A Rule-based Approach to the Implementation of Evaluation Strategies

Mircea Marin[1] and Temur Kutsia[2⋆]

[1] Johann Radon Institute for Computational and Applied Mathematics
Austrian Academy of Sciences
A-4040 Linz, Austria
Mircea.Marin@oeaw.ac.at

[2] Research Institute for Symbolic Computation
Johannes Kepler University
A-4232 Hagenberg, Austria
Temur.Kutsia@risc.uni-linz.ac.at

Abstract. We describe a new methodology to program evaluation strategies, which relies on some advanced features of the rule-based programming language ρLog. We illustrate how our approach works for a number of important evaluation strategies.

1 Introduction

We address the problem of programming evaluation mechanisms which can be expressed as sequences of transformation steps which follow a given strategy. Roughly speaking, a strategy prescribes the order how the transformation steps are applied, and the (sub)expressions on which they act. Since programming and applying such strategies is a topic of rule-based programming, it is natural to ask ourselves how suitable rule-based programming is for this purpose.

Recently, we proposed a rule-based programming language called ρLog [1, 8]. The language was conceived mainly as a convenient tool for programming deduction systems, but we realized soon that it's advanced pattern matching capabilities provide an elegant mechanism for programming evaluation strategies. This paper constitutes our first systematic account to this observation.

To be concrete, we proceed as follows:

- we consider the following well-known evaluation strategies: innermost, outermost, parallel innermost and parallel outermost rewriting, and some evaluation mechanisms of the pure lambda calculus, and
- we illustrate how these evaluation strategies can be specified in the rule-based programming style of ρLog [8].

In Section 2 we give a brief account to the notions of rule-based programming and to the reasons why we chose ρLog as illustrative rule-based system. Section 3 indicates how the evaluation strategies mentioned before can be specified in ρLog. Section 4 concludes.

⋆ Temur Kutsia has been supported by the Austrian Science Foundation FWF under the project SFB 1302.

2 Preliminaries

Rule-based programming emerged as a practice of AI to the design of expert systems [10] or knowledge based systems. The traditional development of such a system follows a pattern which flows from a tightly defined problem scope to a system specification. Then, through the clever use of algorithms and data structures, a workable system emerges to handle the specified problem. Several characteristics of these programs are noteworthy:

- the flow of control is procedural,
- each possible variation or case in the problem domain has to be modelled in conditional logic statements and loops in the program,
- given even a small variation in the problems that might be encountered, the conditional logic involved in handling these grows combinatorially,
- maintaining the integrity of these systems is problematic.

In the AI community, the rule-based programming style offers a different approach to the development of complex, knowledge intensive systems. Many of the problems of procedural programming are avoided because:

- one does not construct or maintain explosive amounts of conditional logic,
- independent statements of "if-then" rules are automatically integrated into a single, efficient conditional logic,
- the logic is automatically maintained as conditions of rules are changed, added and removed from a potentially very large collection of rules.

Since the basic concepts of rule-based programming are so pervasive in computer science, the rule-based programming style became a field of research on its own, which is experiencing a period of growth with the emergence of new concepts and systems that allow a better understanding and usability. According to the community of researchers involved in this kind of research:

> The rule-based programming paradigm is characterized by the repeated, localized transformation of a shared data object such as a term, graph, proof, or constraint store. The transformations are described by rules which separate the description of the sub-object to be replaced (the pattern) from the calculation of the replacement. Optionally, rules can have further conditions that restrict their applicability. The transformations are controlled by explicit or implicit strategies. [2]

In this paper we analyze the suitability of rule-based programming for implementing various evaluation strategies. In order to be more specific and illustrate how these strategies can be implemented, we consider the system ρLog [1, 8].

2.1 The ρLog Language

ρLog has the following remarkable features:

1. It is a based on a calculus for higher-order rule-based programming with function variables and sequence variables [7]: function variables are placeholders for functional terms, and sequence variables are placeholders for (possibly empty) sequences of expressions. We will show that both kinds of variables are essential for giving concise specifications of rewrite strategies,
2. It supports the following operators for building strategies: non-deterministic choice, sequential composition, repetition, committed choice, and normal form computation,
3. It is implemented in MATHEMATICA [11], and thus it inherits all the pattern matching and computing capabilities of MATHEMATICA.

The set of rules induced by a set $\mathcal{B}$ of basic rules is defined by the grammar

$$
\begin{array}{lll}
\ell ::= & & \textit{rules:} \\
 & \mid b & \text{basic rule} \\
 & \mid \ell_1 \circ \ell_2 & \text{composition} \\
 & \mid \ell_1 \mid \ldots \mid \ell_n & \text{nondeterministic choice} \\
 & \mid \texttt{Fst}[\ell_1, \ell_2] & \text{or else choice} \\
 & \mid \texttt{Repeat}[\ell_1, \ell_2] & \text{repetition} \\
 & \mid \texttt{Until}[\ell_1, \ell_2] & \text{repetition} \\
 & \mid \texttt{NF}[\ell_1] & \text{normalization}
\end{array}
$$

where $b \in \mathcal{B}$. Each rule ℓ identifies a reduction relation $\to_\ell$ on expressions, where the expressions are defined over a signature of function symbols and a set $\mathcal{V} = \mathcal{V}_o \cup \mathcal{V}_s$ of variables, like in the language of MATHEMATICA. We distinguish between the set $\mathcal{V}_o$ of *ordinary* variables and the set $\mathcal{V}_s$ of *sequence* variables, and write $vars(E)$ for the set of variables which occur in an expression E.

We distinguish three kinds of atomic formulas: *reducibility formulas* $E \to_\ell F$; *equalities* $E \equiv F$; all the other atomic formulas, called *atomic constraints*. An *irreducibility formula* is of the form $\nexists_F.(E \to_\ell F)$ with $F \notin vars(E)$, abbreviated $E \not\to_\ell$. An *elementary formula* is either an atomic formula or an irreducibility formula. The existential (resp. universal) closure of a formula F is denoted by $\exists F$ (resp. $\forall F$), and the reflexive-transitive closure of $\to_\ell$ is denoted by $\to_\ell^*$.

The following definition will play an important role in the rest of this paper.

Definition 1 (X-admissibility). *Let X be a set of variables and $F = cond_1 \wedge \ldots \wedge cond_p$ a conjunction of elementary formulas. The formula F is* X-admissible *if it satisfies the following restrictions for all $1 \leq i \leq p$:*

- *if $cond_i$ is $s_i \to_{\ell_i} t_i$ or $s_i \equiv t_i$ or $s_i \not\to_{\ell_i}$ then $vars(s_i) \subseteq X \cup \bigcup_{k=1}^{i-1} vars(cond_k)$,*
- *if $cond_i$ is an atomic constraint then $vars(cond_i) \subseteq X \cup \bigcup_{k=1}^{i-1} vars(cond_k)$.*

Basic rules. The logic semantics of $\equiv$ is defined by the clause $\forall x.x \equiv x$. The logic semantics of the binary relation $\to_b$ corresponding to a rule $b \in \mathcal{B}$ is defined by a clause $\mathcal{C}$ of the form

$$\forall\ (cond_1 \wedge \ldots \wedge cond_p \Rightarrow s \to_b t) \tag{1}$$

where $cond_1, \ldots, cond_p$ ($p \geq 0$) are atomic formulas. For computational purposes, ρLog imposes the following restrictions on the syntactic structure of (1): $cond_1 \wedge \ldots \wedge cond_p$ is $vars(s)$-admissible, and $vars(t) \subseteq vars(s) \cup \bigcup_{k=1}^{p} vars(cond_k)$.

Thus, the clauses which define the predicates $\rightarrow_b$ for $b \in \mathcal{B}$ can be regarded as general logic clauses in a higher-order language of terms.

Similar to logic programming, we define *rule-based programming* by specifying

1. a *goal* (or a *query*) as a formula $\exists F$ with F an $\emptyset$-admissible formula,
2. a *rule-based program* as a set $\mathcal{P}$ of clauses of form (1). It is assumed that $\mathcal{P}$ provides definitions for all the basic rules used in the goal,
3. *rule operators* with built-in meanings for reducibility formulas.

The clauses of $\mathcal{P}$ are fetched into a ρLog session via successive calls of the `DeclareRule` method. A clause $\mathcal{C}$ of the form (1) is declared by a call

$\texttt{DeclareRule}[\mathcal{C}^{\texttt{Mma}}]$

where $\mathcal{C}^{\texttt{Mma}}$ is the MATHEMATICA encoding of $\mathcal{C}$ which is defined as follows: suppose $\mathcal{C} = \forall \mathcal{C}_0$ where $\mathcal{C}_0$ is of the form

$$F_1 \wedge (s_1 \rightarrow_{\ell_1} t_1) \wedge F_2 \wedge \ldots \wedge F_p \wedge (s_p \rightarrow_{\ell_p} t_p) \wedge F_{p+1} \Rightarrow (t_0 \rightarrow_b t)$$

such that F_i ($1 \leq i \leq p+1$) are possibly empty conjunctions of formulas which are elementary and not reducible. Let

$$\mathcal{C}_0^{\texttt{M}} = t_0^{\texttt{M}} \rightarrow_b t /; F_1 \wedge (s_1 \rightarrow_{\ell_1} t_1^{\texttt{M}}) \wedge F_2 \wedge \ldots \wedge F_p \wedge (s_p \rightarrow_{\ell_p} t_p^{\texttt{M}}) \wedge F_{p+1}$$

where each $t_i^{\texttt{M}}$ is obtained from t_i by replacing all occurrences of variables from $vars(t_i) \setminus \bigcup_{j=0}^{i-1} vars(t_j)$ with corresponding MATHEMATICA patterns. Then

$$\mathcal{C}^{\texttt{Mma}} = \begin{cases} \texttt{ForAll}[\{x_1, \ldots, x_m\}, \mathcal{C}_0^{\texttt{M}}] & \text{if } vars(\mathcal{C}) \setminus vars(t_0) = \{x_1, \ldots, x_m\} \neq \emptyset, \\ \mathcal{C}_0^{\texttt{M}} & \text{otherwise.} \end{cases}$$

For example, if $\mathcal{C}$ is

$$\forall_{\overline{x}, y, z, t}. (g[\overline{x}] \rightarrow_{\texttt{"b1"}} y) \wedge (y > 0) \wedge (h[y] \rightarrow_{\texttt{"b2"}} u[z, t]) \Rightarrow f[\overline{x}] \rightarrow_{\texttt{"a"}} t$$

where $\overline{x}$ is a sequence variable and y, z, t are ordinary variables, then $\mathcal{C}^{\texttt{Mma}}$ is:[3]

$$\texttt{ForAll}[\{y, z, t\}, f[x\_\_\_] \rightarrow_{\texttt{"a"}} t /; (g[x] \rightarrow_{\texttt{"b1"}} y\_) \wedge (y > 0) \wedge (h[y] \rightarrow_{\texttt{"b2"}} u[z\_, t\_])].$$

A query $\mathcal{G}$ has the MATHEMATICA encoding $\mathcal{G}^{\texttt{Mma}}$ defined as follows: suppose $\mathcal{G} = \exists \mathcal{G}_0$ where $\mathcal{G}_0$ is of the form

$$F_1 \wedge (s_1 \rightarrow_{\ell_1} t_1) \wedge F_2 \wedge \ldots \wedge F_p \wedge (s_p \rightarrow_{\ell_p} t_p) \wedge F_{p+1}$$

such that F_i ($1 \leq i \leq p+1$) are possibly empty conjunctions of formulas which are elementary and not reducible. Let

$$\mathcal{G}_0^{\texttt{M}} = F_1 \wedge (s_1 \rightarrow_{\ell_1} t_1^{\texttt{M}}) \wedge F_2 \wedge \ldots \wedge F_p \wedge (s_p \rightarrow_{\ell_p} t_p^{\texttt{M}}) \wedge F_{p+1}$$

[3] In MATHEMATICA patterns, a sequence variable x is written $x\_\_\_$, and an ordinary variable y is written $y\_$.

where each $t_i^{\mathtt{M}}$ is obtained from t_i by replacing all occurrences of variables from $vars(t_i) \setminus \bigcup_{j=1}^{i-1} vars(t_j)$ with corresponding MATHEMATICA patterns. Then

$$\mathcal{G}^{\mathtt{Mma}} = \begin{cases} \mathtt{Exists}[\{x_1, \ldots, x_m\}, \mathcal{G}_0^{\mathtt{M}}] & \text{if } vars(\mathcal{G}_0) = \{x_1, \ldots, x_m\} \neq \emptyset, \\ \mathcal{G}_0^{\mathtt{M}} & \text{otherwise.} \end{cases}$$

Composed rules (strategies). ρLog recognizes six rule operators. Their logic semantics is shown below:

1. $\forall\ (E \rightarrow_{\ell_1|\ldots|\ell_n} F \Leftrightarrow (E \rightarrow_{\ell_1} F \vee \ldots \vee E \rightarrow_{\ell_n} F))$,
2. $\forall\ (E \rightarrow_{\ell_1 \circ \ell_2} F \Leftrightarrow \exists_G.(E \rightarrow_{\ell_1} G \wedge G \rightarrow_{\ell_2} F))$,
3. $\forall\ (E \rightarrow_{\mathtt{Fst}[\ell_1,\ell_2]} F \Leftrightarrow (E \rightarrow_{\ell_1} F \vee (E \nrightarrow_{\ell_1} \wedge E \rightarrow_{\ell_2} F)))$,
4. $\forall\ (E \rightarrow_{\mathtt{Repeat}[\ell_1,\ell_2]} F \Leftrightarrow \exists_G.(E \rightarrow_{\ell_1}^{*} G \wedge G \rightarrow_{\ell_2} F))$,
5. $\forall\ (E \rightarrow_{\mathtt{Until}[\ell_2,\ell_1]} F \Leftrightarrow \exists_G.(E \rightarrow_{\ell_1}^{*} G \wedge G \rightarrow_{\ell_2} F))$,
6. $\forall\ (E \rightarrow_{\mathtt{NF}[\ell]} F \Leftrightarrow E \rightarrow_{\ell}^{*} F \wedge F \nrightarrow_{\ell})$.

Queries. ρLog provides means to decide the validity of a goal $\mathcal{G}$ with respect to a rule-based program $\mathcal{P}$. Suppose $\mathcal{G} = \exists\mathcal{G}_0$ and $vars(\mathcal{G}_0) = \{x_1, \ldots, x_n\}$. Then we can call one of the following methods:

1. $\mathtt{Query}[\mathtt{Exists}[\{x_1, \ldots, x_n\}, \mathcal{G}_0^{\mathtt{M}}]]$;
   this call computes a pair $\{\mathtt{True}, \{\langle v_1\rangle, \ldots, \langle v_n\rangle\}\}$ which indicates the fact that the substitution $\theta = \{x_1 \rightarrow v_1, \ldots, x_n \rightarrow v_n\}$ is a solution, i.e. the ground formula $\mathcal{G}_0\theta$ holds.[4] Otherwise, the call yields `False`. Note that some x_i may be sequence variables, and then v_i may be sequences of expressions. In order to avoid parsing ambiguities, `Query` yields a result in which each v_i is displayed between $\langle$ and $\rangle$.
2. $\mathtt{QueryAll}[\mathtt{Exists}[\{x_1, \ldots, x_n\}, \mathcal{G}_0^{\mathtt{M}}]]$;
   the call produces an enumeration $\{\tau_1, \tau_2, \ldots\}$ with $\tau_i = \{\langle v_{i,1}\rangle, \ldots, \langle v_{i,n}\rangle\}$ such that $\{\tau_1^{\mathtt{M}}, \tau_2^{\mathtt{M}}, \ldots\}$ with $\tau_i^{\mathtt{M}} = \{x_1 \rightarrow v_{i,1}, \ldots, x_n \rightarrow v_{i,n}\}$ is a complete set of solutions of $\mathcal{G}$.
   The call runs forever if the query has an infinite number of solutions. If $\mathtt{QueryAll}[\mathtt{Exists}[\{x_1, \ldots, x_n\}, \mathcal{G}_0^{\mathtt{M}}]]$ yields a non-empty list $\{\tau_1, \ldots\}$ then $\mathtt{Query}[\mathtt{Exists}[\{x_1, \ldots, x_n\}, \mathcal{G}_0^{\mathtt{M}}]]$ yields $\{\mathtt{True}, \tau_1\}$.
3. $\mathtt{ApplyRuleList}[E, \ell]$;
   where E is a ground expression.
   This call computes the enumeration $\{x\tau_1^{\mathtt{M}}, x\tau_2^{\mathtt{M}}, \ldots\}$ where $\{\tau_1, \tau_2, \ldots\}$ is the enumeration computed by $\mathtt{QueryAll}[\mathtt{Exists}[\{x\}, E \rightarrow_{\ell} x\_]]$. In other words, this call computes an enumeration of all expressions F for which $E \rightarrow_{\ell} F$.
4. $\mathtt{ApplyRule}[E, \ell]$;
   where E is a ground expression. This call yields E if $E \nrightarrow_{\ell}$. Otherwise, it yields the first F enumerated by $\mathtt{ApplyRuleList}[E, \ell]$ such that $E \rightarrow_{\ell} F$.

[4] We write $\mathcal{G}_0\theta$ for the simultaneous substitution of the occurrences of variables x_i in $\mathcal{G}_0$ by $\theta(x_i)$.

Enumeration strategies. For programming purposes, it is important to know what is the result returned by `Query` or `ApplyRule`. In the sequel we will write $E \rightharpoonup_\ell E'$ iff $E \rightarrow_\ell E'$ and E' is the result computed by the call $\texttt{ApplyRule}[E, \ell]$. According to what we said before, the following conditions are equivalent:

(1) $E \rightharpoonup_\ell E'$,
(2) $E \rightarrow_\ell E'$ and $E' = \texttt{ApplyRule}[E, \ell]$,
(3) E' is the first element of the list computed by the call $\texttt{ApplyRuleList}[E, \ell]$,
(4) $E' = \tau_1^{\texttt{M}}(x)$ where τ_1 is the first element of the list computed by the call $\texttt{QueryAll}[\texttt{Exists}[\{x\}, E \rightarrow_\ell x\_]]$.

Hence, in order to specify the relation $\rightharpoonup_\ell$, it is enough to specify the enumeration $\mathcal{E}_\mathcal{G} = \{\tau_1^{\texttt{M}}, \tau_2^{\texttt{M}}, \ldots\}$ for $\mathcal{G} = \exists \mathcal{G}_0$ with $vars(\mathcal{G}_0) = \{x_1, \ldots, x_n\}$ and $\mathcal{G}_0 = cond_1 \wedge \ldots \wedge cond_p$, where $\texttt{QueryAll}[\mathcal{G}^{\texttt{Mma}}] = \{\tau_1, \tau_2, \ldots\}$:

- If $cond_1$ is an atomic constraint or an irreducible formula then $cond_1$ is a ground formula because $cond_1 \wedge \ldots \wedge cond_p$ is $\emptyset$-admissible.
  - If $cond_1$ is invalid then $\mathcal{E}_\mathcal{G} = \{\}$.
  - Otherwise $cond_1$ is valid and $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\exists(cond_2 \wedge \ldots \wedge cond_p)}$.
- Otherwise, if $cond_1$ is an equality $E \equiv F$ then E is necessarily ground. Let $\theta_1, \theta_2, \ldots$ be the Mathematica enumeration of all matchers between E and F, and $\mathcal{G}_i = \exists(cond_2\theta_i \wedge \ldots \wedge cond_p\theta_i)$. Then $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\mathcal{G}_1}^{\theta_1} \cup \mathcal{E}_{\mathcal{G}_2}^{\theta_2} \cup \ldots$ where $\mathcal{E}_{\mathcal{G}_i}^{\theta_i} = \{\gamma\theta_i \mid \gamma \in \mathcal{E}_{\mathcal{G}_i}\}$.
- Otherwise $cond_1$ is a reducibility formula $E \rightarrow_\ell F$ with $vars(E) = \emptyset$.
  1. If $\ell = b \in \mathcal{B}$ and $\mathcal{P}$ contains the clause $\forall\ (cond'_1 \wedge \ldots \wedge cond'_m \Rightarrow s \rightarrow_b t)$ then let $\theta_1, \theta_2, \ldots$ be the Mathematica enumeration of the matchers between E and s, and $\mathcal{G}_i = \exists(cond'_1\theta_i \wedge \ldots \wedge cond'_m\theta_i \wedge (t\theta \equiv F) \wedge cond_2 \wedge \ldots \wedge cond_p)$. Then $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\mathcal{G}_1}^{\theta_1} \cup \mathcal{E}_{\mathcal{G}_2}^{\theta_2} \cup \ldots$
  2. If $\ell = \ell_1 \mid \ldots \mid \ell_n$ then $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\mathcal{G}_1} \cup \ldots \cup \mathcal{E}_{\mathcal{G}_n}$ where $\mathcal{G}_i = \exists((E \rightarrow_{\ell_i} F) \wedge cond_2 \wedge \ldots \wedge cond_p)$.
  3. If $\ell = \ell_1 \circ \ell_2$ then $\mathcal{E}_\mathcal{G} = \{\gamma\restriction_{vars(\mathcal{G})} \mid \gamma \in \mathcal{E}_{\mathcal{G}_1}\}$ where $\mathcal{G}_1 = \exists(E \rightarrow_{\ell_1} X \wedge X \rightarrow_{\ell_2} F \wedge cond_2 \wedge \ldots \wedge cond_p)$ with X is a fresh ordinary variable.
  4. If $\ell = \texttt{Repeat}[\ell_1, \ell_2]$ then we consider the rules $\ell'_1, \ell'_2, \ldots$ where $\ell'_i = (\ell_1 \circ \ell'_{i+1}) \mid \ell_2$ for all $i \in \mathbb{N}$. Then $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\exists(E \rightarrow_{\ell'_1} F \wedge cond_2 \wedge \ldots \wedge cond_p)}$.
  5. If $\ell = \texttt{Until}[\ell_2, \ell_1]$ then let $\ell'_1, \ell'_2, \ldots$ be the rules defined by $\ell'_i = \ell_2 \mid (\ell_1 \circ \ell'_{i+1})$ for all $i \in \mathbb{N}$. Then $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\exists(E \rightarrow_{\ell'_1} F \wedge cond_2 \wedge \ldots \wedge cond_p)}$.
  6. If $\ell = \texttt{Fst}[\ell_1, \ell_2]$ then we distinguish two situations:
     - $\exists_X.E \rightarrow_{\ell_1} X$ is valid. Then $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\exists(E \rightarrow_{\ell_1} F \wedge cond_2 \wedge \ldots \wedge cond_p)}$.
     - $E \not\rightarrow_{\ell_1}$ is valid. Then $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\exists(E \rightarrow_{\ell_2} F \wedge cond_2 \wedge \ldots \wedge cond_p)}$.
  7. If $\ell = \texttt{NF}[\ell_1]$ then we consider the rules $\ell'_1, \ell'_2, \ldots$ where ℓ'_0 is defined by the clause $\forall_x.(x \rightarrow_{\ell'_0} x)$ and $\ell'_i = \texttt{Fst}[\ell_1 \circ \ell'_{i+1}, \ell'_0]$ for all $i > 0$. In this case, $\mathcal{E}_\mathcal{G} = \mathcal{E}_{\exists(E \rightarrow_{\ell'_1} F \wedge cond_2 \wedge \ldots \wedge cond_p)}$.

The following facts are easy to prove:

- $s \rightharpoonup_{\ell_1 \mid \ell_2} t$ iff $s \rightharpoonup_{\ell_1} t$ or else $s \rightharpoonup_{\ell_2} t$.
- $s \rightharpoonup_{\texttt{NF}[\ell]} t$ iff $s \rightharpoonup_\ell^* t$ and $t \not\rightarrow_\ell$.

- $s \rightarrow_{\ell_1 \circ \ell_2} t$ iff $s \rightarrow_{\ell_1} u$ and $u \rightharpoonup_{\ell_2} t$, where u is the first term in the list enumerated by $\texttt{ApplyRuleList}[s, \ell_1]$ such that $u \rightarrow_{\ell_2} t$.
- $s \rightharpoonup_{\texttt{Repeat}[\ell_1,\ell_2]} t$ iff $s \rightharpoonup_{\ell_1 \circ \texttt{Repeat}[\ell_1,\ell_2]} t$ or else $s \rightharpoonup_{\ell_2} t$.
- $s \rightharpoonup_{\texttt{Until}[\ell_2,\ell_1]} t$ iff $s \rightharpoonup_{\ell_2} t$ or else $s \rightharpoonup_{\ell_1 \circ \texttt{Until}[\ell_2,\ell_1]} t$,

We illustrate the expressive power of ρLog with a problem from term rewriting:

Given a term rewriting system, *compute* the set of all its critical pairs.

To be more specific, we assume that $\mathcal{R}$ is a set of rewrite rules of the form $s \rightarrow t$ where s and t are first-order terms defined by the following grammar:

$$
\begin{array}{lll}
\mathit{Term} ::= & & \mathit{terms}: \\
 & n & \text{variable} \\
 | & f & \text{constant} \\
 | & f[\mathit{Term}_1, \ldots, \mathit{Term}_k] &
\end{array}
$$

where $n \in \mathbb{N}$ and $f \in \mathcal{F}$ is a function symbol of arity k. We assume that $\mathbb{N} \cap \mathcal{F} = \emptyset$. A term $n \in \mathbb{N}$ stands for the n-th variable. We represent the elements of $\mathcal{F}$ by MATHEMATICA symbols which have no predefined meaning. Each rule $s \rightarrow t$ satisfies the condition $\mathit{vars}(t) \subseteq \mathit{vars}(s)$, where the set $\mathit{vars}(s)$ of variables in s is defined in the usual way.

Note that, in this language, the substitutions are finite mappings of the form $\{n_1 \rightarrow t_1, \ldots, n_k \rightarrow t_k\}$ where $n_1, \ldots, n_k \in \mathbb{N}$ and $t_1, \ldots, t_k$ are terms.

A *critical pair* of $\mathcal{R}$ is a pair of terms $\langle t_1\theta, s_1[t_2]_p\theta\rangle$ for which there exist $s_1 \rightarrow t_1, s_2 \rightarrow t_2 \in \mathcal{R}$, a non-variable position p in s_1 and a most general unifier θ of $s_1|_p$ and s_2. The root position of a term is denoted by ϵ. To compute the set $\mathit{CPair}(\mathcal{R})$ of all critical pairs, it is enough to specify five rules:

1. "cp1" such that $\langle r_1, r_2\rangle \rightarrow_{\texttt{"cp1"}} \langle u, v\rangle$ iff $r_1 = s_1 \rightarrow t_1$, $r_2 = s_2 \rightarrow t_2$ with s_1, s_2 non-variable terms, p a non-variable position in s_1, $\theta = \mathit{mgu}(s_1|_p, s_2)$, $u = t_1\theta$, and $v = s_1[t_2]_p\theta$.
2. "cp2" such that $\langle r_1, r_2\rangle \rightarrow_{\texttt{"cp2"}} \langle u, v\rangle$ iff $r_1 = s_1 \rightarrow t_1$, $r_2 = s_2 \rightarrow t_2$ with s_1, s_2 non-variable terms, $p \neq \epsilon$ a non-variable position in s_2, $\theta = \mathit{mgu}(s_1, s_2|_p)$, $u = s_2[t_1]_p\theta$, and $v = t_2\theta$.
3. "narrow" such that $\langle s, l \rightarrow r\rangle \rightarrow_{\texttt{"narrow"}} \langle \theta, r\theta\rangle$ iff θ is an mgu of s and l.
4. "recN" such that $\langle s, l \rightarrow r\rangle \rightarrow_{\texttt{"recN"}} \langle \theta, u\rangle$ iff there exists a non-variable position p of s such that θ is an mgu of $s|_p$ and l and $u = s[r]_p\theta$.
5. "CP" such that $\mathcal{R} \rightarrow_{\texttt{"CP"}} \langle u, v\rangle$ iff there exist two rules r_i, r_j in the list $\mathcal{R}$ of rules such that $\langle u, v\rangle$ is a critical pair of r_i and r_j.

To simplify the presentation, we assume that the rules of $\mathcal{R}$ are renamed apart, that is, they don't share common variables.

With ρLog, the specification of the rules mentioned before is straightforward:

$\texttt{DeclareRule}[\texttt{ForAll}[\{cp\}, \{\_\_\_, \rho1\_, \_\_\_, \rho2\_, \_\_\_\} \rightarrow_{\texttt{"CP"}}$
$\quad cp/; (\langle \rho1, \rho2\rangle \rightarrow_{\texttt{"cp1"}|\texttt{"cp2"}} cp\_)]];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{\theta\}, \langle s, l\_ \rightarrow r\_\rangle \rightarrow_{\texttt{"narrow"}}$
$\quad \langle \theta, r/.\theta\rangle/; \texttt{mgu}[\langle s, l\rangle] \equiv \theta\_\texttt{List}]];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{\theta, u\}, \langle f\_[as\_\_\_, s\_, bs\_\_\_], \rho\_\rangle \rightarrow_{\texttt{"recN"}}$
$\quad \langle \theta, f[as, u, bs]/.\theta\rangle/; (\langle s, \rho\rangle \rightarrow_{\texttt{"narrow"}|\texttt{"recN"}} \langle \theta\_, u\_\rangle)]];$

$$\begin{array}{l}\texttt{DeclareRule}[\texttt{ForAll}[\{\theta, u\}, \langle l\_ \to r\_, \rho\_\rangle \to_{\texttt{"cp1"}} \\ \qquad\qquad \langle r/.\theta, u\rangle /; (\langle l, \rho\rangle \to_{\texttt{"narrow"}|\texttt{"recN"}} \langle \theta\_, u\_\rangle)]]; \\ \texttt{DeclareRule}[\texttt{ForAll}[\{\theta, u\}, \langle \rho\_, l\_ \to r\_\rangle \to_{\texttt{"cp2"}} \\ \qquad\qquad \langle u, r/.\theta\rangle /; (\langle l, \rho\rangle \to_{\texttt{"recN"}} \langle \theta\_, u\_\rangle)]];\end{array}$$

where the auxiliary function $\texttt{mgu}[s, t]$ computes the mgu of s and t if s and t are two unifiable non-variable terms, and 0 otherwise.

To illustrate how ρLog works, we consider the rewrite system[5]

$$\begin{array}{rll}\mathcal{R} = \{ & w[f[1, g[2,3]]] \to h1[1,2,3], & g[4, f[5,6]] \to h2[4,5,6], \\ & f[a,b] \to a, & g[c,d] \to c\}.\end{array}$$

$\mathcal{R}$ has three critical pairs which can be computed via the call

$\texttt{ApplyRuleList}[\mathcal{R}, \texttt{"CP"}]$

The call returns the list of critical pairs

$$\{\langle h1[1,4,f[5,6]], w[f[1,h2[4,5,6]]]\rangle,\ \langle h2[4,a,b], g[4,a]\rangle,\ \langle h1[1,c,d], w[f[1,c]]\rangle\}.$$

These critical pairs are generated by the following alternative reductions:

$$\begin{cases} \underline{w[f[1, g[4, f[5,6]]]]} \to_{\{2\to 4, 3\to f[5,6]\}, w[f[1,g[2,3]]]\to h1[1,2,3]} h1[1,4,f[5,6]], \\ w[f[1, \underline{g[4, f[5,6]]}]] \to_{\{\}, g[4,f[5,6]]\to h2[4,5,6]} w[f[1, h2[4,5,6]]]. \end{cases}$$

$$\begin{cases} \underline{g[4, f[a,b]]} \to_{\{5\to a, 6\to b\}, g[4,f[5,6]]\to h2[4,5,6]} h2[4,a,b], \\ g[4, \underline{f[a,b]}] \to_{\{\}, f[a,b]\to a} g[4,a]. \end{cases}$$

$$\begin{cases} \underline{w[f[1, g[c,d]]]} \to_{\{2\to c, 3\to d\}, w[f[1,g[2,3]]]\to h1[1,2,3]} h1[1,c,d], \\ w[f[1, \underline{g[c,d]}]] \to_{\{\}, g[c,d]\to c} w[f[1,c]]. \end{cases}$$

Thus, we succeeded to specify the set of critical pairs of a rewrite system in five lines of code! Such a concise specification is possible because of the availability of function variables and of sequence variables in the specification language of rules. For example, we employed the function variable f and the sequence variables as and bs in writing the pattern $f\_[as\_\_\_, s\_, bs\_\_\_]$ of the rule "recN".

A final remark: in this example, we did not consider the critical pairs produced by overlapping a rule with itself. To compute these critical pairs too, we can define the rule "CPext" = "CP" | "CP0" where the rule "CP0" is defined by

$\texttt{DeclareRule}[\{\_\_\_, \rho\_, \_\_\_\} \to_{\texttt{"CP0"}} cp/; (\langle \rho, \rho\rangle \to_{\texttt{"cp2"}} cp\_), \texttt{"CP0"}];$

3 Evaluation Strategies

In this section we analyze how ρLog can be used to specify various evaluation strategies. We assume that one evaluation step corresponds to performing a

[5] Recall that we represent variables by numbers.

reduction step with respect to $\rightharpoonup_\ell$ where ℓ is some rule. To illustrate, we assume $\ell = \texttt{"fst"} \mid \texttt{"fromN"}$ where the rules "fst" and "fromN" are defined by

$\texttt{DeclareRule}[\texttt{Fst}[\texttt{c}[x\_, \_]] \to_{\texttt{"fst"}} x];$
$\texttt{DeclareRule}[\texttt{from}[n\_] \to_{\texttt{"fromN"}} \texttt{c}[n, \texttt{from}[\texttt{s}[n]]]];$

and the expressions $s_1 = \texttt{Fst}[\texttt{from}[0]]$ and $s_2 = \texttt{Fst}[\texttt{c}[\texttt{Fst}[0, \texttt{s}[0]], \texttt{Fst}[\texttt{c}[\texttt{s}[0], 0]]]]$.

3.1 Innermost Rewriting

An innermost rewrite step with respect to a rule ℓ is a transformation of the form $E \to E[F]_p$ where p is the leftmost innermost position of E for which the subexpression $E|_p$ can be transformed with ℓ, $E|_p \rightharpoonup_\ell F$ and $E[F]_p$ is the result of inserting F at the position p in E. In the sequel we write $E \rightharpoonup_{\texttt{inr}[\ell]} F$ iff F is the result of applying to E an innermost rewrite step with respect to ℓ.

Innermost rewriting is the basic evaluation mechanism of several interpreters: the value of an expression E is defined as the normal form of E with respect to the reduction relation $\rightharpoonup_{\texttt{inr}[\ell]}$. Therefore, an interesting question is whether there is a rule-based specification of $\texttt{inr}[\ell]$.

Consider the rule declaration

$\texttt{DeclareRule}[\texttt{ForAll}[\{R\}, f\_[a\_\_\_, x\_, b\_\_\_] \to_{\texttt{"rwI"}} f[a, R, b] /; (x \to_{\texttt{"rwI"}|\ell} R\_)]];$

According to the enumeration strategy of ρLog, $s \rightharpoonup_{\texttt{"rwI"}|\ell} t$ holds iff t is produced from s by a leftmost innermost rewrite step with respect to $\rightharpoonup_\ell$. Thus, $\rightharpoonup_{\texttt{inr}[\ell]} = \rightharpoonup_{\texttt{"rwI"}|\ell}$, where "rwI" is defined as above.

The leftmost innermost evaluation of a term s is the term t for which the relation $s \rightharpoonup_{\texttt{NF}[\texttt{"rwI"}|\ell]} t$ holds. This value can be computed with ρLog by calling

$\texttt{ApplyRule}[s, \texttt{NF}[\texttt{"rwI"} \mid \ell]];$

For example, the expression s_1 can not be evaluated by leftmost innermost rewriting because of the infinite derivation

$$\begin{aligned} s_1 = \; & \texttt{Fst}[\underline{\texttt{from}[0]}] \rightharpoonup_{\texttt{"rwI"}} \texttt{Fst}[\texttt{c}[0, \underline{\texttt{from}[\texttt{s}[0]]}]] \rightharpoonup_{\texttt{"rwI"}} \\ & \texttt{Fst}[\texttt{c}[0, \texttt{c}[\texttt{s}[0], \underline{\texttt{from}[\texttt{s}[\texttt{s}[0]]]}]]] \rightharpoonup_{\texttt{"rwI"}} \ldots \end{aligned}$$

where the subexpressions rewritten by $\rightharpoonup_\ell$ are underlined.

The expression s_2 is evaluated by leftmost innermost rewriting as follows:

$$\begin{aligned} s_2 = \; & \texttt{Fst}[\texttt{c}[\underline{\texttt{Fst}[0, \texttt{s}[0]]}, \texttt{Fst}[\texttt{c}[\texttt{s}[0], 0]]]] \rightharpoonup_{\texttt{"rwI"}} \\ & \texttt{Fst}[\texttt{c}[0, \underline{\texttt{Fst}[\texttt{c}[\texttt{s}[0], 0]]}]] \rightharpoonup_{\texttt{"rwI"}} \underline{\texttt{Fst}[0, \texttt{s}[0]]} \rightharpoonup_{\texttt{"rwI"}} 0. \end{aligned}$$

3.2 Outermost Rewriting

A leftmost outermost rewrite step with respect to a rule ℓ is a transformation of the form $E \to E[F]_p$ where p is the leftmost outermost position of E for which the subexpression $E|_p$ can be transformed with ℓ and $E|_p \rightharpoonup_\ell F$.

We write $E \rightharpoonup_{\texttt{outr}[\ell]} F$ iff F is the result of applying to E an outermost rewrite step with respect to ℓ. We will show how $\texttt{outr}[\ell]$ can be specified in the rule-based programming style of ρLog.

Consider the rule declaration

$\texttt{DeclareRule}[\texttt{ForAll}[\{R\}, f\_[a\_\_\_, x\_, b\_\_\_] \to_{\texttt{"rwO"}} f[a, R, b]/; (x \to_{\ell|\texttt{"rwO"}} R\_)]];$

According to the enumeration strategy of ρLog, $s \rightharpoonup_{\ell|\texttt{"rwO"}} t$ holds iff t is produced from s by a leftmost outermost rewrite step with respect to $\rightharpoonup_\ell$. Thus, $\rightharpoonup_{\texttt{outr}[\ell]} = \rightharpoonup_{\ell|\texttt{"rwO"}}$, where "rwO" is defined as above.

The leftmost outermost evaluation of a term s is the term t for which the relation $s \rightharpoonup_{\texttt{NF}[\ell|\texttt{"rwO"}]} t$ holds. This value can be computed with ρLog by calling

$\texttt{ApplyRule}[s, \texttt{NF}[\ell \mid \texttt{"rwO"}]];$

For example, s_1 is evaluated by leftmost outermost rewriting to 0, since

$$s_1 = \texttt{Fst}[\underline{\texttt{from}[0]}] \rightharpoonup_{\texttt{"rwO"}} \underline{\texttt{Fst}[\texttt{c}[0, \texttt{from}[\texttt{s}[0]]]]} \rightharpoonup_{\texttt{"rwO"}} 0.$$

The expression s_2 is evaluated by leftmost innermost rewriting to 0 as follows:

$$s_2 = \underline{\texttt{Fst}[\texttt{c}[\texttt{Fst}[0, \texttt{s}[0]], \texttt{Fst}[\texttt{c}[\texttt{s}[0], 0]]]]} \rightharpoonup_{\texttt{"rwO"}} \underline{\texttt{Fst}[0, \texttt{s}[0]]} \rightharpoonup_{\texttt{"rwO"}} 0.$$

A short look at the one-step rewriting rules "rwI" and "rwO" reveals the similarities and the differences between leftmost innermost and leftmost outermost rewriting:

similarity: both strategies select the leftmost rewritable subterm denoted by $x\_$ in the pattern $f\_[a\_\_\_, x\_, b\_\_\_]$. The binding of x to the leftmost rewritable subterm is guaranteed by the built-in pattern matching mechanism of the Mathematica interpreter [11, Section 2.3],

difference: leftmost innermost rewriting attempts to find a leftmost reducible subterm of x before applying rule ℓ: this behavior is witnessed by the reducibility formula $x \to_{\texttt{"rwI"}|\ell} R\_$ in the conditional side of "rwI". Leftmost outermost rewriting attempts to apply rule ℓ to x before looking for a leftmost reducible subterm of x: this behavior is witnessed by the reducibility formula $x \to_{\ell|\texttt{"rwO"}} R\_$ in the conditional side of "rwO".

3.3 Parallel Innermost Rewriting

Parallel innermost rewriting evaluates all the arguments of an expression in parallel, in a bottom-up fashion. In this section we will provide a rule-based specification of this strategy, i.e., a rule "PIRW" such that $E \rightharpoonup_{\texttt{"PIRW"}} F$ iff F is the result of applying a parallel innermost rewrite step to E. To simplify the presentation, we will write $E{\downarrow_1}$ for the expression E if $E \not\rightharpoonup_{\texttt{"PIRW"}}$, or else for the expression F such that $E \rightharpoonup_{\texttt{"PIRW"}} F$. Then $E \rightharpoonup_{\texttt{"PIRW"}} F$ iff

1. $E \equiv f[a_1, \ldots, a_n]$, $F \equiv f[a_1{\downarrow_1}, \ldots, a_n{\downarrow_1}]$ and there exists $i \in \{1, \ldots, n\}$ such that $a_i \rightharpoonup_{\texttt{"PIRW"}} a_i{\downarrow_1}$,
2. or else $E \rightharpoonup_\ell F$.

Based on this observation, we can define "PIRW" as an alias to the rule "PIRWn" | ℓ where the rule "PIRWn" is specified by the clause

$$\forall_{f,\overline{xs},\overline{rs}}.\langle \overline{xs} \rangle \to_{\texttt{"T"}} \langle \overline{rs} \rangle \Rightarrow f[\overline{xs}] \to_{\texttt{"PIRWn"}} f[\overline{rs}].$$

and the rule "T" will be specified by a clause which guarantees that

$$\langle a_1, \ldots, a_n\rangle \rightharpoonup_{\texttt{"T"}} \langle b_1, \ldots, b_n\rangle$$

iff $b_i = a_i{\downarrow}_1$ for $1 \leq i \leq n$ and there exists an i such that $a_i \rightharpoonup_{\texttt{"PrwI"}} a_i{\downarrow}_1$. A simple solution is to define "T" as an alias for the rule "Ta" | "Tb" where the rules "Ta" and "Tb" are defined by the clauses

$$\begin{aligned}
&\forall_{x,\overline{xs},r,\overline{rs}}.x \rightarrow_{\texttt{"PIRW"}} r \wedge \langle\overline{xs}\rangle \rightarrow_{\texttt{"Tw"}} \langle\overline{rs}\rangle \Rightarrow \langle x,\overline{xs}\rangle \rightarrow_{\texttt{"Ta"}} \langle r,\overline{rs}\rangle.\\
&\forall_{x,\overline{xs},\overline{rs}}.\langle\overline{xs}\rangle \rightarrow_{\texttt{"T"}} \langle\overline{rs}\rangle \Rightarrow \langle x,\overline{xs}\rangle \rightarrow_{\texttt{"Tb"}} \langle x,\overline{rs}\rangle.
\end{aligned}$$

and "Tw" is specified such that $\langle a_1, \ldots, a_n\rangle \rightharpoonup_{\texttt{"Tw"}} \langle b_1, \ldots, b_n\rangle$ iff $b_i = a_i{\downarrow}_1$ for $1 \leq i \leq n$. We achieve this by defining "Tw" as an alias for "Tw0" | "Tw1" where the auxiliary rules "Tw0" and "Tw1" are specified by the clauses shown below.

$$\begin{aligned}
&\langle\rangle \rightarrow_{\texttt{"Tw0"}} \langle\rangle.\\
&\forall_{x,\overline{xs},r,\overline{rs}}.x \rightarrow_{\texttt{"PIRW"|"I"}} r \wedge \langle\overline{xs}\rangle \rightarrow_{\texttt{"Tw"}} \langle\overline{rs}\rangle \Rightarrow \langle x,\overline{xs}\rangle \rightarrow_{\texttt{"Tw1"}} \langle r,\overline{rs}\rangle.\\
&\forall_{x}.x \rightarrow_{\texttt{"I"}} x.
\end{aligned}$$

The ρLog declarations of these clauses are:

$\texttt{SetAlias}[\texttt{"PIRWn"} \mid \ell, \texttt{"PIRW"}];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{rs\}, f\_[xs\_\_] \rightarrow_{\texttt{"PIRWn"}} f[rs]/;(\langle xs\rangle \rightarrow_{\texttt{"T"}} \langle rs\_\_\rangle)]];$
$\texttt{SetAlias}[\texttt{"Ta"} \mid \texttt{"Tb"}, \texttt{"T"}];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{r, rs\}, \langle x\_, xs\_\_\_\rangle \rightarrow_{\texttt{"Ta"}}$
$\quad\langle r, rs\rangle/;(x \rightarrow_{\texttt{"PIRW"}} r\_) \wedge (\langle xs\rangle \rightarrow_{\texttt{"Tw"}} \langle rs\_\_\_\rangle)]];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{rs\}, \langle x\_, xs\_\_\_\rangle \rightarrow_{\texttt{"Tb"}} \langle x, rs\rangle/;(\langle xs\rangle \rightarrow_{\texttt{"T"}} \langle rs\_\_\_\rangle)]];$
$\texttt{SetAlias}[\texttt{"Tw0"} \mid \texttt{"Tw1"}, \texttt{"Tw"}];$
$\texttt{DeclareRule}[\langle\rangle \rightarrow_{\texttt{"Tw0"}} \langle\rangle];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{r, rs\}, \langle x\_, xs\_\_\_\rangle \rightarrow_{\texttt{"Tw1"}}$
$\quad\langle r, rs\rangle/;(x \rightarrow_{\texttt{"PIRW"|"I"}} r\_) \wedge (\langle xs\rangle \rightarrow_{\texttt{"Tw"}} \langle rs\_\_\_\rangle)]];$
$\texttt{DeclareRule}[x\_ \rightarrow_{\texttt{"I"}} x];$

Now, the evaluation of any expression E to a normal form using the parallel innermost rewriting strategy can be computed by calling

$$\texttt{ApplyRule}[E, \texttt{NF}[\texttt{"PIRW"}]] \quad \text{or} \quad \texttt{ApplyRule}[E, \texttt{Repeat}[\texttt{"PIRW"}, \texttt{"I"}]].$$

For example, the value of s_1 with respect to this evaluation strategy is undefined because the leftmost innermost rewrite derivation of s_1 given in Section 3.1 is also a parallel innermost rewrite derivation. The value of s_2 with respect to parallel innermost rewriting is 0, and it is computed by the derivation

$$s_2 = \texttt{Fst}[\texttt{c}[\underline{\texttt{Fst}[0, \texttt{s}[0]]}, \underline{\texttt{Fst}[\texttt{c}[\texttt{s}[0], 0]]}]] \rightharpoonup_{\texttt{"PIRW"}} \underline{\texttt{Fst}[\texttt{c}[0, \texttt{s}[0]]]} \rightharpoonup_{\texttt{"PIRW"}} 0.$$

3.4 Parallel Outermost Rewriting

This strategy evaluates all the arguments of an expression in parallel, in a top-down fashion. By following a line of reasoning similar to that for parallel innermost rewriting, we conclude that this strategy corresponds to the reduction relation $\rightharpoonup_{\texttt{"PORW"}}$ induced by "PORW", where the rule "PORW" and those related to it have the following ρLog declarations:

$\texttt{SetAlias}[\ell \mid \texttt{"PORWn"}, \texttt{"PORW"}];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{rs\}, f\_[xs\_\_\_] \rightarrow_{\texttt{"PORWn"}} f[rs]/; (\langle xs\rangle \rightarrow_{\texttt{"T"}} \langle rs\_\_\_\rangle)]];$
$\texttt{SetAlias}[\texttt{"Ta"} \mid \texttt{"Tb"}, \texttt{"T"}];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{r, rs\}, \langle x\_, xs\_\_\_\rangle \rightarrow_{\texttt{"Ta"}}$
$\quad \langle r, rs\rangle/; (x \rightarrow_{\texttt{"PORW"}} r\_) \wedge (\langle xs\rangle \rightarrow_{\texttt{"Tw"}} \langle rs\_\_\_\rangle)]];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{rs\}, \langle x\_, xs\_\_\_\rangle \rightarrow_{\texttt{"Tb"}} \langle x, rs\rangle/; (\langle xs\rangle \rightarrow_{\texttt{"T"}} \langle rs\_\_\_\rangle)]];$
$\texttt{SetAlias}[\texttt{"Tw0"} \mid \texttt{"Tw1"}, \texttt{"Tw"}];$
$\texttt{DeclareRule}[\langle\rangle \rightarrow_{\texttt{"Tw0"}} \langle\rangle];$
$\texttt{DeclareRule}[\texttt{ForAll}[\{r, rs\}, \langle x\_, xs\_\_\_\rangle \rightarrow_{\texttt{"Tw1"}}$
$\quad \langle r, rs\rangle/; (x \rightarrow_{\texttt{"PORW"}\mid\texttt{"I"}} r\_) \wedge (\langle xs\rangle \rightarrow_{\texttt{"Tw"}} \langle rs\_\_\_\rangle)]];$
$\texttt{DeclareRule}[x\_ \rightarrow_{\texttt{"I"}} x];$

This strategy evaluates the terms s_1 and s_2 as follows:

$$s_1 = \texttt{Fst}[\underline{\texttt{from}[0]}] \rightharpoonup_{\texttt{"PORW"}} \texttt{Fst}[\texttt{c}[0, \texttt{from}[\texttt{s}[0]]]] \rightharpoonup_{\texttt{"PORW"}} 0.$$
$$s_2 = \underline{\texttt{Fst}[\texttt{c}[\underline{\texttt{Fst}[0, \texttt{s}[0]]}, \texttt{Fst}[\texttt{c}[\texttt{s}[0], 0]]]]]} \rightharpoonup_{\texttt{"PORW"}} \underline{\texttt{Fst}[0, \texttt{s}[0]]} \rightharpoonup_{\texttt{"PORW"}} 0.$$

Let's compare parallel innermost rewriting and parallel outermost rewriting, from the perspective of rule-based programming:

- a parallel innermost rewrite step is defined as an alias to "PIRWn" | ℓ. This means, it tries first to apply a parallel rewriting of all subterms by using rule "PIRWn", and if this attempt fails then it tries to apply rule ℓ,
- a parallel outermost rewrite step is defined as an alias to ℓ | "PORWn". This means, it tries first to apply rule ℓ, and if this attempt fails then it attempts to perform parallel rewriting of all subterms by using rule "PORWn".

The other rules are used for navigating through the structure of the expressions, and they have identical definitions for both parallel rewriting strategies.

3.5 Lambda Calculus

The lambda calculus is a formal system designed to investigate the notions of function definition, function application and recursion. It was introduced by Church [4] and Kleene [6] in the 1930s and is often regarded as the smallest universal language.[6]

We consider here the pure lambda calculus, defined over a countably infinite set of variables $\mathcal{V}$. For a concrete representation, we identify $\mathcal{V}$ with the set of Mathematica symbols without a predefined meaning, and define a λ-term as an element of the syntax domain defined by the following context-free grammar:

$$\begin{array}{lll} T ::= & & \textit{lambda terms:} \\ & \mid\ x & \text{variable} \\ & \mid\ \lambda[x, T] & \text{abstraction} \\ & \mid\ \texttt{app}[T_1, T_2] & \text{application} \end{array}$$

[6] The lambda calculus is universal in the sense that any computable function can be expressed and evaluated using this formalism.

An abstraction $\lambda[x, T]$ expresses the notion of function with formal parameter x which returns the result of evaluating T with respect to x. Over the set of λ-terms an equivalence relation (here denoted as $=_\alpha$) is defined that captures the intuition that two λ-terms denote the same function. This equivalence relation is defined by the so-called *α-conversion* rule.

The α-conversion rule states that if $x, y \in \mathcal{V}$, T is a λ-term and $T/.\{x \to y\}$ means the term T with every free occurrence of x in T replaced with y then $\lambda[x, T] =_\alpha \lambda[y, E/.\{x \to y\}]$ if y does not appear freely in T and y is not bound by a λ in T whenever it replaces a y. This rule tells us for example that $\lambda[x, \texttt{app}[\lambda[x, x], x]]$ is the same as $\lambda[y, \texttt{app}[\lambda[x, x], y]$. The binary relation $=_\alpha$ is an equivalence on the set of λ-terms. Because of the intended interpretation of abstractions, it is natural to identify λ-terms which are $=_\alpha$-equivalent.

The notion of evaluation is defined via the β-conversion rule. β-conversion expresses the idea of function application. It states that $\texttt{app}[\lambda[x, T_1], T_2]$ reduces to the instance $T_1/.\{x \to T_2\}$ if all free variable occurrences in T_2 remain free in $T_1/.\{x \to T_2\}$. Since $\mathcal{V}$ is infinite, we can always compute a λ-term $\lambda[x, T_1']$ such that (1) $\lambda[x, T_1] =_\alpha \lambda[x, T_1']$ and (2) $\texttt{app}[\lambda[x, T_1'], T_2] = T_1'/.\{x \to T_2\}$. If we assume that the computation of $\lambda[x, T_1']$ is carried out by the function $\alpha\texttt{Convert}[\lambda[s, T_1], T_2]$, then the β-conversion rule can be specified as follows:

$$\texttt{DeclareRule}[\texttt{app}[\lambda[x\_, T1\_], T2\_] \to_{\texttt{"}\beta\texttt{"}} \alpha\texttt{Convert}[\lambda[x, T1], T2]/.\{x \to T2\}]$$

A *redex* is a λ-term s for which there exists t such that $s \to_{\texttt{"}\beta\texttt{"}} t$ holds. The λ-term t is called the *contraction* of s.

In lambda calculus, an evaluation strategy prescribes the order in which the redexes of an expression are contracted until a normal form is eventually reached.

One common practice is to consider all abstractions as values; this means that we do not contract redexes below an occurrence of λ. The other redexes are contracted by following either a lazy or an eager evaluation strategy. Both evaluation strategies have straightforward specifications: lazy β-reduction corresponds to the relation $\rightharpoonup_{\texttt{"}\beta\texttt{"}|\texttt{"}\beta\texttt{L"}}$, and eager β-reduction corresponds to the relation $\rightharpoonup_{\texttt{"}\beta\texttt{E"}|\texttt{"}\beta\texttt{"}}$, where the auxiliary rules "βL" and"βE" are specified below.

$$\texttt{DeclareRule}[\texttt{ForAll}[\{u\}, \texttt{app}[s\_, t\_] \to_{\texttt{"}\beta\texttt{L"}} \texttt{app}[u, t]/; (s \to_{\texttt{"}\beta\texttt{"}|\texttt{"}\beta\texttt{L"}} u\_)]];$$
$$\texttt{DeclareRule}[\texttt{ForAll}[\{u\}, \texttt{app}[s\_, t\_] \to_{\texttt{"}\beta\texttt{E"}} \texttt{app}[s, u]/; (t \to_{\texttt{"}\beta\texttt{E"}|\texttt{"}\beta\texttt{"}} u\_)]];$$

To illustrate the behavior of these two evaluation strategies, we consider the λ-term $t = \texttt{app}[\lambda[x, \lambda[y, \texttt{app}[x, y]]], \texttt{app}[\lambda[z, z], \texttt{app}[\lambda[u, v], w]]]$. The ρLog call

$$\texttt{ApplyRule}[t, \texttt{NF}[\texttt{"}\beta\texttt{"} \mid \texttt{"}\beta\texttt{L"}]]$$

employs the lazy evaluation strategy to reduce t as follows:

$$t = \underline{\texttt{app}[\lambda[x, \lambda[y, \texttt{app}[x, y]]], \texttt{app}[\lambda[z, z], \texttt{app}[\lambda[u, v], w]]]} \rightharpoonup_{\texttt{"}\beta\texttt{L"}} \lambda[y, \texttt{app}[x, \texttt{app}[\lambda[z, z], \texttt{app}[\lambda[u, v], w]]]]$$

whereas the call

$$\texttt{ApplyRule}[t, \texttt{NF}[\texttt{"}\beta\texttt{E"} \mid \texttt{"}\beta\texttt{"}]]$$

employs the eager evaluation strategy to reduce t as follows:

$$
\begin{aligned}
t = {} & \mathtt{app}[\lambda[x, \lambda[y, \mathtt{app}[x, y]]], \mathtt{app}[\lambda[z, z], \underline{\mathtt{app}[\lambda[u, v], w]}]] \rightharpoonup_{\text{"}\beta\mathtt{E}\text{"}} \\
& \mathtt{app}[\lambda[x, \lambda[y, \mathtt{app}[x, y]]], \underline{\mathtt{app}[\lambda[z, z], v]}] \rightharpoonup_{\text{"}\beta\mathtt{E}\text{"}} \\
& \underline{\mathtt{app}[\lambda[x, \lambda[y, \mathtt{app}[x, y]]], v]} \rightharpoonup_{\text{"}\beta\mathtt{E}\text{"}} \lambda[y, \mathtt{app}[v, y]].
\end{aligned}
$$

4 Conclusions

The notion of value depends on the evaluation strategy under consideration. For instance, s_1 can not be evaluated with innermost and parallel innermost rewriting, whereas outermost and parallel outermost rewriting reduce it to 0.

There are many other rule-based programming languages which provide programmatic support for rewriting strategies, but their programming constructs may differ significantly. In [5], Dolstra and Visser analyze the suitability of Stratego [9] for building interpreters with rewriting strategies. The programming style ρLog is more similar to Elan [3] where the main concept is the notion of labelled transformation rule. We claim that ρLog is more suitable than Elan for programming evaluation strategies in a natural and concise way. The additional expressive power of ρLog stems from the availability of function variables and sequence variables. These features provide a simple and powerful mechanism to explore the structure of the expressions to be evaluated.

References

1. http://www.ricam.oeaw.ac.at/people/page/marin/RhoLog/index.html.
2. Fifth International Symposium on Rule-based Programming (RULE 2004). http://www-i2.informatik.rwth-aachen.de/RULE04/.
3. P. Borovansky, C. Kirchner, H. Kirchner, P. E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In *Proceedings of the First International Workshop on Rewriting Logic*, volume 4. Elsevier, 1996.
4. Alonzo Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1:40–41 and 101–102, 1936.
5. Eelco Dolstra and Eelco Visser. Building Interpreters with Rewriting Strategies. Technical Report UU-CS-2002-022. Institute of Information and Computing Sciences, Utrecht University. May 2002.
6. Stephen C. Kleene. λ-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.
7. Mircea Marin and Temur Kutsia. A Calculus for Higher-order Rule-based Programming with Sequence Variables. 2004. To appear. Available from http://www.ricam.oeaw.ac.at/people/page/marin/papers/MarinKutsiaAPLAS.ps.
8. Mircea Marin and Florina Piroi. Rule-based programming with Mathematica. In *Sixth Mathematica Symposium (IMS 2004)*, Banff, Alberta, Canada, August 1-6 2004.
9. Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. *Lecture Notes in Computer Science*, 2051:357–362, 2001.
10. Donald A. Waterman. *The guide to expert systems.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1985.
11. Stephen Wolfram. *The Mathematica Book.* Wolfram Media Inc. Champaign, Illinois, USA and Cambridge University Press, 1999.