©Springer-Verlag
In *Proc. of the 4th Fuji International Symposium on Functional and Logic Programming (FLOPS'99)*, Tsukuba (Japan).
Springer LNCS 1722, pp. 353–368, 1999

A Semantics for Program Analysis in Narrowing-Based Functional Logic Languages

Michael Hanus[1]⋆ Salvador Lucas[2]⋆⋆

[1] Informatik II, RWTH Aachen, D-52056 Aachen, Germany
hanus@informatik.rwth-aachen.de
[2] DSIC, UPV, Camino de Vera s/n, E-46022 Valencia, Spain.
slucas@dsic.upv.es

Abstract. We introduce a denotational characterization of narrowing, the computational engine of many functional logic languages. We use a functional domain for giving a denotation to the narrowing space associated to a given initial expression under an arbitrary narrowing strategy. Such a semantic description highlights (and favours) the operational notion of evaluation instead of the more usual model-theoretic notion of interpretation as the basis for the semantic description. The motivation is to obtain an abstract semantics which encodes information about the real operational framework used by a given (narrowing-based) functional logic language. Our aim is to provide a general, suitable, and accurate framework for the analysis of functional logic programs.

Keywords: domain theory, functional logic languages, narrowing, program analysis, semantics.

1 Introduction

The ability of reasoning about program properties is essential in software design, implementations, and program manipulation. Program analysis is the task of producing (usually approximated) information about a program without actually executing it. The analysis of functional logic programs is one of the most challenging problems in declarative programming. Many works have already addressed the analysis of certain run-time properties of programs (e.g., [3, 11, 13, 15, 23]). Nevertheless, most of these approaches have been done in a rather ad hoc setting, gearing the analysis towards the application on hand. Up to now, there is no general approach for formulating and analyzing arbitrary properties of functional logic programs in an arbitrary operational framework. In this paper we address this problem.

The key of our approach is domain theory [19, 20] since it provides a junction between semantics (spaces of points = denotations of *computational processes*)

⋆ Partially supported by the German Research Council (DFG) grant Ha 2457/1-1.
⋆⋆ Partially supported by EEC-HCM grant ERBCHRXCT940624, Spanish CICYT grant TIC 98-0445-C03-01, and Acción Integrada hispano–alemana HA1997-0073.

and logics (lattices of properties of processes) [2, 20, 22]. The computational process we are interested in is evaluation. In a programming language, the notion of *evaluation* emphasizes the idea that there exists a distinguished set of syntactic elements (the values) which have a predefined mathematical interpretation [10]. The other syntactic elements take meaning from the program definitions *and* the operational framework for the program's execution. In this way, the evaluation process (under a given operational framework) maps general input expressions (having an *a priori* unknown meaning) to values. This point of view favours the operational notion of evaluation instead of the more usual model-theoretic notion of interpretation as the basis for the semantic description.

Since functional logic languages with a complete operational semantics are based on narrowing, we center our attention on it. The idea of using narrowing as an *evaluation mechanism* for integrated languages comes from Reddy's [18]: narrowing is the operational principle which computes the *non-ground value* (*ngv*) of an input expression. Given a domain D, a *ngv* is a *mapping* from valuations (on D) to values (in D). In moving valuations from being parameters of semantic functions (as usual in many approaches, e.g., [9, 16]) to be components of a semantic domain, we understand narrowing as an *evaluation* mechanism which incorporates the instantiation of variables as a part of such evaluation mechanism. Since *ngv*'s are functional values, we use the domain-theoretic notion of *approximable mapping* [19, 20] to give them a computable representation. We argue that this is a good starting point for expressing and managing *observable* properties of functional logic programs (along the lines of [2, 22]). Moreover, it reveals that, within an integrated framework, there exist semantic connections between purely functional and logic properties of programs. Termination and groundness are examples of such related properties. On the other hand, thanks to including operational information into the semantic description, we are able to derive interesting optimizations for program execution.

Section 2 gives some preliminary definitions. Section 3 introduces a domain theoretic approach to pure rewriting and narrowing computations. Section 4 discusses a semantic-based analysis framework for functional logic languages. Section 5 contains our conclusions.

2 Preliminaries

In this section, we give some preliminary definitions (further details in [6, 21]). Given sets A, B, B^A is the set of mappings from A to B and $\mathcal{P}(A)$ denotes the set of all subsets of A. An *order* $\sqsubseteq$ on a set A is a reflexive, transitive and antisymmetric relation. An element $\bot$ of an ordered set $(A, \sqsubseteq)$ is called a *least element* (or a *minimum*) if $\bot \sqsubseteq a$ for all $a \in A$. If such an element exists, then $(A, \sqsubseteq, \bot)$ is called a *pointed ordered set*. Given $S \subseteq A$, an element $a \in A$ is an *upper bound* of S if $x \sqsubseteq a$ for all $x \in S$. In this case we also say that S is a *consistent set*. An upper bound of S is a *least upper bound* (or *lub*, written $\bigsqcup S$) if, for all upper bounds b of S, we have $\bigsqcup S \sqsubseteq b$. A set $S \subseteq A$ is downward (upward) closed if whenever $a \in S$ and $b \sqsubseteq a$ ($a \sqsubseteq b$), we have that $b \in S$. If

$S = \{x, y\}$, we write $x \sqcup y$ instead of $\bigsqcup S$. A non-empty set $S \subseteq A$ is *directed* if, for all $a, b \in S$, there is an upper bound $c \in S$ of $\{a, b\}$. An ideal is a downward closed, directed set and $Id(A)$ is the set of ideals of an ordered set A. A pointed ordered set $(A, \sqsubseteq, \bot)$ is a *complete partial order* (*cpo*) if every directed set $S \subseteq A$ has a *lub* $\bigsqcup S \in A$. An element $a \in A$ of a *cpo* is called *compact* (or *finite*) if, whenever $S \subseteq A$ is a directed set and $a \sqsubseteq \bigsqcup S$, then there is $x \in S$ such that $a \sqsubseteq x$. The set of compact elements of a cpo A is denoted as $K(A)$. A cpo A is *algebraic* if for each $a \in A$, the set $\mathsf{approx}(a) = \{x \in K(A) \mid x \sqsubseteq a\}$ is directed and $a = \bigsqcup \mathsf{approx}(a)$. An algebraic cpo D is a *domain* if, whenever the set $\{x, y\} \subseteq K(D)$ is consistent, then $x \sqcup y$ exists in D. Given ordered sets $(A, \sqsubseteq_A)$, $(B, \sqsubseteq_B)$, a function $f : A \to B$ is monotonic if $\forall a, b \in A, a \sqsubseteq_A b \Rightarrow f(a) \sqsubseteq_B f(b)$; $f : A \to A$ is idempotent if $\forall a \in A, f(f(a)) = f(a)$.

By V we denote a countable set of *variables*; Σ denotes a *signature*, i.e., a set of *function symbols* $\{\mathtt{f}, \mathtt{g}, \ldots\}$, each with a fixed *arity* given by a function $ar : \Sigma \to \mathbb{N}$. We assume $\Sigma \cap V = \emptyset$. We denote by $\mathcal{T}(\Sigma, V)$ the set of (finite) terms built from symbols in the signature Σ and variables in V. A k-tuple $t_1, \ldots, t_k$ of terms is denoted as $\overline{t}$, where k will be clarified from the context. Given a term t, $\mathcal{V}ar(t)$ is the set of variable symbols in t. Sometimes, we consider a fresh constant $\bot$ and $\Sigma_\bot = \Sigma \cup \{\bot\}$. Terms from $\mathcal{T}(\Sigma_\bot, V)$ are ordered by the usual *approximation ordering* which is the least ordering $\sqsubseteq$ satisfying $\bot \sqsubseteq t$ for all t and $f(\overline{t}) \sqsubseteq f(\overline{s})$ if $\overline{t} \sqsubseteq \overline{s}$, i.e., $t_i \sqsubseteq s_i$ for all $1 \leq i \leq ar(f)$.

Terms are viewed as labeled trees in the usual way. *Positions* $p, q, \ldots$ are represented by chains of positive natural numbers used to address subterms of t. By Λ, we denote the empty chain. The set of positions of a term t is denoted by $\mathcal{P}os(t)$. A *linear term* is a term having no multiple occurrences of the same variable. The subterm of t at position p is denoted by $t|_p$. The set of positions of non-variable symbols in t is $\mathcal{P}os_\Sigma(t)$, and $\mathcal{P}os_V(t)$ is the set of variable positions. We denote by $t[s]_p$ the term t with the subterm at the position p replaced by s.

A *substitution* is a mapping $\sigma : V \to \mathcal{T}(\Sigma, V)$ which homomorphically extends to a mapping $\sigma : \mathcal{T}(\Sigma, V) \to \mathcal{T}(\Sigma, V)$. We denote by ε the "identity" substitution: $\varepsilon(x) = x$ for all $x \in V$. The set $\mathcal{D}om(\sigma) = \{x \in V \mid \sigma(x) \neq x\}$ is called the *domain* of σ and $\mathcal{R}ng(\sigma) = \cup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\sigma(x))$ its *range*. $\sigma_{|U}$ denotes the *restriction* of a substitution σ to a subset of variables $U \subseteq \mathcal{D}om(\sigma)$. We write $\sigma \leq \sigma'$ if there is θ such that $\sigma' = \theta \circ \sigma$. A *unifier* of two terms t_1, t_2 is a substitution σ with $\sigma(t_1) = \sigma(t_2)$. A *most general unifier* (*mgu*) of t_1, t_2 is a unifier σ with $\sigma \leq \sigma'$ for all other unifiers σ' of t_1, t_2.

A *rewrite rule* (labeled α) is an ordered pair (l, r), written $\alpha : l \to r$ (or just $l \to r$), with $l, r \in \mathcal{T}(\Sigma, V)$, $l \notin V$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. l and r are called *left-hand side* (*lhs*) and *right-hand side* (*rhs*) of the rule, respectively. A *term rewriting system* (*TRS*) is a pair $\mathcal{R} = (\Sigma, R)$ where R is a set of rewrite rules. A TRS (Σ, R) is *left-linear*, if for all $l \to r \in R$, l is a linear term. Given $\mathcal{R} = (\Sigma, R)$, we consider Σ as the disjoint union $\Sigma = \mathcal{C} \uplus \mathcal{F}$ of symbols $c \in \mathcal{C}$, called *constructors*, and symbols $f \in \mathcal{F}$, called *defined functions*, where $\mathcal{F} = \{f \mid f(\overline{l}) \to r \in R\}$ and $\mathcal{C} = \Sigma - \mathcal{F}$. A *constructor-based* TRS (*CB-TRS*) is a TRS with $l_1, \ldots, l_n \in \mathcal{T}(\mathcal{C}, V)$ for all rules $f(l_1, \ldots, l_n) \to r$.

For a given TRS $\mathcal{R} = (\Sigma, R)$, a term t *rewrites* to a term s (at position p), written $\overset{[p,\alpha]}{\rightarrow}_{\mathcal{R}}$ (or just $t \overset{p}{\rightarrow}_{\mathcal{R}} s$, $t \rightarrow_{\mathcal{R}} s$, or $t \rightarrow s$) if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some rule $\alpha : l \rightarrow r \in R$, position $p \in \mathcal{P}os(t)$ and substitution σ. A term t is in *normal form* if there is no term s with $t \rightarrow_{\mathcal{R}} s$. A TRS $\mathcal{R}$ (or the rewrite relation $\rightarrow_{\mathcal{R}}$) is called *confluent* if for all terms t, t_1, t_2 with $t \rightarrow^*_{\mathcal{R}} t_1$ and $t \rightarrow^*_{\mathcal{R}} t_2$, there exists a term t_3 with $t_1 \rightarrow^*_{\mathcal{R}} t_3$ and $t_2 \rightarrow^*_{\mathcal{R}} t_3$. A term t *narrows* to a term s, written $t \leadsto_{[p,\alpha,\sigma]} s$ (or just $t \leadsto_\sigma s$), if there is $p \in \mathcal{P}os_\Sigma(t)$ and a variant (i.e., a renamed version) of a rule $\alpha : l \rightarrow r$ such that $t|_p$ and l unify with (idempotent) mgu σ, and $s = \sigma(t[r]_p)$. A narrowing derivation $t \leadsto^*_\sigma s$ is such that either $t = s$ and $\sigma = \varepsilon$ or $t \leadsto_{\sigma_0} t_1 \leadsto_{\sigma_1} \cdots t_{n-1} \leadsto_{\sigma_{n-1}} s$ and $\sigma = \sigma_{n-1} \circ \cdots \circ \sigma_1 \circ \sigma_0$. In order to show the progress of a narrowing derivation w.r.t. the computed answer and the evaluated goal, we also define the narrowing relation on substitution/term pairs by $\langle \sigma, t \rangle \leadsto \langle \sigma', s \rangle$ if $t \leadsto_\theta s$ and $\sigma' = \theta_{|\mathcal{V}ar(t)} \circ \sigma$ (i.e., we consider only the substitution of goal variables).

3 The Semantic Approach

A (first-order) program $\mathcal{P} = (\mathcal{R}, t)$ consists of a TRS $\mathcal{R}$ (which establishes the distinction between constructor and defined symbols of the program), and an initial expression t to be *fully* evaluated. We make t explicit since the differences between the purely functional and functional logic styles arise in the different status of the variables occurring in the initial expression: in functional programming, those variables are not allowed (or they are considered as constants and cannot be instantiated). Functional logic languages deal with expressions having *logic* variables and narrowing provides for the necessary instantiations.

We characterize the information which is *currently* couched by a term by means of a mapping $(\!|\ |\!)$ from terms to (partial) values (remind that values are expected to be especial syntactic objects). We call $(\!|\ |\!)$ an *observation* mapping. The adequacy of a given mapping $(\!|\ |\!)$ for observing computations performed by a given operational mechanism should be ensured by showing that $(\!|\ |\!)$ is a homomorphism between the relation among syntactic objects induced by the operational mechanism and the approximation ordering on values. This means that the operational mechanism refines the meaning of an expression as the computation continues.

As a preliminary, simple example, consider pure rewriting computations: The syntactic objects are terms $t \in \mathcal{T}(\Sigma_\perp, V)$ and the values are taken from $(\mathcal{T}^\infty(\mathcal{C}_\perp), \sqsubseteq, \perp)$, the domain of infinite, ground constructor (partial) terms[1]. $(\mathcal{T}^\infty(\mathcal{C}_\perp, V), \sqsubseteq, \perp)$ is the domain $(\mathcal{T}^\infty(\mathcal{C}_\perp \cup V), \sqsubseteq, \perp)$, where $\forall x \in V, ar(x) = 0$. For functional computations, we use $(\!|\ |\!)_F : \mathcal{T}(\Sigma_\perp, V) \rightarrow \mathcal{T}(\mathcal{C}_\perp, V)$ given by

$$\begin{array}{rcll@{\qquad}rcll}
(\!|x|\!)_F &=& x & & (\!|\perp|\!)_F &=& \perp & \\
(\!|c(\overline{t})|\!)_F &=& c((\!|\overline{t}|\!)_F) & \text{if } c \in \mathcal{C} & (\!|f(\overline{t})|\!)_F &=& \perp & \text{if } f \in \mathcal{F}
\end{array}$$

[1] Formally, $(\mathcal{T}^\infty(\mathcal{C}_\perp), \sqsubseteq, \perp)$ is obtained from $\mathcal{T}(\mathcal{C}_\perp)$, which is not even a cpo, as (isomorphic to) its ideal completion $(Id(\mathcal{T}(\mathcal{C}_\perp)), \subseteq, \{\perp\})$ (see [21]).

Proposition 1 (Reduction increases information). *Let $\mathcal{R}$ be a TRS and $t, s \in \mathcal{T}(\Sigma_\perp, V)$. If $t \to^* s$, then $(\!|t|\!)_F \sqsubseteq (\!|s|\!)_F$.*

The function $Rew : \mathcal{T}(\Sigma_\perp, V) \to \mathcal{P}(\mathcal{T}(\mathcal{C}_\perp, V))$ gives a representation $Rew(t) = \{(\!|s|\!)_F \mid t \to^* s\}$ of the rewriting space of a given term t.

Proposition 2. *Let $\mathcal{R}$ be a confluent TRS. For all $t \in \mathcal{T}(\Sigma_\perp, V)$, $Rew(t)$ is a directed set.*

The semantic function $CRew^\infty : \mathcal{T}(\Sigma_\perp, V) \to \mathcal{T}^\infty(\mathcal{C}_\perp, V)$ gives the meaning of a term under evaluation by rewriting: $CRew^\infty(t) = \bigsqcup Rew(t)$. Thus, $CRew^\infty(t)$ is the most defined (possibly infinite) value which can be obtained (or approximated) by issuing rewritings from t. Note that the use of infinite terms in the codomain of $CRew^\infty$ is necessary for dealing with non-terminating programs.

3.1 Narrowing as the Evaluation Mechanism

In the context of a program, a term t with variables denotes a continuous function $t_D \in [D^V \to D]$ which yields the evaluation of t under each possible valuation[2] $\phi \in D^V$ of its variables on a domain D. This is called a *non-ground value* (*ngv*) in [18] and a *derived operator* in [8].

Given domains D and E, the set $[D \to E]$ of (strict) continuous functions from D to E (pointwise) ordered by $f \sqsubseteq g$ iff $\forall x \in V,\ f(x) \sqsubseteq g(x)$, is a domain [10, 21]. For proving that $[D^V \to D]$ is a domain whenever D is, assume that V contains a distinguished (unused) variable $\perp$. Thus, V supplied by the least ordering $\sqsubseteq$ such that $\perp \sqsubseteq x$ and $x \sqsubseteq x$ for all $x \in V$ is a domain. The set $D^{V - \{\perp\}}$ of arbitrary valuations from $V - \{\perp\}$ to D is isomorphic to the domain $[V \to_\perp D]$ of continuous, strict valuations. We assume this fact from now on by removing $\perp$ from V and considering that D^V is a domain. Therefore, $[D^V \to D]$ is a domain and, in particular, $[\mathcal{T}^\infty(\mathcal{C}_\perp)^V \to \mathcal{T}^\infty(\mathcal{C}_\perp)]$ also is.

Our syntactic objects, now, are substitution/term pairs $\langle \sigma, t \rangle$. We could naïvely extend $(\!|\ |\!)_F$ to deal with those pairs: $(\!|\langle \sigma, s \rangle|\!)_F = \langle (\!|\sigma|\!)_F, (\!|s|\!)_F \rangle$ where $(\!|\sigma|\!)_F$ is a substitution given by $(\!|\sigma|\!)_F(x) = (\!|\sigma(x)|\!)_F$ for all $x \in V$. Unfortunately, the semantic progress of a narrowing evaluation might not be captured by the computational ordering $\sqsubseteq$ (extended by $(\phi, \delta) \sqsubseteq (\phi', \delta')$ iff $\forall x \in V. \phi(x) \sqsubseteq \phi'(x)$ and $\delta \sqsubseteq \delta'$) and such an extension of $(\!|\ |\!)_F$.

Example 1. Consider the TRS

```
0+x     → x              0    ≤ x    → true
s(x)+y  → s(x+y)         s(x) ≤ s(y) → x ≤ y
```

and the narrowing step $\langle \varepsilon, \texttt{[x,x+y]} \rangle \rightsquigarrow \langle \{\texttt{x}\mapsto\texttt{0}\}, \texttt{[0,y]} \rangle$ ($\texttt{[}\cdot\texttt{,}\cdot\texttt{]}$ denotes a 2-element list). We have $(\!|\langle \varepsilon, \texttt{[x,x+y]} \rangle|\!)_F = \langle \varepsilon, \texttt{[x,}\perp\texttt{]} \rangle$ and $(\!|\langle \{\texttt{x}\mapsto\texttt{0}\}, \texttt{[0,y]} \rangle|\!)_F = \langle \{\texttt{x}\mapsto\texttt{0}\}, \texttt{[0,y]} \rangle$. Therefore, we do not get the desired increasing computation, because $\varepsilon \not\sqsubseteq \{\texttt{x}\mapsto\texttt{0}\}$ and $\texttt{[x,}\perp\texttt{]} \not\sqsubseteq \texttt{[0,y]}$.

[2] By abuse, we say that the 'domain' of a valuation $\phi \in D^V$ is $\mathcal{D}om(\phi) = \{x \in V \mid \phi(x) \neq \perp\}$.

The problem is that narrowing introduces a new computational mechanism for increasing the information associated to a given term, i.e., instantiation of *logic* variables. Thus, we introduce the observation mapping $(\!|\ |\!)_{FL} : \mathcal{T}(\Sigma_\perp, V) \to \mathcal{T}(\mathcal{C}_\perp)$ which interprets uninstantiated variables as least defined elements:

$$\begin{array}{rcll} (\!|x|\!)_{FL} &=& \perp & \\ (\!|c(\overline{t})|\!)_{FL} &=& c((\!|\overline{t}|\!)_{FL}) & \text{if } c \in \mathcal{C} \end{array} \qquad \begin{array}{rcll} (\!|\perp|\!)_{FL} &=& \perp & \\ (\!|f(\overline{t})|\!)_{FL} &=& \perp & \text{if } f \in \mathcal{F} \end{array}$$

Note that $(\!|t|\!)_{FL} = \perp_{Subst}((\!|t|\!)_F)$ and $(\!|\sigma|\!)_{FL} = \perp_{Subst} \circ (\!|\sigma|\!)_F$.

Example 2. Now, $(\!|\langle \varepsilon, \texttt{[x,x+y]}\rangle|\!)_{FL} = \langle \perp_{Subst}, \texttt{[}\perp\texttt{,}\perp\texttt{]}\rangle \sqsubseteq \langle \{\texttt{x}\mapsto\texttt{0}\}, \texttt{[0,}\perp\texttt{]}\rangle = (\!|\langle\{\texttt{x}\mapsto\texttt{0}\}, \texttt{[0,y]}\rangle|\!)_{FL}$, i.e., $(\!|\ |\!)_{FL}$ satisfies the desired property.

Narrowing computations are compatible with the new observation mapping.

Proposition 3. *Let $\mathcal{R}$ be a TRS. If $\langle\sigma, t\rangle \leadsto^* \langle\sigma', s\rangle$, then $(\!|\langle\sigma, t\rangle|\!)_{FL} \sqsubseteq (\!|\langle\sigma', s\rangle|\!)_{FL}$.*

3.2 The Narrowing Space as an Approximable Mapping

Analogously to *Rew*(t), we can build a semantic description *Narr*(t) of the narrowing evaluation of t. Nevertheless, since *Narr*(t) is intended to be a representation of a *ngv*, i.e., a *functional* value, we need to use the corresponding standard Scott's construction of *approximable mappings* [20, 21].

A *precusl* is a structure $P = (P, \sqsubseteq, \sqcup, \perp)$ where $\sqsubseteq$ is a preorder, $\perp$ is a distinguished minimal element, and $\sqcup$ is a partial binary operation on P such that, for all $a, b \in P$, $a \sqcup b$ is defined if and only if $\{a, b\}$ is consistent in P and then $a \sqcup b$ is a (distinguished) supremum of a and b [21]. Approximable mappings allow us to represent arbitrary continuous mappings between domains on the representations of those domains (their compact elements) as relations between approximations of a given argument and approximations of its value at that argument [21].

Definition 1. [21] *Let $P = (P, \sqsubseteq, \sqcup, \perp), P' = (P', \sqsubseteq', \sqcup', \perp')$ be precusl's. A relation $f \subseteq P \times P'$ is an approximable mapping from P to P' if*

1. $\perp\ f\ \perp'$.
2. *$a\ f\ b$ and $a\ f\ b'$ imply $a\ f\ (b \sqcup b')$.*
3. *$a\ f\ b$, $a \sqsubseteq a'$, and $b' \sqsubseteq' b$ imply $a'\ f\ b'$.*

The ideal completion $(Id(P), \subseteq, \{\perp\})$ of a precusl is a domain (see [21]). An approximable mapping defines a continuous function between $Id(P)$ and $Id(P')$: $\overline{f} : Id(P) \to Id(P')$ is given by $\overline{f}(I) = \{b \in P' \mid \exists a \in I \text{ with } a\ f\ b\}$.

Proposition 4. *Let $P = (P, \sqsubseteq, \sqcup, \perp), P' = (P', \sqsubseteq', \sqcup', \perp')$ be precusl's, and $f, f' \subseteq P \times P'$ be approximable mappings from P to P'. If $f \subseteq f'$, then $\overline{f} \sqsubseteq \overline{f'}$.*

Given a term t, *NDeriv*(t) is the set of narrowing derivations issued from t. We associate an approximable mapping $Narr^A(t)$ to a given narrowing derivation $A \in NDeriv(t)$.

Definition 2. *Given a term $t \in \mathcal{T}(\Sigma_\perp, V)$ and a narrowing derivation*

$$A : \langle \varepsilon, t \rangle = \langle \sigma_0, t_0 \rangle \leadsto \langle \sigma_1, t_1 \rangle \leadsto \cdots \leadsto \langle \sigma_{n-1}, t_{n-1} \rangle \leadsto \langle \sigma_n, t_n \rangle$$

we define $Narr^A(t) = \cup_{0 \leq i \leq n} Narr_i^A(t)$ where:

$$Narr_i^A(t) = \{\langle \varsigma, \delta \rangle \mid \exists \phi \in \mathcal{T}(\mathcal{C}_\perp)^V . (\!|\phi \circ \sigma_i|\!)_{FL} \sqsubseteq \varsigma \wedge \delta \sqsubseteq (\!|\phi(t_i)|\!)_{FL}\}$$

Proposition 5. *Let $\mathcal{R}$ be a TRS, t be a term, and A be a narrowing derivation starting from t. Then, $Narr^A(t)$ is an approximable mapping.*

Definition 3. *Given a term $t \in \mathcal{T}(\Sigma_\perp, V)$, we define the relation $Narr(t) \subseteq \mathcal{T}(\mathcal{C}_\perp)^V \times \mathcal{T}(\mathcal{C}_\perp)$ to be $Narr(t) = \bigcup_{A \in NDeriv(t)} Narr^A(t)$.*

Unfortunately, these semantic definitions are not consistent w.r.t. rewriting.

Example 3. Consider the TRS

```
f(f(x)) → a
c       → b
```

and $A : \langle \varepsilon, t \rangle = \langle \varepsilon, \texttt{f(x)} \rangle \leadsto \langle \{\texttt{x} \mapsto \texttt{f(x')}\}, \texttt{a} \rangle$. If $m = Narr^A(t)$, then $\{\texttt{x} \mapsto \texttt{a}\}\ m\ \texttt{a}$ (we take $\phi = \perp_{Subst}$, $\sigma = \{\texttt{x} \mapsto \texttt{f(x')}\}$ in Definition 2; hence, $(\!|\phi \circ \sigma|\!)_{FL} = \perp_{Subst} \sqsubseteq \{\texttt{x} \mapsto \texttt{a}\} = \varsigma$). Thus, $\overline{Narr^A(t)}(\{\texttt{x} \mapsto \texttt{a}\}) = \texttt{a}$. However, $\{\texttt{x} \mapsto \texttt{a}\}(t) = \texttt{f(a)} \not\to^* \texttt{a}$.

The problem here is that $(\!|\ |\!)_{FL}$ identifies (as $\perp$) parts of the bindings $\sigma(x)$ of a computed substitution σ which can be semantically refined by instantiation (of the variables in $\sigma(x)$) and other which cannot be further refined by instantiation (the operation-rooted subterms in $\sigma(x)$). If we deal with left-linear CB-TRS's and choose (idempotent) *mgu*'s as unifiers for the narrowing process, the substitutions which we deal with are *linear* constructor substitutions, i.e., for all narrowing derivations $\langle \varepsilon, t \rangle \leadsto^* \langle \sigma, s \rangle$ and all $x \in V$, $\sigma(x)$ is a constructor term and $\{\sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$ is a linear set of terms (i.e., no variable appears twice within them). Hence, the substitutions computed by narrowing have *no* partial information apart from the variable occurrences. In this case, $(\!|\sigma|\!)_F = \sigma$, $(\!|\sigma|\!)_{FL} = \perp_{Subst} \circ (\!|\sigma|\!)_F = \perp_{Subst} \circ \sigma$, and we have the following result.

Proposition 6. *Let σ be a linear constructor substitution and $\phi, \varsigma \in \mathcal{T}(\mathcal{C}_\perp)^V$. If $\phi \circ \sigma \sqsubseteq \varsigma$, then there exists $\phi' \in \mathcal{T}(\mathcal{C}_\perp)^V$ such that $\phi \sqsubseteq \phi'$ and $\phi' \circ \sigma = \varsigma$.*

Thus, we obtain a simpler, more readable expression for the approximable mapping which is associated to a given left-linear CB-TRS by noting that

$$\begin{aligned} Narr_i^A(t) &= \{\langle \varsigma, \delta \rangle \mid \exists \phi \in \mathcal{T}(\mathcal{C}_\perp)^V . (\!|\phi \circ \sigma_i|\!)_{FL} \sqsubseteq \varsigma \wedge \delta \sqsubseteq (\!|\phi(t_i)|\!)_{FL}\} \\ &= \{\langle \varsigma, \delta \rangle \mid \exists \phi \in \mathcal{T}(\mathcal{C}_\perp)^V . \phi \circ \sigma_i = \varsigma \wedge \delta \sqsubseteq (\!|\phi(t_i)|\!)_{FL}\} \end{aligned}$$

The union of approximable mappings (considered as binary relations) need not to be an approximable mapping. Nevertheless, we have the following result.

Proposition 7. *Let $\mathcal{R}$ be a left-linear, confluent CB-TRS and t be a term. Then $Narr(t)$ is an approximable mapping.*

We define the semantic function $CNarr^{\infty} : \mathcal{T}(\Sigma_{\perp}, V) \to [\mathcal{T}^{\infty}(\mathcal{C}_{\perp})^{V} \to \mathcal{T}^{\infty}(\mathcal{C}_{\perp})]$ as follows: $CNarr^{\infty}(t) = \overline{Narr(t)}$, i.e., $CNarr^{\infty}(t)$ is the continuous mapping associated to the approximable mapping $Narr(t)$ which represents the narrowing derivations starting from t. This semantics is consistent w.r.t. rewriting.

Theorem 1. *Let $\mathcal{R}$ be a left-linear, confluent CB-TRS. For all $t \in \mathcal{T}(\Sigma_{\perp}, V)$, $\phi \in \mathcal{T}(\mathcal{C}_{\perp})^{V}$, $CNarr^{\infty}(t)\ \phi = CRew^{\infty}(\phi(t))$.*

Narrowing strategies. A narrowing strategy $\mathcal{N}$ is a restriction on the set of possible narrowing steps. Given a narrowing strategy $\mathcal{N}$ and a term t, we can consider the set $\mathit{NDeriv}_{\mathcal{N}}(t) \subseteq \mathit{NDeriv}(t)$ of derivations which start from t and conform to $\mathcal{N}$. By Proposition 5, each $A \in \mathit{NDeriv}_{\mathcal{N}}(t)$ defines an approximable mapping $Narr^{A}(t)$ which is obviously contained in $Narr(t)$. By Proposition 4, $\overline{Narr^{A}(t)} \sqsubseteq \overline{Narr(t)} = CNarr^{\infty}(t)$. Therefore, $\{\overline{Narr^{A}(t)} \mid A \in \mathit{NDeriv}_{\mathcal{N}}(t)\}$ is bounded by $CNarr^{\infty}(t)$. Since $[\mathcal{T}^{\infty}(\mathcal{C}_{\perp})^{V} \to \mathcal{T}^{\infty}(\mathcal{C}_{\perp})]$ is a domain, it is consistently complete, i.e., the *lub* of every bounded subset actually exists (Theorem 3.1.10 in [21]). Thus, for left-linear CB-TRSs, we fix

$$CNarr^{\infty}_{\mathcal{N}}(t) = \bigsqcup\{\overline{Narr^{A}(t)} \mid A \in \mathit{NDeriv}_{\mathcal{N}}(t)\}$$

to be the meaning of t when it is evaluated under the narrowing strategy $\mathcal{N}$. Clearly, for all narrowing strategies $\mathcal{N}$, $CNarr^{\infty}_{\mathcal{N}} \sqsubseteq CNarr^{\infty}$. Thus, $CNarr^{\infty}$ provides a semantic reference for narrowing strategies. Strategies that satisfy $CNarr^{\infty}_{\mathcal{N}} = CNarr^{\infty}$ can be thought of as correct strategies.

Remark 1. Narrowing is able to yield *the graph* of a function f by computing $CNarr^{\infty}(f(\overline{x}))$, where $x_1, \ldots, x_{ar(f)}$ are different variables. This gives an interesting perspective of narrowing as an operational mechanism which computes denotations of *functions* as a whole, rather than only values of particular function calls. A similar observation can be made for narrowing strategies.

3.3 Computational Interpretation of the Semantic Descriptions

Our semantic descriptions are intended to provide a clear computational interpretation of the semantic information. This is essential for defining accurate analyses by using the semantic description.

Proposition 8. *Let $\mathcal{R}$ be a confluent TRS, $t \in \mathcal{T}(\Sigma_{\perp}, V)$, and $\delta = CRew^{\infty}(t)$. If $\delta \in \mathcal{T}(\mathcal{C}, V)$, then $t \to^{*} \delta$.*

Concerning narrowing computations, we have the following result.

Proposition 9. *Let $\mathcal{R}$ be a left-linear, confluent CB-TRS. Let t be a term, $\varsigma \in \mathcal{T}(\mathcal{C}_{\perp})^{V}$, $m = CNarr^{\infty}(t)$, and $\delta = m(\varsigma)$.*

1. *If $\delta \in \mathcal{T}(\mathcal{C}_\perp)$, there exists a narrowing derivation $\langle \varepsilon, t\rangle \leadsto^* \langle \sigma, s\rangle$ such that $\phi \circ \sigma = \varsigma$ and $\delta = (\!|\phi(s)|\!)_{FL}$.*
2. *For every narrowing derivation $\langle \varepsilon, t\rangle \leadsto^* \langle \sigma, s\rangle$ such that $\phi \circ \sigma = \varsigma$, it is $(\!|\phi(s)|\!)_{FL} \sqsubseteq \delta$.*
3. *If $\delta \in \mathcal{T}(\mathcal{C})$, then there exists a narrowing derivation $\langle \varepsilon, t\rangle \leadsto^* \langle \sigma, s\rangle$ such that $s \in \mathcal{T}(\mathcal{C}, V)$, $\phi \circ \sigma = \varsigma$, and $\delta = \phi(s)$.*

We are able to refine the computational information couched by the narrowing semantics by introducing a small modification on it.

Definition 4. *Given a term $t \in \mathcal{T}(\Sigma_\perp, V)$, and a narrowing derivation*

$$A : \langle \varepsilon, t\rangle = \langle \sigma_0, t_0\rangle \leadsto \langle \sigma_1, t_1\rangle \leadsto \cdots \leadsto \langle \sigma_{n-1}, t_{n-1}\rangle \leadsto \langle \sigma_n, t_n\rangle$$

we define $BNarr^A(t) = \cup_{0 \le i \le n} BNarr_i^A(t)$ where:

$$BNarr_i^A(t) = \{\langle \varsigma, \delta\rangle \mid (\!|\sigma_i|\!)_{FL} \sqsubseteq \varsigma \wedge \delta \sqsubseteq (\!|t_i|\!)_{FL}\}$$

Proposition 10. *Let $\mathcal{R}$ be a TRS, t be a term and A be a narrowing derivation starting from t. Then $BNarr^A(t)$ is an approximable mapping.*

If we define $BNarr(t) = \bigcup_{A \in NDeriv(t)} BNarr^A(t)$, we have the following result.

Proposition 11. *Let $\mathcal{R}$ be a left-linear, confluent CB-TRS and t be a term. Then $BNarr(t)$ is an approximable mapping.*

The basic description $BNarr^\infty(t) = \overline{BNarr(t)}$ is closer to the computational mechanism of narrowing. The following proposition formalizes this claim.

Proposition 12. *Let $\mathcal{R}$ be a left-linear, confluent CB-TRS, t be a term, $\varsigma \in \mathcal{T}(\mathcal{C}_\perp)^V$, $m = BNarr^\infty(t)$, and $\delta = m(\varsigma)$.*

1. *If $\delta \in \mathcal{T}(\mathcal{C}_\perp)$, there exists a narrowing derivation $\langle \varepsilon, t\rangle \leadsto^* \langle \sigma, s\rangle$ such that $\phi \circ \sigma = \varsigma$ and $\delta = (\!|s|\!)_{FL}$.*
2. *For every narrowing derivation $\langle \varepsilon, t\rangle \leadsto^* \langle \sigma, s\rangle$ such that $(\!|\sigma|\!)_{FL} \sqsubseteq \varsigma$, it is $(\!|s|\!)_{FL} \sqsubseteq \delta$.*

Proposition 13. *Let $\mathcal{R}$ be a left-linear, confluent CB-TRS, t be a term, and $m = BNarr^\infty(t)$. If $\langle \varepsilon, t\rangle \leadsto^* \langle \sigma, \delta\rangle$ and $\delta \in \mathcal{T}(\mathcal{C})$, then $m((\!|\sigma|\!)_{FL}) = \delta$.*

Since each $BNarr_i^A(t)$ is a special case of $Narr_i^A(t)$, by Proposition 11 and Proposition 4, $BNarr^\infty(t) \sqsubseteq CNarr^\infty(t)$.

4 A Semantics-Based Analysis Framework

Domain theory provides a framework for formulating properties of programs and discussing about them [2, 20]: A property π of a program $\mathcal{P}$ whose denotation $[\![\mathcal{P}]\!]$ is taken from a domain D (i.e., $[\![\mathcal{P}]\!] \in D$) can be identified with a predicate $\pi : D \to \mathbf{2}$, where $\mathbf{2}$ is the two point domain $\mathbf{2} = \{\perp, \top\}$ ordered by $\perp \sqsubseteq \top$

(where $\perp$ can be thought of as false and $\top$ as true). A program $\mathcal{P}$ satisfies π if $\pi([\![\mathcal{P}]\!]) = \top$ (alternatively, if $[\![\mathcal{P}]\!] \in \pi^{-1}(\top)$). As usual in domain theory, we require continuity of π for achieving computability (or *observability*, see [22]). The set $[D \to \mathbf{2}]$ of observable properties is (isomorphic to) the family of open sets of the *Scott's topology* associated to D [2]. A *topology* is a pair (X, τ) where X is a set and $\tau \subseteq \mathcal{P}(X)$ is a family of subsets of X (called the *open* sets) such that [21]: $X, \emptyset \in \tau$; if $U, V \in \tau$, then $U \cap V \in \tau$; and if $U_i \in \tau$ for $i \in I$, then $\bigcup_{i \in I} U_i \in \tau$. The Scott's topology associated to a domain D is given by the set of upward closed subsets $U \subseteq D$ such that, whenever $A \subseteq D$ is directed and $\bigsqcup A \in U$, then $\exists x \in A.x \in U$ [21].

The family τ of open sets of a given topology (X, τ) ordered by inclusion is a *complete lattice*. The top element of the lattice is X. Note that, when considering the Scott's topology (D, τ_D) of a domain D, the open set D denotes a trivial property which every program satisfies; $\emptyset$, the least element of lattice τ_D, denotes the 'impossible' property, which no program satisfies.

4.1 Analysis of Functional Logic Programs

A program analysis consists in the definition of a continuous function $\alpha : D \to A$ between topologic spaces (D, τ_D) and (A, τ_A) which expresses concrete and abstract properties, respectively. By the topological definition of continuity, each open set $V \in \tau_A$ maps to an open set $U \in \tau_D$ via α^{-1}, i.e., $\alpha^{-1} : \tau_A \to \tau_D$ is a mapping from abstract properties (open sets of τ_A) to concrete properties (open sets of τ_D). It is easy to see that $(D, \{\alpha^{-1}(V) \mid V \in \tau_A\})$ is a subtopology of D (i.e., $\{\alpha^{-1}(V) \mid V \in \tau_A\} \subseteq \tau_D$). Therefore, each analysis distinguishes a subset of properties of D which is itself a topology. For instance, the Scott's topology of $\mathbf{2}$ is given by $\tau_{\mathbf{2}} = \{\emptyset, \{\top\}, \mathbf{2}\}$. Such a topology permits to express only one non-trivial property, namely, the one which corresponds to the open set $\{\top\}$.

In functional logic languages, the semantic domain under observation is $[D^V \to D]$. Observable properties of functional logic programs are open sets of its Scott's topology. Approximations to such properties can be obtained by abstracting $[D^V \to D]$ into a suitable abstract domain (see below).

Every continuous function $f : D \to E$ maps observable properties of the codomain E into observable properties of D (by $f^{-1} : \tau_E \to \tau_D$). In particular, elements of $[D^V \to D]$, i.e., denotations of functional logic programs, map properties of D (we call them 'functional' properties) into properties of D^V ('logic' properties). This provides an additional, interesting analytic perspective: By rephrasing Dybjer [7], we can computationally interpret this correspondence as establishing the extent that a 'logic property' (concerning valuations) needs to be ensured to guarantee a property of its functional part (computed value). There is a simple way to obtain an abstraction of the logic part D^V of $[D^V \to D]$ from an abstraction of its functional part D.

Definition 5. *Let D, V, A be sets. Let $\alpha_F : D \to A$ be a mapping. Then, $\alpha_L : D^V \to A^V$ given by $\alpha_L(\phi) = \alpha_F \circ \phi$, for all $\phi \in D^V$, is called the logic abstraction induced by α_F.*

If $\alpha_F : D \to A$ is strict (surjective, continuous), then α_L is strict (surjective, continuous). Whenever α_F is a *continuous* mapping from a domain D to $\mathbf{2}$, α_F expresses, in fact, a single observable property $\alpha^{-1}(\{\top\})$ of D. We can thought of α_F as a *functional* property. Thus, Definition 5 associates an abstraction α_L of D^V to a given property identified by α_F. Thus, each functional property induces a related *set* of logic properties which is a *subtopology* of τ_{D^V}. In Section 4.3 we show that groundness (a logic property), is induced by the functional property of termination.

4.2 Approximation of Functions

Abstractions $\alpha_D : D \to A$ and $\alpha_E : E \to B$ (A and B being algebraic lattices), induce *safety* and *liveness* abstractions $\alpha^S_{D\to E}, \alpha^L_{D\to E} : (D \to E) \to (A \to B)$, of continuous mappings by [1]

$$\alpha^S_{D\to E}(f)(d) = \sqcup\{(\alpha_E \circ f)(d') \mid \alpha_D(d') \sqsubseteq d\}, \text{ and}$$
$$\alpha^L_{D\to E}(f)(d) = \sqcap\{(\alpha_E \circ f)(d') \mid \alpha_D(d') \sqsupseteq d\}$$

where the following correctness result holds:

Theorem 2 (The semi-homomorphism property [1]**).** *Let* $f : D \to E$, $f^S = \alpha^S_{D\to E}(f)$, *and* $f^L = \alpha^L_{D\to E}(f)$. *Then,* $f^L \circ \alpha_D \sqsubseteq \alpha_E \circ f \sqsubseteq f^S \circ \alpha_D$.

Consider an abstraction $\alpha_E : E \to \mathbf{2}$ which can be thought of as a *property* of elements of the codomain E of $f : D \to E$. For analytic purposes, the correctness condition $f^S \circ \alpha_D \sqsupseteq \alpha_E \circ f$ ensures that, for all $x \in D$, whenever the abstract computation $f^S(\alpha_D(x))$ yields $\bot$, the concrete computation $f(x)$ does *not* satisfy the property α_E, i.e., $\alpha_E(f(x)) = \bot$. On the other hand, the correctness condition $f^L \circ \alpha_D \sqsubseteq \alpha_E \circ f$ ensures that, whenever $f^L(\alpha_D(x))$ yields $\top$, the concrete computation $f(x)$ actually satisfies α_E, i.e., $\alpha_E(f(x)) = \top$. We use this computational interpretation later.

4.3 Termination Analysis and Groundness Analysis

The functional structure of the semantic domain of *ngv*'s reveals connections between apparently unrelated analyses. Consider $h_t : \mathcal{T}^\infty(\mathcal{C}_\bot) \to \mathbf{2}$ defined by

$$h_t(\delta) = \begin{cases} \top \text{ if } \delta \in \mathcal{T}(\mathcal{C}) \\ \bot \text{ otherwise} \end{cases}$$

and let $h_g : \mathcal{T}^\infty(\mathcal{C}_\bot)^V \to \mathbf{2}^V$ be the logic abstraction induced by h_t. Note that both h_t and h_g are strict and continuous. Abstractions h_t and h_g express the observable properties of termination and groundness, respectively: Recall that the only nontrivial open set of the Scott's topology of $\mathbf{2}$ is $\{\top\}$. By continuity of h_t, $h_t^{-1}(\{\top\})$ is the (open) set of finite, totally defined values which actually corresponds to terminating successful evaluations[3]. On the other hand, each open set of $\mathbf{2}^V$ is (isomorphic to) an upward closed collection of sets of variables ordered by inclusion. In this case, $h_g^{-1}(F)$ for a given open set F is a set of substitutions whose bindings for variables belonging to $X \in F$ are ground. This formally relates groundness and termination: groundness is the 'logic' property which corresponds to the 'functional' property of termination. In fact, $\mathbf{2}^V$ is the standard abstract domain for *groundness* analysis in logic programming.

4.4 Using Semantic Information for Improving the Evaluation

Groundness information can be used to improve the narrowing evaluation of a term $t = C[t_1, \ldots, t_n]$: if we know that every successful evaluation of t_i grounds the variables of t_j, for some $1 \leq i, j \leq n$, $i \neq j$, then it is sensible to evaluate t by first narrowing t_i (up to a value) and next evaluating t'_j (i.e., t_j after instantiating its variables using the bindings created by the evaluation of t_i) by *rewriting* because, after evaluating t_i, we know that t'_j is ground and we do not need to provide code for unification, instantiation of other variables, etc.

Example 4. Consider the following TRS:

```
0+x          → x              if(true,x,y)  → x
s(x)+y       → s(x+y)         if(false,x,y) → y

even(0)      → true           even(s(s(x))) → even(x)
even(s(0))   → false
```

For an initial (conditional) expression "`if even(x) then x+x else s(x+x)`" (we use the more familiar notation `if then else` for `if` expressions), it is clear that `x` becomes ground after every successful narrowing evaluation of the condition `even(x)`. Thus, we can evaluate `x+x` by rewriting instead of narrowing.

Additionally, we need to ensure that the evaluation of t_i is safe under the context C (i.e., that failing evaluations of t_i do not prevent the evaluation of t). Eventually, we should also ensure that the complete evaluation of t'_j is safe. Strictness information can be helpful here: if the (normalizing) narrowing strategy is not able to obtain any value, this means that the whole expression does not have a value. However, we should only use non-contextual strictness analyses (like Mycroft's [17] is). In this way, we ensure that the strict character of an argument is not altered after a possible instantiation of its surrounding context.

In order to ensure that *every* successful narrowing derivation grounds a given variable $x \in \mathcal{V}ar(t)$, we use the safety abstraction $m^S \in \mathbf{2}^V \to \mathbf{2}$ of $m = BNarr^\infty(t)$ (based on h_t and h_g).

[3] h_t and Mycroft's abstraction: $halt(d) = \begin{cases} \top \text{ if } d \neq \bot \\ \bot \text{ if } d = \bot \end{cases}$ for termination analysis [17] are similar. However, $halt$ only expresses termination if $\mathcal{C}$ only contains constant symbols. It is easy to see that, in this case, $h_t = halt$.

Example 5. (Continuing Example 4) For $t = \texttt{even(x)}$, we have:

$$\begin{array}{lll} BNarr^{\infty}(t) = \{ & \{\texttt{x} \mapsto \bot\} \mapsto \bot, & \{\texttt{x} \mapsto \texttt{0}\} \mapsto \texttt{true}, \\ & \{\texttt{x} \mapsto \texttt{s(}\bot\texttt{)}\} \mapsto \bot, & \{\texttt{x} \mapsto \texttt{s(0)}\} \mapsto \texttt{false}, \\ & \{\texttt{x} \mapsto \texttt{s(s(}\bot\texttt{))}\} \mapsto \bot, & \{\texttt{x} \mapsto \texttt{s(s(0))}\} \mapsto \texttt{true}, \ldots \} \end{array}$$

In general, if we can prove that, for all abstract substitutions $\phi^{\#} \in \mathbf{2}^V$ with $\phi^{\#}(x) = \bot$, it is $m^S(\phi^{\#}) = \bot$, then we can ensure that x is grounded in every successful derivation from t. To see this point, consider a successful derivation $\langle \varepsilon, t \rangle \rightsquigarrow^* \langle \sigma, \delta \rangle$ such that $\delta \in \mathcal{T}(\mathcal{C})$ and $\sigma(x) \notin \mathcal{T}(\mathcal{C})$, i.e., x is not grounded. By Proposition 13, $m((\!|\sigma|\!)_{FL}) = \delta$. By definition of m^S, $m^S(h_g((\!|\sigma|\!)_{FL})) = \top$. Since $(\!|\sigma|\!)_{FL}(x) \notin \mathcal{T}(\mathcal{C})$, we have $h_g((\!|\sigma|\!)_{FL})(x) = h_t((\!|\sigma|\!)_{FL}(x)) = \bot$, thus contradicting (a case of) our initial assumption, $m^S(h_g((\!|\sigma|\!)_{FL})) = \bot$.

Example 6. (Continuing Example 5) For $t = \texttt{even(x)}$, we have $m^S = \{\{x \mapsto \bot\} \mapsto \bot, \{x \mapsto \top\} \mapsto \top\}$. Thus, x is grounded in every successful derivation of even(x).

The previous considerations make clear that the semantic dependency expressed by the *ngv*'s has the corresponding translation for the analysis questions.

5 Related Work and Concluding Remarks

The idea of giving denotational descriptions of different operational frameworks is not new. For instance, [5] assigns different *fixpoint* semantics for a program under either call-by-name or call-by-value strategies. This shows that, in some sense, the semantic descriptions also (silently) assume some underlying operational approach (usually, call-by-name like).

In [18], the notion of *ngv* as the semantic object that a narrowing computation should compute was already introduced. It was also noted that narrowing only computes a *representation* of the object, not the object itself. However, it was not clearly explained how this connection can be done.

In [16], domains are used to give semantics to the functional logic language BABEL. However, the style of the presentation is model-theoretic: all symbols take meaning from a given interpretation and the connection between the declarative and operational semantics (lazy narrowing) are given by means of the usual completeness/correctness results. The semantic domain is different from ours because valuations are just a parameter of the semantic functions rather than a component of the domain. Thus, the *Herbrand domain* $\mathcal{T}^{\infty}(\mathcal{C}_{\bot})$ is the semantic domain in [16].

The semantic approach in [9] is much more general than [16] (covering non-deterministic computations), but the style of the presentation is model-theoretic too. The basic semantic domain is also different from ours: no functional domain for denotations is used and, in fact, bounded completeness, which is essential in our setting to deal with the functional construction and with narrowing strategies, is not required in [9].

In [23], a denotational description of a particular narrowing strategy (the needed narrowing strategy [4]) is given. The semantics is nicely applied to demandedness analysis but nothing has been said about how to use it for more general analysis problems. This question is important since the notion of demandedness pattern is essential for the definition of the semantics itself.

We have presented a domain-theoretic approach for describing the semantics of integrated functional logic languages based on narrowing. Our semantics is parameterized by the narrowing strategy which is used by the language. The semantics is not 'model-theoretic' in the sense that we let within the operational mechanism (the narrowing strategy) to establish the 'real' meaning of the functions defined by the program rules. In this way, we are able to include more operational information into the semantic description. As far as we know, previous works have not explicitly considered different arbitrary strategies for parameterizing the semantics of functional logic languages, that is, the operational-oriented denotational description formalized in this work is novel in the literature of the area.

Another interesting point of our work is its applicability to the analysis of functional logic programs. Since we use a functional domain (the domain of *non-ground-values*), we are able to associate a denotation to a term with variables. Thus, narrowing is reformulated as an evaluation mechanism which computes the denotation of the input expression. This was already suggested by Reddy [18] but it is only formally established in this paper by using approximable mappings. Thanks to this perspective, we can easily use the standard frameworks for program analysis based on the denotational description of programs. In other words, the approximation of the domain of non-ground values enables the analysis of functional logic programs. Our description also reveals unexplored connections between purely functional and logic properties. These connections suggest that, within the functional logic setting, we have ascertained a kind of 'duality' between purely functional and purely logic properties. As far as we know, this had not been established before.

Future work includes a more detailed study about how to use this semantics to develop practical methods for the analysis of functional logic programs. Another interesting task is to extend this semantics to more general computation models for declarative languages [12].

References

1. S. Abramsky. Abstract Interpretation, Logical Relations, and Kan Extensions. *Journal of Logic and Computation* 1(1):5-40, 1990.
2. S. Abramsky. Domain Theory in Logic Form. *Annals of Pure and Applied Logic* 51:1-77, 1991.
3. M. Alpuente, M. Falaschi, and F. Manzo. Analyses of Unsatisfiability for Equational Logic Programming. *Journal of Logic Programming*, 22(3):221-252, 1995.
4. S. Antoy, R. Echahed and M. Hanus. A needed narrowing strategy. In *Conference Record of the ACM Symposium on Principles of Programming Languages, POPL'94*, pages 268-279. ACM Press, 1994.

5. J.W. de Bakker. Least Fixed Points Revisited. *Theoretical Computer Science*, 2:155-181, 1976.
6. F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, 1998.
7. P. Dybjer. Inverse Image Analysis Generalises Strictness Analysis. *Information and Computation* 90:194-216, 1991.
8. J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial Algebra Semantics and Continuous Algebras. *Journal of the ACM* 24(1):68-95, 1977.
9. J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. *Journal of Logic Programming* 40(1):47-87, 1999.
10. C.A. Gunter. Semantics of Programming Languages. The MIT Press, Cambridge, MA, 1992.
11. M. Hanus. Towards the Global Optimization of Functional Logic Programs. In P.A. Fritzson, editor, *Proc. 5th International Conference on Compiler Construction, CC'94*. LNCS 786:68-82, Springer Verlag, Berlin, 1994.
12. M. Hanus. A Unified Computation Model for Functional and Logic Programming. In *Conference Record of the 24th Symposium on Principles of Programming Languages POPL'97*, pages 80-93, ACM Press, 1997.
13. M. Hanus and F. Zartmann. Mode Analysis of Functional Logic Programs. In B. Le Charlier, editor, *Proc. of 1st International Static Analysis Symposium, SAS'94*. LNCS 864:26-42, Springer Verlag, Berlin, 1994.
14. J.-M. Hullot. Canonical forms and unification. In *Proc. 5th Conference on Automated Deduction, CADE'80*, LNCS:318-334, Springer-Verlag, Berlin, 1980.
15. J.J. Moreno-Navarro, H. Kuchen, J. Mariño, S. Winkler and W. Hans. Efficient Lazy Narrowing using Demandedness Analysis. In M. Bruynooghe and J. Penjam, editors, *Proc. of 5th International Symposium on Programming Language Implementation and Logic Programming, PLILP'93*. LNCS 714:167-183, Springer-Verlag, Berlin, 1993.
16. J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic programming with functions and predicates: the language BABEL. *Journal of Logic Programming*, 12:191-223, 1992.
17. A. Mycroft. The theory and practice of transforming call-by-need into call-by-value. In B. Robinet, editor, *Proc. of the 4th International Symposium on Programming*, LNCS 83:269-281, Springer-Verlag, Berlin, 1980.
18. U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. of IEEE International Symposium on Logic Programming*, pages 138-151, 1985.
19. D. Scott. Domains for Denotational Semantics. In M. Nielsen and E.M. Schmidt, editors, *Proc. of 9th International Colloquium on Automata, Languages and Programming, ICALP'82*, LNCS 140:577-613, Springer-Verlag, Berlin, 1982.
20. D. Scott. Lectures on a mathematical theory of computation. Monograph PRG-19. Computing Laboratory, Oxford University, 1981.
21. V. Stoltenberg-Hansen, I. Lindström, and E.R. Griffor. Mathematical Theory of Domains. Cambridge University Press, 1994.
22. S. Vickers. Topology via Logic. Cambridge University Press, 1989.
23. F. Zartmann. Denotational Abstract Interpretation of Functional Logic Programs. In P. Van Hentenryck, editor, *Proc. of the 4th International Static Analysis Symposium, SAS'97*, LNCS 1302:141-159, Springer-Verlag, Berlin, 1997.