Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 328-339

A Hierarchy of Semantics for Non-deterministic Term Rewriting Systems*

Juan Rodríguez-Hortalá
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
juanrh@fdi.ucm.es

ABSTRACT. Formalisms involving some degree of nondeterminism are frequent in computer science. In particular, various programming or specification languages are based on term rewriting systems where confluence is not required. In this paper we examine three concrete possible semantics for non-determinism that can be assigned to those programs. Two of them –call-time choice and run-time choice– are quite well-known, while the third one –plural semantics– is investigated for the first time in the context of term rewriting based programming languages. We investigate some basic intrinsic properties of the semantics and establish some relationships between them: we show that the three semantics form a hierarchy in the sense of set inclusion, and we prove that call-time choice and plural semantics enjoy a remarkable compositionality property that fails for run-time choice; finally, we show how to express plural semantics within run-time choice by means of a program transformation, for which we prove its adequacy.

1 Introduction

Term rewriting systems (*TRS's*) [4] have a long tradition as a suitable basic formalism to address a wide range of tasks in computer science, in particular, many specification languages [5, 7], theorem provers [21, 6] and programming languages are based on TRS's. For instance, the syntax and theory of TRS's was the basis of the first formulations of *functional logic programming* (*FLP*) through the idea of narrowing [9]. On the other hand, non-determinism is an expressive feature that has been used for a long time in system specification (e.g., non-deterministic Turing machines or automata) or for programming (the constructions of McCarthy and Dijkstra are classical examples). One of the appeals of term rewriting is its elegant way to express non-determinism through the use of a non-confluent TRS, obtaining a clean and high level representation of complex systems. In the field of FLP, non-confluent TRS's are used as programs to support non-strict non-deterministic functions, which are one of the most distinctive features of the paradigm [8, 3]. Those TRS's follow the constructor discipline also, corresponding to a value-based semantic view, in which the purpose of computations is to produce values made of constructors.

Therefore non-confluent constructor-based TRS's can be used as a common syntactic framework for FLP and rewriting. The set of rewrite rules constitutes a program and so we also call them *program rules*. Nevertheless the behaviour of current implementations of FLP

*This work has been partially supported by the Spanish projects Merit-Forms-UCM (TIN2005-09207-C03-03), Promesas-CAM (S-0505/TIC/0407) and FAST-STAMP (TIN2008-06622-C03-01/TIN).

and rewriting differ substantially, because the introduction of non-determinism in a functional setting gives rise to a variety of semantic decisions, that were explored in [20]. There the different language variants that result after adding non-determinism to a basic functional language were expounded, structuring the comparison as a choice among different options over several dimensions: strict/non-strict functions, angelic/demonic/erratic non-deterministic choices and *singular/plural semantics* for parameter passing. In the present paper we assume non-strict angelic non-determinism, and we are concerned about the last dimension only. The key difference is that under a singular semantics, in the substitutions used to instantiate the program rules for function application, the variables of the program rules should range over single objects of the set of values considered; in a plural semantics those range over sets of objects. This has been traditionally identified with the distinction between *call-time choice* and *run-time choice* [11] parameter passing mechanisms. Under call-time choice a value for each argument is computed before performing parameter passing, this corresponds to call-by-value in a strict setting and to call-by-need in a non-strict setting, in which a partial value instead of a total value is computed. On the other hand, run-time-choice corresponds to call-by-name, each argument is copied without any evaluation and so the different copies of any argument may evolve in different ways afterwards. Thus, traditionally it has been considered that call-time choice parameter passing inducts a singular semantics while run-time choice inducts a plural semantics.

EXAMPLE 1. *Consider the TRS* $\mathcal{P} = \{f(c(X)) \to d(X,X), X\ ?\ Y \to X, X\ ?\ Y \to Y\}$. *With call-time choice/singular semantics to compute a value for the term* $f(c(0?1))$ *we must first compute a (partial) value for* $c(0?1)$*, and then we may continue the computation with* $f(c(0))$ *or* $f(c(1))$ *which yield* $d(0,0)$ *or* $d(1,1)$. *Note that* $d(0,1)$ *and* $d(1,0)$ *are not correct values for* $f(c(0?1))$ *in that setting.*
On the other hand with run-time choice/plural semantics to evaluate the term $f(c(0?1))$*:*

- *Under the run-time choice point of view, the step* $f(c(0?1)) \to d(0?1, 0?1)$ *is sound, hence not only* $d(0,0)$ *and* $d(1,1)$ *but also* $d(0,1)$ *and* $d(1,0)$ *are valid values for* $f(c(0?1))$.
- *Under the plural semantics point of view, we consider the set* $\{c(0), c(1)\}$ *which is a subset of the set of values for* $c(0?1)$ *in which every element matches the argument pattern* $c(X)$. *Therefore the set* $\{0,1\}$ *can be used for parameter passing obtaining a kind of "set expression"* $d(\{0,1\},\{0,1\})$*, which evaluation yields the values* $d(0,0)$*,* $d(1,1)$*,* $d(0,1)$ *and* $d(1,0)$.

In general, call-time choice/singular semantics produces less results than run-time choice/ plural semantics.

A standard formulation for call-time choice[†] in FLP is the *CRWL*[‡] logic [8], which is implemented by current FLP languages like Toy [15] or Curry [10]; traditional term rewriting may be considered the standard semantics for run-time choice[§], and is the basis for the semantics of languages like Maude [5], but has been rarely [1] thought as a valuable global alternative to call-time choice for the value-based view of FLP. However, there might be

[†]In fact angelic non-strict call-time choice.
[‡]Constructor-based **ReW**riting **L**ogic.
[§]In fact angelic non-strict run-time choice.

parts in a program or individual functions for which run-time choice could be a better option, and therefore it would be convenient to have both possibilities (run-time/call-time) at programmer's disposal [13]. Nevertheless the use of an operational notion like term rewriting as the semantic basis of a FLP language can lead us to confusing situations, not very compatible with the value-based semantic view that we wanted for the constructor-based TRS's used in FLP.

EXAMPLE 2. *Starting with the TRS of Example 1 we want to evaluate the expression $f(c(0)$? $c(1))$ with run-time choice/plural semantics:*

- *Under the run-time choice point of view, that is, using term rewriting, the evaluation of the subexpression $c(0)?c(1)$ is needed in order to get an expression that matches the left hand side $f(c(X))$. Hence the derivations $f(c(0)?c(1)) \to f(c(0)) \to d(0,0)$ and $f(c(0)?c(1)) \to f(c(1)) \to d(1,1)$ are sound and compute the values $d(0,0)$ and $d(1,1)$, but neither $d(0,1)$ nor $d(1,0)$ are correct values for $f(c(0)?c(1))$.*
- *Under the plural semantics point of view, we consider the set $\{c(0), c(1)\}$ which is a subset of the set of values for $c(0)?c(1)$ in which every element matches the argument pattern $c(X)$. Therefore the set $\{0,1\}$ can be used for parameter passing obtaining a kind of "set expression" $d(\{0,1\},\{0,1\})$ that yields the values $d(0,0)$, $d(1,1)$, $d(0,1)$ and $d(1,0)$.*

Which of these is the more suitable perspective for FLP?

This problem did not appear in [20] because no pattern matching was present, nor in [11] because only call-time choice was adopted (and this conflict does not appear between the call-time choice and the singular semantics views). Choosing the run-time choice perspective of term rewriting has some unpleasant consequences. First of all the expression $f(c(0?1))$ has more values than the expression $f(c(0)?c(1))$, even when the only difference between them is the subexpressions $c(0?1)$ and $c(0)?c(1)$, which have the same values both in call-time choice, run-time choice and plural semantics. This is pretty incompatible with the value-based semantic view we are looking for in FLP. And this has to do with the loss of some desirable properties, present in *CRWL*, when switching to run-time choice. We will see how plural semantics recovers those properties, which are very useful for reasoning about computations. Furthermore it allows natural encodings of some programs that need to do some collecting work, as we will see later (Example 8). Hence we claim that the plural semantics perspective is more suitable for a value-based programming language.

The rest of the paper is organized as follows. Section 2 contains some technical preliminaries and notations about *CRWL* and term rewriting systems. In Section 3 we introduce π*CRWL*, a variation of *CRWL* to express plural semantics, and present some of its properties. In Section 4 we discuss about the different properties of these semantics and prove the inclusion chain *CRWL* $\subseteq$ rewriting $\subseteq$ π*CRWL*, that constitutes a hierarchy of semantics for non-determinism. Section 5 recalls that no straight simulation of *CRWL* in term rewriting can be done by a program transformation, and vice versa, and shows a novel transformation to simulate π*CRWL* using term rewriting. Finally Section 6 summarizes some conclusions and future work. Fully detailed proofs, including some auxiliary results, can be found in [19].

2 Preliminaries

2.1 Constructor based term rewriting systems

We consider a first order signature $\Sigma = CS \cup FS$, where CS and FS are two disjoint set of *constructor* and defined *function* symbols respectively, all them with associated arity. We write CS^n (FS^n resp.) for the set of constructor (function) symbols of arity n. We write $c, d, \ldots$ for constructors, $f, g, \ldots$ for functions and $X, Y, \ldots$ for variables of a numerable set $\mathcal{V}$. The notation $\overline{o}$ stands for tuples of any kind of syntactic objects. Given a set $\mathcal{A}$ we denote by $\mathcal{A}^*$ the set of finite sequences of elements of that set. For any sequence $a_1 \ldots a_n \in \mathcal{A}^*$ and function $f : \mathcal{A} \to \{true, false\}$, by $a_1 \ldots a_n \mid f$ we denote the sequence constructed taking in order every element from $a_1 \ldots a_n$ for which f holds.

The set Exp of *expressions* is defined as $Exp \ni e ::= X \mid h(e_1, \ldots, e_n)$, where $X \in \mathcal{V}$, $h \in CS^n \cup FS^n$ and $e_1, \ldots, e_n \in Exp$. The set $CTerm$ of *constructed terms* (or *c-terms*) is defined like Exp, but with h restricted to CS^n (so $CTerm \subseteq Exp$). The intended meaning is that Exp stands for evaluable expressions, i.e., expressions that can contain function symbols, while $CTerm$ stands for data terms representing **values**. We will write $e, e', \ldots$ for expressions and $t, s, \ldots$ for c-terms. The set of variables occurring in an expression e will be denoted as $var(e)$. We will frequently use *one-hole contexts*, defined as $Cntxt \ni \mathcal{C} ::= [\,] \mid h(e_1, \ldots, \mathcal{C}, \ldots, e_n)$, with $h \in CS^n \cup FS^n$. The application of a context $\mathcal{C}$ to an expression e, written by $\mathcal{C}[e]$, is defined inductively as $[\,][e] = e$ and $h(e_1, \ldots, \mathcal{C}, \ldots, e_n)[e] = h(e_1, \ldots, \mathcal{C}[e], \ldots, e_n)$.

Substitutions $\theta \in Subst$ are finite mappings $\theta : \mathcal{V} \longrightarrow Exp$, extending naturally to $\theta : Exp \longrightarrow Exp$. We write ϵ for the identity (or empty) substitution. We write $e\theta$ for the application of θ to e, and $\theta\theta'$ for the composition, defined by $X(\theta\theta') = (X\theta)\theta'$. The domain and range of θ are defined as $dom(\theta) = \{X \in \mathcal{V} \mid X\theta \neq X\}$ and $vran(\theta) = \bigcup_{X \in dom(\theta)} var(X\theta)$. If $dom(\theta_0) \cap dom(\theta_1) = \emptyset$, their disjoint union $\theta_0 \uplus \theta_1$ is defined by $(\theta_0 \uplus \theta_1)(X) = \theta_i(X)$, if $X \in dom(\theta_i)$ for some θ_i; $(\theta_0 \uplus \theta_1)(X) = X$ otherwise. Given $W \subseteq \mathcal{V}$ we write $\theta|_W$ for the restriction of θ to W, and $\theta|_{\backslash D}$ is a shortcut for $\theta|_{(\mathcal{V} \backslash D)}$. We will sometimes write $\theta = \sigma[W]$ instead of $\theta|_W = \sigma|_W$. *C-substitutions* $\theta \in CSubst$ verify that $X\theta \in CTerm$ for all $X \in dom(\theta)$.

A *constructor-based term rewriting system* $\mathcal{P}$ (CS) is a set of c-rewrite rules of the form $f(\overline{t}) \to r$ where $f \in FS^n$, $e \in Exp$ and $\overline{t}$ is a linear n-tuple of c-terms, where linearity means that variables occur only once in $\overline{t}$. In the present work we restrict ourselves to CS's not containing *extra variables*, i.e., CS's for which $var(r) \subseteq var(f(\overline{t}))$ holds for any rewrite rule; the extension of this work to rules with extra variables is a subject of future work. We assume that every CS $\mathcal{P}$ contains the rules $\{X \,?\, Y \to X, X \,?\, Y \to Y, if\ true\ then\ X \to X\}$, defining the behaviour of $\_?\_ \in FS^2$, $if\_then\_ \in FS^2$, both used in mixfix mode, and that those are the only rules for that function symbols. For the sake of conciseness we will often omit these rules when presenting a CS.

Given a TRS $\mathcal{P}$, its associated *rewrite relation* $\to_{\mathcal{P}}$ is defined as: $\mathcal{C}[l\sigma] \to_{\mathcal{P}} \mathcal{C}[r\sigma]$ for any context $\mathcal{C}$, rule $l \to r \in \mathcal{P}$ and $\sigma \in Subst$. We write $\overset{*}{\to}_{\mathcal{P}}$ for the reflexive and transitive closure of the relation $\to_{\mathcal{P}}$. In the following, we will usually omit the reference to $\mathcal{P}$ or denote it by $\mathcal{P} \vdash e \to e'$ and $\mathcal{P} \vdash e \to^* e'$.

$$
\begin{array}{llllll}
\textbf{(RR)} & \dfrac{}{X \rightarrowtriangle X} & X \in \mathcal{V} & \textbf{(DC)} & \dfrac{e_1 \rightarrowtriangle t_1 \ldots e_n \rightarrowtriangle t_n}{c(e_1,\ldots,e_n) \rightarrowtriangle c(t_1,\ldots,t_n)} & c \in CS^n \\
\textbf{(B)} & \dfrac{}{e \rightarrowtriangle \perp} & & \textbf{(OR)} & \dfrac{e_1 \rightarrowtriangle p_1\theta \ldots e_n \rightarrowtriangle p_n\theta \;\; r\theta \rightarrowtriangle t}{f(e_1,\ldots,e_n) \rightarrowtriangle t} & \begin{array}{l} f(p_1,\ldots,p_n) \to r \in \mathcal{P} \\ \theta \in CSubst_\perp \end{array}
\end{array}
$$

Figure 1: Rules of *CRWL*

2.2 The CRWL framework

In the *CRWL* framework [8], programs are CS's, also called *CRWL-programs* (or simply 'programs') from now on. To deal with non-strictness at the semantic level, we enlarge Σ with a new constant constructor symbol $\perp$. The sets $Exp_\perp$, $CTerm_\perp$, $Subst_\perp$, $CSubst_\perp$ of partial expressions, etc., are defined naturally. Notice that $\perp$ does not appear in programs. Partial expressions are ordered by the *approximation* ordering $\sqsubseteq$ defined as the least partial ordering satisfying $\perp \sqsubseteq e$ and $e \sqsubseteq e' \Rightarrow \mathcal{C}[e] \sqsubseteq \mathcal{C}[e']$ for all $e, e' \in Exp_\perp, \mathcal{C} \in Cntxt$. This partial ordering can be extended to substitutions: given $\theta, \sigma \in Subst_\perp$ we say $\theta \sqsubseteq \sigma$ if $X\theta \sqsubseteq X\sigma$ for all $X \in \mathcal{V}$.

The semantics of a program $\mathcal{P}$ is determined in *CRWL* by means of a proof calculus able to derive reduction statements of the form $e \rightarrowtriangle t$, with $e \in Exp_\perp$ and $t \in CTerm_\perp$, meaning informally that t is (or approximates to) a *possible value* of e, obtained by iterated reduction of e using $\mathcal{P}$ under call-time choice. The *CRWL*-proof calculus is presented in Figure 1. Rule **B** (bottom) allows any expression to be undefined or not evaluated (non-strict semantics). Rule **OR** (outer reduction) expresses that to evaluate a function call we must choose a compatible program rule, perform parameter passing (by means of a $CSubst_\perp$ θ) and then reduce the right-hand side. The use of partial c-substitutions in OR is essential to express call-time choice, as only single partial values are used for parameter passing.

We write $\mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t$ to express that $e \rightarrowtriangle t$ is derivable in the *CRWL*-calculus using the program $\mathcal{P}$. Given a program $\mathcal{P}$, the *CRWL-denotation* of an expression $e \in Exp_\perp$ is defined as $[\![e]\!]^{sg}_{\mathcal{P}} = \{t \in CTerm_\perp \mid \mathcal{P} \vdash_{CRWL} e \rightarrowtriangle t\}$. In the following, we will usually omit the reference to $\mathcal{P}$.

3 $\pi CRWL$: a plural semantics for FLP

The new calculus $\pi CRWL$ is defined by modifying the rules of *CRWL* to consider sets of partial values for parameter passing instead of single partial values: hence, only the rule OR should be modified. To avoid the need to extend the syntax with new constructions to represent those "set expressions" that we talked about in the introduction, we will exploit the fact that $[\![e_1 \;?\; e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!]$. Therefore the substitutions used for parameter passing will map variables to "disjunctions of values". We define the set $CSubst^?_\perp = \{\theta \in Subst_\perp \mid \forall X \in dom(\theta), \theta(X) = t_1 \;?\; \ldots \;?\; t_n$ such that $t_1, \ldots, t_n \in CTerm_\perp, n > 0\}$, for which $CSubst_\perp \subseteq CSubst^?_\perp \subseteq Subst_\perp$ obviously holds. The operator $? : CSubst^*_\perp \to CSubst^?_\perp$ constructs the $CSubst^?_\perp$ corresponding to a non empty sequence of $CSubst_\perp$, and is defined as $?(\theta_1 \ldots \theta_n)(X) = X$ if $X \notin \bigcup_{i \in \{1,\ldots,n\}} dom(\theta_i)$; $?(\theta_1 \ldots \theta_n)(X) = \rho_1(X) \;?\; \ldots \;?\; \rho_m(X)$, where $\rho_1 \ldots \rho_m = \theta_1 \ldots \theta_n \mid \lambda\theta.(X \in dom(\theta))$, otherwise. Then $dom(?(\theta_1 \ldots \theta_n)) = \bigcup_i dom(\theta_i)$. This

$$
\textbf{(RR)}\ \frac{}{X \rightarrowtriangle X}\quad X \in \mathcal{V}
\qquad
\textbf{(DC)}\ \frac{e_1 \rightarrowtriangle t_1 \ldots e_n \rightarrowtriangle t_n}{c(e_1,\ldots,e_n) \rightarrowtriangle c(t_1,\ldots,t_n)}\quad c \in CS^n
$$

$$
\textbf{(B)}\ \frac{}{e \rightarrowtriangle \perp}
\qquad
\textbf{(POR)}\ \frac{\begin{array}{ccc} e_1 \rightarrowtriangle p_1\theta_{11} & e_n \rightarrowtriangle p_n\theta_{n1} & \\ \ldots & \ldots & \ldots \\ e_1 \rightarrowtriangle p_1\theta_{1m_1} & e_n \rightarrowtriangle p_n\theta_{nm_n} & r\theta \rightarrowtriangle t \end{array}}{f(e_1,\ldots,e_n) \rightarrowtriangle t}
$$

$$
(f(\overline{p}) \to r) \in \mathcal{P}, \theta = ?\{\theta_{11},\ldots,\theta_{1m_1}\} \uplus \ldots \uplus ?\{\theta_{n1},\ldots,\theta_{nm_n}\}
$$
$$
\forall i,j\ \theta_{ij} \in CSubst_\perp \wedge dom(\theta_{ij}) = var(p_i), \forall i\ m_i > 0
$$

Figure 2: Rules of $\pi CRWL$

operator is overloaded to handle non empty sets $\Theta \subseteq CSubst_\perp$ as $?\Theta = ?(\theta_1 \ldots \theta_n)$ where the sequence $\theta_1 \ldots \theta_n$ corresponds to an arbitrary reordering of the elements of Θ.

The $\pi CRWL$-proof calculus is presented in Figure 2. The only difference with the calculus in Figure 1 is that the rule OR has been replaced by **POR** (plural outer reduction), in which we may compute more that one partial value for each argument, and then use a substitution from $CSubst^?_\perp$ instead of $CSubst_\perp$ for parameter passing, achieving a plural semantics[¶]. This calculus derives reduction statements of the form $\mathcal{P} \vdash_{\pi CRWL} e \rightarrowtriangle t$ that express that t is (or approximates to) a possible value for e in this semantics, under the program $\mathcal{P}$. The *$\pi CRWL$-denotation* of an expression $e \in Exp_\perp$ under a program $\mathcal{P}$ in $\pi CRWL$ is defined as $[\![e]\!]^{pl}_{\mathcal{P}} = \{t \in CTerm_\perp \mid \mathcal{P} \vdash_{\pi CRWL} e \rightarrowtriangle t\}$.

EXAMPLE 3. *Consider the program of Example 1, that is $\mathcal{P} = \{f(c(X)) \to d(X,X), X \mathbin{?} Y \to X, X \mathbin{?} Y \to Y\}$. The following is a $\pi CRWL$-proof for the statement $f(c(0)?c(1)) \rightarrowtriangle d(0,1)$ (some steps have been omitted for the sake of conciseness):*

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{0 \rightarrowtriangle 0}\,DC}{c(0) \rightarrowtriangle c(0)}\,DC \quad \dfrac{}{c(1) \rightarrowtriangle \perp}\,B \quad c(0) \rightarrowtriangle c(0)}{c(0)?c(1) \rightarrowtriangle c(0)}\,POR \quad c(0)?c(1) \rightarrowtriangle c(1) \quad \dfrac{0?1 \rightarrowtriangle 0 \quad 0?1 \rightarrowtriangle 1}{d(0?1,0?1) \rightarrowtriangle d(0,1)}\,DC}{f(c(0)?c(1)) \rightarrowtriangle d(0,1)}\,POR
$$

$\pi CRWL$ enjoys some nice properties, like the following monotonicity property, where for any proof we define its *size* as the number of applications of rules of the calculus.

LEMMA 4. *For any CRWL-program, $e, e' \in Exp_\perp$, $t, t' \in CTerm_\perp$ if $e \sqsubseteq e'$ and $t' \sqsubseteq t$ then $\mathcal{P} \vdash_{\pi CRWL} e \rightarrowtriangle t$ implies $\mathcal{P} \vdash_{\pi CRWL} e' \rightarrowtriangle t'$ with a proof of the same size or smaller.*

One of the most important properties is its compositionality, a property very close to the DET-additivity property for algebraic specifications of [11]:

THEOREM 5. *For any CRWL-program, $\mathcal{C} \in Contx$ and $e \in Exp_\perp$, $[\![\mathcal{C}[e]]\!]^{pl} = \bigcup_{\{t_1,\ldots,t_n\} \subseteq [\![e]\!]^{pl}} [\![\mathcal{C}[t_1 \mathbin{?} \ldots \mathbin{?} t_n]]\!]^{pl}$, for any arrangement of the elements of $\{t_1,\ldots,t_n\}$ in $t_1 \mathbin{?} \ldots \mathbin{?} t_n$.*

The proof for that theorem is based upon the commutativity, associativity of ?, and the idempotence of its partial application (see [19]). With Theorem 5 at hand is very easy to prove the following distributivity property for $\pi CRWL$, also known as the *bubbling* operational rule [2]:

[¶]In fact angelic non-strict plural non-determinism.

THEOREM 6.*[Correctness of bubbling] For any CRWL-program, $\mathcal{C} \in Contx$ and $e_1, e_2 \in Exp_\perp$, $[\![\mathcal{C}[e_1 \mathbin{?} e_2]]\!]^{pl} = [\![\mathcal{C}[e_1] \mathbin{?} \mathcal{C}[e_2]]\!]^{pl}$.*

$\pi CRWL$ also has some monotonicity properties related to substitutions. We define the preorder $\sqsubseteq_\pi$ over $CSubst^?_\perp$ by $\theta \sqsubseteq_\pi \theta'$ iff $\forall X \in \mathcal{V}$, given $\theta(X) = t_1 \mathbin{?} \ldots \mathbin{?} t_n$ and $\theta(X) = t'_1 \mathbin{?} \ldots \mathbin{?} t'_m$ then $\forall t \in \{t_1, \ldots, t_n\} \exists t' \in \{t'_1, \ldots, t'_m\}$ such that $t \sqsubseteq t'$; and the preorder $\trianglelefteq$ over $Subst_\perp$ by $\sigma \trianglelefteq \sigma'$ iff $\forall X \in \mathcal{V}$, $[\![\sigma(X)]\!]^{pl} \subseteq [\![\sigma'(X)]\!]^{pl}$.

LEMMA 7. *For any CRWL-program, $e \in Exp_\perp$, $t \in CTerm_\perp$, $\sigma, \sigma' \in Subst_\perp$, $\theta, \theta' \in CSubst^?_\perp$:*

1. ***Strong monotonicity of*** $\boldsymbol{Subst_\perp}$*: If $\forall X \in \mathcal{V}, s \in CTerm_\perp$ given $\mathcal{P} \vdash_{\pi CRWL} \sigma(X) \rightarrowtriangle s$ with size K we also have $\mathcal{P} \vdash_{\pi CRWL} \sigma'(X) \rightarrowtriangle s$ with size $K' \leq K$, then $\vdash_{\pi CRWL} e\sigma \rightarrowtriangle t$ with size L implies $\vdash_{\pi CRWL} e\sigma' \rightarrowtriangle t$ with size $L' \leq L$.*
2. ***Monotonicity of*** $\boldsymbol{CSubst_\perp}$*: If $\theta, \theta' \in CSubst_\perp$ and $\theta \sqsubseteq \theta'$ then $\mathcal{P} \vdash_{\pi CRWL} e\theta \rightarrowtriangle t$ with size K implies $\mathcal{P} \vdash_{\pi CRWL} e\theta' \rightarrowtriangle t$ with size $K' \leq K$.*
3. ***Monotonicity of*** $\boldsymbol{Subst_\perp}$*: If $\sigma \trianglelefteq \sigma'$ then $[\![e\sigma]\!]^{pl} \subseteq [\![e\sigma']\!]^{pl}$.*
4. ***Monotonicity of*** $\boldsymbol{CSubst^?_\perp}$*: If $\theta \sqsubseteq_\pi \theta'$ then $[\![e\theta]\!]^{pl} \subseteq [\![e\theta']\!]^{pl}$.*

We end this section with an example of the use of $\pi CRWL$ to model problems in which some collecting work has to be done.

EXAMPLE 8. *We want to represent the database of a bank in which we hold some data about its employees, this bank has several branches and we want to organize the information according to them. So we define a non-deterministic function $branches$ to represent the set of branches: a set is identified then with a non-deterministic expression. In this line we define a non-deterministic function $employees$ which conceptually returns the set of records containing the information regarding the employees that work in a branch. Now, to search for the names of two clerks we define the function $twoclerks$ which is based upon $find$, which forces the desired pattern $e(N, S, clerk)$ over the set defined by $employees(branches)$.*

$\mathcal{P} = \{branches \to madrid, branches \to vigo, employees(madrid) \to e(pepe, men, clerk), employees(madrid) \to e(paco, men, boss), employees(vigo) \to e(maria, women, clerk), employees(vigo) \to e(jaime, women, boss), twoclerks \to find(employees(branches)), find(e(N, S, clerk)) \to (N, N)\}$

With term rewriting $twoclerks \to find(employees(branches)) \not\to^* (pepe, maria)$, because in that expression the evaluation of $branches$ is needed and so one of the branches must be chosen. On the other hand with $\pi CRWL$ (some steps have been omitted for the sake of conciseness):

$$\cfrac{\begin{array}{cc} \cfrac{\ldots}{employees(branches) \rightarrowtriangle e(pepe, \perp, clerk)}\,POR & \\ & \cfrac{\ldots}{(pepe \mathbin{?} maria, pepe \mathbin{?} maria) \rightarrowtriangle (pepe, maria)}\,DC \\ \cfrac{\ldots}{employees(branches) \rightarrowtriangle e(maria, \perp, clerk)}\,POR & \end{array}}{\cfrac{find(employees(branches)) \rightarrowtriangle (pepe, maria)}{twoclerks \rightarrowtriangle (pepe, maria)}\,POR}\,POR$$

where

$$\cfrac{\cfrac{}{branches \rightarrowtriangle madrid}\,POR \quad \cfrac{\ldots}{e(pepe, men, clerk) \rightarrowtriangle e(pepe, \perp, clerk)}\,DC}{employees(branches) \rightarrowtriangle e(pepe, \perp, clerk)}\,POR$$

4 Comparison: a hierarchy of semantics

When comparing these semantics is not surprising finding that $CRWL$ and $\pi CRWL$ have similar properties. For example the monotonicity Lemma 4 also holds for $CRWL$; this lemma does not even make sense for term rewriting, as it only works with total terms. On the other hand term rewriting is closed under $Subst$ ($e \rightarrow^* e'$ implies $e\sigma \rightarrow^* e'\sigma$, for any $\sigma \in Subst$); $CRWL$ is not closed under $Subst$ but under $CSubst_\perp$, as corresponds to call-time choice; $\pi CRWL$ is closed under $CSubst_\perp$ too (see [19]), and we conjecture that for $\theta \in CSubst^?_\perp$ if $\vdash_{\pi CRWL} e \twoheadrightarrow t$ then $[\![t\theta]\!]^{pl} \subseteq [\![e\theta]\!]^{pl}$. For $CRWL$ a compositionality result similar to Theorem 5 also holds, and bubbling is correct too [14]. This is not the case for term rewriting, as we saw when switching from $f(c(0?1))$ to $f(c(0)?c(1))$ in examples 1 and 2.

4.1 The hierarchy

As $\pi CRWL$ is a modification of $CRWL$, the relation between them is very direct.

THEOREM 9. *For any CRWL-program $\mathcal{P}$, $e \in Exp_\perp, t \in CTerm_\perp$ given a CRWL-proof for $\mathcal{P} \vdash e \twoheadrightarrow t$ we can build a πCRWL-proof for $\mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ just replacing every **OR** step by the corresponding **POR** step. As a consequence $[\![e]\!]^{sg}_{\mathcal{P}} \subseteq [\![e]\!]^{pl}_{\mathcal{P}}$.*

Concerning the relation of $CRWL$ and $\pi CRWL$ with term rewriting, we will use the notion of *shell $|e|$ of an expression e* that represents the outer constructor (and thus computed) part of e, defined as $|\perp| = \perp$, $|X| = X$, $c(e_1, \ldots, e_n) = c(|e_1|, \ldots, |e_n|)$, $|f(e_1, \ldots, e_n)| = \perp$ (for $X \in \mathcal{V}, c \in CS, f \in FS$). We also define the denotation of $e \in Exp$ under term rewriting as $[\![e]\!]^{rw} = \{t \in CTerm_\perp \mid \exists e' \in Exp \,.\, e \rightarrow^* e' \wedge t \sqsubseteq |e'|\}$. In a previous joint work the author explored the relation between $CRWL$ and term rewriting ([12], Theorem 9), recast in the following theorem:

THEOREM 10. *For any CRWL-program $\mathcal{P}$, $e \in Exp$, $[\![e]\!]^{sg} \subseteq [\![e]\!]^{rw}$. The converse inclusion does not hold in general.*

As we saw in Example 1, in general call-time choice semantics like $CRWL$ produce less results than run-time choice semantics like the one induced by term rewriting. We will see that this kind of relation also holds for term rewriting and $\pi CRWL$.

THEOREM 11. *For any CRWL-program $\mathcal{P}$, $e \in Exp$, $[\![e]\!]^{rw} \subseteq [\![e]\!]^{pl}$. The converse inclusion does not hold in general.*

The key for proving Theorem 11 is a lemma stating that $\forall e, e' \in Exp$ if $e \rightarrow e'$ then $[\![e']\!]^{pl} \subseteq [\![e]\!]^{pl}$, that is, that every rewriting step is sound wrt. $\pi CRWL$. The evident corollary for these theorems is the announced inclusion chain.

COROLLARY 12. *For any CRWL-program $\mathcal{P}$, $e \in Exp$, $[\![e]\!]^{sg} \subseteq [\![e]\!]^{rw} \subseteq [\![e]\!]^{pl}$. Hence $\forall t \in CTerm$, $\vdash_{CRWL} e \twoheadrightarrow t$ implies $e \rightarrow^* t$ which implies $\vdash_{\pi CRWL} e \twoheadrightarrow t$.*

5 Simulating plural semantics

In [12, 13] it was shown that neither $CRWL$ can be simulated by term rewriting with a simple program transformation, nor vice versa. Nevertheless, plural semantics can be simulated by rewriting using the transformation presented in the current section, which could be used as the basis for a first implementation of $\pi CRWL$ that we might use for experimentation. First we will present a naive version of this transformation, and show its adequacy; later we will propose some simple optimizations for it.

5.1 A simple transformation

DEFINITION 13. *Given a CRWL-program P, for every rule* $(f(p_1,\dots,p_n) \to r) \in \mathcal{P}$ *such that* $f \notin \{\_?\_, if\_then\_\}$ *we define its transformation as:*

$$pST(f(p_1,\dots,p_n) \to r) = f(Y_1,\dots,Y_n) \to if\ match(Y_1,\dots,Y_n)\ then\ r[\overline{X_{ij}/project_{ij}(Y_i)}]$$

- $\forall i \in \{1,\dots,n\}$, $\{X_{i1},\dots,X_{ik_i}\} = var(p_i) \cap var(r)$ *and* $Y_i \in \mathcal{V}$ *is fresh.*
- $match \in FS^n$ *fresh is defined by the rule* $match(p_1,\dots,p_n) \to true$.
- *Each* $project_{ij} \in FS^1$ *is a fresh symbol defined by the single rule* $project_{ij}(p_i) \to X_{ij}$.

For $f \in \{\_?\_, if\_then\_\}$ *the transformation leaves its rules untouched.*

The function $match$ is used to impose a "guard" that enforces the matching of each argument with its corresponding pattern. If we dropped this condition the translation of, for example, to rule $(null(nil) \to true)$, would be $(null(Y) \to true)$, which is clearly unsound as then $null(0 : nil) \to true$. Besides each pattern p_i has been replaced by a fresh variable Y_i, to which any expression can match, hence the arguments may be replicated freely by the rewriting process without demanding any evaluation and thus keeping its denotation untouched: this is the key to achieve completeness wrt. $\pi CRWL$. Later on, the functions $project_{ij}$ will just make the projection of each variable when needed.

EXAMPLE 14. *Applying this to Example 1 we get* $\{f(Y) \to if\ match(Y)\ then\ d(project(Y), project(Y)), match(c(X)) \to true, project(c(X)) \to X\}$ *, under which we can do:*

$$\underline{f(c(0)?c(1))} \to if\ \underline{match(c(0)?c(1))}\ then\ d(project(c(0)?c(1)), project(c(0)?c(1)))$$
$$\to^* \underline{if\ true\ then\ d(project(c(0)?c(1)), project(c(0)?c(1)))}$$
$$\to d(project(\underline{c(0)?c(1)}), project(\underline{c(0)?c(1)})) \to^* d(\underline{project(c(0))}, \underline{project(c(1))}) \to^* d(0,1)$$

Concerning the adequacy of the transformation:

THEOREM 15. *For any CRWL-program* $\mathcal{P}$, $e \in Exp_\perp$ *built up on the signature of* $\mathcal{P}$, $[\![e]\!]^{pl}_{pST(\mathcal{P})} \subseteq [\![e]\!]^{pl}_{\mathcal{P}}$.

THEOREM 16. *For any CRWL-program* $\mathcal{P}$, $e \in Exp, t \in CTerm_\perp$ *built up on the signature of* $\mathcal{P}$, *if* $\mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ *then exists some* $e' \in Exp$ *built using symbols of the signature of* $pST(\mathcal{P})$ *such that* $pST(\mathcal{P}) \vdash e \to^* e'$ *and* $t \sqsubseteq |e'|$.

COROLLARY 17. *For any CRWL-program* $\mathcal{P}$, $e \in Exp$ *built using symbols of the signature of* $\mathcal{P}$, $[\![e]\!]^{pl}_{\mathcal{P}} = [\![e]\!]^{rw}_{pST(\mathcal{P})}$. *Hence* $\forall t \in CTerm\ \mathcal{P} \vdash_{\pi CRWL} e \twoheadrightarrow t$ *iff* $pST(\mathcal{P}) \vdash e \to^* t$.

5.2 An optimized transformation

Concerning the transformation, if a pattern is ground then no parameter passing will be done for it and so no transformation is needed: for $null(nil) \to true$ we get $\{null(Y) \to if\ match(Y)\ then\ true, match(nil) \to true\}$, which is equivalent. Besides, if the pattern is a variable then any expression matches it and the projection functions are trivial, so no transformation is needed neither, as happens with $pair(X) \to (X,X)$ for which $\{pair(Y) \to if\ match(Y)\ then\ (project(Y), project(Y)), match(X) \to true, project(X) \to X\}$ are returned.

DEFINITION 18.*Given a CRWL-program P, for every rule* $(f(p_1,\dots,p_n) \to r) \in \mathcal{P}$ *we define its transformation as:*

$$pST(f(p_1,\dots,p_n) \to r)$$
$$= \begin{cases} f(p_1,\dots,p_n) \to r & \text{if } \rho_1\dots\rho_m \text{ is empty} \\ f(\tau(p_1),\dots,\tau(p_n)) \to \begin{array}{l} if\ match(Y_1,\dots,Y_m) \\ \quad then\ r[\overline{X_{ij}/project_{ij}(Y_i)}] \end{array} & \text{otherwise} \end{cases}$$

where $\rho_1\dots\rho_m = p_1\dots p_n \mid \lambda p.(p \notin \mathcal{V} \wedge var(p) \neq \emptyset)$.
- $\forall \rho_i$, $\{X_{i1},\dots,X_{ik_i}\} = var(\rho_i) \cap var(r)$ *and* $Y_i \in \mathcal{V}$ *is fresh.*
- $\tau : CTerm \to CTerm$ *is defined by* $\tau(p) = p$ *if* $p \in \mathcal{V} \vee var(p) = \emptyset$ *and* $\tau(p) = Y_i$ *otherwise, for* $p \equiv \rho_i$.
- $match \in FS^m$ *fresh is defined by the rule* $match(\rho_1,\dots,\rho_m) \to true$.
- Each $project_{ij} \in FS^1$ *is a fresh symbol defined by the single rule* $project_{ij}(\rho_i) \to X_{ij}$.

We will not give a formal proof for the adequacy of the optimization. Nevertheless note how this transformation leaves untouched the rules for $\_?\_$ and $if\_then\_$ without defining an special case for them. As the simple transformation worked well for that rules that suggests that we are doing the right thing. We end this section with an example application of the optimized transformation, over the program of Example 8:

EXAMPLE 19. *The only rule modified is the one for* $find$*:* $\{find(Y) \to if\ match(Y)\ then\ (project(Y), project(Y)), match(e(N,s,clerk)) \to true, project(e(N,s,clerk)) \to N\}$ *so:*

$$\begin{array}{l}
\underline{twoclerks} \to \underline{find(employees(branches))} \\
\to if\ match(\underline{employees(branches)})\ then\ (project(employees(branches)), project(employees(branches))) \\
\to^* \underline{if\ match(e(pepe,men,clerk))\ then\ (project(employees(branches)), project(employees(branches)))} \\
\to^* (project(\underline{employees(branches)}), project(\underline{employees(branches)})) \\
\to^* (\underline{project(e(pepe,men,clerk))}, \underline{project(e(maria,women,clerk))}) \to^* (pepe, maria)
\end{array}$$

6 Conclusions

In this work we have pointed the different interpretations of run-time choice and plural semantics caused by pattern matching. To the best of our knowledge this distinction is stablished in the present paper for the first time, because in [20] no pattern matching was present and in [11] only call-time choice was adopted. We argue that the run-time choice semantics induced by term rewriting is not the best option for a value-based programming language like current implementations of FLP. For that context a plural semantics has been proposed for which the compositionality properties lost when turning from call-time choice to rewriting are recovered. Nevertheless, for other kind of rewriting based languages like Maude, which are not limited to constructor-based TRS's, term rewriting has been proven to be an effective formalism.

Our concrete contributions can be summarized as follows:
- We have presented the proof calculus $\pi CRWL$, a novel formulation of plural semantics for left-linear constructor-based TRS's, which are the kind of TRS's used in FLP. Some basic properties of the new semantics have been stated and proved, and by some examples we have shown how it allows natural encodings of some programs that need to do some collecting work (Sect. 3).

- We have compared the new calculus with *CRWL* and term rewriting, which are standard formulations for call-time choice and run-time choice respectively. The different properties of these calculi have been discussed and the inclusion chain *CRWL* $\subseteq$ rewriting $\subseteq$ π*CRWL* has been proved (Sect. 4).
- We have recalled some previous results about the impossibility of a straight simulation of *CRWL* in term rewriting or viceversa by a simple program transformation. Besides we have proposed a novel program transformation to simulate plural semantics with term rewriting, and proved its adequacy (Sect. 5).

From a practical point of view, it might be unrealistic to think that a monolithic semantic view is adequate for addressing all non-determinism present in a large program. In [13] we have started to investigate the combination of call-time choice and run-time choice in a unified framework. But as π*CRWL* seems to be more suitable than run-time choice for a value-based language, we are planning to extend that work to plural semantics.

We contemplate other relevant subjects of future work:
- Extending the current results to programs with extra variables, that is, with rules $l \rightarrow r$ in which $var(r) \subseteq var(l)$ does not hold in general. We should also deal with conditional rules and equality constraints to cover the basic features of FLP languages.
- Studying the relation between the determinism of programs under *CRWL* [12] and π*CRWL*, which we conjecture is equivalent. We also conjecture that for deterministic programs $\forall e \in Exp, [\![e]\!]^{sg} = [\![e]\!]^{rw} = [\![e]\!]^{pl}$. Getting results about the relation of confluence and determinism of programs could be useful for analyzing the confluence of a TRS through its determinism. In the same line, the inclusion chain *CRWL* $\subseteq$ rewriting $\subseteq$ π*CRWL* could be used to study the termination of a TRS through its termination in *CRWL* and π*CRWL*.
- Developing a more operational rewrite notion for π*CRWL* in the line of [12], which could be extended to narrowing like in [14]. A complexity study would be needed to ensure that the extra nondeterminism does not preclude the design of an efficient implementation. On the other hand the natural value for π*CRWL* seems to be $\mathcal{P}(CTerm_\perp)$ instead of $CTerm_\perp$, a formulation in the line of [16] could be useful to forget about the tricky use of _?_.
- Finally, for the immediate future, it could be interesting implementing the transformation to simulate π*CRWL* in some term rewriting based language like Maude [5]. Maybe the context-sensitive rewriting [18] features of Maude could be used to improve the laziness of the transformed program like in [17]. Besides, the matching-module capacities of Maude could be used to enhance the expressivity of plural semantics.

Acknowledgements: The author would like to thank Paco López Fraguas and Jaime Sánchez Hernández for their support and their useful suggestions. I would also like to thank the referees for their very valuable comments.

References

[1] S. Antoy. Optimal non-deterministic functional logic computations. In *Proc. ALP'97*, pages 16–30. Springer LNCS 1298, 1997.

[2] S. Antoy, D. Brown, and S. Chiang. Lazy context cloning for non-deterministic graph rewriting. In Proc. Termgraph'06, pages 61–70. ENTCS, 176(1), 2007.

[3] S. Antoy and M. Hanus. Functional logic design patterns. In Proc. FLOPS'02, pages 67–87. Springer LNCS 2441, 2002.

[4] F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, United Kingdom, 1998.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Springer LNCS 4350, 2007

[6] M. Clavel, M. Palomino, and A. Riesco. Introducing the itp tool: a tutorial. J. UCS 12(11), pages 1618–1650, 2006.

[7] R. Diaconescu and K. Futatsugi. An overview of CafeOBJ. ENTCS 15,1998.

[8] J. C. González-Moreno, T. Hortalá-González, F. López-Fraguas, and M. Rodríguez-Artalejo. An approach to declarative programming based on a rewriting logic. J. Log. Program. 40(1), pages 47–87, 1999.

[9] M. Hanus. The integration of functions into logic programming: From theory to practice. J. Log. Program. 19/20, pages 583–628, 1994.

[10] M. Hanus (ed.). Curry: An integrated functional logic language (version 0.8.2). Available at *http://www.informatik.uni-kiel.de/~curry/report.html*, March 2006.

[11] H. Hussmann. Non-Determinism in Algebraic Specifications and Algebraic Programs. Birkhäuser Verlag, 1993.

[12] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A simple rewrite notion for call-time choice semantics. In Proc. PPDP'07, pages 197–208. ACM Press, 2007.

[13] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. A flexible framework for programming with non-deterministic functions (Extended version). Tech. Rep. SIC-9-08, Universidad Complutense de Madrid, 2008. `http://gpd.sip.ucm.es/juanrh/pubs/tchrRTCT08.pdf`.

[14] F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. Rewriting and call-time choice: the HO case. In Proc. FLOPS'08, pages 147–162. Springer LNCS 4989, 2008.

[15] F. López-Fraguas and J. Sánchez-Hernández. $\mathcal{TOY}$: A multiparadigm declarative system. In Proc. RTA'99, pages 244–247. Springer LNCS 1631, 1999.

[16] F. López-Fraguas and J. Sánchez-Hernández. A proof theoretic approach to failure in functional logic programming. TPLP, 4(1&2), pages 41–74, 2004.

[17] S. Lucas. Needed reductions with context-sensitive rewriting. In Proc. ALP/HOA'97, pages 129–143. Springer LNCS 1298, 1997.

[18] S. Lucas. Context-sensitive computations in functional and functional logic programs. J. Fun. Log. Program 1998(1), 1998.

[19] J. Rodríguez-Hortalá. A Hierarchy of Semantics for Non-deterministic Term Rewriting Systems (Extended version). Tech. Rep. SIC-10-08, Universidad Complutense de Madrid, 2008. `http://gpd.sip.ucm.es/juanrh/pubs/tchrFSTTCS08.pdf`.

[20] H. Søndergaard and P. Sestoft. Non-determinism in functional languages. The Computer Journal 35(5), pages 514–523, 1992.

[21] M. Wenzel. The isabelle/isar reference manual. `http://isabelle.in.tum.de/dist/Isabelle99-2/doc/isar-ref.pdf`.