bellegarde [1995], which is able to perform tupling and deforestation.

A different PE method for a rewriting-based, FL language is presented in Lafave and Gallagher [1997b] which relies on a slight adaptation of our recursive notion of

[9]Narrowing can be seen as a "linear" restriction on completion, similar to "linear resolution."

closedness. The specialization uses a (restricted form of) unification-based propagation information, which differs from the operational principle of the language (based on term rewriting). Lafave and Gallagher consider the specialization of arbitrary expressions just as we can do in our framework, and thus a finer control strategy for specializing complex goals (in the style of Albert et al. [1998] and Leuschel et al. [1996]) is needed to achieve good specialization for some involved problems such as tupling.

8.3 Partial Deduction

Within our paradigm, we are able to achieve more general results than those already obtained in partial deduction. In fact, if we consider the straightforward embedding of logic programs into functional logic programming, by means of boolean-valued functions (with constructor function calls as arguments) representing predicates, our method essentially boils down to standard partial deduction. Namely, in Section 3.1 we have shown that, if we use a simple, nonrecursive definition of closedness, our results are essentially equivalent to those for partial deduction of pure positive logic programs. Of greater interest, when we consider the generalized closedness which recurses over the structure of the terms, what we obtain in return is a framework which is essentially as powerful as the conjunctive extension to partial deduction presented in Leuschel et al. [1996] and which can be similarly speeded up by a fine treatment of predefined symbols similar to the partitioning techniques of Glück et al. [1996] and Leuschel et al. [1996], as is done in Albert et al. [1998].

8.4 Summary

To summarize, our framework defines the first semantics-based PE scheme for functional logic programs which is able to automatically improve program performance without changing the computational meaning and which still guarantees termination. We think that this work gives a good example of how to bring the advantages of partial deduction and positive supercompilation together.

9. CONCLUSIONS AND FURTHER RESEARCH

Few attempts have been made to study the relationship between PE techniques used in logic and functional languages (e.g., see Alpuente et al. [1998b], Glück and Sørensen [1994], and Pettorossi and Proietti [1996]). On the contrary, there is a parallel development of program transformation ideas within the functional programming and the logic programming communities with too little discussion between them. This separation has the negative consequence of duplicated work, since developments are not shared and since many similarities are overlooked. As pointed out by Sørensen et al. [1996], a direct comparison of methods is often blurred due to different language paradigms and different perspectives. We think that the unified (functional logic) treatment of the problem lays the ground for comparisons and brings the different methodologies closer, which could generate new insights for further developments in both fields.

In this article, we have investigated the semantic properties of a novel, narrowing-driven PE transformation and the conditions under which the technique terminates, is sound and complete, and is also generally applicable to a wide class of programs. Since our basic transformation technique and the operational semantics of func-

tional logic languages coincide, we can use all known results about (the different kinds of) narrowing, and thus correctness proofs benefit from this. We hope that our correctness proofs can be of value in themselves, not only because they establish some results that are not as simple as expected, but also because they give a kind of proof structure which may be of relevance for other similar studies.

We have also presented an automatic, on-line PE algorithm for functional logic programs, whose behavior does not depend on the eager or lazy nature of the narrower, and we have shown that it guarantees closedness of the residual program as well as termination of the transformation. Useful instances of this generic algorithm can be easily defined by considering a fixed narrowing strategy. In this work, we have considered the case of normalizing innermost narrowing, since it has been shown that this strategy is a reasonable improvement over pure logic SLD resolution [Fribourg 1985; Hanus 1994b]. Another instance of the algorithm can be found in Alpuente et al. [1997], where a lazy narrowing strategy is considered. We think that our results can give support and induce new research in hybrid approaches to specialization.

To provide empirical evidence of the practicality of our approach, we have presented some experimental results with an implementation of the NPE method. The system INDY allows the user to select either a call-by-value or a call-by-name evaluation strategy. The results from this preliminary implementation are quite reasonable and show that efficiency as well as specialization can be achieved for several interesting program transformation problems.

We conclude by mentioning some further research. Turchin's supercompiler does not just propagate positive information (by applying unifiers), but also propagates negative information which can restrict the values that the variables can take by using environments of positive and negative bindings (bindings which do not hold) [Glück and Klimov 1993; Lafave and Gallagher 1997a; Sørensen et al. 1994; Turchin 1986]. We think that we can strengthen this effect in the setting of (equational) constraint logic programming [Alpuente et al. 1995; Jaffar and Lassez 1987] by using some kind of narrowing procedure with disunification, such as the ones defined in Arenas et al. [1994], Bert and Echahed [1995], Fernández [1992], and Ramírez and Falaschi [1993], in order to propagate (negative) bindings which can be gathered during the transformation as (disequality) constraints.

Our results are also relevant for the definition of a compositional semantics for functional logic programming which is useful for modular program analysis and transformation [Bossi et al. 1994]. We are currently studying how to define this semantics as the result of partial evaluation of the program using derivations that end at *open* function symbols (i.e., symbols to be defined in other program modules).

We also mention the investigation of the application of our framework to optimal versions of lazy narrowing strategies, such as needed narrowing [Antoy et al. 1994] (and its extension to a higher-order framework), which has been proposed as the basic operational principle of Curry [Hanus et al. 1995], a language which is intended to become a standard in the functional logic programming community.

APPENDIX

A. CORRECTNESS PROOFS

In this appendix we give the proofs of some technical results in Section 3. We also introduce some auxiliary notions and results which are needed to prove them.

A.1 Proof of Lemma 3.3

Let us recall the definition of *parallel composition* of substitutions, denoted by $\Uparrow$ [Hermenegildo and Rossi 1989; Palamidessi 1990]. Informally, this operation corresponds to the notion of unification generalized to substitutions. Here $\widehat{\theta}$ denotes the *equational representation* of the substitution θ, i.e., if $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ then $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$.

Definition A.1 *[Palamidessi 1990].* Let θ_1 and θ_2 be two idempotent substitutions. Then, we define $\Uparrow$ as follows:

$$\theta_1 \Uparrow \theta_2 = mgu(\widehat{\theta}_1 \cup \widehat{\theta}_2)$$

We have the following property for the $\Uparrow$ operator.

LEMMA A.2 [LASSEZ ET AL. 1988; PALAMIDESSI 1990]. *Let θ_1 and θ_2 be idempotent substitutions. Then*

$$\theta_1 \Uparrow \theta_2 = \theta_1 mgu(\widehat{\theta}_2\theta_1) = \theta_2 mgu(\widehat{\theta}_1\theta_2).$$

Now we can prove the desired lemma.

LEMMA 3.3. *Let $\mathcal{R}$ be a CTRS and $r' \equiv (s\theta \to t \Leftarrow C)$ be a resultant for s in $\mathcal{R}$, obtained from the following derivation:*

$$\mathcal{D}_s \equiv [s = y \overset{\theta}{\leadsto}{}^* C, t = y]$$

Let g be an equational goal such that there exists $u \in \overline{O}(g)$ with $g_{|u} \equiv s\gamma$. For every narrowing derivation for g in $\mathcal{R}$

$$g \overset{\sigma}{\leadsto}{}^* g'$$

which employs the same rules and in the same order as $\mathcal{D}_s$ does (at the corresponding positions), the narrowing step

$$g \overset{\upsilon}{\leadsto} g''$$

can be proven using the resultant r', where $g' = g''$ and $\sigma = \upsilon$ $[Var(g)]$.

PROOF. We prove this lemma for a resultant constructed from a one-step derivation. The extension to derivations of arbitrary length follows easily by induction.

Let us consider the following narrowing step for $s = y$:

$$s = y \overset{[p,\theta,r]}{\leadsto} (C, s[\rho]_p = y)\theta$$

where $r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$ and $\theta = mgu(\{s_{|p} = \lambda\})$. Since $y \notin Dom(\theta)$, the resultant r' is defined as follows:

$$r' \equiv (s\theta \to s[\rho]_p\theta \Leftarrow C\theta)$$

Let $g_{|u} \equiv s\gamma$ and assume that the following narrowing step can be proved:

$$g[s\gamma]_u \overset{[u.p,\sigma,r]}{\leadsto} (C, g[s\gamma[\rho]_p]_u)\sigma$$

where $\sigma = mgu(\{g_{|u.p} = \lambda\}) = mgu(\{s\gamma_{|p} = \lambda\})$. Now, we want to prove that the following narrowing step

$$g[s\gamma]_u \overset{[u,\upsilon,r']}{\leadsto} (C\theta, g[s[\rho]_p\theta]_u)\upsilon$$

can be done with $\upsilon = mgu(\{s\gamma = s\theta\})$, $(C, g[s\gamma[\rho]_p]_u)\sigma = (C\theta, g[s[\rho]_p\theta]_u)\upsilon$, and $\upsilon = \sigma\ [Var(g)]$. The correspondence between υ and σ is easily derived from the following equivalences:

$$\begin{array}{rll}
\upsilon_{\restriction Var(s\gamma)} & = mgu(\{s\gamma = s\theta\})_{\restriction Var(s\gamma)} & \\
& = mgu(\widehat{\gamma} \cup \widehat{\theta})_{\restriction Var(s\gamma)} & \\
& = (\gamma \Uparrow \theta)_{\restriction Var(s\gamma)} & \text{(by Definition A.1)} \\
& = (\gamma mgu(\widehat{mgu}(\{s_{|p} = \lambda\})\gamma))_{\restriction Var(s\gamma)} & \text{(by Lemma A.2)} \\
& = mgu(\widehat{mgu}(\{s_{|p} = \lambda\})\gamma)_{\restriction Var(s\gamma)} & \text{(since } Dom(\gamma) \cap Var(s\gamma) = \emptyset) \\
& = mgu(\widehat{mgu}(\{s\gamma_{|p} = \lambda\}))_{\restriction Var(s\gamma)} & \text{(since } Dom(\gamma) \cap Var(\lambda) = \emptyset) \\
& = mgu(\{s\gamma_{|p} = \lambda\})_{\restriction Var(s\gamma)} & \\
& = \sigma_{\restriction Var(s\gamma)} &
\end{array}$$

Hence, it follows that $\upsilon = \sigma\ [Var(g)]$. The following equivalence is verified: $\gamma\sigma = \gamma mgu(\{s_{|p}\gamma = \lambda\}) = \gamma mgu(\{s_{|p} = \lambda\}\gamma) = \gamma mgu(\widehat{\theta}\gamma) = \gamma \Uparrow \theta = \theta mgu(\widehat{\gamma}\theta)$. Then, since $Dom(\gamma) \cap Var(C) = \emptyset$, it holds that $C\sigma = C\gamma\sigma = C\theta mgu(\widehat{\gamma}\theta) = C\theta mgu(\widehat{mgu}(\{s\gamma = s\})\theta) = C\theta mgu(\{s\gamma = s\}\theta) = C\theta mgu(\{s\gamma = s\theta\}) = C\theta\upsilon$ (analogously, it holds that $\rho\sigma = \rho\theta\upsilon$). Putting all pieces together, we have that $(C, g[s\gamma[\rho]_p]_u)\sigma = (C\sigma, g\sigma[s\gamma[\rho]_p\sigma]_u) = (C\theta\upsilon, g\upsilon[s\theta[\rho\theta]_p\upsilon]_u) = (C\theta, g[s[\rho]_p\theta]_u)\upsilon$, which concludes the proof. □

A.2 Proof of Proposition 3.5

We first need the following definition [Bockmayr and Werner 1995].

Definition A.4 (*Reduction without Evaluating Conditions*). Let $\mathcal{R}$ be a CTRS. For sequences of equations g and g', the conditional term-rewriting relation without evaluation of the premise $\rightarrowtail_{\mathcal{R}}$ is defined by $g \rightarrowtail_{\mathcal{R}} g'$ if there exists a position $u \in O(g)$, a variant $(\lambda \to \rho \Leftarrow C) \ll \mathcal{R}_+$ of a rule in $\mathcal{R}_+$, and a substitution σ such that $g_{|u} = \lambda\sigma$ and $g' = (C\sigma, g[\rho\sigma]_u)$.

Definition A.5. Let $\mathcal{R}$ be a CTRS. We let $\mathcal{R}_u$ denote the "unconditional part" of $\mathcal{R}$, that is, the set of rules obtained by removing the conditions (i.e., the rules of $\mathcal{R}_u$ are the heads of the rules of $\mathcal{R}$).

Lemma A.6 [Bockmayr and Werner 1995]. *Let $\mathcal{R}$ be a CTRS and g, g' be equational goals. If $g \overset{\theta}{\leadsto}{}^* g'$ then $g\theta \rightarrowtail^*_{\mathcal{R}} g'$.*

Now we can proceed with the proof of Proposition 3.5.

Proposition 3.5. *The n-CTRS obtained as the partial evaluation of a term in a noetherian n-CTRS is noetherian, for $n \in \{1, 2\}$.*

PROOF. The derivations considered in the definition of narrowing-driven PE have the form

$$s = y \ \overset{\theta}{\leadsto}{}^{*} \ g, true \ \text{ or } \ s = y \ \overset{\theta}{\leadsto}{}^{*} \ g, t = y.$$

We consider these two cases separately.

(1) We can split the derivation $(s = y \ \overset{\theta}{\leadsto}{}^{*} \ g, true)$ into

$$s = y \ \overset{\gamma_1}{\leadsto}{}^{+} \ g^{\star}, t = y \ \overset{\sigma}{\leadsto} \ g^{\star}, true \ \overset{\gamma_2}{\leadsto}{}^{*} \ g, true$$

with associated resultant $((s \to y)\gamma_1\sigma\gamma_2 \Leftarrow g)$. Since $(s = y \ \overset{\gamma_1}{\leadsto}{}^{+} \ g^{\star}, t = y)$, then by Lemma A.6, there exists a conditional rewriting sequence without evaluating the conditions:

$$(s = y)\gamma_1 \rightarrowtail^{*}_{\mathcal{R}} g^{\star}, t = y$$

which implies that

$$s\gamma_1 \to^{*}_{\mathcal{R}_u} t.$$

Since $\mathcal{R}$ is noetherian, there exists a monotonic well-founded ordering $\succ$ over $\tau(\Sigma \cup V)$ which is stable under substitution (i.e., such that, if $l \succ r$, then $l\sigma \succ r\sigma$ for all substitutions σ for variables in l and r) such that $\lambda \succ \rho$, for all $(\lambda \to \rho \Leftarrow C) \in \mathcal{R}$ [Dershowitz and Jouannaud 1990]. Together with monotonicity, this condition ensures that $t \succ s$ whenever t rewrites to s for terms t and s in $\tau(\Sigma \cup V)$. Hence, we have that $s\gamma_1 \succ t$. Assume that the rules of $\mathcal{R}$ do not contain extra variables (i.e., variables in C or ρ that do not occur in λ), since the case when extra variables appear in C is analogous. Then, $Var(t) \subseteq Var(s\gamma_1)$ and $Var(g^{\star}) \subseteq Var(s\gamma_1)$.

We have to prove that $s\gamma_1\sigma\gamma_2 \succ y\gamma_1\sigma\gamma_2$, $Var(y\gamma_1\sigma\gamma_2) \subseteq Var(s\gamma_1\sigma\gamma_2)$, and $Var(g) \subseteq Var(s\gamma_1\sigma\gamma_2)$. Now we consider two cases:

(a) If σ is of the form $\{y/t\}$, then $y\gamma_1\sigma \equiv y\sigma \equiv t$ and $s\gamma_1\sigma \equiv s\gamma_1$, since $y \notin Var(s)$. By the stability of $\succ$, we have $s\gamma_1 \succ t \Rightarrow s\gamma_1\gamma_2 \succ t\gamma_2$, i.e., $s\gamma_1\sigma\gamma_2 \succ y\gamma_1\sigma\gamma_2$. Now, the facts $Var(t) \subseteq Var(s\gamma_1)$ and $Var(g^{\star}) \subseteq Var(s\gamma_1)$ imply that $Var(t\gamma_2) \subseteq Var(s\gamma_1\sigma\gamma_2)$ and $Var(g) \subseteq Var(s\gamma_1\sigma\gamma_2)$, respectively.

(b) If $t \equiv x, x \in V$ and σ is of the form $\{x/y\}$, then $y\gamma_1\sigma \equiv y$, since $y \notin Var(s)$. By the stability of $\succ$, we have $s\gamma_1 \succ t \Rightarrow s\gamma_1 \succ x \Rightarrow s\gamma_1\sigma \succ y \Rightarrow s\gamma_1\sigma\gamma_2 \succ y\gamma_2$, i.e., $s\gamma_1\sigma\gamma_2 \succ y\gamma_1\sigma\gamma_2$. Now we have that $Var(t) \subseteq Var(s\gamma_1) \Rightarrow x \in Var(s\gamma_1) \Rightarrow x\sigma \equiv y \in Var(s\gamma_1\sigma) \Rightarrow Var(y\gamma_2) \subseteq Var(s\gamma_1\sigma\gamma_2)$. Analogously, $Var(g^{\star}) \subseteq Var(s\gamma_1) \Rightarrow Var(g) \subseteq Var(s\gamma_1\sigma\gamma_2)$.

(2) We consider the derivation

$$s = y \ \overset{\theta}{\leadsto}{}^{+} \ g, t = y$$

which produces the resultant $((s \to y)\theta \Leftarrow g)\sigma$, where $\sigma = mgu(\{t = y\})$. The proof is perfectly analogous to the proof of (a), considering that the computed resultant could be produced by a derivation of the form $(s = y \ \overset{\theta}{\leadsto}{}^{+} \ g, t = y \overset{\sigma}{\leadsto} g, true)$ as well.

□

A.3 Proofs of Completeness Using the Basic Closedness

Lemma 3.12 is an easy consequence of the following switching lemmata (they are a reformulation of Lemma 26 and Lemma 37 in Middeldorp et al. [1996]). Given a sequence (of equations) C, we let $|C|$ denote the length of C (i.e., the number of equations in it).

LEMMA A.8. *Let $\mathcal{R}$ be a CTRS and g_1 be a goal containing distinct equations e_1 and e_2 at positions i and j, respectively. Let $r_1 \equiv (\lambda_1 \to \rho_1 \Leftarrow C_1)$, $r_2 \equiv (\lambda_2 \to \rho_2 \Leftarrow C_2) \in \mathcal{R}$, with $|C_1| = k_1$ and $|C_2| = k_2$. For every conditional narrowing derivation*

$$g_1 \overset{[i.u_1,r_1,\sigma_1]}{\leadsto} g_2 \overset{[(j+k_1).u_2,r_2,\sigma_2]}{\leadsto} g_3$$

with $j.u_2 \in \overline{O}(g_1)$, there exists a conditional narrowing derivation

$$g_1 \overset{[j.u_2,r_2,\sigma_2']}{\leadsto} g_2' \overset{[(i+k_2).u_1,r_1,\sigma_1']}{\leadsto} g_3'$$

such that g_3 and g_3' are equal (up to reordering) and $\sigma_1\sigma_2 = \sigma_2'\sigma_1'$.

PROOF. It is perfectly analogous to the proof of Lemma 26 in Middeldorp et al. [1996]. The assumption that rules include conditions does not override the proof, since both e_1 and e_2 belong to the initial goal g_1. By a final reordering of the introduced conditions C_1 and C_2, we get the equivalence with g_3. □

LEMMA A.9. *Let $\mathcal{R}$ be a CTRS and g_1 be a goal containing distinct equations e_1 and e_2 at positions i and j, respectively. Let $r_1 \equiv (\lambda_1 \to \rho_1 \Leftarrow C_1)$, $r_2 \equiv (\lambda_2 \to \rho_2 \Leftarrow C_2) \in \mathcal{R}$, with $|C_1| = k_1$ and $|C_2| = k_2$. For every conditional narrowing refutation*

$$g \overset{\theta_1}{\leadsto}{}^* g_1 \overset{[i.u_1,r_1,\sigma_1]}{\leadsto} g_2 \overset{[(j+k_1).u_2,r_2,\sigma_2]}{\leadsto} g_3 \overset{\theta_2}{\leadsto}{}^* \top$$

with $\theta_1\sigma_1\sigma_2\theta_2{\restriction_{Var(g)}}$ normalized, there exists a conditional narrowing refutation

$$g \overset{\theta_1}{\leadsto}{}^* g_1 \overset{[j.u_2,r_2,\sigma_2']}{\leadsto} g_2' \overset{[(i+k_2).u_1,r_1,\sigma_1']}{\leadsto} g_3' \overset{\theta_2}{\leadsto}{}^* \top$$

such that g_3 and g_3' are equal (up to reordering) and $\theta_1\sigma_1\sigma_2\theta_2 = \theta_1\sigma_2'\sigma_1'\theta_2$.

PROOF. It follows by Lemma A.8, since the normalization of the computed answer guarantees that $j.u_2 \in \overline{O}(g_1)$ (see the proof of Lemma 37 in Middeldorp et al. [1996]). □

Now, Lemma 3.12 is easily derived.

LEMMA 3.12. *Let $\mathcal{S}$ be an arbitrary selection function. For every conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$ that produces a normalized substitution there exists a conditional narrowing refutation $\mathcal{D}_{\mathcal{S}} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$ which respects $\mathcal{S}$.*

PROOF. Immediate by repeated application of Lemma A.9. □

A.4 Proofs of Completeness Using the Generalized Closedness

Let us first recall the formal definition of *level-canonicity* for CTRS's [Middeldorp and Hamoen 1994].

Definition A.11. Let $\mathcal{R}$ be a CTRS. We inductively define TRS's $\mathcal{R}_n$ for $n \geq 0$ as follows:

$$\begin{aligned}\mathcal{R}_0 &= \{x = x \rightarrow true\},\\ \mathcal{R}_{n+1} &= \{\lambda\sigma \rightarrow \rho\sigma \mid (\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R} \text{ and } e \overset{\sigma}{\leadsto}{}^{*} \, true \text{ in } \mathcal{R}_n \text{ for all } e \in C\}\end{aligned}$$

Definition A.12. A CTRS is called *level-confluent* if each $\mathcal{R}_n$, $n \geq 0$, is confluent. We call $\mathcal{R}$ *level-canonical* if $\mathcal{R}$ is noetherian and level-confluent.

Of course, level-confluence implies confluence. Level-confluence can be ensured by syntactic conditions on programs [Suzuki et al. 1995].

Now, we recall the formal definition of *decreasingness* for CTRS's [Middeldorp and Hamoen 1994].

Definition A.13. A 1-CTRS is *decreasing* if there exists a well-founded extension $\succ$ of the rewrite relation $\rightarrow_{\mathcal{R}}$ with the following properties:

(1) $\succ$ has the *subterm property*, i.e., $t \succ t_{|u}$ for all $u \in O(t) - \{\Lambda\}$, and
(2) if $(\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R}$ and σ is a substitution, then $\lambda\sigma \succ \rho\sigma$ and $\lambda\sigma \succ s\sigma, \lambda\sigma \succ t\sigma$ for all $s = t$ in C.

Decreasingness can be seen as the natural extension of noetherianity to the case of CTRS's which ensures the finiteness of recursive evaluation. It is known to be essential for the completeness of some optimized forms of narrowing, such as basic or innermost conditional narrowing, which are the operational basis of a number of functional logic languages like ALF [Hanus 1990], LPG [Bert and Echahed 1986; 1995], and SLOG [Fribourg 1985].

Now we formalize *basic conditional narrowing* in the style of Hullot [1980] and Middeldorp and Hamoen [1994]. Given a natural number k and a set O of positions, we let $k + O = \{(k+i).u \mid i.u \in O,\ i \in I\!N\}$. We use this notation to recompute the set of basic positions of the goal when k new equations are added by a narrowing step.

Definition A.14 (*Basic Conditional Narrowing*). Let

$$\mathcal{D} \equiv g_1 \overset{[u_1,r_1,\sigma_1]}{\leadsto} g_2 \overset{[u_2,r_2,\sigma_2]}{\leadsto} \dots \overset{[u_{n-1},r_{n-1},\sigma_{n-1}]}{\leadsto} g_n$$

be a conditional narrowing derivation, with $r_i \equiv (\lambda_i \rightarrow \rho_i \Leftarrow C_i)$, and $|C_i| = k_i$, $i = 1, \dots, n-1$. $\mathcal{D}$ is *basic* if $u_i \in B_i$ for $1 \leq i \leq n-1$, where the sets of positions $B_1, \dots, B_{n-1}$ are inductively defined as follows:

$$\begin{aligned}B_1 &= \overline{O}(g_1),\\ B_{i+1} &= \mathcal{B}(B_i, u_i, r_i) \text{ for } 1 \leq i \leq n-1\end{aligned}$$

Here $\mathcal{B}(B_i, u_i, r_i)$ abbreviates $\overline{O}(C_i) \cup (k_i + (B_i - \{w \in B_i \mid u_i \leq w\})) \cup (k_i + \{u_i.w \mid w \in \overline{O}(\rho_i)\})$. Positions in B_i $(1 \leq i \leq n-1)$ are referred to as *basic positions.*

$\mathcal{D}$ is *based* on a set of positions $B_1 \subseteq \overline{O}(g_1)$ if $u_i \in B_i$ for $1 \leq i \leq n-1$, with $B_2, \dots, B_{n-1}$ defined as above.

Now we proceed with the proof of Lemma 3.20.

LEMMA 3.20. *Let $\mathcal{S}$ be an arbitrary selection function. For every basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$ there exists a basic conditional narrowing refutation $\mathcal{D}_{\mathcal{S}} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$ which respects $\mathcal{S}$.*

PROOF. It is immediate to see that the requirement $j.u_2 \in \overline{O}(g_1)$ in Lemma A.8 is always satisfied for a basic conditional narrowing derivation. Also note that the exchange of the two steps preserves basicness. Therefore, $\mathcal{D}_{\mathcal{S}}$ is constructed by a finite number of applications of Lemma A.8. Since the transformation in Lemma A.8 preserves the computed answer substitutions, it follows that the answers of $\mathcal{D}$ and $\mathcal{D}_{\mathcal{S}}$ do coincide. □

Next follows the statement and proof of Lemma 3.24.

LEMMA 3.24. *Let S be a finite set of terms and g be an S-closed goal. Then, an S-flattening of g always exists (although it might not be unique).*

PROOF. An S-flattening of a given goal g can be obtained by using the following algorithm:

Input: a set of terms S and a goal g
Output: a set of S-flattenings G_S of g
Initialization: $i := 0$; $G_0 := \{g\}$
While exists $g \in G_i$ such that $closed^-(S, g)$ does not hold **do**:

(1) let $g \in G_i$ such that $closed^-(S, g)$ does not hold;
(2) let $u \in \overline{O}(g)$ be a non-root position of g such that $g_{|u}$ is an outer subterm of g headed by a defined function symbol and $closed^-(S, g_{|u})$ does not hold;
(3) for all term $s \in S$ such that $g_{|u} \equiv s\theta$ do:
    for all $x/t \in \theta$, perform an elementary flattening of g by extracting together all subterms $t = g_{|u.p}$ s.t. $s_{|p} = x$;
(4) let G' be the resulting set of flattened goals, then $G_{i+1} := (G_i - \{g\}) \cup G'$;
(5) $i := i + 1$

Return $G_S := G_i$

Note that this algorithm simply proceeds by following the definition of the generalized closedness. Then, it is straightforward to demonstrate that, given an input S-closed goal g, the algorithm always terminates and returns at least one S-flattening of g. Of course, the algorithm might not return a singleton, since the definition of generalized closedness is nondeterministic (which is reflected, in step (3) of the algorithm, in the possibility that $g_{|u}$ is covered by different terms in S). □

The following lemmata are needed to prove Propositions 3.25 and 3.26. They establish a precise correspondence between the basic derivations for a goal g and those for an elementary flattening of g.

LEMMA A.17. *Let $\mathcal{R}$ be a program in $I\!R_b$, g a goal, and $g' \in flat(g)$ an elementary flattening of g. If there exists a basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$, then there exists a basic conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^* \top]$ such that $\theta' \leq \theta$ $[Var(g)]$.*

PROOF. Since $g \overset{\theta}{\leadsto}^* \top$, by Lemma A.6 we have that $g\theta \rightarrowtail^*_{\mathcal{R}} \top$. Let us consider an elementary flattening $g' \equiv (t = x, g[x, \ldots, x]_{u_1,\ldots,u_k})$, $k \geq 1$, with $\{u_1, \ldots, u_k\} \subseteq \breve{O}(g)$, x a fresh variable, and $g_{|u_1} \equiv \ldots \equiv g_{u_k} \equiv t$. Let $\vartheta = \{x/t\}\theta$ (note that $\vartheta = \{x/(t\theta)\} \cup \theta = \theta\ [Var(g)]$, since $x \notin Dom(\theta)$). Then, there exists a sequence $g'\vartheta \rightarrowtail^*_{\mathcal{R}} \top$. Now, there is a normalized substitution $\vartheta' = \{x/(x\vartheta)\downarrow \mid x \in Dom(\vartheta)\}$ such that $g'\vartheta' \rightarrowtail^*_{\mathcal{R}} \top$, where $\vartheta' = \{x/(t\theta)\downarrow\} \cup \theta = \theta\ [Var(g)]$ (since $\theta_{\lceil Var(g)}$ is normalized). From the completeness of basic conditional narrowing, we have a basic narrowing refutation $g' \overset{\theta'}{\leadsto}^* \top$, such that $\theta' \leq \vartheta'\ [Var(g')]$. Finally, since $\vartheta'_{\lceil Var(g')}$ is normalized and $\theta' \leq \vartheta'\ [Var(g')]$, then $\theta'_{\lceil Var(g')}$ is also normalized, and, since $\vartheta' = \theta\ [Var(g)]$, then $\theta' \leq \theta\ [Var(g)]$. □

The following auxiliary, technical definition is helpful. Given a goal g, a set of positions Pos of g, and one of its flattenings g', we denote by $flat_pos(Pos, g')$ the *corresponding positions* of Pos in g'. Formally, if $g' \equiv (t = x, g[x, \ldots, x]_{u_1,\ldots,u_k})$, then $flat_pos(B, g') = \{1.1.p \mid i \in \{1, \ldots, k\} \wedge u_i.p \in B\} \cup (1 + (B - \{w \in B \mid i \in \{1, \ldots, k\} \wedge u_i \leq w\}))$.

LEMMA A.18. *Let $\mathcal{R}$ be a CTRS, g a goal, and $g' \in flat(g)$ an elementary flattening of g. Let $B \subseteq \overline{O}(g)$ be a set of positions of g and $B' = flat_pos(B, g')$. If there exists a conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}^* \top]$ based on B', then there exists a conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}^* \top]$ based on B such that $\theta' = \theta\ [Var(g)]$.*

PROOF. This result follows easily from Lemma 3.20.

Let $g' \equiv (t = x, g[x, \ldots, x]_{u_1,\ldots,u_k})$, with $\{u_1, \ldots, u_k\} \subseteq \breve{O}(g)$, x a fresh variable, and $g_{|u_1} \equiv \ldots \equiv g_{|u_k} \equiv t$. By Lemma 3.20, for any fixed selection function $\mathcal{S}$, there exists a narrowing refutation $g' \overset{\theta'}{\leadsto}^* \top$ respecting $\mathcal{S}$ which is based on B'. Without loss of generality, we consider the following derivation for g', which computes θ' by selecting equations according to a fixed left-to-right selection rule:

$$\mathcal{D}' \equiv (g' \equiv g'_1 \overset{[w_1,r_1,\sigma_1]}{\leadsto} g'_2 \overset{[w_2,r_2,\sigma_2]}{\leadsto} \ldots \overset{[w_{m-1},r_{m-1},\sigma_{m-1}]}{\leadsto} g'_m \equiv \top), \quad \theta' = \sigma_1 \ldots \sigma_{m-1},$$

where $r_i \equiv (\lambda_i \rightarrow \rho_i \Leftarrow C_i) \ll \mathcal{R}_+$, $w_i \in B_i$, $\sigma_i = mgu(\{g'_{i|w_i} = \lambda_i\})$, $g'_{i+1} = (C_i, g'_i[\rho_i]_{w_i})\sigma_i$, $i = 1, \ldots, m-1$, $m > 2$. Note that $m > 2$, since the number of equations in the flattened goal g' is greater than or equal to two. The proof is done by induction on the number $n = (m-1)$ of steps of this derivation.

Let $n = 2$. Then $w_1 \equiv 1, w_2 \equiv 2$, and the unification rule $(x = x \rightarrow true)$ has been applied twice. Then,

$$g'_1 \equiv (t = x, g[x, \ldots, x]_{u_1,\ldots,u_k}) \overset{[1,(x=x\rightarrow true),\{x/t\}]}{\leadsto} g \overset{[2,(x=x\rightarrow true),\theta]}{\leadsto} \top,$$

and the result follows with $\theta = mgu(g) = \theta'_{\lceil Var(g)}$.

Now we consider the inductive case $n > 2$. We have two possibilities:

(1) Let $w_1 \equiv 1$. Then $g'_1 \equiv (t = x, g[x, \ldots, x]_{u_1,\ldots,u_k}) \overset{[1,x=x\rightarrow true,\{x/t\}]}{\leadsto} g$, and the result follows.

(2) Let $w_1 = 1.1.w' \in B_1$, with $w' \in \overline{O}(g_{|u_i})$, $i = 1, \ldots, k$. Then $g'_2 \equiv (C_1, t[\rho_1]_{w'} =$

$x, g[x, \ldots, x]_{u_1,\ldots,u_k})\sigma_1$. Since $w' \in \overline{O}(g_{|u_i})$, $i = 1, \ldots, k$, then

$$\begin{array}{lll} g & \overset{[u_1.w', r_1, \sigma_1]}{\leadsto} & (C_1, g[t[\rho_1]_{w'}, t, \ldots, t]_{u_1,u_2,\ldots,u_k})\sigma_1 \\ & \overset{[u_2.w', r_1, \epsilon]}{\leadsto} & \ldots \\ & \overset{[u_k.w', r_1, \epsilon]}{\leadsto} & (C_1, \ldots, C_1, g[t[\rho_1]_{w'}, \ldots, t[\rho_1]_{w'}]_{u_1,\ldots,u_k})\sigma_1 \equiv h_2. \end{array}$$

Trivially, all steps in this derivation except for the first one could be considered as rewriting steps, since no instantiation of goal variables is done. Let $g_2 = (C_1, g[t[\rho_1]_{w'}, \ldots, t[\rho_1]_{w'}]_{u_1,\ldots,u_k})\sigma_1$ (i.e., g_2 is equal to h_2 except for the possible repetition of some conditions). It is immediate that g_2 and h_2 compute the same answers. Since $x \notin Dom(\sigma_1)$, we have $g'_2 \in flat(g_2)$. Let $\sigma' = \sigma_2 \ldots \sigma_{n-1}$; we have that $g'_2 \overset{\sigma'}{\leadsto}{}^* \top$ is based on B_2. By the induction hypothesis, $g_2 \overset{\sigma}{\leadsto}{}^* \top$, with $\sigma = \sigma'$ $[Var(g_2)]$, and thus $g \overset{\theta}{\leadsto}{}^* \top$, with $\theta = \sigma_1 \ldots \sigma_{n-1} = \theta'$ $[Var(g)]$, which ends the proof.

□

Now we can lift the previous results to the case of an S-flattening.

PROPOSITION 3.25. *Let $\mathcal{R}$ be a program in $I\!R_b$ and S a finite set of terms. Let g be a goal and g' an S-flattening of g. Then, for each basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$, there exists a basic conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^* \top]$ such that $\theta' \leq \theta$ $[Var(g)]$.*

PROOF. Immediate by repeated application of Lemma A.17. □

PROPOSITION 3.26. *Let $\mathcal{R}$ be a CTRS and S a finite set of terms. Let g be a goal and g' an S-flattening of g. Then, for each basic conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^* \top]$, there exists a basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$ such that $\theta = \theta'$ $[Var(g)]$.*

PROOF. It follows by repeated application of Lemma A.18, since the initial derivation is basic, and thus it is based on $\overline{O}(g)$. Note that, by definition of S-flattening, g' can be obtained through the application of a finite number of elementary flattenings. □

The following example shows that a similar result to Proposition 3.25, guaranteeing that $\theta \in \mathcal{O}_{\mathcal{R}}(g)$ implies $\theta' \in \mathcal{O}_{\mathcal{R}}(g')$, with $\theta' \leq \theta$ $[Var(g)]$, does not hold for unrestricted conditional narrowing under the conditions for the completeness of this calculus.

Example A.21. Consider again the program $\mathcal{R}$ in Example 3.17 and the goal $g \equiv (g(a, a) = c)$. Now, there exists the following unrestricted conditional narrowing refutation

$$\begin{array}{rcl} g(a, \underline{a}) = c & \overset{\epsilon}{\leadsto} & \underline{g(a, b)} = c \\ & \overset{\epsilon}{\leadsto} & \underline{c = c} \\ & \overset{\epsilon}{\leadsto} & \top \end{array}$$

which yields the normalized CAS ϵ in $\mathcal{R}$. However, it is easy to see that narrowing does not compute a more general answer for the flattened goal $g' = (a = x,\ g(x,x) = c)$ in $\mathcal{R}$. Note that the derivation

$$\begin{array}{rcl} g' = (\underline{a = x},\ g(x,x) = c) & \stackrel{\{x/a\}}{\leadsto} & true,\ g(a,\underline{a}) = c \\ & \stackrel{\epsilon}{\leadsto} & true,\ \underline{g(a,b)} = c \\ & \stackrel{\epsilon}{\leadsto} & true,\ \underline{c = c} \\ & \stackrel{\epsilon}{\leadsto} & \top \end{array}$$

computes the substitution $\{x/a\}$, which satisfies $\{x/a\} = \epsilon\ [Var(g)]$ but is not normalized.

We finally consider the proof of Proposition 3.29. The proof of Proposition 3.30 is analogous (using Proposition 3.26 in place of Proposition 3.25).

PROPOSITION 3.29. *Let S be a finite set of terms and $\mathcal{R}$ be an S-closed program in $I\!R_b$. Let $\mathcal{R}_{flat}$ be an S-flattening of $\mathcal{R}$ and g be a goal. Then, for each basic narrowing refutation $[g \stackrel{\theta}{\leadsto}^* \top]$ for g in $\mathcal{R}$, there exists a basic narrowing refutation $[g \stackrel{\theta'}{\leadsto}^* \top]$ for g in $\mathcal{R}_{flat}$ such that $\theta' \leq \theta\ [Var(g)]$.*

PROOF. Assume the following basic narrowing refutation for g in $\mathcal{R}$:

$$\mathcal{D} \equiv [g \equiv g_0 \stackrel{[u_1,r_1,\theta_1]}{\leadsto} g_1 \stackrel{[u_2,r_2,\theta_2]}{\leadsto} \ldots \stackrel{[u_n,r_n\theta_n]}{\leadsto} g_n \equiv \top]$$

with $\theta = \theta_1\theta_2\ldots\theta_n$. We make the proof by induction on the number n of steps of the derivation.

Since the base case is trivial, let us consider the inductive case $n > 1$. Since the lhs's of the rules in $\mathcal{R}$ and $\mathcal{R}_{flat}$ are identical, then we have that $g_0 \stackrel{[u_1,r'_1,\theta_1]}{\leadsto} g'_1$ in $\mathcal{R}_{flat}$ where $r'_1 \in \mathcal{R}_{flat}$ is a S-flattened version of $r_1 \in \mathcal{R}$. Now, we consider two cases:

—If $r_1 \equiv r'_1$ (i.e., the rule r_1 was already S-closed with the nonrecursive notion of closedness), then the claim follows by the inductive hypothesis.

—Otherwise, assume that $r_1 \equiv (\lambda_1 \to \rho_1 \Leftarrow C_1)$. Then, by definition of S-flattening for programs, we have that $r'_1 \equiv (\lambda_1 \to \rho'_1 \Leftarrow C'_1)$, where the goal $(C'_1, \rho'_1 = y)$ is an S-flattening of $(C_1, \rho_1 = y)$, with $y \notin (Var(C_1) \cup Var(\rho_1))$. Then, it is immediate to see that $g'_1 \equiv (C'_1, g_0[\rho'_1]_{u_1})\theta_1$ is an S-flattening of $g_1 \equiv (C_1, g_0[\rho_1]_{u_1})\theta_1$. Hence, by Proposition 3.25, there exists a basic narrowing refutation $[g'_1 \stackrel{\sigma}{\leadsto}^* \top]$ for g'_1 in $\mathcal{R}$, with $\sigma \leq \theta_2 \ldots \theta_n$, and the claim follows by inductive hypothesis.

□

A.5 Proofs of Strong Soundness and Completeness

In order to prove Lemma 3.34, we first need the following auxiliary definition, which formalizes the notion of *covering set* for a term t with respect to a set S. Roughly speaking, we say that each element of the covering set is also a set (*closure set*), formed by the (nested) positions of t which are considered when the definition of closedness is used to inductively test if t is S-closed in all possible ways, i.e., by

exploiting all possibilities to cover t using the elements of S. This definition is also useful to ease the implementation of the tests of closedness.

Definition A.23. Let S be a finite set of terms and t be an S-closed term. We define the *covering set* of t with respect to S as follows:

$$CSet(S,t) = \{O \mid O \in c_set(S,t),\ u.0 \notin O,\ u \in I\!N^*\}$$

where the auxiliary function *c_set*, used to compute each closure set O, is defined inductively as

$$c_set(S,t) \ni \begin{cases} \emptyset & \text{if } t \in V \text{ or } t \equiv c \in \mathcal{C}, \\ \bigcup_{i \in \{1,\ldots,n\}} \{i.p \mid p \in c_set(S,t_i)\} & \text{if } t \equiv c(t_1,\ldots,t_n),\ c \in \mathcal{C}, \\ \{\Lambda\} \cup \{u.p \mid s_{|u} = x,\ p \in c_set(S, x\theta)\} & \text{if } t \equiv f(t_1,\ldots,t_n),\ f \in \mathcal{F}, \\ & x \in V \text{ and } \exists s \in S.\ s\theta = t, \\ \{0\} & \text{otherwise.} \end{cases}$$

Note that positions ending with the mark 0 identify the situation in which some subterm of t is not an instance of any of the terms in S. That is, a set of positions containing a position of the form $u.0$ is not considered a closure set. The function $CSet$ is extended to equations and goals in the natural way.

We note that, given a set of terms S, the way in which a term t can be proved S-closed is not unique. This is why the set-valued function $CSet$ can compute several closure sets for the given term. The following example illustrates this point.

Example A.24. Let $S = \{f(x), g(x), f(g(x))\}$ and consider the term $t = f(g(0))$. Then, there are two possibilities to verify that t is S-closed:

(1) Consider $f(g(x)) \in S$. Then $t = f(g(x))\{x/0\}$ and, trivially, $closed^+(S, \{0\})$. Here, the associated closure set is $\{\Lambda\}$.

(2) Now consider $f(x) \in S$. Then $t = f(x)\{x/g(0)\}$. Now, we can verify that $closed^+(S, \{g(0)\})$, since $g(x) \in S$ and $closed^+(S, \{0\})$. In this case, the associated closure set is $\{\Lambda, 1\}$.

Hence, we have $CSet(S,t) = \{\{\Lambda\}, \{\Lambda, 1\}\}$.

The following auxiliary lemma is also needed.

Lemma A.25. *If S is an independent set of terms, then a term t can be S-closed in a unique way, i.e., the set $CSet(S,t)$ is a singleton.*

Proof. The proof is by contradiction. Assume that there exists a term t which might be S-closed in two distinct forms. By definition of closedness, we can easily show by structural induction that this can only happen if there exists a subterm $t_{|u}$ of t which is an instance of two (distinct) terms $s, s' \in S$. Thus, there exist two substitutions θ, θ' such that $t_{|u} = s\theta$ and $t_{|u} = s'\theta'$. Hence, since variables in the terms of S are renamed apart, the substitution $\vartheta = \theta \cup \theta'$ is a unifier of s and s' ($s\vartheta = s'\vartheta$), which contradicts the independence of S. $\square$

Now we can prove the desired lemma:

LEMMA 3.34. *Let S be an independent set of terms, t an S-closed term, and $t_{|u}$ a subterm of t such that $u \in \overline{O}(t)$. If $t_{|u}$ unifies with some term $s \in S$, then $s \leq t_{|u}$.*

PROOF. The proof is by contradiction. Let us assume that there exists a subterm $t_{|u}$ of t, $u \in \overline{O}(t)$, such that $t_{|u}$ unifies with $s \in S$ and $s \not\leq t_{|u}$.

Let $\{u_1, \ldots, u_n\} \in CSet(S, t)$. Hence $\forall i \in \{1, \ldots, n\}$. $\exists s_i \in S$. $s_i \leq t_{|u_i}$, $n \geq 0$. By Lemma A.25, this set of positions is unique. Thus, we can consider two cases:

(1) $u \in \{u_1, \ldots, u_n\}$. Then, there exists k, $1 \leq k \leq n$, such that $u \equiv u_k$ and $s_k \leq t_{|u_k}$. Since there exists a term $s \in S$ such that $t_{|u_k}$ unifies with s and such that $s \not\leq t_{|u_k}$, then $s_k \not\equiv s$. Finally, since $s_k \leq t_{|u_k}$ and $t_{|u_k}$ unify with s, then s_k and s unify, which contradicts the independence of S.

(2) $u \notin \{u_1, \ldots, u_n\}$. In this case, by definition of closedness and the fact that the set of closure positions $\{u_1, \ldots, u_n\}$ is unique, there exists i, $1 \leq i \leq n$, and w such that $u = u_i.w$, $s_i \leq t_{|u_i}$. If we choose number i such that u_i is the greatest position which satisfies the previous requirements, then $s_{i|w} \notin V$. Thus, since $s_i \leq t_{|u_i}$ and $u_i.w = u$, then $s_{i|w} \leq t_{|u}$. Finally, we have to distinguish two cases:
   (a) $s \not\equiv s_i$. Since $s_{i|w} \leq t_{|u}$ and s unifies with $t_{|u}$, then $s_{i|w}$ unifies with s, $s_{i|w} \notin V$, and hence, there is an overlapping of two terms in S, which contradicts the hypothesis of independence for S.
   (b) $s \equiv s_i$. Then $s_{|w} \leq t_{|u}$, and given that $t_{|u}$ unifies with s, there exists a proper subterm $s_{|w}$ of s which unifies with s, contradicting again the independence of S.

□

We can finally prove the strong soundness of partial evaluation.

THEOREM 3.35. *Let $\mathcal{R}$ be a program, g a goal, S a finite and independent set of terms, and $\mathcal{R}'$ a partial evaluation of $\mathcal{R}$ with respect to S such that $\mathcal{R}' \cup \{g\}$ is S-closed. Then, for all $\theta' \in \mathcal{O}_{\mathcal{R}'}(g)$ there exists $\theta \in \mathcal{O}_{\mathcal{R}}(g)$ such that $\theta = \theta'$ $[Var(g)]$.*

PROOF. In order to prove this theorem, it suffices to prove the following fact. Given an S-closed term t such that $(t = z) \overset{[u,r',\theta]}{\leadsto} (C, t[\rho]_u = z)\theta$, where $r' \equiv (s\vartheta \to \rho \Leftarrow C) \ll \mathcal{R}'$, $s \in S$, there exists the following narrowing derivation

$$t = z \overset{\theta'}{\leadsto}{}^n (C, t[\rho]_u = z)\theta'$$

which uses the rules of the original program $\mathcal{R}$ and such that $\theta = \theta'$ $[Var(t)]$.

Firstly, since $t = z \overset{[u,r',\theta]}{\leadsto} (C, t[\rho]_u = z)\theta$, then $\theta = mgu(\{t_{|u} = s\vartheta\}) \not\equiv fail$. Let us consider that the resultant r' has been produced from the following narrowing derivation for s in $\mathcal{R}$:

$$s = y \overset{\vartheta}{\leadsto}{}^n (C, \rho = y)$$

Since $t_{|u}$ and $s\vartheta$ unify, then also $t_{|u}$ and s unify. Hence, by Lemma 3.34, it holds that $s \leq t_{|u}$, and therefore there exists a substitution γ such that $s\gamma = t_{|u}$. Following an argument similar to that in the proof of Lemma 3.3, it is immediate to prove that, since there exists the derivation $s = y \overset{\vartheta}{\leadsto}{}^n (C, \rho = y)$ in $\mathcal{R}$ and $\theta = mgu(\{t_{|u} =$

$s\vartheta\}) = mgu(\{s\gamma = s\vartheta\}) \not\equiv fail$, then the following derivation can be proven:

$$t = z \overset{\theta'}{\leadsto}{}^n (C, t[\rho]_u = z)\theta'$$

using the same rules and in the same order as in the derivation for $(s = y)$ (at the corresponding positions), in such a way that $\theta = \theta'$ $[Var(t)]$. By Lemma 3.34, it follows that no additional answer can be produced, since it is not possible to use resultants which are computed by starting from terms which are more instantiated than the selected subterm $t_{|u}$. This completes the proof. □

Now we consider the proof of Proposition 3.36. The following lemma is auxiliary. It mainly establishes that, when we consider a linear set of calls, closedness is not lost after extracting a single occurrence of a subterm by elementary flattening.

LEMMA A.28. *Let S be a finite and linear set of terms, t an S-closed term, and $x \notin Var(t)$ a fresh variable. Then, for all $O \in CSet(S,t)$ and position u of O, the term $t[x]_u$ is S-closed.*

PROOF. Let $O \in CSet(S,t)$, $O \not\equiv \emptyset$. We prove the lemma by structural induction over the structure of the term t.

(1) if $t \in V$ or $t \equiv c$ then O is empty, and the claim follows by vacuity.
(2) if $t \equiv c(t_1, t_2, \ldots, t_n)$ then let us assume without loss of generality that $t[x]_u = c(t_1[x]_{u'}, t_2, \ldots, t_n)$. Then, by structural induction, the terms $t_1[x]_{u'}, t_2, \ldots, t_n$ are S-closed, and hence the term $c(t_1[x]_{u'}, t_2, \ldots, t_n)$ is also S-closed.
(3) if $t \equiv f(t_1, t_2, \ldots, t_n)$, where $f \in \mathcal{F}$, then there exists $s \in S$ such that $s\theta = t$ with $\theta = \{x_1/t_{|p_1}, \ldots, x_k/t_{|p_k}\}$, $\{p_1, \ldots, p_k\} \subseteq O$, and the terms $t_{|p_1}, \ldots, t_{|p_k}$ are S-closed. Now, let us assume without loss of generality that $p_1 \leq u$ (the case $u \equiv \Lambda$ is trivial). Then, we can construct a new substitution $\theta' = \{x_1/(t[x]_u)_{|p_1}, \ldots, x_k/t_{|p_k}\}$. Finally, by structural induction, the terms $(t[x]_u)_{|p_1}, \ldots, t_{|p_k}$ are S-closed, and since s is linear, it holds that $s\theta' = t[x]_u$, which completes the proof.

□

The next lemma is a strengthening of Lemma A.17 for the case when only single occurrences of terms in g are extracted by elementary flattening.

LEMMA A.29. *Let $\mathcal{R}$ be a program in $I\!R_b$, g a goal, and $g' \in flat(g)$ an elementary flattening of g such that only one occurrence of the considered term is extracted. Let $B \subseteq \overline{O}(g)$ be a set of positions of g and $B' = flat_pos(B, g')$. If there exists a conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^* \top]$ based on B, then there exists a conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^* \top]$ based on B' such that $\theta' = \theta$ $[Var(g)]$.*

PROOF. Let $\mathcal{D}$ be the following successful derivation for g in $\mathcal{R}$ based on B:

$$g \equiv g_1 \overset{[u_1, r_1, \theta_1]}{\leadsto} g_2 \overset{[u_2, r_2, \theta_2]}{\leadsto} \ldots \overset{[u_{m-1}, r_{m-1}, \theta_{m-1}]}{\leadsto} g_m \equiv \top, \quad \theta = \theta_1 \ldots \theta_{m-1},$$

where $r_i \equiv (\lambda_i \to \rho_i \Leftarrow C_i) \ll \mathcal{R}_+$, $u_i \in B_i$ (given by the definition of basic narrowing), $\theta_i = mgu(\{g_{i|u_i} = \lambda_i\})$, $g_{i+1} = (C_i, g_i[\rho_i]_{u_1})\theta_i$, $i = 1, \ldots, m-1$,

$m > 1$. We prove the claim by induction on the number $n = (m - 1)$ of steps of this derivation $\mathcal{D}$.

The base case $n = 1$ is trivial, since then the goal g has been solved using the rule $(x = x \;\rightarrow\; true)$, and a corresponding basic narrowing refutation for g' in $\mathcal{R}$ can be built by employing the rule $(x = x \;\rightarrow\; true)$ twice.

Let us consider the inductive case $n > 1$. Since $g' \in flat(g)$, then $g'_1 \equiv (s = x, g_1[x]_p)$, where $s = g_{1|p}$, $x \notin Var(g_1)$, and $p \in \overline{O}(g_1)$. We split the derivation $\mathcal{D}$ into $\mathcal{D}_1\mathcal{D}_2$ where $\mathcal{D}_1 \equiv g_1[s]_p \overset{[u_1, r_1, \theta_1]}{\leadsto} g_2$, and $\mathcal{D}_2 = g_2 \overset{\theta_2}{\leadsto} \ldots \overset{\theta_n}{\leadsto} \top$. Now, we have the following possibilities:

(1) If $p \leq u_1$, then $u_1 = p.u'$ and $g_2 = (C_1, g_1[s[\rho_1]_{u'}]_p)\theta_1$. Thus, the following narrowing step can be proved for g'_1:

$$g'_1 \equiv (s = x, g_1[x]_p) \overset{[1.1.u', r_1, \theta_1]}{\leadsto} g'_2$$

where $g'_2 = (C_1, s[\rho_1]_{u'} = x, g_1[x]_p)\theta_1$. Now, since $x \notin Var(g_1)$, then $x \notin Dom(\theta_1)$, and thus $g'_2 = (C_1\theta_1, (s[\rho_1]_{u'})\theta_1 = x, g_1\theta_1[x]_p)$ and $g'_2 \in flat(g_1)$. Since $\mathcal{D}_2$ is based on B_2 then, by the induction hypothesis, there exists a successful derivation for g'_2 in $\mathcal{R}$ based on B'_2 with CAS θ''. Therefore we obtain the narrowing refutation $g' \overset{\theta'}{\leadsto}{}^{*} \top$ with $\theta' = \theta_1\theta'' = \theta_1 \ldots \theta_n = \theta\ [Var(g_1)]$, and the claim follows.

(2) If $p \perp u_1$, the proof is perfectly analogous to case (1).

(3) If $u_1 \leq p$, the claim is trivial, since we can perform the following step on g'_1

$$g'_1 \overset{\{x/g_{|p}\}}{\leadsto} g_1,$$

and thus we can construct a basic narrowing refutation for g' in $\mathcal{R}$ by computing exactly the same steps as in $\mathcal{D}$ (note that $g_{|p}$ cannot be reduced either in $\mathcal{D}$ or in $\mathcal{D}'$, since the position p is not basic).

□

Finally, Proposition 3.36 follows easily from the previous lemma.

PROPOSITION 3.36. *Let $\mathcal{R}$ be a program in $I\!R_b$ and S a finite and linear set of terms. Let g be a goal and g' an S-flattening of g. Then, for each basic conditional narrowing refutation $\mathcal{D} \equiv [g \overset{\theta}{\leadsto}{}^{*} \top]$, there exists a basic conditional narrowing refutation $\mathcal{D}' \equiv [g' \overset{\theta'}{\leadsto}{}^{*} \top]$ such that $\theta' = \theta\ [Var(g)]$.*

PROOF. Since $\mathcal{D}$ is basic, then $\mathcal{D}$ is based on $B = \overline{O}(g)$. Moreover, it is straightforward to see that, under the linearity assumption, an S-flattening g' for the goal g can be obtained by extracting a single occurrence of a subterm of g. Hence, the claim follows by repeated application of Lemma A.29. □

B. PROOFS FOR THE NARROWING-DRIVEN PE ALGORITHM

In this appendix we give the proofs of some technical results in Sections 4 and 5.

B.1 Proof of Theorem 4.6

In order to prove this theorem, we first need the following lemma which establishes the transitivity of the closedness relation. Here we use the notation $depth(t)$ to denote the maximum number of nested symbols in the term t. Formally, if t is a constant or a variable, then $depth(t) = 1$. Otherwise, $depth(f(t_1, \ldots, t_n)) = 1 + max(\{depth(t_1), \ldots, depth(t_n)\})$.

LEMMA B.1. *If term t is S_1-closed, and S_1 is S_2-closed, then t is S_2-closed.*

PROOF. We proceed by structural induction on t.

Since the base case $depth(t) = 1$ is trivial, we proceed with the inductive case. Let $depth(t) = k+1$, $k > 0$, and the property holds for all t' such that $depth(t') \leq k$. Since t is S_1-closed (and $depth(t) > 0$), following the definition of closedness, we need to distinguish two cases:

(1) if t is headed by a constructor symbol, then $t \equiv c(t_1, \ldots, t_n)$, $c \in \mathcal{C}$, and $\{t_1, \ldots, t_n\}$ is S_1-closed. Since $depth(t_i) \leq k$, $i = 1, \ldots, n$, then by the induction hypothesis, $closed^+(S_2, t_i)$ holds for all $i = 1, \ldots, n$. Hence, by definition of closedness, t is also S_2-closed.

(2) If t is headed by a defined function symbol, then $t \equiv f(t_1, \ldots, t_n)$, $f \in \mathcal{F}$, and there exists $s_1 \in S_1$ such that $s_1\theta_1 = t$ and $closed^+(S_1, s')$ holds for all $x/s' \in \theta_1$. Since $s_1 \in S_1$, then by hypothesis we have that s_1 is S_2-closed. Hence, there exists $s_2 \in S_2$ such that $s_2\theta_2 = s_1$ (since s_1 is also headed by a defined function symbol), and $closed^+(S_2, s'')$ holds for all $x/s'' \in \theta_2$. Then, we have that there is a term $s_2 \in S_2$ such that $s_2\theta_2\theta_1 = s_1\theta_1 = t$. To prove that t is S_2-closed we only need to show that $closed^+(S_2, t')$ holds for all x/t' in $\theta_1\theta_2$. Since $depth(s') \leq k$ for all $x/s' \in \theta_1$, then by the induction hypothesis, we have that $closed^+(S_2, s')$ also holds for all $x/s' \in \theta_1$. Finally, since $closed^+(S_2, s'')$ holds for all $x/s'' \in \theta_2$, it is immediate that $x\theta_2\theta_1$ is S_2-closed for all $x \in Dom(\theta_2\theta_1)$, which ends the proof.

□

Now, the desired result can be proved.

THEOREM 4.6. *Let $\mathcal{R}$ be a program and g be a goal. If $\mathcal{P}(\mathcal{R}, g)$ terminates computing the set of terms S, then $\mathcal{R}' \cup \{g\}$ is S-closed, where the specialized program is given by $\mathcal{R}' = U_\varphi(S, \mathcal{R})$.*

PROOF. Assume that $\mathcal{P}(\mathcal{R}, g)$ terminates computing S. From Definition 4.5, we have $c[S] \longmapsto c[S]$, with $\mathcal{R}' = U_\varphi(S, \mathcal{R})$ and $c[S] = abstract(c[S], \mathcal{R}'_{calls})$. By the definition of abstraction operator (Definition 4.3), for all $t \in \mathcal{R}'_{calls}$, we have that t is S-closed. Hence, $\mathcal{R}'$ is S-closed.

Now we show that g is also S-closed. Let $c_1[S_1] = abstract(c_0, g_{calls})$. Since $S_1 = g_{calls}$, it trivially follows that g is S_1-closed. By definition of abstraction operator (Definition 4.3), for all $i > 0$, if $c_i[S_i] \longmapsto_{\mathcal{P}} c_{i+1}[S_{i+1}]$ then S_i is S_{i+1}-closed. Finally, since we have a path $c_1[S_1] \longmapsto^*_{\mathcal{P}} c[S]$ and g is S_1-closed, the claim follows by transitivity (Lemma B.1). □

B.2 Termination Proofs

We first introduce the following preliminary notations. We define the complexity $\mathcal{M}_S$ of a set S as the finite multiset of natural numbers corresponding to the depth of the elements of S. Formally, $\mathcal{M}_S = \{depth(s) \mid s \in S\}$. We consider the well founded total ordering $<_{mul}$ over multiset complexities by extending the well founded ordering $<$ on $I\!N$ to the set $M(I\!N)$ of finite multisets over $I\!N$. The set $M(I\!N)$ is well founded under the ordering $<_{mul}$, since $I\!N$ is well founded under $<$ [Dershowitz and Jouannaud 1990]. Let $\mathcal{M}, \mathcal{M}'$ be multiset complexities, then $\mathcal{M} <_{mul} \mathcal{M}' \;\Leftrightarrow\; \exists X \subseteq \mathcal{M}, X' \subseteq \mathcal{M}'$ such that $\mathcal{M} = (\mathcal{M}' - X') \cup X$ and $\forall n \in X,\ \exists n' \in X'.\ n < n'$.

Lemma 5.14. *The function abstract* is an abstraction operator.*

Proof. Let q be a PE* configuration, and let T be a set of terms. Assume that S_q is the set of terms contained in q. Then, following the definition of abstraction operator (Definition 4.3), we need to prove that

(1) if $s \in abstract^*(q, T)$ then there exists $t \in (S_q \cup T)$ such that $t_{|p} = s\theta$ for some position p and substitution θ and
(2) for all $t \in (S_q \cup T)$, t is closed with respect to $abstract^*(q, T)$.

Condition (1) is trivially fulfilled, since $abstract^*$ only applies the msg operator, which cannot introduce function symbols not appearing in S_q or T in the resulting configuration. Now we prove condition (2) by well-founded induction on $S_q \cup T$.

If $S_q \cup T \equiv \emptyset$, then $abstract^*(\emptyset, \emptyset) = \emptyset$, and the proof is done.

Let us consider the inductive case $S_q \cup T \neq \emptyset$, and assume that T is not empty (otherwise the proof is trivial). Then, $T \equiv T_0 \cup \{t\}$, with $T_0 = \{t_1, \ldots, t_{n-1}\}$. By the inductive hypothesis, the property holds for all $q^\star$ and for all $T^\star$ such that $\mathcal{M}_{S_{q^\star} \cup T^\star} <_{mul} \mathcal{M}_{S_q \cup T}$. Following the definition of $abstract^*$, we now consider three cases:

(1) If t is a variable symbol, or t is a constant value $c \in \mathcal{C}$, then

$$\begin{aligned} q' &= abstract^*(q, T) \\ &= abstract^*(q, T_0 \cup \{t\}) \\ &= abstract^*(abstract^*(\ldots abstract^*(q, t_1), \ldots, t_{n-1}), t) \\ &= abstract^*(abstract^*(q, T_0), t) \\ &= abstract^*(q, T_0). \end{aligned}$$

Since $\mathcal{M}_{S_q \cup T_0} <_{mul} \mathcal{M}_{S_q \cup T_0 \cup \{t\}} = \mathcal{M}_{S_q \cup T}$ then, by the inductive hypothesis, $S_q \cup T_0$ is closed with respect to the terms in q', and by the definition of closedness, so is $S_q \cup T_0 \cup \{t\} = S_q \cup T$.

(2) If $t \equiv c(s_1, \ldots, s_m)$, $c \in \mathcal{C}$, $m > 0$, then by definition of $abstract^*$

$$\begin{aligned} q' &= abstract^*(q, T) \\ &= abstract^*(q, T_0 \cup \{t\}) \\ &= abstract^*(abstract^*(q, T_0), c(s_1, \ldots, s_m)) \\ &= abstract^*(abstract^*(q, T_0), \{s_1, \ldots, s_m\}) \\ &= abstract^*(\ldots abstract^*(abstract^*(q, T_0), s_1), \ldots, s_m) \\ &= abstract^*(q, T_0 \cup \{s_1, \ldots, s_m\}). \end{aligned}$$

Since $\mathcal{M}_{S_q \cup T_0 \cup \{s_1,\ldots,s_m\}} <_{mul} \mathcal{M}_{S_q \cup T_0 \cup \{c(s_1,\ldots,s_m)\}} = \mathcal{M}_{S_q \cup T}$ then, by the inductive hypothesis, $S_q \cup T_0 \cup \{s_1, \ldots, s_m\}$ is closed with respect to the terms in q', and by the definition of closedness, so is $S_q \cup T_0 \cup \{c(s_1, \ldots, s_m)\} = S_q \cup T$.

(3) If $t \equiv f(s_1, \ldots, s_m)$, $f \in \mathcal{F}$, $m \geq 0$, then by definition of *abstract**

$$\begin{aligned} q' &= abstract^*(q, T) \\ &= abstract^*(q, T_0 \cup \{t\}) \\ &= abstract^*(abstract^*(q, T_0), t) \\ &= abs_call(abstract^*(q, T_0), t). \end{aligned}$$

Assume that $abstract^*(q, T_0)$ returns $q'' \equiv (q_1, \ldots, q_p)$, $p \geq 1$ (since the case when $abstract^*(q, T_0)$ is *nil* is straightforward). Here we distinguish three cases, which correspond to the three case values of the function *abs_call* in definition of *abstract**, given by $abs_call(q'', t) =$

(a) (q'', t). Since $\mathcal{M}_{S_q \cup T_0} <_{mul} \mathcal{M}_{S_q \cup T}$ then, by the inductive hypothesis, $S_q \cup T_0$ is closed with respect to the terms in $S_{q''}$, and therefore it is closed with respect to $q' \equiv (q'', t)$. Then, the claim follows trivially from the fact that t is closed with respect to the terms in (q'', t).

(b) $abstract^*((q_1, \ldots, q_p), S)$, where there exists $i \in \{1, \ldots, p\}$ such that $q_i\theta = t$ and $S = \{s \mid x/s \in \theta\}$. First, we have that $abstract^*((q_1, \ldots, q_p), S) = abstract^*(abstract^*(q, T_0), S) = abstract^*(q, T_0 \cup S)$. Since $\mathcal{M}_{S_q \cup T_0 \cup S} <_{mul} \mathcal{M}_{S_q \cup T_0 \cup \{t\}} = \mathcal{M}_{S_q \cup T}$ then, by the inductive hypothesis, $S_q \cup T_0 \cup S$ is closed with respect to the terms in $abstract^*(q, T_0 \cup S)$ and hence, since we have shown that $q' = abstract^*(q, T_0 \cup S)$, we derive that the terms in $S_q \cup T_0 \cup S$ are closed with respect to the terms in q'. By definition of *abstract**, it is immediate to see that $\mathcal{M}_{S_{q''}} \leq_{mul} \mathcal{M}_{S_q \cup T_0}$, since q'' can at most contain the original terms in q and T_0 (or a set of terms with a lower complexity). Therefore $\mathcal{M}_{S_{q''} \cup S} <_{mul} \mathcal{M}_{S_q \cup T_0 \cup \{t\}} = \mathcal{M}_{S_q \cup T}$. Then, by the inductive hypothesis, $S_{q''} \cup S$ is closed with respect to the terms in q'. Finally, the claim follows by Lemma B.1, since t is closed with respect to the terms in $S_{q''} \cup S$ by definition.

(c) $abstract^*((q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_p), S)$, where there exists $i \in \{1, \ldots, p\}$ such that $msg(\{q_i, t\}) = \langle w, \{\theta_1, \theta_2\}\rangle$, and $S = \{w\} \cup \{s \mid x/s \in \theta_1 \text{ or } x/s \in \theta_2\}$. The argument which proves this case is similar to that of case (2), using the fact that, by definition of closedness and *msg*, the set of terms $\{q_i, t\}$ is closed with respect to $(\{w\} \cup \{s \mid x/s \in \theta_1 \text{ or } x/s \in \theta_2\})$.

□

Finally, we proceed with the proof of Theorem 5.15.

THEOREM 5.15. *The narrowing-driven PE algorithm terminates for the domain State* of PE* configurations and the nonembedding unfolding rule (Definition 5.8) and abstraction operator (Definition 5.11).*

PROOF. Let $\mathcal{R}$ be a program and g_0 be a goal. Consider the following computation: $q_0 \longmapsto_{\mathcal{P}} \ldots \longmapsto_{\mathcal{P}} q_n \ldots$ for g_0 in $\mathcal{R}$, where $q_0 = abstract^*(nil, g_{0_{calls}})$ and $q_i = abstract^*(q_{i-1}, \mathcal{R}^{i-1}_{calls})$, with $\mathcal{R}^{i-1} = U^{\trianglelefteq}_{\varphi}(S_{q_{i-1}}, \mathcal{R})))$, $S_{q_{i-1}}$ the terms in q_{i-1}, and $i \geq 1$.

First, we prove that each subsequence of comparable terms in a PE* configuration $q_i \equiv (q_{i1}, \ldots, q_{im_i})$ satisfies the following property (*nonembedding property*):

$$\forall j, k \in \{1, \ldots m_i\}.\ (j < k \ \wedge\ comparable(q_{ij}, q_{ik})) \Rightarrow q_{ij} \not\trianglelefteq q_{ik})$$

This suffices to prove the theorem, as it follows directly from the following facts:

—The number of sequences of incomparable terms which can be formed using a finite number of defined function symbols is finite.

—By Theorem 5.2 and the nonembedding property stated above, the subsequences of comparable terms of any PE* configuration are finite.

—By Theorem 5.6, the sets $\mathcal{R}^i_{calls}$, $i \geq 1$, are all finite, and therefore the computation of each PE* configuration is also finite.

—The function *abstract** is well defined, in the sense that the computation of the new configuration always terminates (see Example 5.12).

We prove the claim by induction on the length n of the computation.

Let $n = 0$. We make the proof by well-founded induction on $T = g_{0_{calls}}$ using the well-founded total ordering $<_{mul}$ over multiset complexities.

If $T \equiv \emptyset$, then $q_0 = abstract^*(nil, \emptyset) = nil$, and the proof is done.

Let us consider the inductive case $T \equiv T_0 \cup \{t\}$, with $T_0 = \{t_1, \ldots, t_{k-1}\}$. By the inductive hypothesis, the property holds for all T' such that $\mathcal{M}_{T'} <_{mul} \mathcal{M}_T$. By definition of *abstract**, we have $q_0 = abstract^*(nil, T) = abstract^*(nil, T_0 \cup \{t\}) = abstract^*(abstract^*(nil, T_0), t))$, and by the induction hypothesis, the sequence of terms $q' = abstract^*(nil, T_0) = (q'_1, \ldots, q'_p)$, $p \geq 0$, fulfils the property that $\forall j, k \in \{1, \ldots, p\}.\ (j < k \ \wedge\ comparable(q'_j, q'_k)) \Rightarrow q'_j \not\trianglelefteq q'_k$. Following the definition of *abstract**, we now consider three cases:

(1) If t is a variable symbol, or t is a constant value $c \in \mathcal{C}$, then by definition of *abstract**, $q_0 = abstract^*(abstract^*(nil, T_0), t) = abstract^*(nil, T_0)$, and the claim follows directly from the induction hypothesis.

(2) If $t \equiv c(s_1, \ldots, s_m)$, $c \in \mathcal{C}$, $m > 0$, then by definition of *abstract**

$$\begin{aligned} q_0 &= abstract^*(abstract^*(nil, T_0), c(s_1, \ldots, s_m)) \\ &= abstract^*(abstract^*(nil, T_0), \{s_1, \ldots, s_m\}) \\ &= abstract^*(\ldots abstract^*(\ldots abstract^*(nil, t_1), \ldots, t_{k-1}), s_1, \ldots, s_m) \\ &= abstract^*(nil, T_0 \cup \{s_1, \ldots, s_m\}). \end{aligned}$$

Since $\mathcal{M}_{T_0 \cup \{s_1, \ldots, s_m\}} <_{mul} \mathcal{M}_{T_0 \cup \{c(s_1, \ldots, s_m)\}} = \mathcal{M}_T$, then the claim follows from the induction hypothesis.

(3) If $t \equiv f(s_1, \ldots, s_m)$, $f \in \mathcal{F}$, $n \geq 0$, then $q_0 = abstract^*(abstract^*(nil, T_0), t) = abs_call(abstract^*(nil, T_0), t)$. Assume that $abstract^*(nil, T_0) = (q'_1, \ldots, q'_p)$, with $p > 0$ (since the case when $abstract^*(nil, T_0)$ is nil is straightforward). Here we distinguish three cases, which correspond to the three case values of the function *abs_call* in definition of *abstract**, given by $abs_call((q'_1, \ldots, q'_p), t) =$

(a) $(q'_1, \ldots, q'_p, t)$;

(b) $abstract^*((q'_1, \ldots, q'_p), S)$, where there exists $i \in \{1, \ldots, p\}$ such that $q'_i\theta = t$ and $S = \{s \mid x/s \in \theta\}$;

(c) $abstract^*(q'', S)$, where there exists $i \in \{1, \ldots, p\}$ such that $msg(\{q'_i, t\}) = \langle w, \{\theta_1, \theta_2\}\rangle$, $q'' = (q'_1, \ldots, q'_{i-1}, q'_{i+1}, \ldots, q'_p)$, and $S = \{w\} \cup \{s \mid x/s \in \theta_1 \text{ or } x/s \in \theta_2\}$.

Let us first consider case (a). Then,

$$\begin{aligned} q_0 &= abs_call(abstract^*(nil, T_0), t) \\ &= abs_call((q'_1, \ldots, q'_p), t) \\ &= (q'_1, \ldots, q'_p, t), \end{aligned}$$

and the following facts hold: (1) by the induction hypothesis, the comparable terms in $(q'_1, \ldots, q'_p)$ satisfy the nonembedding property and (2) by the premise of case (a), the considered term t does not infringe the nonembedding property for any of the comparable terms in $(q'_1, \ldots, q'_p)$.

Now we consider case (b). Then,

$$\begin{aligned} q_0 &= abs_call(abstract^*(nil, T_0), t) \\ &= abs_call((q'_1, \ldots, q'_p), t) \\ &= abstract^*((q'_1, \ldots, q'_p), S) \\ &= abstract^*(abstract^*(nil, T_0), S) \\ &= abstract^*(nil, T_0 \cup S), \end{aligned}$$

with $q'_i\theta = t$. Since $\mathcal{M}_{(T_0 \cup S)} <_{mul} \mathcal{M}_T$ then, by the induction hypothesis, the comparable terms in q_0 satisfy the nonembedding property.

The argument which proves case (c) is similar to that of case (2), using the fact that $\mathcal{M}_{(S_{q''} \cup S)} <_{mul} \mathcal{M}_T$, where $S_{q''}$ is the set of terms in q''..

Let $n > 0$. Assume $q_n = abstract^*(q_{n-1}, \mathcal{R}^{n-1})$. By the induction hypothesis, each subsequence of comparable terms of q_{n-1} satisfies the nonembedding property, and then the proof is perfectly analogous to the base case. □

ACKNOWLEDGEMENTS

We wish to thank Jean Pierre Jouannaud, Robert Nieuwenhuis, Albert Rubio, and Morten Sørensen for useful discussions. Part of this research was done while the third author was visiting the University of Padova and of Udine with a fellowship from the HCM project Console. Germán Vidal gratefully acknowledges the hospitality of these departments.

We acknowledge the anonymous referees for their detailed comments and helpful suggestions, which have allowed us to substantially improve this article.

REFERENCES

Albert, E., Alpuente, M., Falaschi, M., Julián, P., and Vidal, G. 1998. Improving control in functional logic program specialization. In *Proceedings of the 5th Static Analysis Symposium, SAS'98*. Lecture Notes in Computer Science. Springer-Verlag, Berlin. To appear.

Albert, E., Alpuente, M., Falaschi, M., and Vidal, G. 1998. INDY user's manual. Tech. Rep. DSIC-II/12/98, UPV. Available from URL: `http://www.dsic.upv.es/users/elp/papers.html`.

Alpuente, M., Falaschi, M., Julián, P., and Vidal, G. 1997. Specialization of lazy functional logic programs. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*. ACM Press, New York, 151–162.

ALPUENTE, M., FALASCHI, M., AND LEVI, G. 1995. Incremental constraint satisfaction for equational logic programming. *Theor. Comput. Sci. 142,* 1, 27–57.

ALPUENTE, M., FALASCHI, M., AND VIDAL, G. 1996a. A compositional semantic basis for the analysis of equational Horn programs. *Theor. Comput. Sci. 165,* 1, 97–131.

ALPUENTE, M., FALASCHI, M., AND VIDAL, G. 1996b. Narrowing-driven partial evaluation of functional logic programs. In *Proceedings of the 6th European Symposium on Programming*, H. R. Nielson, Ed. Lecture Notes in Computer Science, vol. 1058. Springer-Verlag, Berlin, 45–61.

ALPUENTE, M., FALASCHI, M., AND VIDAL, G. 1998a. Experiments with the call-by-value partial evaluator. Tech. Rep. DSIC-II/13/98, UPV. Available from `http://www.dsic.upv.es/users/elp/papers.html`.

ALPUENTE, M., FALASCHI, M., AND VIDAL, G. 1998b. A unifying view of functional and logic program specialization. *ACM Comput. Surv.* To appear.

ANTOY, S., ECHAHED, R., AND HANUS, M. 1994. A needed narrowing strategy. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages.* ACM Press, New York, 268–279.

ARENAS, P., GIL, A., AND LÓPEZ, F. 1994. Combining lazy narrowing with disequality constraints. In *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming.* Lecture Notes in Computer Science, vol. 844. Springer-Verlag, Berlin, 385–399.

BAADER, F. AND NIPKOW, T. 1998. *Term Rewriting and All That.* Cambridge University Press, Cambridge, U.K.

BELLEGARDE, F. 1995. ASTRE: Towards a fully automated program transformation system. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, J. Hsiang, Ed. Lecture Notes in Computer Science, vol. 914. Springer-Verlag, Berlin, 403–407.

BENKERIMI, K. AND HILL, P. 1993. Supporting transformations for the partial evaluation of logic programs. *J. Logic Comput. 3,* 5, 469–486.

BENKERIMI, K. AND LLOYD, J. 1990. A partial evaluation procedure for logic programs. In *Proceedings of the 1990 North American Conference on Logic Programming*, S. Debray and M. Hermenegildo, Eds. The MIT Press, Cambridge, MA, 343–358.

BERT, D. AND ECHAHED, R. 1986. Design and implementation of a generic, logic and functional programming language. In *Proceedings of 1st European Symposium on Programming.* Lecture Notes in Computer Science, vol. 213. Springer-Verlag, Berlin, 119–132.

BERT, D. AND ECHAHED, R. 1995. On the operational semantics of the algebraic and logic programming language LPG. In *Recent Trends in Data Type Specifications.* Lecture Notes in Computer Science, vol. 906. Springer-Verlag, Berlin, 132–152.

BOCKMAYR, A. AND WERNER, A. 1995. LSE narrowing for decreasing conditional term rewrite systems. In *Proceedings of Conditional Term Rewriting Systems, CTRS'94.* Lecture Notes in Computer Science, vol. 968. Springer-Verlag, Berlin.

BOL, R. 1993. Loop checking in partial deduction. *J. Logic Program. 16,* 1&2, 25–46.

BONACINA, M. 1988. Partial evaluation by completion. In *Conferenza dell'Associazione Italiana per il Calcolo Automatico*, G. Italiani et al., Eds.

BONDORF, A. 1988. Towards a self-applicable partial evaluator for term rewriting systems. In *Proceedings of the International Workshop on Partial Evaluation and Mixed Computation*, D. Bjørner, A. Ershov, and N. Jones, Eds. North-Holland, Amsterdam, 27–50.

BONDORF, A. 1989. A self-applicable partial evaluator for term rewriting systems. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development, TAPSOFT'89*, J. Diaz and F. Orejas, Eds. Lecture Notes in Computer Science, vol. 352. Springer-Verlag, Berlin, 81–95.

BOSSI, A., GABBRIELLI, M., LEVI, G., AND MARTELLI, M. 1994. The s-semantics approach: Theory and applications. *J. Logic Program. 19-20*, 149–197.

BOWEN, K. AND KOWALSKI, R. 1982. Amalgamating language and metalanguage in logic programming. In *Logic Programming*, K. Clark and S. Tärnlund, Eds. Academic Press, New York, 153–172.

Bruynooghe, M., De Schreye, D., and Martens, B. 1992. A general criterion for avoiding infinite unfolding. *New Gen. Comput. 11,* 1, 47–79.

Burstall, R. and Darlington, J. 1977. A transformation system for developing recursive programs. *J. ACM 24,* 1, 44–67.

Cheong, P. and Fribourg, L. 1992. A survey of the implementations of narrowing. In *Declarative Programming. Workshops in Computing*, J. Darlington and R. Dietrich, Eds. Springer-Verlag and BCS, Berlin, 177–187.

Chin, W. 1993. Towards an automated tupling strategy. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'93.* ACM Press, New York, 119–132.

Consel, C. and Danvy, O. 1991. Static and dynamic semantics processing. In *Proceedings of the 18th Annual ACM Symposium on Principles of Programming Languages.* ACM Press, New York, 14–24.

Consel, C. and Danvy, O. 1993. Tutorial notes on partial evaluation. In *Proceedings of the 20th Annual ACM Symposium on Principles of Programming Languages.* ACM Press, New York, 493–501.

Darlington, J. 1982. Program transformations. In *Functional Programming and Its Applications*, J. Darlington, P. Henderson, and D. A. Turner, Eds. Cambridge University Press, Cambridge, U.K., 193–215.

Darlington, J. and Pull, H. 1988. A program development methodology based on a unified approach to execution and transformation. In *Proceedings of the International Workshop on Partial Evaluation and Mixed Computation*, D. Bjørner, A. Ershov, and N. Jones, Eds. North-Holland, Amsterdam, 117–131.

Dershowitz, N. 1995. Goal solving as operational semantics. In *Proceedings of the 1995 International Logic Programming Symposium*, J. Lloyd, Ed. The MIT Press, Cambridge, MA, 3–17.

Dershowitz, N. and Jouannaud, J.-P. 1990. Rewrite systems. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B, Formal Models and Semantics. Elsevier, Amsterdam, 243–320.

Dershowitz, N. and Jouannaud, J.-P. 1991. Notations for rewriting. *Bull. Euro. Assoc. Theor. Comp. Sci. 43*, 162–172.

Dershowitz, N. and Reddy, U. 1993. Deductive and inductive synthesis of equational programs. *J. Symbol. Comput. 15*, 467–494.

Echahed, R. 1988. On completeness of narrowing strategies. In *Proceedings of CAAP'88, the 13th Colloquium on Trees in Algebra and Programming.* Lecture Notes in Computer Science, vol. 299. Springer-Verlag, Berlin, 89–101.

Fernández, M. 1992. Narrowing based procedures for equational disunification. *Appl. Alg. Eng. Commun. Comput. 3*, 1–26.

Fribourg, L. 1985. SLOG: A logic programming language interpreter based on clausal superposition and rewriting. In *Proceedings of the 2nd IEEE International Symposium on Logic Programming.* IEEE Press, New York, 172–185.

Gallagher, J. 1993. Tutorial on specialisation of logic programs. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'93.* ACM Press, New York, 88–98.

Gallagher, J. and Bruynooghe, M. 1990. Some low-level source transformations for logic programs. In *Proceedings of the 2nd Workshop on Meta-Programming in Logic*, M. Bruynooghe, Ed. Department of Computer Science, KU Leuven, Belgium, 229–246.

Glück, R. and Klimov, A. 1993. Occam's razor in metacomputation: The notion of a perfect process tree. In *Proceedings of the 3rd International Workshop on Static Analysis, WSA'93*, P. Cousot, M. Falaschi, G. Filè, and A. Rauzy, Eds. Lecture Notes in Computer Science, vol. 724. Springer-Verlag, Berlin, 112–123.

Glück, R. and Sørensen, M. 1994. Partial deduction and driving are equivalent. In *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming.* Lecture Notes in Computer Science, vol. 844. Springer-Verlag, Berlin, 165–181.

Glück, R. and Sørensen, M. 1996. A roadmap to metacomputation by supercompilation. In *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation*, O. Danvy, R. Glück, and P. Thiemann, Eds. Lecture Notes in Computer Science, vol. 1110. Springer-Verlag, Berlin, 137–160.

Glück, R., Jørgensen, J., Martens, B., and Sørensen, M. 1996. Controlling conjunctive partial deduction of definite logic programs. In *Proceedings of the 8th International Symposium on Programming Languages: Implementations, Logics and Programs*. Lecture Notes in Computer Science, vol. 1140. Springer-Verlag, Berlin, 152–166.

Hanus, M. 1990. Compiling logic programs with equality. In *Proceedings of the 2nd International Workshop on Programming Language Implementation and Logic Programming*. Lecture Notes in Computer Science, vol. 456. Springer-Verlag, Berlin, 387–401.

Hanus, M. 1991. Efficient implementation of narrowing and rewriting. In *Proceedings of the International Workshop on Processing Declarative Knowledge*. Lecture Notes in Artificial Intelligence, vol. 567. Springer-Verlag, Berlin, 344–365.

Hanus, M. 1992. Improving control of logic programs by using functional logic languages. In *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, M. Bruynooghe and M. Wirsing, Eds. Lecture Notes in Computer Science, vol. 631. Springer-Verlag, Berlin, 1–23.

Hanus, M. 1994a. Combining lazy narrowing with simplification. In *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming*. Lecture Notes in Computer Science, vol. 844. Springer-Verlag, Berlin, 370–384.

Hanus, M. 1994b. The integration of functions into logic programming: From theory to practice. *J. Logic Program. 19&20*, 583–628.

Hanus, M. 1995. On extra variables in (equational) logic programming. In *Proceedings of the 20th International Conference on Logic Programming*. The MIT Press, Cambridge, MA, 665–678.

Hanus, M., Kuchen, H., and Moreno-Navarro, J. 1995. Curry: A truly functional logic language. In *Proceedings of the ILPS'95 Workshop on Visions for the Future of Logic Programming*. 95–107.

Hermenegildo, M. and Rossi, F. 1989. On the correctness and efficiency of independent And-parallelism in logic programs. In *Proceedings of the 1989 North American Conference on Logic Programming*, E. Lusk and R. Overbeck, Eds. The MIT Press, Cambridge, MA, 369–389.

Hölldobler, S. 1989. *Foundations of Equational Logic Programming*. Lecture Notes in Artificial Intelligence, vol. 353. Springer-Verlag, Berlin.

Hullot, J. 1980. Canonical forms and unification. In *Proceedings of the 5th International Conference on Automated Deduction*. Lecture Notes in Computer Science, vol. 87. Springer-Verlag, Berlin, 318–334.

Hussman, H. 1985. Unification in conditional-equational theories. In *Proceedings of the European Conference on Computer Algebra, EUROCAL'85*. Lecture Notes in Computer Science, vol. 204. Springer-Verlag, Berlin, 543–553.

Jaffar, J. and Lassez, J.-L. 1987. Constraint logic programming. In *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages*. ACM Press, New York, 111–119.

Jones, N. 1994. The essence of program transformation by partial evaluation and driving. In *Logic, Language and Computation*, N. Jones, M. Hagiya, and M. Sato, Eds. Lecture Notes in Computer Science, vol. 792. Springer-Verlag, Berlin, 206–224.

Jones, N., Gomard, C., and Sestoft, P. 1993. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Englewood Cliffs, NJ.

Jørgensen, J., Leuschel, M., and Martens, B. 1996. Conjunctive partial deduction in practice. In *Proceedings of the International Workshop on Logic Program Synthesis and Transformation, LOPSTR'96*, J. Gallager, Ed. Lecture Notes in Computer Science, vol. 1207. Springer-Verlag, Berlin, 59–82.

Klop, J. 1992. Term rewriting systems. In *Handbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. Maibaum, Eds. Vol. I. Oxford University Press, Oxford, 1–112.

KOMOROWSKI, H. 1982. Partial evaluation as a means for inferencing data structures in an applicative language: A theory and implementation in the case of Prolog. In *Proceedings of the 9th ACM Symposium on Principles of Programming Languages.* ACM Press, New York, 255–267.

LAFAVE, L. AND GALLAGHER, J. 1997a. Constraint-based partial evaluation of rewriting-based functional logic programs. In *Proceedings of the 7th International Workshop on Logic Program Synthesis and Transformation, LOPSTR'97.* Lecture Notes in Computer Science. Springer-Verlag, Berlin. To appear.

LAFAVE, L. AND GALLAGHER, J. 1997b. Partial evaluation of functional logic programs in rewriting-based languages. Tech. Rep. CSTR-97-001, Department of Computer Science, University of Bristol, Bristol, England.

LAM, J. AND KUSALIK, A. 1991. A comparative analysis of partial deductors for pure Prolog. Tech. Rep., Department of Computational Science, University of Saskatchewan, Canada. Revised April 1991.

LASSEZ, J.-L., MAHER, M. J., AND MARRIOTT, K. 1988. Unification revisited. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, Los Altos, CA, 587–625.

LEUSCHEL, M. 1998. The ECCE partial deduction system and the DPPD library of benchmarks. Tech. Rep., Accessible via `http://www.cs.kuleuven.ac.be/~lpai`.

LEUSCHEL, M. AND MARTENS, B. 1996. Global control for partial deduction through characteristic atoms and global trees. In *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation*, O. Danvy, R. Glück, and P. Thiemann, Eds. Lecture Notes in Computer Science, vol. 1110. Springer-Verlag, Berlin, 263–283.

LEUSCHEL, M., DE SCHREYE, D., AND DE WAAL, A. 1996. A conceptual embedding of folding into partial deduction: Towards a maximal integration. In *Proceedings of the Joint International Conference and Symposium on Logic Programming, JICSLP'96*, M. Maher, Ed. The MIT Press, Cambridge, MA, 319–332.

LEUSCHEL, M., MARTENS, B., AND DE SCHREYE, D. 1998. Controlling generalization and polyvariance in partial deduction of normal logic programs. *ACM Trans. Program. Lang. Syst. 20,* 1, 208–258.

LEVI, G. AND SIROVICH, F. 1975. Proving program properties, symbolic evaluation and logical procedural semantics. In *Proceedings of the 4th International Symposium on Mathematical Foundations of Computer Science, MFCS'75.* Lecture Notes in Computer Science, vol. 32. Springer-Verlag, Berlin, 294–301.

LLOYD, J. AND SHEPHERDSON, J. 1991. Partial evaluation in logic programming. *J. Logic Program. 11*, 217–242.

MARTENS, B. AND GALLAGHER, J. 1995. Ensuring global termination of partial deduction while allowing flexible polyvariance. In *Proceedings of the 12th International Conference on Logic Programming*, L. Sterling, Ed. The MIT Press, Cambridge, MA, 597–611.

MIDDELDORP, A. AND HAMOEN, E. 1994. Completeness results for basic narrowing. *Appl. Alg. Eng. Commun. Comput. 5*, 213–253.

MIDDELDORP, A., OKUI, S., AND IDA, T. 1996. Lazy narrowing: Strong completeness and eager variable elimination. *Theor. Comput. Sci. 167,* 1,2, 95–130.

MINIUSSI, A. AND SHERMAN, D. J. 1996. Squeezing intermediate construction in equational programs. In *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation*, O. Danvy, R. Glück, and P. Thiemann, Eds. Lecture Notes in Computer Science, vol. 1110. Springer-Verlag, Berlin, 284–302.

MORENO-NAVARRO, J. AND RODRÍGUEZ-ARTALEJO, M. 1992. Logic programming with functions and predicates: The language Babel. *J. Logic Program. 12,* 3, 191–224.

NUTT, W., RÉTY, P., AND SMOLKA, G. 1989. Basic narrowing revisited. *J. Symbol. Comput. 7*, 295–317.

PADAWITZ, P. 1988. *Computing in Horn Clause Theories.* EATCS Monographs on Theoretical Computer Science, vol. 16. Springer-Verlag, Berlin.

PALAMIDESSI, C. 1990. Algebraic properties of idempotent substitutions. In *Proceedings of 17th International Colloquium on Automata, Languages and Programming, CAAP'90*, M. Paterson, Ed. Lecture Notes in Computer Science, vol. 443. Springer-Verlag, Berlin, 386–399.

PETTOROSSI, A. AND PROIETTI, M. 1994. Transformation of logic programs: Foundations and techniques. *J. Logic Program. 19,20*, 261–320.

PETTOROSSI, A. AND PROIETTI, M. 1996. A comparative revisitation of some program transformation techniques. In *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation*, O. Danvy, R. Glück, and P. Thiemann, Eds. Lecture Notes in Computer Science, vol. 1110. Springer-Verlag, Berlin, 355–385.

RAMÍREZ, M. AND FALASCHI, M. 1993. Conditional narrowing with constructive negation. In *Proceedings of the 3rd International Workshop on Extensions of Logic Programing, ELP'92*, E. Lamma and P. Mello, Eds. Lecture Notes in Artificial Intelligence, vol. 660. Springer-Verlag, Berlin, 59–79.

REDDY, U. 1985. Narrowing as the operational semantics of functional languages. In *Proceedings of 2nd IEEE International Symposium on Logic Programming.* IEEE Press, New York, 138–151.

RÉTY, P. 1987. Improving basic narrowing techniques. In *Proceedings of the 1987 Conference on Rewriting Techniques and Applications.* Lecture Notes in Computer Science, vol. 256. Springer-Verlag, Berlin, 228–241.

ROMANENKO, A. 1991. Inversion and metacomputation. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'91.* ACM Press, New York, 12–22.

SANDS, D. 1995. Higher order expression procedures. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'95.* ACM Press, New York, 178–189.

SCHERLIS, W. 1981. Program improvement by internal specialization. In *Proceedings of the 8th Annual ACM Symposium on Principles of Programming Languages.* ACM Press, New York, 41–49.

SLAGLE, J. 1974. Automated theorem-proving for theories with simplifiers, commutativity and associativity. *J. ACM 21,* 4, 622–642.

SØRENSEN, M. 1994. Turchin's supercompiler revisited: An operational theory of positive information propagation. Tech. Rep. 94/7, Master's Thesis, DIKU, University of Copenhagen, Denmark.

SØRENSEN, M. AND GLÜCK, R. 1995. An algorithm of generalization in positive supercompilation. In *Proceedings of the 1995 International Logic Programming Symposium*, J. Lloyd, Ed. The MIT Press, Cambridge, MA, 465–479.

SØRENSEN, M., GLÜCK, R., AND JONES, N. 1994. Towards unifying partial evaluation, deforestation, supercompilation, and GPC. In *Proceedings of the 5th European Symposium on Programming*, D. Sannella, Ed. Lecture Notes in Computer Science, vol. 788. Springer-Verlag, Berlin, 485–500.

SØRENSEN, M., GLÜCK, R., AND JONES, N. 1996. A positive supercompiler. *J. Funct. Program. 6,* 6, 811–838.

SUZUKI, T., MIDDELDORP, A., AND IDA, T. 1995. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications.* Lecture Notes in Computer Science, vol. 914. Springer-Verlag, Berlin, 179–193.

TURCHIN, V. 1986. The concept of a supercompiler. *ACM Trans. Program. Lang. Syst. 8,* 3, 292–325.

TURCHIN, V. 1988. The algorithm of generalization in the supercompiler. In *Proceedings of the International Workshop on Partial Evaluation and Mixed Computation*, D. Bjørner, A. Ershov, and N. Jones, Eds. North-Holland, Amsterdam, 531–549.

TURCHIN, V. 1996. Metacomputation: Metasystem transitions plus supercompilation. In *Proceedings of the 1996 Dagstuhl Seminar on Partial Evaluation.* Lecture Notes in Computer Science, vol. 1110. Springer-Verlag, Berlin, 481–509.

Vidal, G. 1996. Semantics-based analysis and transformation of functional logic programs. Ph.D. thesis, DSIC, Universidad Politécnica de Valencia, Spain. In spanish.

Wadler, P. 1990. Deforestation: Transforming programs to eliminate trees. *Theor. Comput. Sci. 73*, 231–248.

Received February 1997; revised March 1998; accepted June 1998