On Narrowing Strategies for Partial Non-Strict Functions*

David de Frutos-Escrig
María-Inés Fernández-Camacho

Departamento de Informática y Automática
Facultad de Ciencias Matemáticas
Universidad Complutense
28040 Madrid, Spain

Abstract

We study completeness of narrowing strategies for a class of programs defining (possibly partial and non-strict) functions by means of equations, with a lazy semantics, so that infinite values are also admissible. We consider a syntactical restriction introduced by Echahed, under which he proved that any narrowing strategy is complete for specifications defining total functions with finite values. Unfortunately things are not so pretty for the larger class of programs that we consider. So we see that laziness of strategies is necessary in order to cope with non-strictness, fairness if we want to compute infinite values, and syntactically complete specifications (those including rules covering all possible patterns for each function) if we are also interested in the computation of partial values.

1 Introduction

Equational Unification [2,5] has been the base to unify relational and functional paradigms in programing. Narrowing [16,7,11] is probably the most natural extension of both rewriting and SLD-resolution in order to obtain an operational view of E-unification. In order to have completeness of narrowing without syntactical restrictions we need very strong semantical restrictions like canonicity [2]; but if we adopt the so called constructor discipline we can relax in several ways the restrictions of confluence and termination. BABEL [8,9,10] is a new language following these ideas, in which two apparently opposed properties like the existence of a clear mathematical semantics and an efficient implementation,

*This work has been partially supported by Acción Integrada Hispano-Alemana 35B, granted by Dirección General de Investigación Científica y Técnica (Spain) and Deutscher Akademischer Austauschdienst (F.R.G.)

have been carefully taken into account along its development in order to obtain a "practical" language in the sense suggested by modern programming metodologies. It has an strong similarity with the language K-LEAF developed by Levy et al. [6].

Unfortunately general narrowing is rather inefficient due to the necessity of doing backtracking along a computation not only on the rule applied, but also on the reduced redex. This problem is well known just for functional programming where it has been proved that some fixed strategies on the selection of the redex to reduce, like innermost, outermost, or some others, are complete under some adequate technical restrictions. In fact, in PROLOG it is exploited the fact that any (particularly fifo) strategy on the selection of the next goal to reduce is complete. So Fribourg has proved in [3] that innermost narrowing is complete under some syntactical conditions. On the other hand in [10] it is proved that lazy narrowing, that was introduced in [14], is complete for BABEL, but this lazy narrowing is not a strategy in the sense considered here, but just a relatively small restriction on the set of redices that should be considered at each step of the computation in order to guarantee completeness.

We are interested in a more strong result, concerning any strategy, or if this is not possible a class of strategies as broad as possible. Padawitz in [12] gave the first step in this direction, obtaining a positive result, under some "uniformity" conditions on the considered programs, that we could classify as semantical, and then difficult to decide. Echahed in [1] has presented a general result valid for any strategy, with just a simple syntactical restriction, the so called principle of free narrowing strategies, on the programs that can be considered. But he also gives a transformer by means of which any program can be set, preserving its semantics, in the adequate form in order to apply the completeness result. The real limitation of this result is that it can only be applied to canonical programs, what in fact is a semantical restriction in the general case. In [13] you can find some other properties of the class of programs fulfilling the principle of free narrowing strategies.

We have generalized Echahed's result to non-canonical programs, covering in fact partial functions, partial values, non-strict functions and infinite objects like in MIRANDA [17], what is done in a functional-logic environment, in BABEL. It is very important to note that narrowing is used in this language in a way that we could say to be "much more functional than logic". In general we will consider the evaluation of arbitrary terms, and not just the solution of equations. In that last case it is well known that the consideration of partial functions, infinite structures and terms with partial values is not justified, as E-unification is non-computable under non-termination. But in a more functional framework like ours, it has perfect sense to consider all these generalizations. Of course this must be done in a non-naive way; for instance, we cannot compute an infinite value, but we can compute all (this means as many as we want) its finite approximations. The same is true for partial values: in general we cannot say "this (partial) value is the value of something", as non-termination makes this undecidable, but we can compute all their finite approximations, including, of course, the value itself if it is finite, of such a partial value.

We have seen that in fact we do not need the large and complex syntax of BABEL, including predicates and functions, conditional equations, the use of new variables in conditions, an equality test, and some other facilities, in order to show the difficulties induced by the semantical generalizations listed above. So we have introduced simple-

BABEL programs that are in fact the same specifications considered by Echahed, but eliminating the canonicity restriction and giving a semantics that covers all those new interesting semantical characteristics. On the other hand, although we have extended Echahed's results, this has not been at all by means of a generalization of his proofs, because they are strongly based on the canonicity of the rewriting system associated to the considered program, property that has been avoided in our work (mainly in that concerning termination.)

We assert, without proof, that all the results in the paper could be extended without any added technical difficulty to whole BABEL, but with awfully long proofs due to the necessity of considering all the syntactical facilities in the language.

The paper is structured as follows : in the next section we give a series of technical definitions, and present the language simple-BABEL including formal definitions of its semantics, both operational and denotational. The third section is the main of the paper, and presents narrowing strategies discussing their completeness. So, we see that we have to impose laziness to cope with non-strictness, fairness if we want to compute (arbitrarily accurate approximations of) infinite values, and the restriction either to syntactically total programs or to the so called goal oriented strategies, if we want to compute (improve) partial values. Finally, in the fourth section, we study a couple of transformers by means of which, from an arbitray program we can obtain another more or less equivalent in the syntactical class of programs for which we have our completeness results of narrowing strategies.

2 Definitions and basic results

We suppose that the reader is familiar with the usual terminology and notation in the logic and functional programming environment. In any case, one can use [4,1,11,9] in order to remind them. We have prefered to omit the basic definitions included there, in order to fix the reader's attention in the less standard ones that we present below.

We will consider along this paper **simple-BABEL** programs or "general specifications" that are defined as follows:

Definition 1 Let C and F be a pair of sets of operation symbols, over which an arity $\alpha : C \cup F \rightarrow \mathbb{N}$ is defined. Let X be a set of variable symbols.

We define:

Data terms $t \in DT$

$$\begin{array}{lll} t & := & X \quad \% \text{ variables} \\ & & c \quad \% \ \alpha(c) = 0 \\ & & c(t_1, \ldots, t_n) \quad \% \ \alpha(c) = n > 0 \end{array}$$

Expressions $E \in EXP$

$$\begin{array}{lll} E & := & t \quad \% \text{ data terms} \\ & & c(E_1, \ldots, E_n) \quad \% \ \alpha(c) = n > 0 \\ & & f(E_1, \ldots, E_n) \quad \% \ \alpha(f) = n > 0 \\ & & f \quad \% \text{ defined constants} \end{array}$$

□

Definition 2

1. Simple-BABEL *rules* are pairs $f(t_1, \ldots, t_n) := E$ verifying the following restrictions
   (a) *Left linearity*: no variable has multiple occurrences along the left hand side.
   (b) *Local determinism*: $vars(\text{lhs}) \supseteq vars(E)$.
2. Simple-BABEL *programs* are (recursively enumerable) sets of simple-BABEL rules verifying the *weak nonambiguity restriction*: If $L_1 := E_1$, $L_2 := E_2$ are in a simple-BABEL program Π and L_1 and L_2 are unifiable via m.g.u. σ, then $E_1(\sigma) = E_2(\sigma)$. (Here $=$ means semantic equality.)

□

Remarks:

1. We say that the rules are flat, as the arguments in lhs's must be data terms, that can not contain defined functions in F.
2. We have imposed just this weak nonambiguity restriction based on semantical instead of syntactical equality, because only the first is necessary in order to have deterministic definitions. Obviously, this semantical equality is undecidable in general, and this is why the syntactical, stronger but trivially checkable is usually imposed; but at the theoretical level it does not matter how the necessary semantical equality is guaranteed.

Declarative or denotational semantics is based on the Herbrand domain for C, H_C, that is obtained adding $\perp$, whose arity is 0, to this set, and considering the set of finite and infinite terms over such extended domain. An ordering based on refining is introduced in H_C reflecting the interpretation of $\perp$ as "undefined": so $t \sqsubseteq t'$ iff t' can be obtained from t substituting some expressions for occurrences of $\perp$. So we obtain a domain in the sense of Scott [15], whose finite elements are just the finite terms.

Definition 3 A *Herbrand interpretation* for $\sum = (C, F)$ is any algebra $I = (H_C, (f_I)_{f \in F})$ where f_I is a continuous function $f_I : H_C^n \longrightarrow H_C$, with $n = \alpha(f)$. □

Remark: We do not impose strictness to the interpretations of defined functions.

Any expression can be evaluated under an interpretation, given an environment ρ defining the value of its (free) variables. So we obtain the evaluation function $[\![\cdot]\!]_I : EXP \longrightarrow H_C$, whose rather straightforward definition can be found for instance (for the full BABEL language) in [8,9].

Definition 4

1. We say that I is a model of Π ($I \models \Pi$) iff it is a model of any rule in Π.
2. We say that I is a model of $L := E$ ($I \models L := E$) iff $[\![L]\!]_I(\rho) = [\![E]\!]_I(\rho)$ for any environment ρ over I.

□

Herbrand interpretations can be ordered by means of the usual ordering between products of function spaces: $I = (H_C, (f_I)) \sqsubseteq I' = (H_C, (f'_I))$ iff

$$\forall f \in F \ \forall h_1, \ldots, h_n \in H_C \quad f(h_1, \ldots, h_n) \sqsubseteq f'(h_1, \ldots, h_n).$$

This leads us to a domain of Herbrand interpretations, *HINT*, which allows us to prove the existence of minimal models for simple-BABEL programs.

Definition 5 The *interpretation transformer* associated to Π is the mapping $\tau_\Pi : HINT \longrightarrow HINT$ defined as follows ($\tau_\Pi(I) = J$)

$$f_J(t_1, \ldots, t_n) = \begin{cases} [\![E]\!]_I & \text{iff } f(t_1, \ldots, t_n) := E \text{ is a ground instance} \\ & \text{of some rule in } \Pi \\ \bot & \text{otherwise.} \end{cases}$$

□

Theorem 1 τ_Π is a continuous function. □

Corollary 1 Π has a minimal model (the minimal fixed point of τ_Π.) □

Definition 6 Declarative semantics of simple-BABEL programs is given by the valuation of ground expressions under any model, and specially by their valuation under its minimal model. □

Due to the existence of infinite objects in the semantic domain, and to the fact that there are expressions with such a value, it is obvious that we can not expect that a (finite) operational semantics will compute those values. Nevertheless we should be able to compute any finite approximation to them. This kind of facts leads us to the definition of a function extracting all definitive information in an expression.

Definition 7 The shell $\mid M \mid$ of any expression is defined, by structural induction, as follows:

$\mid c \mid = c \quad \forall c \in C \quad \alpha(c) = 0$

$\mid c(M_1, \ldots, M_n) \mid = c(\mid M_1 \mid, \ldots, \mid M_n \mid) \quad \forall c \in C \quad \alpha(c) > 0$

$\mid N \mid = \bot$ for any other expression.

□

Operational semantics of simple-BABEL is based on lazy narrowing. This is defined by restricting the narrowing steps to those corresponding to lazy redices.

First we have to introduce an extended unification algorithm that includes the definition of pending argument, in which lazy narrowing is based.

Definition 8 (Extended Linear Unification Algorithm)

INPUT : $c(M_1, \ldots, M_n)$, $c(t_1, \ldots, t_n)$

(This is the only case in which we are interested due to flatness and left linearity of rules.)

STEP 1 (INITIALIZATION): Set $U_0 = \{M_1 \downarrow_1 t_1, \ldots, M_n \downarrow_n t_n\}$, $\sigma_0 = \epsilon$

($\downarrow_i$'s are just syntactical marks in order to note that the corresponding unification problem corresponds to the i-th argument of the input.)

STEP 2 (REDUCTION): Don't care nondeterministically rewrite the configuration (U_i, σ_i) using the rules that follow until a nonreducible configuration (U_n, σ_n) is reached.

UR1 : $(\{c(N_1, \ldots, N_m) \downarrow_i c(N'_1, \ldots, N'_m)\} \cup U, \sigma) \longrightarrow$
$(\{N_1 \downarrow_i N'_1, \ldots, N_m \downarrow_i N'_m\} \cup U, \sigma) \qquad \alpha(c) \geq 0$

UR2 : $(\{X \downarrow_i t\} \cup U, \sigma) \longrightarrow (U[t/X], \sigma[t/X])$

UR3 : $(\{M \downarrow_i X\} \cup U, \sigma) \longrightarrow (U[M/X], \sigma[M/X])$

UR4 : $(\{c(N_1, \ldots, N_m) \downarrow_i c'(N'_1, \ldots, N'_k)\} \cup U, \sigma) \longrightarrow (\{FAILURE\}, \sigma)$
$c \neq c'$ (In fact, left linearity and the fact that $c(M_1, \ldots, M_n)$ and $c(t_1, \ldots, t_n)$ do not share variables will lead us when we apply this algorithm to $U[M/X] = U$ in case 3.)

STEP 3 (OUTPUT): If $U_n = \emptyset$ report *SUCCESS* and output σ_n as m.g.u.

If $U_n = \{FAILURE\}$ report *FAILURE*.

Otherwise let $I = \{i \mid \exists f(N_1, \ldots, N_k) \downarrow_i c(N'_1, \ldots, N'_l) \in U_n\}$ and report *PENDING* with such a set as set of pending arguments.

□

Definition 9 A rule $L := R$ standing apart from N *applies to, fails for* or *is pending for* N iff the unification of N, L yields a *SUCCESS, FAILURE* or *PENDING* outcome, respectively. If the unification succeeds with m.g.u. σ, we say that $L := R$ is applicable to N via σ. If it yields a pending outcome with I as set of demanding arguments indices, we say that $L := R$ is pending at i for any i in I. □

Redex occurrences are the occurrences of an expression where some rule is applicable. But in order to bind the set of possible computations, and also to avoid incomplete strategies that would be obtained, as we will show below, if too inner redices would be selected, we introduce lazy redices that are defined as follows:

Definition 10 For any expression M the set $LR(M)$ of *lazy redices* for M is defined, by structural induction, by the following clauses

- $LR(f(M_1, \ldots, M_n)) = \{\epsilon \mid \text{some rule in } \Pi \text{ applies to } f(M_1, \ldots, M_n)\} \cup \bigcup_{i \in I} i \cdot LR(M_i)$ where I is the set of argument indexes at which some rule in Π is pending for $f(M_1, \ldots, M_n)$
- $LR(c(M_1, \ldots, M_n)) = \bigcup_{i=1}^{n} i \cdot LR(M_i)$.

□

Definition 11

1. If $u \in LR(M)$ and $L := R$ is a rule that applies to M/u via σ we say that M narrows in one step to $M[u \leftarrow R]\sigma$, and write $M \overset{N}{\longrightarrow}_{\sigma_M} M[u \leftarrow R]\sigma$, where $\sigma_M = \sigma \mid_{vars(M)}$.

2. We say that M reduces to N via lazy narrowing and σ_M, iff there is some narrowing sequence $M = M_0 \overset{N}{\longrightarrow}_{\sigma_M^1} M_1 \overset{N}{\longrightarrow} \cdots M_i \overset{N}{\longrightarrow}_{\sigma_M^{i+1}} M_{i+1} \overset{N}{\longrightarrow} \cdots M_n = N$ with $\sigma_M = \sigma_M^1 \circ \cdots \circ \sigma_M^n$.
   This fact will be indicated by the notation $M \overset{N}{\overset{*}{\longrightarrow}}_{\sigma_M} N$.

□

Operational and denotational semantics of simple-BABEL are related by means of the following soundness and completeness theorems.

Theorem 2 Any reduction $M \overset{N}{\overset{*}{\longrightarrow}}_{\sigma_M} N$ is sound in the sense that $[\![M\sigma_M]\!]_I(\rho) = [\![N\sigma_M]\!]_I(\rho)$ for any model I of Π, and any environment ρ over I.

Proof: See [9] . □

Corollary 2 Any reduction $M \overset{N}{\overset{*}{\longrightarrow}}_{\sigma_M} N$ is sound in the sense that $[\![M\sigma_M]\!]_I(\rho) \sqsupseteq \mid N\rho \mid$ for any model I of Π, and any environment ρ over I. □

Theorem 3 Let M be any expression, s a partial term, and θ a data-substitution with $\mathrm{dom}(\theta) \subseteq \mathrm{var}(M)$ such that $[\![M\theta]\!]_I(\rho) \sqsupseteq s$ holds for any I model of Π and any environment over I; then, there exists a (lazy) narrowing sequence $M \overset{N}{\overset{*}{\longrightarrow}}_{\sigma_M} N$ and a substitution λ such that $\theta = \sigma_M \circ \lambda$ and $\mid N\lambda \mid \sqsupseteq s$.

Proof: See [18] or [10] . □

Definition 12 Two terms t and t' are said to be *sub-unifiable* iff there exists an occurrence u in $O(t) \cap O(t')$ such that

1. t/u and t'/u are unifiable (via σ_u),

2. for all occurrences w with $w < u$, t/w and t'/w have the same root symbol.

□

Definition 13 Two terms t and t' are said to be *strictly sub-unifiable* iff there exists an occurrence u where they are sub-unifiable, and the corresponding m.g.u. σ_u is neither a variable renaming nor the empty substitution. □

Finally we introduce the elements to define the syntactical restriction under which Echahed has proved in [1] that any narrowing strategy is complete, for the restricted class of programs that he considered.

Definition 14 We say that Π verifies the *principle of free narrowing strategies* (pfns) iff for any pair of rules in Π their lhs's are not strictly subunifiable. □

Remark: If we forbid the appearance in a program of two rules having the same or equivalent (the names of the variables does not matter) lhs, this condition is stronger than our weak nonambiguity restriction. But we have included this one and not the former in the definition of simple-BABEL programs, because we obtain in this way a broader class of programs, and although we will need the stronger restriction in order to obtain the results about completeness of narrowing strategies, in Section 4 we will prove that this results can be extended in some way to our broader class of programs, in the sense that any of these can be transformed into another equivalent, fulfilling the pfns.

3 Completeness of narrowing strategies

As we said in the introduction, R. Echahed [1] has proved that, under the hypothesis of free narrowing strategies, any narrowing strategy is complete for a kind of specifications rather simpler than ours. One can also find there some simple counterexamples showing that the restriction to this class of programs is necessary in order to guarantee completeness of arbitrary narrowing strategies. The main differences between both classes of specifications are termination, that we do not impose at all, and the use of just totally defined functions: we allow partial defined functions and we admit also infinite values; in addition, strictness is not imposed. Let us first refresh the notion of narrowing strategy.

Definition 15 A *narrowing strategy* is a partial function from expressions to their occurrences $NST : EXP \longrightarrow \mathbb{N}^*$ such that it is defined iff the argument is narrowable, choosing a narrowing point in it. □

Definition 16 A narrowing sequence is *admissible under a narrowing strategy NST* iff any of its narrowing steps $M_i \overset{N}{\longrightarrow}_{\sigma_M^{i+1}} M_{i+1}$ is made on the redex $NST(M_i)$. □

For the class of specifications considered by Echahed any strategy is complete. This is no more true for simple-BABEL programs verifying pfns. It is due to non-strictness and partial defined functions in one hand, and to the admission of infinite and partial defined functions on the other hand. Let us see three different examples illustrating the different kinds of patological situations that can be presented.

The first example shows the kind of problems that we can find, even if we are only interested on the computation of finite and total values of expressions.

Example 1 Let Π_1 be the program defined in Table 1, and take as E_1 the expression $f(g(y))$. We have two possible narrowing points ϵ and 1. If our strategy chooses the

$C = \{0, s\}$	% Natural numbers
$F = \{f, g\}$	$\alpha(f) = 1 \;\; \alpha(g) = 1$
$f(x) := 0$	% Non-strict
$g(0) := s(0)$	% Partially defined

Table 1: Definition of Π_1

second, then we should unify y and 0 : we would get the value 0 for the evaluated expression, corresponding to the case $y = 0$, but all the remaining zero values corresponding to any $y > 0$ would be lost. As we said, non-strictness of f in association with a partially (syntactically) defined function g has caused the problem. The solution in this case will be to allow just strategies choosing lazy redices. Check that we have $LR(E_1) = \epsilon$. □

The second example presents the new difficulties that we find, if we also want to compute (arbitrary accurate approximations of) infinite but total values.

Example 2 Let Π_2 be the program defined in Table 2, and take as E_2 the expression $p(i, i)$. We have $[\![E_2]\!]_I = p(\infty, \infty)$ under any Herbrand interpretation that was a model

$C = \{0, s\} \cup \{p\}$	% Natural numbers and the pairing constructor
$F = \{i\}$	$\alpha(i) = 0$ % "Infinite" defined constant
$i := s(i)$	% Infinite value

Table 2: Definition of Π_2

of Π_2. Finite approximations of this value are the pseudo-expressions $p(\overline{m}, \overline{n})$ where m and n are natural numbers and $\overline{m}$ denotes the m-th successor of $\perp$ formally defined by $\overline{0} = \perp$, $\overline{s(m)} = s(\overline{m})$. Then if we want completeness in the sense of Theorem 3, we would need narrowing sequences $E_2 \overset{*}{\underset{N}{\longrightarrow}} E_2'$ with $\mid E_2' \mid\sqsupseteq p(\overline{m}, \overline{n})$, for any $m, n \in \mathbb{N}$. But, if we work under a strategy NST such that $NST(p(s_n(i), i)) = 1 \cdot 1_n$, where $s_n(i)$ is defined by $s_0(i) = i$, $s_{s(n)}(i) = s(s_n(i))$, and 1_n by $1_0 = E$, $1_{s(n)} = 1 \cdot 1_n$, then any narrowing sequence under such a strategy would lead us to some $E_2' = p(s_n(i), i)$, with $n \in \mathbb{N}$ and obviously for none of them we have, for instance, $\mid E_2' \mid\sqsupseteq p(\perp, s(\perp))$. Now the problem has been caused by infinite values, and the solution will be to impose in some way that the strategy only will lead us to fair narrowing sequences. □

Finally the third example shows a new kind of problem, that could appear if we desired to compute approximations of any kind, including partial ones, of values, covering so the full completeness result presented in our Theorem 2.

Example 3 Let Π_3 be the program defined in Table 3, and take as E_3 the expression $p(f(x), g(x))$; for $x = 1$ its value is $p(\perp, 1)$, but it could not be computed by narrowing

$C = \{0, 1\} \cup \{p\}$	% Two constants and the pairing constructor
$F = \{f, g\}$	$\alpha(f) = 1 \quad \alpha(g) = 1$
$f(0) := 0$	% Partially defined
$g(x) = 1$	

Table 3: Definition of Π_3

under a strategy NST such that $NST(E_3) = 1$, because this would lead us to the substitution $x = 0$. In this case the problem comes from the partial definedness of f; but as p

is a constructor we can not solve it just by restricting narrowing to lazy redices: in fact 1 is such a redex.

Two solutions are possible: the first consists on a syntactical completion of syntactically partially defined functions, f, by means of the rule $f(x) := f(x)$, or more exactly of a development of it in order to maintain pfns. In our example we would add rules $f(1) := f(1)$, $f(p(x,y)) := f(p(x,y))$. The second possibility is more pragmatic but less elegant; it consists in a generalization of the strategy notion, under which a strategy could select a set of redices (in practice we would take sets as small as possible) that should be complete what roughly means that the set of substitutions covered by m.g.u.'s corresponding to the application of applicable rules under those redices should be maximal. □

Next we will formally present the three completeness theorems covering the three cases presented above, and illustrated by our examples.

We begin with the case in which we only are interested in the computation of finite and total values.

Definition 17 We say that a strategy is *lazy* if it always selects lazy redices. □

Theorem 4 Let Π be a simple-BABEL program verifying pfns, E an expression and θ a data-substitution with $dom(\theta) = var(E)$ such that $[\![E\theta]\!]_I = t$ with t a (total, finite and ground) term, under any Herbrand interpretation (specially under the minimal one); then, for any lazy strategy $LNST$ there exists a narrowing sequence under it $E \overset{*}{\underset{N}{\longrightarrow}}_{\sigma_M} t'$, and a substitution λ such that $\theta = \sigma_M \circ \lambda$ and $t'\lambda = t$.

The proof is based on the following lemma that states the main property of lazy redices, and more generally of applicable and pending rules, for programs verifying pfns.

Lemma 1 Let Π be a simple-BABEL program verifying pfns and $E = f(E_1, \ldots, En)$ an expression. Then if $r = L := R$ is pending for E on I for some $r \in \Pi$, any other $r' \in \Pi$ is either pending for E on I, or fails for it.

Proof: If $r' = L' := R'$ is applicable to E, or is pending for I' with $I \not\subset I'$ then for $i \in I$, in the first case, and $i \in I - I'$ in the second, we will have, as r is pending on i, some occurrence u such that $root(E/iu) \in F$ and $\forall v < u\ root(E/iv) \in C$. But as r' is applicable or not pending for E on i, we must have some $v \leq u$ such that L'/iv is a variable, and then L and L' would be strictly subunifiable, against the fact that Π verifies pfns. □

Corollary 3 Let Π be a simple-BABEL program verifying pfns and $u \in LR(E)$; if $E = M_0 \overset{*}{\underset{N}{\longrightarrow}}_{\sigma_E} M_n$ is a lazy narrowing sequence such that in none of its steps we have applied a rule on the redex u, then we have

1. $\forall v \leq u\ \ \forall m \leq n\ \ root(M_m/v) = root(E/v)$
2. We have neither applied any rule on any redex $v \geq u$.

Proof:

1. As $u \in LR(E)$ we have that $\forall v < u$ such that $root(E/v) \in F$ if i is such that $vi \leq u$ then we have some rule in Π pending for E/v on I. Then Lemma 1 tells us that $\forall v < u$ $root(E/v) \in F$ we do not have applicable rules. So the first rule that we will apply along our narrowing sequence will leave $root(E/v)$ unchanged for any $v \leq u$. After the application of the corresponding narrowing step, for any $v < u$ with $root(E/v)$ none formerly pending rule for it can become applicable as M_1 and E are equal along the path given by u. Then either u remains in $LR(M_1)$ or there is some $v \leq u$ such that any rule fails for M_1/v. In the first case we will follow the same reasoning to the remaining steps of the sequence; in the second the laziness of the computation assures us that we will not apply any rule on any $w \geq u$, and Lemma 1 tells us that we will not do it on any $w < u$.

2. As $u \in LR(E)$ we have some rule that applies to E/u, and then we can not have pending rules, so that $LR(E/u) = \{\epsilon\}$.

□

Let us go now to the proof of Theorem 4:

Proof: As we know that lazy narrowing is complete (Theorem 3) we must have a narrowing sequence $E \mathrel{\mathrm{N}}\!\!\longrightarrow_{\sigma_E} t'$, and a substitution λ such that $\theta = \sigma_E \circ \lambda$ and $t'\lambda = t$. But we have to prove that we can get such a sequence under the strategy *LNST*; this will be obtained from the former by a sequence of careful modifications. We will construct it by induction on the length l of the given narrowing sequence.

If $l = 0$ we have an empty sequence that of course is allowable under *LNST*.

Otherwise, we consider $u = LNST(E)$ and we have two possibilities: we make narrowing on u on the first step of the given sequence, or not.

In the first case the same step is allowable by *LNST* and then the result follows by induction hypothesis.

Otherwise, by Corollary 3 we know that either we have applied a rule on u along the sequence or $root(t'/u) = root(E/u)$ but this second case is impossible because $u \in LNST(E)$, and then $root(E/u) \in F$, so that for any substitution λ we would have that $\mid t'\lambda \mid$ is not total, and then $\mid t'\lambda \mid \neq t$. So let $E = M_0 \mathrel{\mathrm{N}}\!\!\overset{*}{\longrightarrow}_{\sigma_a}$ $M_i \mathrel{\mathrm{N}}\!\!\overset{<u,r>}{\longrightarrow}_{\sigma_i} M_{i+1} \mathrel{\mathrm{N}}\!\!\overset{*}{\longrightarrow}_{\sigma_b} M_l = t'$, where i is the first step in which we have applied a rule on u, and $\sigma_E = \sigma_a \circ \sigma_i \circ \sigma_b$. We have the following facts :

1. $M_i/u = (E/u)\sigma_a$, as an immediate consequence of Corollary 3.

2. Let $r = L := R$, then L, E/u are unifiable via m.g.u. σ'_i and $\sigma'_i < \sigma_a \circ \sigma_i$. This follows from 1 and definition of m.g.u.

3. $E = M_0 \mathrel{\mathrm{N}}\!\!\overset{<u,r>}{\longrightarrow}_{\sigma'_i} M'_1$ is an admissible narrowing step under *LNST*.

4. The steps before the i-th on the given computation work on occurrences that are not modified, but perhaps by some substitution, by the application of r to E. Note particularly, because it is very important in order to maintain the length of the sequence, the use of part ii. in Corollary 3 in order to obtain this fact, and consequently the use of the fact that the given sequence is lazy.

5. Those steps can be given after M'_1 obtaining a sequence (of the same length!) $M'_1 \overset{*}{\underset{N}{\longrightarrow}}_{\sigma'_a} M'_{i+1}$ where $\sigma'_i \circ \sigma'_a \equiv \sigma_a \circ \sigma_i$ so that M'_{i+1} is a variant of M_{i+1}.

6. We can extend the sequence by means of the corresponding variant of the sequence $M_{i+1} \overset{*}{\underset{N}{\longrightarrow}}_{\sigma_b} M_l = t'$, to reach M'_l by σ'_b variant of σ, so that we have some λ' such that $\mid M_l\lambda' \mid = t$.

7. Finally we can apply the induction hypothesis to the sequence $M'_1 \overset{*}{\underset{N}{\longrightarrow}}_{\sigma'_a\sigma'_b} M''_l$, whose length is $l-1$ and is lazy as the original sequence was, and as we saw in the proof of Corollary 3, the data-substitutions induced by narrowing steps just can slightly change the set of lazy narrowing points: If we narrow E at u getting E' through the substitution σ, then $\forall v \not\geq u$ we have $v \in LR(E') \Longrightarrow u \in LR(E)$, and as $LR(E')$ is prefix-free we have that if some $v \not\geq u$ belongs to $LR(E) - LR(E')$ then we would have $[\![E'/v]\!](\rho) = \bot$ under any substitution, and as in order to apply any rule at any $w < v$ in E we need a defined value of E/v, we have the same for $E'\sigma$, getting that $[\![E\sigma]\!](\rho)$ would be a partial value for any substitution ρ. So if this would be the case for our sequence we would have some $\sigma \leq \sigma_E$ such that for any ρ $[\![E\sigma]\!](\rho)$ is partial, contradicting the fact that $[\![E\theta]\!]_I$ is total for some $\theta \geq \sigma_E$.

□

In order to extend our results to the computations of (any finite approximation) of infinite values, we introduce a dynamic notion of strategy generalizing the static notion introduced in Definition 15.

Definition 18 A *dynamic strategy DS* is a function from narrowing sequences to occurrences, such that if $q = M \overset{*}{\underset{N}{\longrightarrow}}_{\sigma} N$, $DS(q)$ is defined iff N is narrowable, and gives in that case a lazy redex in N. □

Remarks:

1. We include the restriction to lazy redices in the definition of dynamic strategies, because we want to generalize Theorem 3 to infinite values, and the restriction of lazy strategies has been necessary even to cover the finite case considered there.

2. It is easy to check that any of Echahed's results on completeness of strategies and also our Theorem 3, are also valid for dynamic strategies, because the static character of strategies is never used along their proofs.

Definition 19 A dynamic strategy is *fair* iff if $q = M \overset{*}{\underset{N}{\longrightarrow}}_{\sigma} N$, $u \in LR(N)$, we cannot have a sequence of computations $q_n = M \overset{*}{\underset{N}{\longrightarrow}}_{\sigma} N_n$ with $n \in \mathbb{N}$, such that $q_0 = q$, and for all $i \in \mathbb{N}$ q_{i+1} strictly extends q_i, $DS(q_i) \not\leq u$ and $u \in LR(N_i)$. □

Intuitively a dynamic strategy is fair iff *it cannot infinitely postpone the choice of a lazy redex* that remains like that forever, from some point, along an admissible computation under the considered strategy.

Note: The reason to introduce dynamic strategies is mainly pragmatic. We could develop this part of our theory just for static strategies, but in practice we will have different kinds

of problems : on one hand it would be difficult (probably undecidable) to decide if an arbitrary (computable) strategy would be fair; on the other hand many simple strategies will be unfair : take for instance outermost or innermost ones; we would have in fact some lazy and fair strategies, like the corresponding version of breadth narrowing, but they should be, in our opinion, too restrictive. We have prefered to introduce dynamic strategies that are thought to be really devised in a dynamic way, getting the fairness constraint in different ways according to the particular taste of the user.

Theorem 5 Let Π be a simple-BABEL program verifying pfns, E an expression and θ a data substitution with $dom(\theta) = var(E)$ such that $[\![E\theta]\!]_I = t$ with t a total, ground but (possibly) infinite term, under any Herbrand interpretation. Then for any lazy fair dynamic strategy *FDS* there exists an infinite narrowing sequence under it $E = M_0 \overset{}{\underset{N}{\longrightarrow}}_{\sigma_0} M_1 \overset{*}{\underset{N}{\longrightarrow}} \cdots M_i \overset{}{\underset{N}{\longrightarrow}}_{\sigma_i} M_{i+1} \overset{\infty}{\longrightarrow} \cdots$ such that $\forall i\ \exists \lambda_i\ \sigma_0 \circ \cdots \circ \sigma_{i-1} \circ \lambda_i = \theta$ and $\langle\!|\ M_i\lambda_i\ |\!\rangle_{i\in\mathbb{N}}$ is a monotonic sequence of finite elements whose limit is $[\![E\theta]\!]_I$.

Proof: As H_C is a domain and the tree of narrowing sequences from E is finitary, we know from Theorem 3 that we have a lazy narrowing infinite sequence q as that looked for. As in the proof of our previous theorem we will convert this sequence in other "more or less" equivalent to it. For we consider first $U = FDS(\epsilon)$, and, as in that proof, we will check that there must be some i such that in the i-th step of q we have narrowed on u. Then we will rearrange the first i steps of the sequence getting an equivalent one E_1 whose first step is allowable under *FDS*. We will repeat this process for the infinitely many steps of the sequences, obtaining a succession of equivalent sequences $\langle q_n \rangle_{n\in\mathbb{N}}$ whose first n steps, for each q_n, are permissible under *FDS*.

Then it is clear that if we consider the sequence limit of this succession q_∞ defined in a natural way, we have an infinite sequence allowed by *FDS*. What it is not trivial is that q_∞ will compute the same value as q; in fact it can be false for arbitrary strategies and this is the reason why we said "more or less" equivalent. Take for instance our Example 2. We can take as q the sequence
$E_2 = M_0 \overset{}{\underset{N}{\longrightarrow}}_{\epsilon} M_1 \overset{*}{\underset{N}{\longrightarrow}}_{\epsilon} \cdots M_j \overset{}{\underset{N}{\longrightarrow}}_{\epsilon} M_{j+1} \overset{*}{\underset{N}{\longrightarrow}}_{\epsilon} \cdots$ where $M_j = p(s_{\frac{i+1}{2}}(i), s_{\frac{i-1}{2}}(i))$ if j is odd and $M_j = p(s_{j/2}(i), s_{j/2}(i))$ if j is even, and each narrowing step has been given over $1_{\frac{i+1}{2}}$ and $2 \cdot 1_{\frac{i-2}{2}}$ respectively. Then, it is easy to check that if we would take as strategy the *NST* described there, σ_∞ would be the (only) sequence making narrowing in j-th step on 1_j and obtaining after j steps $M_j^{NST} = p(s_j(i), i)$, so that $\bigsqcup_{j\in\mathbb{N}} |\ M_j^{NST} \models p(\infty, \bot) \not\sqsubseteq p(\infty, \infty)$.

So what happens in the general case is that q_∞ is less or equivalent to q, but not necessarily equivalent, because some step of q could have been infinitely delayed and so avoided by q_∞. But that is exactly what our notion of fair strategy avoids, at least for programs satisfying pfns. Indeed, if we consider the redex u_1 corresponding to the first step of q, we have that $u_1 \in LR(E)$, and if Π verifies pfns we have that it will remain in the set of lazy redices until a narrowing step will be done on it; otherwise, as we saw in our previous proof, the value of the corresponding subexpression E/u_1 would be $\bot$, and that of E partial against the hypothesis. But then our fairness restriction assures us that sooner or later this redex will be selected by the strategy, so let us say that in the n-th step of q_n we would apply on u the same rule that was applied in the first step of q. But along q all the constructors in t will be computed, so that for any depth h, all the

constructors in t up to this depth will be calculated after some (finite) number of steps s_h. Then, after iterating the reasoning presented upwards for the first step of q, we would get some n_h such that if $E = M_0 \mathrel{\mathrm{N}}\!\!\xrightarrow{*}_{\sigma_h} M_{n_h}$ is the subsequence of q_{n_h} constituted by its n_h first steps, we have that there is some λ_h such that $\mid M_{n_h}\lambda_h \mid$ is equal to t up to depth h. □

Let us finish with our third case. Again we obtain a broader generalization, but for it we have to restrict more the class of admissible strategies. As we said, there are several ways to do it, that we present in the following.

Definition 20 Let Π be a simple-BABEL program over $\sum = (C, F)$. We say that

1. f is syntactically totally defined by Π iff for each tuple of data terms $(t_1, \ldots, t_n)$ with $\alpha(f) = n$, we have some $r = L := R \in \Pi$ applicable to $f(t_1, \ldots, t_n)$.
2. Π is syntactically complete iff for all f in F, f is syntactically totally defined by Π.

□

Theorem 6 Let Π be a simple-BABEL program syntactically complete and verifying pfns, E an expression, θ a data substitution with $dom(\theta) = var(E)$, and s a finite, ground and (possibly) partial term, such that $[\![E\theta]\!]_I \sqsupseteq s$ under any Herbrand interpretation; then for any lazy fair dynamic strategy FDS there exists a narrowing sequence under it $q : E \mathrel{\mathrm{N}}\!\!\xrightarrow{*}_{\sigma_E} N$ and a substitution λ such that $\theta = \sigma_E \cdot \lambda$ and $\mid N\lambda \mid \sqsupseteq s$.

Let us begin noting why fairness is needed even if we would know that $[\![E\theta]\!]_I$ is finite. For let us modify our program in Example 3, taking $f(x) = f(x)$ as the definition of this function. Then Π verifies all the hypothesis of the theorem, and $[\![p(f(1), g(1))]\!]_I = p(\bot, 1)$. But if we consider the static strategy selecting (always) 1 as redex, we would obtain a unique narrowing sequence that does not make any progress at all so that any reachable expression would be the initial one, whose shell is $p(\bot, \bot) \not\sqsupseteq p(\bot, 1)$.

Now we introduce a formal concept that will be used to express the fact we are only interested in obtaining a value greater than s but not more.

Definition 21 Let E be an expression and s a partial value such that we are interested on the evaluation of the partial substitutions θ such that $[\![E\theta]\!]_I(\rho) \sqsupseteq s$ under any Herbrand interpretation I, and for any environment ρ. Let $u \in LR(E)$, we say that u is unnecessary for computing s iff if w is the greatest prefix of u contained in the defined (labelled by some constructor) occurrences of s, then for all $w' \leq w$ we have $root(E/w') \in C$ (in fact we will have $root(E/w') = root(s/w')$.) □

We can proceed now with the proof of the theorem.

Proof: We consider, as in the proof of Theorem 4, a lazy narrowing sequence $q : E = M_0 \mathrel{\mathrm{N}}\!\!\xrightarrow{*}_{\sigma} s'$ and a substitution λ such that $\theta = \sigma \circ \lambda$ and $\mid s'\lambda \mid \sqsupseteq s$. It must exist by Theorem 3. It is easy to check that we can suppose that it does not contain unnecessary steps (those applied on unnecessary redices on each intermediate expression M_i, in order to compute s.) Indeed, if we have not yet computed s by M_i, and then we make narrowing on an unnecessary redex u, we can just eliminate this and all the following steps on redices

$u' \geq u$, and changing the substitutions accordingly we would obtain a (shorter) sequence with the same properties as the original. We have just to iterate the argument in order to remove all the unnecessary steps.

Then we try to convert it in an admissible sequence under the given strategy. For let u be the redex selected by it in the beginning; there are two possibilities: either u is a necessary redex on E to compute s or it is unnecessary. In the first case reasoning as in the proof of Theorem 3 we have that u must be the selected redex in one of the steps of q; otherwise the path along q will remain unchanged at s', and as u is necessary on E to compute s we would have $\mid s'\lambda \mid \not\trianglerighteq s$ for any substitution λ. Then we can proceed as in that proof getting a sequence q_1 equivalent to q, but whose first step is allowed by our strategy.

On the other hand, if u is unnecessary to compute s, and there is not any step of q narrowing on u, we know by Corollary 3 that there will be neither any step narrowing in any occurrence comparable with u. Then we would take the (or any in general) rule in Π that is applicable to $(E/u)\theta$. If σ' is the substitution used to narrow E/u by this rule on its root, we have that $\sigma' \leq \theta$ so that σ and σ' are compatible. Then we can execute q after this first step, by means of the corresponding substitutions getting a sequence $q_1 : E \overset{N}{\longrightarrow}_{\sigma_1} s'$. Note that as q contains no unnecessary steps, the new sequence remains lazy : if a necessary redex would become no more lazy redex after a narrowing step, this would mean than the value of some subexpression containing this redex would become undefined, but as the redex is necessary in order to compute s we could not do it any more, contradicting our hypothesis. Oppositely if we would have unnecessary steps in q some of them could have become non lazy without contradiction as it would just imply that the value of some subexpression that is no needed in order to compute s is undefined. So we can get in both cases some q_1 equivalent to q, but whose first step is allowed by our strategy.

We then have just to iterate the construction, but how long? Obviously until exhausting q, in which moment the constructed sequence q' will be allowable under the current strategy and will verify the thesis of the theorem. When we are in the first of the two former cases we indeed use one step of q in order to compute the following narrowing sequence; this is not true in the other case, but it cannot be repeated infinitely often, because some (at least the first) of the remaining steps will correspond to lazy redices that will remain like that after any narrowing step on an unnecessary redex, and then sooner or later some of them must be selected by our fair strategy. □

Remarks:

1. In this case we can not extend the completeness result, without changing the notion of admissible strategy, to the case in which $dom(\theta) \subset var(E)$. For let us consider a new modification of Example 3 : take now $f(0) = 0$; $f(1) = 1$; $f(p(x, y)) = 0$ as the definition of this function. Then if we take $E = p(f(x), g(x))$ and as θ the empty substitution, we have that for any data substitution ρ $[\![E]\!]_I(\rho) \sqsupseteq p(\bot, 1)$. But if our strategy would choose 1 as first redex we would have to bound x to 0, 1 or $p(x', y)$, so that any substitution σ corresponding to an extension of any of these computations would be not $\sigma < \theta$, and the completeness thesis would not be fulfilled.

2. We could syntactically complete any partial definition by adding the clause $f(x) := f(x)$. Clearly this would not modify the meaning of the program, but, of course, if

f was not totally syntactically undefined, we would obtain after this modification a program not verifying the pfns. In the next section we will prove that we can transform each program in an equivalent one verifying pfns. Besides, if the original program is syntactically complete the transformed one also has this property, and so we would be in the hypothesis of applying Theorem 6.

You can see that the necessity of fairness, the problem reported in part **1** of the previous remark, and the necessity to restrict ourselves to syntactically complete programs are all of them due to the admission by the strategy of narrowing steps on occurrences that are unnecessary for the searched result.

In order to avoid these restrictions, we could introduce the notion of goal oriented strategy. This would be defined given a goal term s that must be (it or any other term improving it) obtained by the evaluation of the considered expression E (see Theorem 6 for the corresponding technical details.) We say that a narrowing strategy GST is goal oriented to s if it never selects unnecessary redices in order to compute s; then, we have the following

Theorem 7 Let Π be a simple-BABEL program verifying pfns, E an expression, θ a data substitution with $dom(\theta) \subseteq var(E)$, and s a finite, ground and (possibly) partial term, such that $[\![E\theta]\!]_I(\rho) \sqsupseteq s$ under any Herbrand interpretation and for any environment ρ over I; then for any goal oriented to s strategy GST there exists a narrowing sequence under it $q : E \mathrel{\mathrm{N}}\!\!\xrightarrow{*}_{\sigma_E} N$ and a substitution λ such that $\theta = \sigma_E \circ \lambda$ and $\mid N\lambda \mid \sqsupseteq s$.

Proof: : Similar to that of previous theorem; just note that syntactical completeness of the program, fairness of the strategy and the restriction to ground expressions $E\theta$ were all just necessary in order to cope with the selection of an unnecessary redex along the narrowing process, so we can leave them out once only goal oriented strategies are admissible. □

We even announced in the introduction a third possibility in order to treat the problem caused by the computation of partial values. Again we have to generalize the notion of strategy in order to describe it.

Definition 22 A *sufficiently general strategy* SGS is a function associating to any expression E a set of its lazy redices $SGS(E)$ such that if E is narrowable on some u under σ, there is some $u' \in SGS(E)$ such that E is narrowable on u' under some σ' with $\sigma \leq \sigma'$. □

Intuitively a general strategy would be the straight generalization of strategies that is obtained by considering functions that select a set of redices instead of a single one. This, of course, would introduce some backtracking on redices, increasing the probability to maintain completeness under some adverse hypothesis as those introduced in the case that we are now studying. On the other hand sufficient generality is captured by the condition on substitutions related with possible narrowing steps, that means that the narrowing steps corresponding to the redices selected by the strategy are sufficiently general, so that no substitution corresponding to other redex will be lost, although just redices in the selected set , will be considered.

Definition 23 A narrowing sequence is *admissible under a sufficiently general strategy* SGS iff any of its narrowing steps $M_i \mathrm{N}\!\!\longrightarrow_{\sigma_M^{i+1}} M_{i+1}$ is made on some redex in $SGS(M_i)$. □

Of course one could argue that these general strategies are not real ones, because in fact the universal one associating to each expression the set of all its redices would be indeed one of them, but it is clear that it does not make any selection at all, what is just the objective of strategies. That is true; we introduced complete strategies, that is to say, strategies preserving completeness of narrowing, in order to reduce backtracking along computations. This is so because we do not need to consider all the narrowing steps from the reducing expression, but just the corresponding to one redex: this selected by the adopted strategy. Then, what we are now proposing is the consideration of strategies selecting sets of redices, that of course will be only used in practice when we do not know any complete ordinary strategy. And obviously in these cases we will look for strategies selecting sets as small as possible (although this is not at all a theoretical imperative.)

We should extent the notion of dynamic strategy to general ones in order to introduce fairness, what can be done in a rather straightforward way. Then we have

Theorem 8 Let Π be a simple-BABEL program verifying pfns, E an expression, θ a data substitution with $dom(\theta) \subseteq var(E)$, and s a finite, ground and (possibly) partial term, such that $[\![E\theta]\!]_I(\rho) \sqsupseteq s$ under any Herbrand interpretation and for any environment ρ over I; then for any sufficiently general strategy SGS there exists a narrowing sequence under it $q : E \mathrm{N}\!\!\overset{*}{\longrightarrow}_{\sigma_E} N$ and a substitution λ such that $\theta = \sigma_E \circ \lambda$ and $\mid N\lambda \mid \sqsupseteq s$. □

4 Transforming specifications

Echahed has made in [1] a detailed presentation of an algorithm that transforms a specification into an "equivalent" one fulfilling the pfns. We will present a simple example in order to illustrate the algorithm, and mainly to explain why we have quoted the word equivalent in our previous sentence.

Example 4 Let us consider the following simple-BABEL program (and also specification in Echahed's sense):

$$\begin{array}{lcl} f(0,0) & := & 0 \\ f(s(x),0) & := & 1 \\ f(x,s(y)) & := & 2 \end{array}$$

Echahed's transformer would transform it, into the following specification:

$$\begin{array}{lcl} f(0,0) & := & 0 \\ f(s(x),0) & := & 1 \\ f(0,s(y)) & := & 2 \\ f(s(x),s(y)) & := & 2 \end{array}$$

□

We need this transformation because in the original specification we had a situation of strictly subunification on the first argument of the function, between the third and the two first equations. In order to eliminate this problem we unfold the x in this third

equation, introducing a new equation for each constructor in the signature over which we are working. Note that in the general case we have to consider all the constructors in that signature, and not just those included in the patterns of the conflicting equations; this has been just a coincidence in our example. Also, in the general case we would have to iterate the algorithm in order to avoid conflicts between a variable and a pattern containing nested constructors: for instance if we had two patterns x and $s(s(y))$ it would not be sufficient to unfold x into 0 and $s(x')$, we would have to iterate the process because a conflict between x' and $s(y)$ remains. In any case, we think that once the idea of the algorithm is understood, it is more or less easy to guess how it would work in the general case, so that we will not include here a more detailed presentation of the algorithm; this should be just a copy of that in [1] .

Once we have understood the transformation, we will probably assert without any doubt that the original and the obtained specifications are equivalent. But this immediate assertion needs a more accurate statement: in the general case S and $TR(S)$ will be equivalent for ground expressions, but not for arbitrary ones. This is exactly the same problem that we reported in part 1 of our last remark. It was not a problem for Echahed because he was only interested in the evaluation of ground terms.

For instance, we had for the original specification in our example $[\![f(x, s(y))]\!]_I(\rho) = 2$ under any Herbrand interpretation, and for any environment ρ. Nevertheless, this expression is undefined under $TR(S)$. This is completely reasonable under our semantics of simple-BABEL programs, because we admit partial values, and of course they can appear not only as results but also as arguments. So we have as possible environment one verifying $\rho(x) = \perp$, and clearly $f(\perp, s(y))$ is undefined for any value of y. Of course, this would only be a problem for us if we are interested on general completeness results covering also non-ground cases.

Fortunately, we have got a new transformer, pointed to us by J.J.Moreno-Navarro, fully preserving the semantics of programs. Again we will begin by showing how it works by means of an example, the same that was used before.

If we denote by TR' this new transformer, $TR'(S)$ would be the following "simple-BABEL" program:

$$\begin{array}{lcl} f(x,0) & := & f'(x,0) \\ f(y,0) & := & f''(y,0) \\ f(x,s(y)) & := & 2 \\ f'(0,0) & := & 0 \\ f''(s(x),0) & := & 1 \end{array}$$

You can see that the philosophy of the transformer is just the opposite of that of the former one. Instead of particularizing variables, we generalize non-variable patterns.

We had also to quote a word in the sentence before the description of the transformed program. This is because, unfortunately, it is not a real simple-BABEL program, because for instance $f(3,0)$ matches both $f(x,0)$ and $f(y,0)$, but $[\![f'(3,0)]\!] = \perp \neq 1 = [\![f''(3,0)]\!]$.

Nevertheless, we can generalize the class of simple-BABEL programs, relaxing the non-ambiguity restriction, by putting instead the following generalized non-ambiguity restriction: If $L_1 := E_1$ and $L_2 := E_2$ are in Π and L_1 and L_2 are unifiable via m.g.u. σ, then either $[\![E_1(\sigma)]\!] = [\![E_2(\sigma)]\!]$ or one of them is undefined.

Of course, we need slight modifications in our definitions in order to cope with this generalization. For instance, in Definition 5 we have to add the condition $[\![E]\!]_I \neq \perp$ to the description of the first case. Also the strong soundness result presented in Theorem 2 is lost, but we have yet its weak version saying $[\![M\sigma_M]\!]_I(\rho) \sqsupseteq [\![N\sigma_M]\!]_I(\rho)$, what is enough taking into account the fact that usually we are only interested in the computations of total values, and the completeness result whose validity remains and says us that selecting the appropiate branch we can ever compute by narrowing the (total) value of an expression.

Let us go now with the formal definition of the new transformer.

Definition 24 Let f be a function defined in a simple-BABEL program Π. We say

1. For each argument i the pattern defining it in a rule $L := R$ corresponding to f is *too particular* iff there exists some other rule defining f, $L' := R'$, with the pattern defining that argument strictly more general than the former; that is to say, if t and t' are those patterns, there exists a non-trivial substitution σ with $t = t'\sigma$.

2. If the pattern t defining the i-th argument of f in a rule $L := R$ is too particular, we call *the most general covering* (mgc) of it in Π any renaming of the pattern t' of that argument on any other rule for f in Π that was not too particular, and verifies $t = t'\sigma$ for some substitution σ.

□

Definition 25 We obtain the transformed version $TR'(\Pi)$ of a simple-BABEL program Π in the following way:

- If none of the patterns in $L := R \in \Pi$ is too particular we include $L := R$ in $TR'(\Pi)$.

- If $tpp(L := R) \neq \emptyset$, where $tpp(r)$ denotes the set of arguments with too particular patterns of a rule, we include in $TR'(\Pi)$ a couple of rules

$$\begin{array}{lcl} f(t'_1, \ldots, t'_n) & := & f'_r(t'_1, \ldots, t'_n) \\ f'_r(t_1, \ldots, t_n) & := & R \end{array}$$

  where $L = f(t_1, \ldots, t_n)$; $t'_i = t_i$ if this argument is not too particular, and $t'_i = mgc(t_i)$ if it is too particular; and f'_r is a new defined function for each so developed rule r. Of course, the sets of variables in each t'_i will be taken disjoint, which is possible by the linearity of the rules in Π.

□

Theorem 9 If Π is a simple-BABEL program, its transformed version $TR'(\Pi)$ is a general simple-BABEL program, verifying the pfns, and such that any (possibly non-ground) expression has the same semantics under both programs. □

Remarks:

1. The size of $TR'(\Pi)$ is always linear in the size of Π; in fact this size can only be increased by a factor of 2 when the new transformer is applied. Oppositely, even if Π has no nested constructors in the patterns of its rules, the size of the definition of a function in a program can increase by a factor of c^n where n is its arity, and c the number of constructors in the signature. One can check that things will be even worse if the patterns include nested constructors, although this is not, indeed, a very usual case in practice.

2. From the pragmatic point of view, general simple-BABEL programs could not be very interesting if traversing of many failing branches in a computation tree implied too many wasted time in useless unifications. But this is not a real problem for the programs obtained by means of our transformer, because the only useless unifications for them will be those leading us to some of the introduced functions f'_r, and immediately the computation would fail if this was indeed a useless path. For instance, if in our example we compute $f(3,0)$, the first new rule will lead us to $f'(3,0)$, and immediately this branch will fail; then we will apply the other possible rule, obtaining $f''(3,0)$, and then 1. Moreover, although this new transformer introduces many new functions, they could be implemented more efficiently than ordinary ones, because they are always just defined by a single rule.

   On the other hand, if we computed the possible values of $f(x,1)$ we would immediately obtain the answer 2, without any binding for x; instead, if we computed it using $TR(\Pi)$ we would obtain a couple of answers 2, with $x = 0$ and $x = s(x')$ as bindings. We have not only lost the most general answer, covering also the case $x = \perp$, but also we have introduced an unnecessary duplication of answers, what could be dramatic if this situation presented repeatedly in a nested way.

5 Acknowledgements

We are very grateful to M. Rodríguez Artalejo that proposed us to work on the subject after reading Echahed's paper along a visit to his institution. Also to J. J. Moreno Navarro that gave us the idea for the new transformer. And to the referees of a previous version of this paper for their encouragement and clever suggestions, after a very careful reading. And finally to K. Indermark and all the members of his group in Aachen, specially to R. Loogen and H. Kuchen, whom we presented the results of this research that was developed during an stay of the authors in their University.

References

[1] R. Echahed *On completeness of narrowing strategies* in CAAP 88, LNCS 299, Springer Verlag (1988)

[2] M. J. Fay *First Order Unification in an Equational Theory* Proc. 4th Workshop on Automated Deduction, Austin, Texas (1979)

[3] L. Fribourg *SLOG: A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting* SLP 85, Boston,1985

[4] G. Huet, D. C. Open *Equations and Rewrite Rules: A Survey* in Formal Language Theory: Perspectives and Open Problems, Academic Press (1980)

[5] J. M. Hullot *Canonical Forms and Unification* 5th CADE, LNCS 87, Springer Verlag (1980)

[6] G. Levi, P.G. Bosco, E. Giovannetti, C. Moiso, C. Palamidesi *A Complete Semantic Characterization of K- LEAF, a Logic Language with Partial Functions* Proc. 4th Symposium on Logic Programming, IEEE Computer Society Press (1987)

[7] G. Lindstrom *Functional Programming and the Logical Variable* 12th ACM POPL, 1985

[8] J. J. Moreno-Navarro, M. Rodríguez-Artalejo *BABEL: A Functional and Logic Programming Language based on Constructor Discipline and Narrowing* in Algebraic and Logic Programming, LNCS 343, Springer Verlag (1989)

[9] J. J. Moreno-Navarro, M. Rodríguez-Artalejo *Logic Programming with Functions and Predicates: The language BABEL* Informe Técnico DIA/89/3, Departamento de Informática y Automática, Universidad Complutense Madrid (1989) (Also to appear in Journal of Logic Programming)

[10] J. J. Moreno-Navarro *Diseño, Semántica e Implementación de BABEL: Un lenguaje que integra la programación funcional y lógica* Ph.D.Thesis, Facultad de Informática, Universidad Politécnica Madrid (1989)

[11] W. Nutt, P. Rety, G. Smolka *Basic Narrowing Revisited* SEKI Report SR 87/07, 1987

[12] P. Padawitz *Strategy Controlled Reduction and Narrowing* RTA 87, LNCS 256, Springer Verlag (1987)

[13] P.Padawitz *Computing in Horn Clause Theories* EATCS Monographs on Theoretical Computer Science, Vol. 16, Springer Verlag (1988)

[14] U.S. Reddy *Narrowing as the Operational Semantics of Functional Languages*, Proc. IEEE International Symposium on Logic Programming, IEEE Computer Society Press (1985)

[15] D. S. Scott *Domains for Denotational Semantics* ICALP 82, LNCS 140, Springer Verlag (1982)

[16] J. R. Slagle *Automated Theorem Proving with Theories with Simplifiers, Conmutativity and Associativity* JACM 21, 1974

[17] D. A. Turner *MIRANDA: A non-strict functional language with polymorphic types* Proc. ACM Conf. on Functional Languages and Computer Architecture 1985, LNCS 201, Springer Verlag (1985)

[18] J. You *Enumerating Outer Narrowing Derivations for Constructor-Based Term Rewriting Systems*, J. Symbolic Computation 7: 319-341 (1989)