A Note on Syntactic Annotations for Narrowing[*]

María Alpuente Santiago Escobar Salvador Lucas
DSIC, Universidad Politécnica de Valencia (Spain)
{alpuente,sescobar,slucas}@dsic.upv.es

Abstract

In this paper, we discuss syntactic annotations for narrowing and propose a unified model that exploits the synergy of combining previous frameworks.

1 Introduction

Computational systems with a reduction-based operational principle, such as rule-based programming languages and equational theorem provers, typically incorporate a predefined evaluation strategy that cuts off the search space but can also modify important program properties such as termination, normalization, and optimality.

Syntactic strategy annotations are commonly used to give the programmer some control over the considered strategy and properties. Functional languages such as Clean [19], Haskell [13], and Lisp [18] encode the strategy as *strictness annotations* to become "more eager", thus more efficient; equational languages such as Maude [2, 3], OBJ2 [7], OBJ3 [9], and CafeOBJ [8] implement them as *replacement restrictions* to become "more lazy", thus improving termination; and functional logic languages such as Curry [12] provide *instantiation restrictions* to control coroutine via logic variables as in logic programming.

Context-sensitive rewriting (*CSR*) [15, 17] provides a basic framework to reason about strategy annotations as replacement (or reduction) restrictions. In *CSR*, a replacement map μ indicates the arguments $\mu(f) \subseteq \{1,\ldots,ar(f)\}$ of function symbols f on which reductions are allowed. These replacement restrictions have proved useful for making programs more efficient (by reducing the number of attempted matchings) and for dealing with infinite data structures (by imposing some controlled laziness; for instance in languages such as Maude or CafeOBJ). They have also turned essential as a tool for achieving termination of programs [4, 16] and for coding complex systems into rewrite rules [2].

Example 1 Consider the following term rewriting system (TRS) $\mathcal{R}$ borrowed from [17]:

```
first(0,Z) → nil
first(s(X),cons(Y,Z)) → cons(Y,first(X,Z))
from(X) → cons(X,from(s(X)))
```

In order to ensure the termination of the rewriting relation for $\mathcal{R}$, we have to restrict reductions on the second argument of the list constructor `cons`. With $\mu(\texttt{cons}) = \{1\}$ and $\mu(f) = \{1,\ldots,ar(f)\}$ for all other symbols f, we forbid replacements on the second argument of `cons` while reductions are still allowed below the arguments of all other symbols (and also below the first argument of `cons`). For instance, although $t_1 = \texttt{first(s(0),from(0))}$ has an infinite reduction sequence by using ordinary rewriting[1]

$$\texttt{first(s(0),}\underline{\texttt{from(0)}}\texttt{)}$$
$$\to \texttt{first(s(0),cons(0,}\underline{\texttt{from(s(0))}}\texttt{))}$$
$$\to \cdots$$

it is possible to prove that *CSR* for

[*]This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN 2004-7943-C04-02, the Generalitat Valenciana under grant GV03/25, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project.

[1]We underline the subterm reduced at each rewriting step (the selected *redex*).

$\mathcal{R}$ above is terminating (e.g., an automatic proof can be obtained with MU-TERM, see `http://www.dsic.upv.es/~slucas/csr/termination/muterm`). ■

Context-sensitive narrowing (*CSN*) [15] is the lifting of context-sensitive rewriting to narrowing. In *CSN*, the replacement map μ indicates the arguments $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ of function symbol f on which narrowing steps are allowed.

Example 2 Consider the program $\mathcal{R}$ and the replacement map μ of Example 1. For the term t_2 = `first(X,from(X))`, only three context-sensitive narrowing sequences can be proved:

$$\begin{array}{l}
\underline{\texttt{first(X,from(X))}}\\
\leadsto_{[\texttt{X}\mapsto\texttt{0}]}\ \texttt{nil}
\end{array}$$

$$\begin{array}{l}
\texttt{first(X,}\underline{\texttt{from(X)}}\texttt{)}\\
\leadsto_{id}\ \underline{\texttt{first(X,cons(X,from(s(X))))}}\\
\leadsto_{[\texttt{X}\mapsto\texttt{0}]}\ \texttt{nil}
\end{array}$$

$$\begin{array}{l}
\texttt{first(X,}\underline{\texttt{from(X)}}\texttt{)}\\
\leadsto_{id}\ \underline{\texttt{first(X,cons(X,from(s(X))))}}\\
\leadsto_{[\texttt{X}\mapsto\texttt{s(X')}]}\ \texttt{cons(s(X'),}\\
\qquad\qquad\qquad\texttt{first(X',from(s(s(X')))))}
\end{array}$$

However, t_2 has an infinite number of (possibly infinite) ordinary narrowing sequences, computing substitutions [`X` $\mapsto$ `0`], [`X` $\mapsto$ `s(0)`], [`X` $\mapsto$ `s(s(0))`], etc. ■

On the other hand, in [11] Hanus proposed a *unified computational model* for functional logic languages that considers *instantiation restrictions* in order to combine narrowing with the so-called *residuation* principle [14] used by some functional logic languages such as Escher or Oz. In Hanus' terminology, this is expressed by means of *rigid* or *flexible* annotations which can be given to the inductive positions of the definitional trees which are used to guide the program execution in this model. In the language Curry *rigid* or *flexible* annotations are directly given to the defined function symbols instead. Informally, a *rigid* annotation for f specifies that no instantiation for the variables of a function call rooted by f are allowed in a reduction step, i.e., reduction of the term is delayed until it is properly instantiated to enable a rewriting step. Function symbols with a functional or equational meaning are typically annotated as *rigid* whereas function symbols with an intended logic meaning are marked as *flexible*.

Example 3 Consider again the program of Example 1 and the following rules:

$$\begin{array}{l}
\texttt{leqOne(0)} \to \texttt{True}\\
\texttt{leqOne(s(0))} \to \texttt{True}
\end{array}$$

Assume that `first` and `from` are marked as *rigid*, whereas `leqOne` is marked as *flexible*. We also consider an auxiliary constructor symbol[2] `and`. Now, consider the term t_3 = `and(leqOne(X),first(X,from(X)))`. The first argument of `first` is a variable, `X`, thus narrowing of the subterm `first(X,from(X))` will be delayed until `X` is properly instantiated, since `first` and `from` are marked as rigid. This can be caused by the narrowing of `leqOne(X)`, as shown in the two following sequences:

$$\begin{array}{l}
\texttt{and(}\underline{\texttt{leqOne(X)}}\texttt{,first(X,from(X)))}\\
\leadsto_{[\texttt{X}\mapsto\texttt{0}]}\ \texttt{and(True,first(0,from(0)))}\\
\leadsto_{id}\ \texttt{and(True,nil)}
\end{array}$$

$$\begin{array}{l}
\texttt{and(}\underline{\texttt{leqOne(X)}}\texttt{,first(X,from(X)))}\\
\leadsto_{[\texttt{X}\mapsto\texttt{s(0)}]}\ \texttt{and(True,first(s(0),}\underline{\texttt{from(s(0))}}\texttt{))}\\
\leadsto_{id}\ \texttt{and(True,}\\
\qquad\quad\underline{\texttt{first(s(0),}}\\
\qquad\qquad\quad\underline{\texttt{cons(s(0),from(s(s(0)))))}}\texttt{)}\\
\leadsto_{id}\ \texttt{and(True,}\\
\qquad\quad\texttt{cons(s(0),}\underline{\texttt{first(0,from(s(s(0))))}}\texttt{))}\\
\leadsto_{id}\ \texttt{and(True,cons(s(0),nil))}
\end{array}$$

However, there are also some infinite (undesired) narrowing sequences from t_3:

$$\begin{array}{l}
\texttt{and(leqOne(X),first(X,}\underline{\texttt{from(X)}}\texttt{))}\\
\leadsto_{id}\ \texttt{and(leqOne(X),}\\
\qquad\quad\texttt{first(X,cons(X,}\underline{\texttt{from(s(X))}}\texttt{)))}\\
\leadsto_{id}\ \cdots
\end{array}$$

which are typically avoided by using a lazy narrowing strategy, as in Curry. Lazy strategies are outside the scope of this paper; instead we consider syntactic strategy annotations which are often simpler. ■

One important motivation for this paper is to endow CafeOBJ or Maude with narrowing ca-

[2]For the sake of simplicity, in this paper we restrict ourselves to one-sorted signatures.

pabilities controlled by syntactic annotations. The inclusion of narrowing in the computational model of Maude is one of the expected forthcoming features of the language (see [2, Table 1]). The previous examples illustrate the usefulness of each annotation model (restricting subterm replacements or restricting the instantiation of variables). The key idea of our approach is to formalize a unified framework which uses mappings $\mu, \nu \in \mathcal{F} \to \mathcal{P}(\mathbb{N})$ to specify the replacement or instantiation restrictions, respectively.

The impact of the instantiation restrictions in the computational behaviour of functional-logic programs is illustrated in the following example.

Example 4 Consider the TRS $\mathcal{R}$ of Example 1 together with the following rules describing bank accounts. Accounts are simply modeled by means of two simple operations `dep` (deposit) and `wit` (withdrawal):

```
account(N,nil) → N
account(N,cons(dep(M),XS))
  → account(add(N,M),XS)
account(N,cons(wit(M),XS))
  → account(sub(N,M),XS)
add(0,Y) → Y
add(s(X),Y) → s(add(X,Y))
sub(X,0) → X
sub(s(X),s(Y)) → sub(X,Y)
client1(cons(dep(50),
        cons(wit(20), nil))) → True
```

Consider the input expression t_4 = `and(account(0,XS),client1(XS))` which we want to narrow, and assume the map μ of Example 1: $\mu(\texttt{cons}) = \{1\}$ and $\mu(f) = \{1, \ldots, ar(f)\}$ otherwise. Assume also that instantiations are not allowed under the arguments of symbols `add` and `sub`, i.e., $\nu(\texttt{add}) = \nu(\texttt{sub}) = \varnothing$, since they play the role of 'built-in' arithmetic operations which require completely evaluated arguments. Then, it is very handy to have a mechanism to forbid instantiations on the second argument of symbol `account`, i.e., $\nu(\texttt{account}) = \{1\}$, since `account` typically relies on the operations generated by the clients; this is a natural choice which avoids infinite narrowing sequences like this one:

$$\begin{array}{l}
\underline{\texttt{account(N,XS)}} \\
\leadsto_{[\texttt{XS}\mapsto\texttt{cons(dep(M),XS')}]}\ \underline{\texttt{account(add(N,M),XS')}} \\
\leadsto_{[\texttt{XS'}\mapsto\texttt{cons(dep(M'),XS'')}]} \\
\qquad\qquad \underline{\texttt{account(add(add(N,M),M'),XS'')}} \\
\leadsto \cdots
\end{array}$$

Note that we are able to restrict instantiations in a concrete argument of a particular symbol, rather than marking a plain function symbol as in Curry. ■

The new model subsumes previous approaches, including *CSR*, *CSN*, and the rigid/flexible annotations as implemented in Curry. We believe that this setting can be specially well-suited for Maude, since it could be smoothly integrated into the current operational scheme. On the other hand, the concurrent execution of objects, which is a strong point of rewriting in Maude, can be supported using a more general model of synchronization by logical variables and narrowing, as illustrated by the example above.

2 Preliminaries

Throughout the paper, $\mathcal{X}$ denotes a countable set of variables and $\mathcal{F}$ denotes a signature, i.e., a set of function symbols, each having a fixed arity given by a mapping $ar : \mathcal{F} \to \mathbb{N}$. Terms are viewed as labelled trees in the usual way. The set of terms built from $\mathcal{F}$ and $\mathcal{X}$ is $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The set of terms built only from $\mathcal{F}$ is $\mathcal{T}(\mathcal{F})$. A substitution is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ for some variables $\mathcal{D}om(\sigma)$. A substitution is usually denoted as $[X_1 \mapsto t_1, \ldots, X_k \mapsto t_k]$, and in such case $\mathcal{D}om(\sigma) = \{X_1, \ldots, X_k\}$, or the identity substitution id s.t. $\forall X \in \mathcal{X} : id(X) = X$. A substitution is homomorphically extended to terms. A term is said to be linear if it has no multiple occurrences of a single variable.

Positions $p, q, \ldots$ of a term t are represented by sequences of positive natural numbers used to address subterms of t. The set of positions of a term t is denoted by $\mathcal{P}os(t)$, $\mathcal{P}os_{\mathcal{F}}(t)$ are the positions of non-variable symbols, and $\mathcal{P}os_{\mathcal{X}}(t)$ are the positions of variables. We denote the root position (empty sequence) by Λ. Given positions p and q, we denote its concatenation as $p.q$. If p is a position, and Q is a set of positions, $p.Q = \{p.q \mid q \in Q\}$. The subterm at position p of t is denoted as $t|_p$, and $t[s]_p$ is the term t with the subterm at position p replaced by s. The symbol labelling the root of t is denoted as $root(t)$.

A rewrite rule is an ordered pair (l, r), written $l \to r$, with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. The left-hand side (*lhs*) of the rule is l and r is its right-hand side (*rhs*). A TRS is a pair $\mathcal{R} = (\mathcal{F}, R)$ where R is a set of rewrite rules. A TRS $\mathcal{R}$ is left-linear if for all $l \to r \in \mathcal{R}$, l is a linear term. Given $\mathcal{R} = (\mathcal{F}, R)$, we consider $\mathcal{F}$ as the disjoint union $\Sigma = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors* and symbols $f \in \mathcal{D}$, called *functions*, where $\mathcal{D} = \{root(l) \mid l \to r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$. Then, $\mathcal{T}(\mathcal{C}, \mathcal{X})$ (resp. is the set of constructor terms. A TRS $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ is a constructor system (CS) if for all $f(l_1, \ldots, l_k) \to r \in R$, $l_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, for $1 \leq i \leq k$.

A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrites to s (at position p), written $t \overset{p}{\to}_{\mathcal{R}} s$ ($t \to_{\mathcal{R}} s$ or $t \to s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some rule $l \to r \in R$, $p \in \mathcal{P}os_{\mathcal{F}}(t)$ and substitution σ. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ narrows to s (at position p with substitution θ), written $t \overset{[p,\theta]}{\leadsto}_{\mathcal{R}} s$ ($t \leadsto_{\mathcal{R}} s$, $t \overset{p}{\leadsto}_{\theta} s$ or $t \leadsto_{\theta} s$), if $\theta(t|_p) = \sigma(l)$ and $s = \theta(t)[\sigma(r)]_p$, for some (renamed apart) rule $l \to r \in R$, $p \in \mathcal{P}os_{\mathcal{F}}(t)$ and substitutions θ and σ. A narrowing derivation $t \leadsto^*_{\theta} s$ is such that either $t = s$ and $\theta = id$ or $t = t_1 \leadsto_{\theta_1} t_2 \leadsto_{\theta_2} \cdots t_n \leadsto_{\theta_n} t_{n+1} = s$ and $\theta = \theta_n \circ \cdots \circ \theta_1$.

2.1 Context-sensitive rewriting (*CSR*)

A mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ is a *replacement map* (or $\mathcal{F}$-map) if $\forall f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ [15]. The set of *μ-positions* $\mathcal{P}os^{\mu}(t)$ of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is: $\mathcal{P}os^{\mu}(t) = \{\Lambda\}$, if $t \in \mathcal{X}$, and $\mathcal{P}os^{\mu}(t) = \{\Lambda\} \cup \bigcup_{i \in \mu(root(t))} i.\mathcal{P}os^{\mu}(t|_i)$, if $t \notin \mathcal{X}$.

In *context-sensitive rewriting* (*CSR* [15]), we (only) contract redexes at replacing positions: t μ-rewrites to s, written $t \overset{p}{\hookrightarrow}_{\mu} s$, if $t \overset{p}{\to}_{\mathcal{R}} s$ and $p \in \mathcal{P}os^{\mu}(t)$, i.e., $\hookrightarrow_{\mu} \subseteq \to_{\mathcal{R}}$. The $\hookrightarrow_{\mu}$-normal forms are called μ-normal forms. Note that the μ-normal forms strictly include all normal forms of $\mathcal{R}$.

The canonical replacement map $\mu^{can}_{\mathcal{R}}$ is *the most restrictive replacement map ensuring that the non-variable subterms of the left-hand sides of the rules of $\mathcal{R}$ are replacing.* Note that $\mu^{can}_{\mathcal{R}}$ is easily obtained from $\mathcal{R}$: $\forall f \in \mathcal{F}$, $\forall i \in \{1, \ldots, ar(f)\}$, $i \in \mu^{can}_{\mathcal{R}}(f)$ iff $\exists l \to r \in \mathcal{R}, \exists p \in \mathcal{P}os_{\mathcal{F}}(l)$ s.t. $root(l|_p) = f$ and $p.i \in \mathcal{P}os_{\mathcal{F}}(l)$. We say that the map μ is less or equally restrictive than the map μ', denoted by $\mu' \sqsubseteq \mu$, if $\forall f \in \mathcal{F}$, $\mu'(f) \subseteq \mu(f)$.

2.2 Context-sensitive narrowing (*CSN*)

The lifting of context-sensitive rewriting to narrowing allows restrictions on narrowing steps. In *context-sensitive narrowing* (*CSN* [15]), we (only) narrow narroxes at replacing positions: t μ-narrows to s, written $t \overset{[p,\theta]}{\leadsto}_{\mu} s$ (or $t \overset{\theta}{\leadsto}_{\mu} s$), if $t \overset{[p,\theta]}{\leadsto}_{\mathcal{R}} s$ and $p \in \mathcal{P}os^{\mu}(t)$, i.e., $\leadsto_{\mu} \subseteq \leadsto_{\mathcal{R}}$ (similarly with $\leadsto^*_{\mu}$).

The canonical *evaluation* replacement map for a subset $\mathcal{B} \subseteq \mathcal{C}$, denoted by $\mu^{\mathcal{B}}_{\mathcal{R}}$, is a replacement map such that $\mu^{can}_{\mathcal{R}} \sqsubseteq \mu^{\mathcal{B}}_{\mathcal{R}}$ and for all $c \in \mathcal{B}$, $\mu^{\mathcal{B}}_{\mathcal{R}}(c) = \{1, \ldots, ar(c)\}$, i.e., it is a canonical map that doesn't restrict constructor symbols in the set $\mathcal{B}$. We write $\vartheta \leq \theta$ for substitutions ϑ and θ with $\mathcal{D}om(\theta) = \mathcal{D}om(\vartheta)$ if there exists substitution ϑ' such that for all $X \in \mathcal{D}om(\theta)$, $\vartheta'(\vartheta(X)) = \theta(X)$. A *normalized substitution* is a substitution θ such that $\theta(X)$ is a $\to$-normal form for any $X \in \mathcal{D}om(\theta)$. One of the most important properties of *CSN* is the following.

Theorem 1 (Completeness) [15] *Let $\mathcal{R}$ be a left-linear CS and μ be such that $\mu^{\mathcal{B}}_{\mathcal{R}} \sqsubseteq \mu$ for some $\mathcal{B} \subseteq \mathcal{C}$. Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \in \mathcal{T}(\mathcal{B}, \mathcal{X})$, and θ, ϑ be normalized substitutions. Then, $t \leadsto^*_{\theta} s$ iff $t \overset{\vartheta}{\leadsto}{}^*_{\mu} s$ and $\vartheta \leq \theta$.*

3 The unified annotation model

We consider the combination of two maps $\langle \mu, \nu \rangle$ which specify the reductions and instantiations that can be done. The mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ is a replacement map whereas the mapping $\nu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ is an instantiation restriction map. In this unified model, we contract only redexes at μ-

positions and instantiate only variables that occur at least in one ν-position. Moreover, those variables can be only instantiated with terms that fulfill the ν restrictions: t $\langle\mu,\nu\rangle$-narrows to s, written $t \overset{[p,\theta]}{\rightsquigarrow}_{\langle\mu,\nu\rangle} s$, if $t \overset{[p,\theta]}{\rightsquigarrow}_{\mathcal{R}} s$ for some rule $l \to r \in \mathcal{R}$, $p \in \mathcal{P}os^{\mu}(t)$, and $\forall x \in \mathcal{D}om(\theta)$, $\exists q \in \mathcal{P}os_x(t|_p)$ s.t. $p.q.\mathcal{P}os_{\mathcal{F}}(\theta(t)|_{p.q}) \subseteq \mathcal{P}os^{\nu}(\theta(t)) \cap p.q.\mathcal{P}os_{\mathcal{F}}(l)$. That is, $\rightsquigarrow_{\langle\mu,\nu\rangle} \subseteq \rightsquigarrow_{\mu} \subseteq \rightsquigarrow$. From now on, we will write ${\rightsquigarrow_{\langle\mu,\nu\rangle}}_{\theta}$ in the examples instead of $\overset{\theta}{\rightsquigarrow}_{\langle\mu,\nu\rangle}$.

Example 5 Consider Example 4 and $t_4 =$ `and(account(0,XS),client1(XS))`. Recall the following maps μ,ν from Example 4: $\mu(\texttt{cons}) = \{1\}$ and $\mu(f) = \{1,\ldots,ar(f)\}$ otherwise; $\nu(\texttt{add}) = \nu(\texttt{sub}) = \varnothing$, $\nu(\texttt{account}) = \{1\}$, and $\nu(f) = \{1,\ldots,ar(f)\}$ otherwise. We can only prove the following $\langle\mu,\nu\rangle$-narrowing step:

$$\begin{array}{l}
\texttt{and(account(0,XS),}\underline{\texttt{client1(XS)}}\texttt{)} \\
{\rightsquigarrow_{\langle\mu,\nu\rangle}}_{[\texttt{X}\mapsto\texttt{cons(dep(50),cons(wit(20),nil))}]} \\
\texttt{and(account(0,cons(dep(50),} \\
\qquad\qquad\qquad \texttt{cons(wit(20),nil))),True)}
\end{array}$$

That is, $2 \in \mathcal{P}os^{\mu}(t_4)$, $\mathcal{P}os_{\texttt{XS}}(t_4|_2) = \{1\}$, $\theta = [\texttt{XS} \mapsto \texttt{cons(dep(50),cons(wit(20),nil))}]$, and $2.1.\mathcal{P}os_{\mathcal{F}}(\theta(t_4)|_{2.1}) \subseteq \mathcal{P}os^{\nu}(\theta(t_4)) \cap 2.1.\mathcal{P}os_{\mathcal{F}}(l)$ with $l =$ `client1(cons(dep(50),cons(wit(20),nil)))`. ■

Not surprisingly, the completeness of the new model can only be proven under conditions that are similar to Theorem 1.

Theorem 2 (Completeness) *Let $\mathcal{R}$ be a left-linear CS and μ,ν be such that $\mu^{\mathcal{B}}_{\mathcal{R}} \sqsubseteq \mu$ and $\mu^{\mathcal{B}}_{\mathcal{R}} \sqsubseteq \nu$ for some $\mathcal{B} \subseteq \mathcal{C}$. Let $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $s \in \mathcal{T}(\mathcal{B},\mathcal{X})$, and θ,ϑ be normalized substitutions. Then, $t \rightsquigarrow^{*}_{\theta} s$ iff $t \overset{\vartheta}{\rightsquigarrow}^{*}_{\langle\mu,\nu\rangle} s$ and $\vartheta \leq \theta$.*

This means that the combined model should be applied under the same conditions as *modes* in the logic programming setting, see [6]. Nevertheless, it is still useful for real applications, and real programming languages, such as Prolog or Curry.

In the following, we show that previous approaches are subsumed by the new model.

3.1 Context-sensitive rewriting (*CSR*)

The rewrite relation $\hookrightarrow_{\mu}$ can be easily simulated in our new model by $\rightsquigarrow_{\langle\mu,\nu\rangle}$, by reusing the replacement map μ and fixing $\nu(f) = \varnothing$ for all $f \in \mathcal{F}$.

3.2 Context-sensitive narrowing (*CSN*)

It is also straightforward to simulate the context-sensitive narrowing relation $\rightsquigarrow_{\mu}$ in the new model by $\rightsquigarrow_{\langle\mu,\nu\rangle}$, by reusing the replacement map μ and fixing $\nu(f) = \{1,\ldots,ar(f)\}$ for all $f \in \mathcal{F}$.

3.3 Rigid/flexible annotations

With respect to rigid/flexible annotations as implemented in Curry, the simulation within the new model is not immediate, since an involved lazy narrowing strategy called *needed narrowing* is used in [1] as the basic strategy to which rigid/flexible annotations are applied. However, if we consider the use of *rigid/flexible* annotations on top of ordinary narrowing, they can be simulated by $\rightsquigarrow_{\langle\mu,\nu\rangle}$, where $\mu(f) = \{1,\ldots,ar(f)\}$ for all $f \in \mathcal{F}$, $\nu(c) = \{1,\ldots,ar(c)\}$ for all constructor symbols $c \in \mathcal{C}$, $\nu(f) = \varnothing$ if $f \in \mathcal{D}$ is marked as *rigid*, and $\nu(f) = \{1,\ldots,ar(f)\}$ if $f \in \mathcal{D}$ is marked as *flexible*.

4 Conclusions

Clearly, more research has to be done in order to assess the properties and benefits of our model of syntactic annotations for narrowing in practice. We (hopefully) expect to have motivated the advantages w.r.t. the state-of-the-art models for syntactic annotations in real programming.

An important open problem is termination of the $\langle\mu,\nu\rangle$-narrowing relation. This is particularly interesting because classes of programs where narrowing terminates are small and rarely useful in practice. Works on termination of logic programs such as [6] could be revisited in the context of termination of $\langle\mu,\nu\rangle$-narrowing or combined with termination of *CSR* [10]. Moreover, termination of the

$\langle\mu,\nu\rangle$-narrowing relation can be very useful for model checking, protocol verification, and theorem proving, since narrowing has proved to be very useful as their basic mechanism [5].

References

[1] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. *Journal of the ACM*, 47(4):776–822, 2000.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, J. M. N. Martí-Oliet, , and C. Talcott. The Maude 2.0 System. In *Proc. of RTA'2003*, LNCS 2706:76–87. Springer-Verlag, 2003.

[4] F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving Termination of Membership Equational Programs. In P. Sestoft and N. Heintze, editors, *Proc. of the ACM SIGPLAN 2004 Symposium on Partial Evaluation and Program Manipulation, PEPM'04*, pages 147-158, ACM Press, New York, 2004.

[5] S. Escobar, J. Meseguer, and P. Thati. Natural Narrowing for General Term Rewriting Systems. In *Proc. of RTA'2005*, LNCS 3467:279-293. Springer-Verlag, 2005.

[6] S. Etalle, A. Bossi, and N. Cocco. Termination of well-moded programs. *Journal of Logic Programming*, 38(2):243–257, 1999.

[7] K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In *Proc. of POPL'85*, pages 52–66. ACM Press, 1985.

[8] K. Futatsugi and A. Nakagawa. An overview of CAFE specification environment – an algebraic approach for creating, verifying, and maintaining formal specification over networks –. In *1st International Conference on Formal Engineering Methods*, 1997.

[9] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: algebraic specification in action*. Kluwer Academic Publishers, 2000.

[10] J. Giesl and A. Middeldorp. Transformation Techniques for Context-Sensitive Rewrite Systems. *Journal of Functional Programming*, 14(4):379-427, 2004.

[11] M. Hanus. A unified computation model for functional and logic programming. In *Proc. of POPL'97*, pages 80–93. ACM, 1997.

[12] M. Hanus, S. Antoy, H. Kuchen, F. López-Fraguas, W. Lux, J. Moreno Navarro, and F. Steiner. Curry: An Integrated Functional Logic Language (version 0.8). 2003.

[13] P. Hudak, S. Peyton–Jones, and P. Wadler. Report on the Functional Programming Language Haskell: a non–strict, purely functional language. *Sigplan Notices*, 27(5), 1992.

[14] J. Lloyd. Combining Functional and Logic Programming Languages. In *Proc. of the International Logic Programming Symposium*, pages 43–57. The MIT Press, 1994.

[15] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1–61, 1998.

[16] S. Lucas. Termination of Rewriting With Strategy Annotations. In *Proc. of LPAR'01*, LNCS 2250:669–684. Springer-Verlag, 2001.

[17] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.

[18] J. McCarthy. Recursive Functions of Symbolic Expressions and their Computations by Machine, Part I. *Communications of the ACM*, 3(4):184–195, 1960.

[19] E. Nöcker, J. Smetsers, M. van Eekelen, and M. Plasmeijer. Concurrent clean. In *Proceedings of PARLE'91*, LNCS 506:202–219. Springer-Verlag, 1991.