Foundations of the Rule-Based System ρLog

Mircea Marin* — **Temur Kutsia****

* *Graduate School of Systems and Information Engineering*
University of Tsukuba
Tsukuba 305-8573, Japan
mmarin@cs.tsukuba.ac.jp

** *Research Institute for Symbolic Computation*
Johannes Kepler University of Linz
A-4232 Castle of Hagenberg, Austria
tkutsia@risc.uni-linz.ac.at

ABSTRACT. *We describe the foundations of a system for rule-based programming which integrates two powerful mechanisms: (1) matching with context variables, sequence variables, and regular constraints for their matching values; and (2) strategic programming with labeled rules. The system is called ρLog, and is built on top of the pattern matching and rule-based programming capabilities of Mathematica.*

KEYWORDS: *Rule-based programming, declarative programming, matching.*

1. Introduction

Rewriting is a general declarative formalism to specify, program, and reason about computational systems. For instance, in equational reasoning and functional programming, rules are understood as oriented equalities; in theorem proving or concurrent programming, rules are understood as inferences or transitions. The increasing awareness of the usefulness of rewriting in non-equational contexts has led to theoretical developments such as rewriting logic [MES 92] and to specialized calculi such as System S [VIS 98], the Rho-calculus [CIR 03] and extensions thereof [BER 04, CIR 04]. These theoretical frameworks led to the emergence of rule-based programming languages such as ELAN [BOR 02], Maude [CLA 02] and Stratego [VIS 04], and to rule-based specification languages such as ASF+SDF [BER 89] and CafeOBJ [DIA 02].

The rule-based programming style builds upon the idea of expressing computations as strategic compositions of basic computational steps, where each basic step can be decomposed into three elementary operations: (1) retrieve information from the input data, as candidates for computing a new result, (2) use some decision crite-

rion to filter the irrelevant candidates, and (3) compute a result from a candidate that has passed the decision criterion. Such a basic computational step can be specified as a transformation rule

$$\texttt{apply}(st, t_1) \rightarrow t_2 \Leftarrow cnd$$

where:

1) t_1 is an expression that describes the structure of the input data,

2) *st* is an expression describing the strategic construct that acts on the input data,

3) *cnd* is a boolean test to filter irrelevant data,

4) t_2 is an expression that describes the computation of a result.

Rule-based programs are collections of transformation rules that provide a declarative semantics for answering queries. The expressive power of a rule-based programming language can be estimated by looking at: (1) the kind of queries that can be answered, (2) the strategic constructs recognized by the language, and (3) the programming constructs for specifying transformation rules.

In this paper we describe the foundations of a rule-based programming system called ρLog. The distinguishing features of ρLog, when compared to other rule-based systems, are:

– Programming with context and sequence variables. Context variables are placeholders for contexts, which are functional expressions whose applicative behavior is to replace a special constant (the context hole) with the expression given as argument. Sequence variables are placeholders for arbitrarily long finite sequences of expressions. These constructs enable a compact representation of expressions whose instances can be terms of arbitrary depth and width.

– Regular constraints for the admissible values of sequence variables and of context variables. A regular constraint specifies a regular grammar of the language where the value of the constrained variable belongs.

– Strategic programming in a language with constructors for most of the strategic relations used in rule-based transformations: composition, choice, reflexive-transitive closure, normalization, and or-else choice.

To our knowledge, ρLog is the first rule-based programming system which integrates programming with sequence variables, context variables, and regular constraints in a single framework. These capabilities enable a highly declarative programming style that is expressive enough to support concise implementations for: specifying and prototyping deductive systems, solvers for various equational theories, tools for querying and translating XML, evaluation strategies, etc. [MAR 04a, MAR 04c, MAR 04d].

However, the integration of these features into a coherent framework requires the acceptance of some syntactic restrictions. For example, unrestricted rule-based programming with sequence variables and context variables is problematic because it requires unification of terms with sequence and context variables, and this unification problem is not finitary. We have overcome this problem by imposing the restriction of *determinism* on programs and goals. Determinism is a notion similar to

well-modedness in logic programming, that allows to compute the answers of queries by iterative matching instead of unification.

The paper is structured as follows. In the next section we introduce the syntax of ρLog. In Section 3 we describe the underlying calculus. To give the reader a better feeling of the generality of our rule based-programming approach and the wide range of application areas, we discuss some applications in Section 4. Section 5 concludes.

2. The Language

The main syntactic categories of our language are: terms, contexts, strategies, regular expressions, regular constraints, programs and queries.

Terms are intended to specify the structure of input and output of the transformation rules of ρLog. They are built from: function symbols from a set $\mathcal{F}$; function variables from a set $\mathcal{V}_{\mathtt{f}}$; individual variables from a set $\mathcal{V}_{\mathtt{i}}$; sequence variables from a set $\mathcal{V}_{\mathtt{s}}$; and context variables from a set $\mathcal{V}_{\mathtt{c}}$. All these sets are assumed to be mutually disjoint. In addition, the sets $\mathcal{V}_{\mathtt{f}}$, $\mathcal{V}_{\mathtt{i}}$, $\mathcal{V}_{\mathtt{s}}$ and $\mathcal{V}_{\mathtt{c}}$ are countably infinite, and contain the following special symbols: the *anonymous individual variable* $\mathtt{\_i} \in \mathcal{V}_{\mathtt{i}}$; the *anonymous function variable* $\mathtt{\_f} \in \mathcal{V}_{\mathtt{f}}$; the *anonymous sequence variable* $\mathtt{\_s} \in \mathcal{V}_{\mathtt{s}}$; and the *anonymous context variable* $\mathtt{\_c} \in \mathcal{V}_{\mathtt{c}}$. The set $\mathcal{F}$ contains the special function symbol $\diamond$, called the *hole* symbol. From now on we assume that metavariables f, g, h range over $\mathcal{F} \setminus \{\diamond\}$; F, G, H range over $\mathcal{V}_{\mathtt{f}}$; x, y, z range over $\mathcal{V}_{\mathtt{i}}$; $\overline{x}, \overline{y}, \overline{z}$ range over $\mathcal{V}_{\mathtt{s}}$; and $\overline{C}$ ranges over $\mathcal{V}_{\mathtt{c}}$.

Terms and term sequences are mutually defined by the grammars:

t	::=		*terms:*
		$\diamond$	hole
	$\mid$	x	individual variable
	$\mid$	$f(ts)$	rigid term
	$\mid$	$F(ts)$	flex term
	$\mid$	$\overline{C}(t)$	context application
ts	::=		*term sequences:*
		$\ulcorner\urcorner$	empty sequence
	$\mid$	t	term
	$\mid$	$\overline{x}$	sequence variable
	$\mid$	$\ulcorner ts_1, ts_2 \urcorner$	sequence of term sequences

The delimiters $\ulcorner$ and $\urcorner$ around term sequences are mainly for readability purposes, and we do not display them inside terms. Terms of the form $f()$ with $f \in \mathcal{F}$ will be abbreviated to f.

We introduce the following definitions: $\mathcal{V}_{\mathtt{f}}^{-} := \mathcal{V}_{\mathtt{f}} \setminus \{\mathtt{\_f}\}$, $\mathcal{V}_{\mathtt{i}}^{-} := \mathcal{V}_{\mathtt{i}} \setminus \{\mathtt{\_i}\}$, $\mathcal{V}_{\mathtt{s}}^{-} := \mathcal{V}_{\mathtt{s}} \setminus \{\mathtt{\_s}\}$, $\mathcal{V}_{\mathtt{c}}^{-} := \mathcal{V}_{\mathtt{c}} \setminus \{\mathtt{\_c}\}$, $\mathcal{V} := \mathcal{V}_{\mathtt{f}} \cup \mathcal{V}_{\mathtt{i}} \cup \mathcal{V}_{\mathtt{s}} \cup \mathcal{V}_{\mathtt{c}}$, $\mathcal{V}_{\mathrm{any}} := \{\mathtt{\_f}, \mathtt{\_i}, \mathtt{\_s}, \mathtt{\_c}\}$, $\mathcal{V}^{-} := \mathcal{V} \setminus \mathcal{V}_{\mathrm{any}}$, and $\mathcal{F}^{-} := \mathcal{F} \setminus \{\diamond\}$. If $M \subseteq \mathcal{F}$ and $V \subseteq \mathcal{V}$ then $T(M, V)$ is the set

of terms and $TS(M, V)$ is the set of term sequences defined by the previous grammar with symbols from M and V.

A *context* is a term with a single occurrence of $\diamond$. For example, $\diamond$, $\overline{C}(\diamond)$ and $f(g(x), \diamond, \overline{x}, F(\overline{C}(a)))$ are contexts. We write $Ctx(\mathcal{F}, \mathcal{V})$ for the set of contexts and assume that the metavariable C ranges over contexts.

Strategies provide a mechanism to describe complex rule-based computations in a highly declarative way. In ρLog, strategies are built from strategy names from a set $\mathcal{F}_{\mathtt{st}}$ that contains special symbols (*predefined strategies*) `Id`, `Compose`, `Choice`, `Closure`, `NormalForm`, `OrElse`, and strategy variables from a countably infinite set $\mathcal{V}_{\mathtt{st}}$. We assume that these sets are disjoint, and also disjoint from the sets $\mathcal{V}$ and $\mathcal{F}$. From now on we assume the metavariable `l` ranges over the set

$$\mathcal{L} := \mathcal{F}_{\mathtt{st}} \setminus \{\mathtt{Id}, \mathtt{Compose}, \mathtt{Choice}, \mathtt{Closure}, \mathtt{NormalForm}, \mathtt{OrElse}\}$$

of what we call *user-definable strategies*, and σ ranges over $\mathcal{V}_{\mathtt{st}}$.

Strategies are defined by the grammar

st	::=		*strategies:*
		$\mathtt{l}(st_1, \ldots, st_m)$	basic strategy
	|	σ	strategy variable
	|	$\mathtt{Id}$	identity
	|	$\mathtt{Compose}(st_1, st_2)$	composition
	|	$\mathtt{Choice}(st_1, st_2)$	choice
	|	$\mathtt{Closure}(st)$	closure
	|	$\mathtt{NormalForm}(st)$	normalization
	|	$\mathtt{OrElse}(st_1, st_2)$	or-else

where $m \geq 0$. We write $St(\mathcal{F}_{\mathtt{st}}, \mathcal{V}_{\mathtt{st}})$ for the set of strategies defined by this grammar, and abbreviate basic strategies of the form $\mathtt{l}()$ to $\mathtt{l}$.

Strategic programming in ρLog is supported by the fact that the meaning of predefined strategies is, as expected, predefined, whereas the meaning of the basic strategies is user-definable. We defer the explanation of the meanings of strategic constructs until Subsection 2.2, after we introduce the notions of program and query.

A *substitution* is a mapping

$$\theta : \mathcal{V}^- \cup \mathcal{V}_{\mathtt{st}} \longrightarrow \mathcal{F}^- \cup \mathcal{V}_{\mathtt{f}} \cup T(\mathcal{F}^-, \mathcal{V}) \cup TS(\mathcal{F}^-, \mathcal{V}) \cup Ctx(\mathcal{F}, \mathcal{V}) \cup St(\mathcal{F}_{st}, \mathcal{V}_{st})$$

that satisfies the following conditions:

– $\theta(F) \in \mathcal{F}^- \cup \mathcal{V}_{\mathtt{f}}$ for all $F \in \mathcal{V}^-_{\mathtt{f}}$,
– $\theta(x) \in T(\mathcal{F}^-, \mathcal{V})$ for all $x \in \mathcal{V}^-_{\mathtt{i}}$,
– $\theta(\overline{x}) \in TS(\mathcal{F}^-, \mathcal{V})$ for all $\overline{x} \in \mathcal{V}^-_{\mathtt{s}}$,
– $\theta(\overline{C}) \in Ctx(\mathcal{F}, \mathcal{V})$ for all $\overline{C} \in \mathcal{V}^-_{\mathtt{c}}$,
– $\theta(\sigma) \in St(\mathcal{F}_{\mathtt{st}}, \mathcal{V}_{\mathtt{st}})$ for all $\sigma \in \mathcal{V}_{\mathtt{st}}$, and

– $\{u \in \mathcal{V}_{\mathtt{f}}^- \cup \mathcal{V}_{\mathtt{i}}^- \cup \mathcal{V}_{\mathtt{s}}^- \cup \mathcal{V}_{\mathtt{st}} \mid \theta(u) \neq u\} \cup \{\overline{C} \in \mathcal{V}_c^- \mid \theta(\overline{C}) \neq \overline{C}(\diamond)\}$
is a finite set. We call this set the *domain* of θ and denote it by $Dom(\theta)$.

We use the standard notation and write substitutions as finite sets of *bindings*.

EXAMPLE 1. — The following two sets are substitutions:

$$\{x \mapsto f(a, \mathtt{\_i}, \overline{y}), \overline{x} \mapsto \ulcorner\urcorner, \overline{y} \mapsto \ulcorner a, \overline{C}(f(b)), x\urcorner, F \mapsto g, \overline{C} \mapsto g(\diamond)\}.$$
$$\{x \mapsto F(\mathtt{\_s}), \overline{x} \mapsto \overline{y}, \overline{C} \mapsto f(\overline{C}(F(\mathtt{\_s}, a, \diamond, b, \mathtt{\_s})))\}.$$

The following sets are not substitutions:

$$\{x \mapsto \ulcorner \overline{C}(f(a, \mathtt{\_i}, \overline{y})), a\urcorner, F \mapsto \diamond\}.$$
$$\{x \mapsto \mathtt{\_s}, \overline{y} \mapsto y, \overline{C} \mapsto \ulcorner \diamond, a\urcorner\}.$$
$$\{\mathtt{\_f} \rightarrow f, \mathtt{\_i} \mapsto f(), \mathtt{\_s} \mapsto \ulcorner f(), g()\urcorner, \mathtt{\_c} \mapsto \overline{C}(\diamond)\}.$$

□

Substitutions can instantiate terms, term sequences, contexts, or strategies. The *instance* $E\theta$ of an expression $E \in T(\mathcal{F}, \mathcal{V}) \cup TS(\mathcal{F}, \mathcal{V}) \cup Ctx(\mathcal{F}, \mathcal{V}) \cup St(\mathcal{F}_{\mathtt{st}}, \mathcal{V}_{\mathtt{st}})$ is the expression obtained by

1) replacing every occurrence of some $u \in \mathcal{V}^- \cup \mathcal{V}_{\mathtt{st}}$ with $\theta(u)$, and

2) evaluating the subexpressions $C(t)$ to $C[t]$ where $C[t]$ indicates the term obtained from context C by replacing the occurrences of $\diamond$ with term t.

2.1. *Regular constraints*

Regular constraints restrict the possible values of a sequence variable or context variable. Such restrictions are defined by associating a variable with a regular expression that represents the set of its admissible values. We consider two kinds of regular expressions: for sequence terms and for contexts. Each such regular expression represents a set of corresponding syntactic objects. Our definitions of the semantics of regular expressions will make use of the operation of language substitution. In general, if $A, A_1, \ldots, A_n$ are sets of objects (i.e., sequence terms or contexts), and $o_1, \ldots, o_n$ are symbols used in the construction of these objects, then the *language substitution* $A\{o_1 \leftarrow A_1, \ldots, o_n \leftarrow A_n\}$ denotes the set consisting of all objects producible from the elements of A by replacing, for all $1 \leq i \leq n$, all occurrences of o_i with (possibly different) elements from A_i. In general, we will consider only *conservative* language substitutions, that is, language substitutions for which the objects of $A\{o_1 \leftarrow A_1, \ldots, o_n \leftarrow A_n\}$ are in the same syntactic category as the objects of A.

We start with defining regular expressions. First we define them for term sequences and then for contexts.

Rs	::=		*regular expressions for term sequences:*
		ts	hole-free term sequence ($ts \in TS(\mathcal{F}^-, \mathcal{V})$)
	|	$\ulcorner \mathsf{Rs}_1, \mathsf{Rs}_2 \urcorner$	concatenation
	|	$\mathsf{Rs}_1 \mid \mathsf{Rs}_2$	choice
	|	Rs^*	closure

The anonymous variables $\_\mathtt{f}$, $\_\mathtt{i}$, $\_\mathtt{s}$ and $\_\mathtt{c}$ have special meanings: They are anonymous placeholders respectively for a function symbol or a function variable, for a term, for a term sequence, and for a context. This interpretation of anonymous variables is taken into account when we define the language $[\![\mathsf{Rs}]\!]$ generated by Rs:

$$\begin{aligned}
[\![ts]\!] &:= \{ts\}\Theta \\
[\![\ulcorner \mathsf{Rs}_1, \mathsf{Rs}_2 \urcorner]\!] &:= \{\ulcorner ts_1, ts_2 \urcorner \mid ts_1 \in [\![\mathsf{Rs}_1]\!], ts_2 \in [\![\mathsf{Rs}_2]\!]\} \\
[\![\mathsf{Rs}_1 \mid \mathsf{Rs}_2]\!] &:= [\![\mathsf{Rs}_1]\!] \cup [\![\mathsf{Rs}_2]\!] \\
[\![\mathsf{Rs}^*]\!] &:= \bigcup_{n\geq 0} [\![\mathsf{Rs}]\!]^n
\end{aligned}$$

where $[\![\mathsf{Rs}]\!]^0 = \{\ulcorner \urcorner\}$, $[\![\mathsf{Rs}]\!]^{n+1} = \{\ulcorner ts_1, ts_2 \urcorner \mid ts_1 \in [\![\mathsf{Rs}]\!],\ ts_2 \in [\![\mathsf{Rs}]\!]^n\}$ for $n \geq 0$, and $\{ts\}\Theta$ is a language substitution with

$$\Theta = \{\_\mathtt{f} \leftarrow \mathcal{F}^- \cup \mathcal{V}_\mathtt{f},\ \_\mathtt{i} \leftarrow T(\mathcal{F}^-, \mathcal{V}),\ \_\mathtt{s} \leftarrow TS(\mathcal{F}^-, \mathcal{V}),\ \_\mathtt{c} \leftarrow Ctx(\mathcal{F}, \mathcal{V})\}.$$

A *regular constraint for a sequence variable* is a pair of a non-anonymous sequence variable and a regular expression for term sequences, written $\overline{x}$:Rs, where $\overline{x} \in \mathcal{V}_\mathtt{s}^-$. We say that $\overline{x}$ is *constrained* by Rs. The purpose of such a constraint is to restrict a possible value of the variable $\overline{x}$ to belong to the language $[\![\mathsf{Rs}]\!]$.

Regular expressions for contexts are defined by the following grammar:

Rc	::=		*regular expressions for contexts:*
		C	context
	|	$\mathsf{Rc}_1.\mathsf{Rc}_2$	composition
	|	$\mathsf{Rc}_1 + \mathsf{Rc}_2$	choice
	|	$\mathsf{Rc}^\star$	closure

We interpret anonymous variables in the same way as before, and define the language $[\![\mathsf{Rc}]\!]$ generated by Rc as follows:

$$\begin{aligned}
[\![C]\!] &:= \{C\}\Theta \\
[\![\mathsf{Rc}_1.\mathsf{Rc}_2]\!] &:= \{C_1[C_2] \mid C_1 \in [\![\mathsf{Rc}_1]\!],\ C_2 \in [\![\mathsf{Rc}_2]\!]\} \\
[\![\mathsf{Rc}_1 + \mathsf{Rc}_2]\!] &:= [\![\mathsf{Rc}_1]\!] \cup [\![\mathsf{Rc}_2]\!] \\
[\![\mathsf{Rc}^\star]\!] &:= \bigcup_{n\geq 0} [\![\mathsf{Rc}]\!]^n
\end{aligned}$$

where $[\![\mathsf{Rc}]\!]^0 = \{\diamond\}$, $[\![\mathsf{Rc}]\!]^{n+1} = \{C_1[C_2] \mid C_1 \in [\![\mathsf{Rc}]\!],\ C_2 \in [\![\mathsf{Rc}]\!]^n\}$ for $n \geq 0$, and $\{C\}\Theta$ is a language substitution with Θ being, like above,

$$\{\_\mathtt{f} \leftarrow \mathcal{F}^- \cup \mathcal{V}_\mathtt{f},\ \_\mathtt{i} \leftarrow T(\mathcal{F}^-, \mathcal{V}),\ \_\mathtt{s} \leftarrow TS(\mathcal{F}^-, \mathcal{V}),\ \_\mathtt{c} \leftarrow Ctx(\mathcal{F}, \mathcal{V})\}.$$

A *regular constraint for a context variable* is a pair of a non-anonymous context variable and a regular expression on contexts, written $\overline{C}$:Rc, where $\overline{C} \in \mathcal{V}_\mathtt{c}^-$. We say that

$\overline{C}$ is *constrained* by Rc. The purpose of such a construct is to restrict the value of $\overline{C}$ to belong to the language $[\![\mathsf{Rc}]\!]$.

For programming purposes, it is often useful to have the syntactic construct Rs^n as an abbreviation to n-fold concatenation $\ulcorner \mathsf{Rs}, \ldots, \mathsf{Rs} \urcorner$, and Rc^n as an abbreviation to n-fold composition $\mathsf{Rc}.\cdots.\mathsf{Rc}$ and, therefore, we will consider them as part of our language of regular expressions.

2.2. *Programs and queries*

The building blocks of programs and queries are the *reducibility atoms*, built from the reducibility predicate $\rightarrow$ applied to 3 arguments: a strategy followed by two terms without holes. We write a reducibility atom $\rightarrow (st, t_1, t_2)$ as $\mathtt{apply}(st, t_1) \rightarrow t_2$ to emphasize the intuition behind it: the application of a strategy *st* on some input t_1 in order to produce an output t_2. The negation of $\mathtt{apply}(st, t_1) \rightarrow t_2$ is written as $\neg(\mathtt{apply}(st, t_1) \rightarrow t_2)$. A *literal* is a pair $\langle L, \mathcal{C}\rangle$ where L is either a reducibility atom or its negation, and $\mathcal{C}$ is a set of regular constraints for variables that occur in L.

A rule-based program is a special kind of general logic program that defines the interpretation of the reducibility predicate $\rightarrow$ used in the construction of reducibility atoms. The general structure of a rule-based program clause is

$$\langle \mathtt{apply}(st, t) \rightarrow t', \mathcal{C}\rangle \ \mathtt{:-}\ L_1, \ldots, L_n$$

where $L_1, \ldots, L_n$ are literals. In rule-based programming, these clauses are interpreted as conditional rewrite rules, because of the aforementioned intended meaning of the reducibility predicate: application of a strategy *st* on some input t_1 in order to produce an output t_2. The constraint part $\mathcal{C}$ is relevant for restricting the values of the matchers considered in the rewrite step. In order to emphasize the rewriting character of reducibility literals, we will abbreviate a literal $\langle \mathtt{apply}(st, t_1) \rightarrow t_2, \mathcal{C}\rangle$ as $t_1 \rightarrow_{st,\mathcal{C}} t_2$ and a literal $\langle \neg(\mathtt{apply}(st, t_1) \rightarrow t_2), \mathcal{C}\rangle$ as $t_1 \not\rightarrow_{st,\mathcal{C}} t_2$. If $\mathcal{C} = \emptyset$ then we simply write $t_1 \rightarrow_{st} t_2$ instead of $t_1 \rightarrow_{st,\emptyset} t_2$, and $t_1 \not\rightarrow_{st} t_2$ instead of $t_1 \not\rightarrow_{st,\emptyset} t_2$. We depict a program clause as a conditional rewrite rule of the form

$$t \rightarrow_{st,\mathcal{C}} t' \Leftarrow \bigwedge_{i=1}^{n} (t_i \rightharpoonup_{st_i,\mathcal{C}_i} t_i')$$

where $n \geq 0$ and $\rightharpoonup \in \{\rightarrow, \not\rightarrow\}$. If $n = 0$ then we elide the conditional part and write the program clause simply as $t \rightarrow_{st,\mathcal{C}} t'$.

ρLog is designed to answer queries of the form $\bigwedge_{i=1}^{n}(t_i \rightharpoonup_{st_i,\mathcal{C}_i} t_i')$ by using a form of SLDNF-resolution. Thus we consider rule-based programming as a special case of general logic programming, where the only defined symbol is the reducibility predicate $\rightarrow$. (See, e.g., [APT 94] for a survey on general logic programming.)

The SLDNF-resolution principle works well for first-order term languages, but is not suitable for the language of ρLog, where sequence variables and context variables

render an infinitary unification problem for terms. We have overcome this problem by identifying a class of queries and programs for which SLDNF-resolution can be performed by matching instead of unification. The advantage of matching versus unification is that matching is finitary and there are known matching algorithms both for unconstrained matching and for matching with regular constraints [KUT 05b].

In ρLog we require programs and queries to be *deterministic*. As we shall see later, in Section 3, determinism is a syntactic restriction that ensures the possibility to carry out SLDNF-resolution by matching instead of unification. In the remainder of this subsection we introduce our notion of determinism. To simplify this presentation, we employ the following notation:

– $vars(E)$ denotes the set of variables of a syntactic object E. We say E is *ground* if $vars(E) = \emptyset$.

– If $\mathcal{C}$ is a set of regular constraints then $cvars(\mathcal{C})$ denotes the set of all variables constrained by regular expressions in $\mathcal{C}$, and $evars(\mathcal{C})$ denotes the set of all non-anonymous variables that occur in regular expressions in $\mathcal{C}$ (the set of *extra variables* of $\mathcal{C}$). We also say that $\mathcal{C}$ *constrains* variables in $cvars(\mathcal{C})$.

DEFINITION 2. — *A program clause* $t'_0 \rightarrow_{st,\mathcal{C}_0} t_{n+1} \Leftarrow \bigwedge_{i=1}^{n}(t_i \rightharpoonup_{st_i,\mathcal{C}_i} t'_i)$ *is* deterministic *if it satisfies the following conditions:*

1) For all $1 \leq i \leq n$, $vars(st_i) \subseteq vars(st)$,

2) For all $1 \leq i \leq n$, *if the* i^{th} *literal of the conditional side of the clause is negative then* $vars(t'_i) \subseteq \mathcal{V}_{\text{any}} \cup \bigcup_{0 \leq j < i} vars(t'_j)$,

3) For all $1 \leq i \leq n+1$, $vars(t_i) \subseteq \bigcup_{0 \leq j < i} vars(t'_j) \setminus \mathcal{V}_{\text{any}}$,

4) For all $0 \leq i \leq n$,

- $\mathcal{C}_i$ *constrains any variable at most once, and*
- $cvars(\mathcal{C}_i) \subseteq vars(t'_i) \setminus \bigcup_{0 \leq j < i} vars(t'_j)$ *and* $evars(\mathcal{C}_i) \subseteq \bigcup_{0 \leq j < i} vars(t'_j)$.

A query $\bigwedge_{i=1}^{n}(t_i \rightharpoonup_{st_i,\mathcal{C}_i} t'_i)$ *is* deterministic *if it satisfies the following conditions for all* $1 \leq i \leq n$*:*

1) $vars(st_i) = \emptyset$ *and* $vars(t_i) \subseteq \bigcup_{1 \leq j < i} vars(t'_j) \setminus \mathcal{V}_{\text{any}}$,

2) If the i^{th} *literal is negative then* $vars(t'_i) \subseteq \mathcal{V}_{\text{any}} \cup \bigcup_{1 \leq j < i} vars(t'_j)$,

3) $\mathcal{C}_i$ *constrains any variable at most once, and*

4) $cvars(\mathcal{C}_i) \subseteq vars(t'_i) \setminus \bigcup_{1 \leq j < i} vars(t'_j)$ *and* $evars(\mathcal{C}_i) \subseteq \bigcup_{1 \leq j < i} vars(t'_j)$.

We will explain the relevance of all these syntactic restrictions in Section 3, when we introduce the resolution principle of ρLog and identify sufficient conditions to ensure the tractability of resolution steps by matching instead of unification.

2.3. *Built-in strategies*

For programming purposes, the meaning of the strategic constructors from $\mathcal{F}_{\mathtt{st}}$ should be built-in. More precisely, we want to ensure the following relationships for all ground hole-free terms t_1, t_2 and ground strategies st, st_1, st_2:

$t_1 \rightarrow_{\mathtt{Id}} t_2$	iff	$t_1 = t_2$,
$t_1 \rightarrow_{\mathtt{Compose}(st_1,st_2)} t_2$	iff	$t_1 \rightarrow_{st_1} t_3$ and $t_3 \rightarrow_{st_2} t_2$ for some ground term t_3,
$t_1 \rightarrow_{\mathtt{Choice}(st_1,st_2)} t_2$	iff	$t_1 \rightarrow_{st_1} t_2$ or $t_1 \rightarrow_{st_2} t_2$,
$t_1 \rightarrow_{\mathtt{Closure}(st)} t_2$	iff	$t_1 = t_2$ or there exists a sequence of ground terms $t'_1, \ldots, t'_n$ such that $t_1 \rightarrow_{st} t'_1, \ldots, t'_{n-1} \rightarrow_{st} t'_n$ and $t'_n = t_2$,
$t_1 \rightarrow_{\mathtt{NormalForm}(st)} t_2$	iff	$t_1 \rightarrow_{\mathtt{Closure}(st)} t_2$ and there is no ground term t_3 such that $t_2 \rightarrow_{st} t_3$,
$t_1 \rightarrow_{\mathtt{OrElse}(st_1,st_2)} t_2$	iff	$t_1 \rightarrow_{st_1} t_2$ or else $t_1 \rightarrow_{st_2} t_2$.

We enforce this interpretation by considering the following built-in program:

$$
\begin{aligned}
\mathcal{P}_{\mathtt{st}} = \{\ & \mathtt{x} \rightarrow_{\mathtt{Id}} \mathtt{x}. \\
& \mathtt{x} \rightarrow_{\mathtt{Compose}(\sigma_1,\sigma_2)} \mathtt{z} \Leftarrow (\mathtt{x} \rightarrow_{\sigma_1} \mathtt{y}) \wedge (\mathtt{y} \rightarrow_{\sigma_2} \mathtt{z}). \\
& \mathtt{x} \rightarrow_{\mathtt{Choice}(\sigma_1,\sigma_2)} \mathtt{y} \Leftarrow (\mathtt{x} \rightarrow_{\sigma_1} \mathtt{y}). \\
& \mathtt{x} \rightarrow_{\mathtt{Choice}(\sigma_1,\sigma_2)} \mathtt{y} \Leftarrow (\mathtt{x} \rightarrow_{\sigma_2} \mathtt{y}). \\
& \mathtt{x} \rightarrow_{\mathtt{Closure}(\sigma)} \mathtt{y} \Leftarrow (\mathtt{x} \rightarrow_{\mathtt{Id}} \mathtt{y}). \\
& \mathtt{x} \rightarrow_{\mathtt{Closure}(\sigma)} \mathtt{y} \Leftarrow (\mathtt{x} \rightarrow_{\sigma} \mathtt{z}) \wedge (\mathtt{z} \rightarrow_{\mathtt{Closure}(\sigma)} \mathtt{y}). \\
& \mathtt{x} \rightarrow_{\mathtt{NormalForm}(\sigma)} \mathtt{y} \Leftarrow (\mathtt{x} \rightarrow_{\mathtt{Closure}(\sigma)} \mathtt{y}) \wedge (\mathtt{y} \not\rightarrow_{\sigma} \mathtt{\_i}). \\
& \mathtt{x} \rightarrow_{\mathtt{OrElse}(\sigma_1,\sigma_2)} \mathtt{y} \Leftarrow (\mathtt{x} \rightarrow_{\sigma_1} \mathtt{y}), \\
& \mathtt{x} \rightarrow_{\mathtt{OrElse}(\sigma_1,\sigma_2)} \mathtt{y} \Leftarrow (\mathtt{x} \not\rightarrow_{\sigma_1} \mathtt{\_i}) \wedge (\mathtt{x} \rightarrow_{\sigma_2} \mathtt{y}).\}
\end{aligned}
$$

where $\mathtt{x}, \mathtt{y}, \mathtt{z} \in \mathcal{V}_{\mathtt{i}}$ and $\sigma, \sigma_1, \sigma_2 \in \mathcal{V}_{\mathtt{st}}$. Note that all clauses of $\mathcal{P}_{\mathtt{st}}$ are deterministic.

3. The ρLog Calculus

ρLog is designed to answer deterministic queries in theories represented by programs made of deterministic clauses. In general, the structure of a ρLog program $\mathcal{P}$ is $\mathcal{P} := \mathcal{P}_{\mathtt{st}} \cup \mathcal{P}_{\mathtt{u}}$ where

– $\mathcal{P}_{\mathtt{st}}$ is the deterministic program introduced in the previous subsection; It provides default interpretation for the strategy operators from $\mathcal{F}_{\mathtt{st}}$.

– $\mathcal{P}_{\mathtt{u}}$ is a program that provides interpretation for user-definable strategies. $\mathcal{P}_{\mathtt{u}}$ is specified by the user, and consists of deterministic clauses of the form

$$t \rightarrow_{st,\mathcal{C}} t' \Leftarrow \bigwedge_{i=1}^{n} (t_i \rightarrow_{st_i,\mathcal{C}_i} t'_i)$$

where *st* is a basic strategy.

The computational model of ρLog is, in essence, SLDNF-resolution with leftmost literal selection: Every program clause $t \rightarrow_{st_0,\mathcal{C}_0} t_{n+1} \Leftarrow \bigwedge_{i=1}^{n}(t_i \rightarrow_{st_i,\mathcal{C}_i} t'_i)$ is logically equivalent to the clause $t \rightarrow_{st_0,\mathcal{C}_0} x \Leftarrow \bigwedge_{i=1}^{n}(t_i \rightarrow_{st_i,\mathcal{C}_i} t'_i) \wedge (t_{n+1} \rightarrow_{\mathtt{Id}} x)$ where x is a fresh individual variable, and therefore we can use resolution with respect to these equivalent clauses. The inference rules that define the resolution calculus of ρLog are shown in Figure 1, where $mcsm(E_1 \ll E_2, \mathcal{C})$ denotes the minimal complete set of matchers of E_1 and E_2 which satisfy the regular constraints from $\mathcal{C}$.

$$\frac{(t \rightarrow_{st,\mathcal{C}} t') \wedge G}{(\bigwedge_{i=1}^{n}(t_i \rightarrow_{st_i,\mathcal{C}_i} t'_i) \wedge G \wedge (t_{n+1} \rightarrow_{\mathtt{Id},\mathcal{C}} t'))\theta}$$

if $st \neq \mathtt{Id}$, where $t_0 \rightarrow_{st_0,\mathcal{C}_0} t_{n+1} \Leftarrow \bigwedge_{i=1}^{n}(t_i \rightarrow_{st_i,\mathcal{C}_i} t'_i)$ is a fresh variant of a clause from $\mathcal{P}_{\mathtt{st}} \cup \mathcal{P}_{\mathtt{u}}$ and $\theta \in mcsm(\mathtt{apply}(st_0, t_0) \ll \mathtt{apply}(st, t), \mathcal{C}_0)$.

$$\frac{(t \rightarrow_{\mathtt{Id},\mathcal{C}} t') \wedge G}{G\theta}$$

where $\theta \in mcsm(t' \ll t, \mathcal{C})$.

$$\frac{(t \not\rightarrow_{st,\mathcal{C}} t') \wedge G}{G}$$

if there exists a finite failed SLDNF-derivation tree of $t \rightarrow_{st,\mathcal{C}} t'$ with respect to the program $\mathcal{P}_{\mathtt{st}} \cup \mathcal{P}_{\mathtt{u}}$.

Figure 1. *The resolution calculus of ρLog.*

The main differences between SLDNF-resolution and the calculus of ρLog are:

– the usage of $mcsm(\mathtt{apply}(st_0, t_0) \ll \mathtt{apply}(st, t), \mathcal{C}_0)$ instead of
$mcsu(t \rightarrow_{st} t' \equiv t_0 \rightarrow_{st_0} x, \mathcal{C}_0)$ when $st \neq \mathtt{Id}$, and
– the usage of $mcsm(t \ll t', \mathcal{C})$ instead of
$mcsu(t \rightarrow_{\mathtt{Id}} t' \equiv x \rightarrow_{\mathtt{Id}} x, \mathcal{C})$ when $st = \mathtt{Id}$

where $mcsu(E_1 \equiv E_2, \mathcal{C})$ is the notation for a minimal complete set of unifiers between E and E' which satisfy the regular constraints from $\mathcal{C}$. We recall that the terms of our language may contain anonymous variables, and therefore the notions of unifier and matcher between terms must be defined with some care: First, every occurrence of an anonymous variable in the terms under consideration is replaced by a fresh new variable of the same sort; after computing the substitution (unifier or matcher), the variables that were newly introduced at the beginning are back-substituted into anonymous variables, and the bindings for anonymous variables are removed.

Unfortunately, $mcsu(E_1 \equiv E_2, \mathcal{C})$ is usually an infinite set, and *this* is the place where deterministic goals and queries overcome this problem: It can be shown that any ρLog-derivation with leftmost literal selection of a deterministic goal with respect to a deterministic program has the following properties:

1) Every selected literal $t \rightarrow_{st,\mathcal{C}} t'$ has $vars(t) = vars(st) = \emptyset$.

2) Every selected literal $t \rightharpoonup_{st,\mathcal{C}} t'$ has $cvars(\mathcal{C}) \subseteq vars(t')$ and $evars(\mathcal{C}) = \emptyset$.

3) Every selected literal $t \nrightarrow_{st,\mathcal{C}} t'$ has $vars(t') \subseteq \mathcal{V}_{\rm any}$.

Therefore, whenever we select an atom $t \rightharpoonup_{st,\mathcal{C}} t'$ for resolution, we have $vars(t) = vars(st) = \emptyset$, $cvars(\mathcal{C}) \subseteq vars(t')$, $evars(\mathcal{C}) = \emptyset$, and are in one of the following two situations:

I) $st \neq \mathtt{Id}$ and resolution is with respect to a fresh variant

$$t_0 \rightharpoonup_{st_0,\mathcal{C}_0} x \Leftarrow \bigwedge_{i=1}^{n} (t_i \rightharpoonup_{st_i,\mathcal{C}_i} t_i') \wedge (t_{n+1} \rightharpoonup_{\mathtt{Id}} x). \tag{1}$$

In this situation $mcsu(t \rightharpoonup_{st} t' \equiv t_0 \rightharpoonup_{st_0} x, \mathcal{C}_0) = \{\theta \cup \{x \mapsto t'\theta\} \mid \theta \in mcsm(\mathtt{apply}(st_0, t_0) \ll \mathtt{apply}(st, t), \mathcal{C}_0)\}$, and therefore SLDNF-resolution with respect to (1) coincides with the corresponding inference rule of ρLog.

II) $st = \mathtt{Id}$ and resolution is with respect to a fresh variant $x \rightharpoonup_{\mathtt{Id}} x$. In this case $mcsu(t \rightharpoonup_{\mathtt{Id}} t' \equiv x \rightharpoonup_{\mathtt{Id}} x, \mathcal{C}_0) = \{\theta \cup \{x \mapsto t\} \mid \theta \in mcsm(t' \ll t, \mathcal{C}_0)\}$ and, again, SLDNF-resolution coincides with the corresponding inference rule of ρLog.

Hence, the resolution calculus of ρLog inherits all the properties of SLDNF-resolution with leftmost literal selection. In particular, our calculus is sound and incomplete because leftmost literal selection is an unfair selection strategy. Completeness can be recovered if we adopt additional restrictions to guarantee termination of $\mathcal{P}_{\mathtt{st}} \cup \mathcal{P}_{\mathtt{u}}$. We do not address the termination problem in this paper.

Property 2) is important for the matching algorithm with regular constraints which computes $mcsm(E \ll E', \mathcal{C})$ [KUT 05b].

Our notion of deterministic clause is a generalization of the notion of deterministic 3-CTRS [OHL 01] to the case when negative literals are allowed in the conditional part. Condition 2) of Definition 2 on program clauses overcomes the well-known problematic interpretation of negative literals in general logic programming with SLDNF-resolution [APT 94], by ensuring the fact that SLDNF-resolution with leftmost literal selection selects only negative literals with property 3) mentioned above.

ρLog may also access powerful external libraries to carry out symbolic or numeric computations. The only requirement is the availability of an interface $eval()$ that enables to evaluate any ground term t to a boolean value $eval(t) \in \{0, 1\}$. This capability is embedded in the framework of ρLog via *boolean atoms*. A *boolean atom* is modeled as a reducibility atom of the form $t \rightharpoonup_{\mathtt{Id}} \mathtt{True}$ where $\mathtt{True} \in \mathcal{F}$ is a special function symbol that indicates the evaluation of this literal should rely on the interface $eval()$ to some external libraries.

The inference rule for boolean atoms is

$$\frac{(t \rightharpoonup_{\mathtt{Id}} \mathtt{True}) \wedge G}{G}$$

if $eval(t) = 1$. In general we will abbreviate the boolean atoms and write t instead of $t \rightharpoonup_{\mathtt{Id}} \mathtt{True}$ inside goals and program clauses.

Notes on implementation

Our implementation of the calculus of ρLog takes advantage of the observation that the resolution steps can be regarded as rewrite steps without evaluating conditions with respect to the conditional term rewriting system $\mathcal{P}_{\mathtt{st}} \cup \mathcal{P}_{\mathtt{u}}$, in the sense defined by Bockmayr in [BOC 90]. This can be easily seen in Figure 1 where, for the case $st \neq \mathtt{Id}$, we compute a matcher between a subterm of the current goal and the left-hand side of the conditional rewrite rule. Since the Mathematica interpreter has a built-in mechanism for this kind of rewriting (with built-in backtracking), we found convenient to implement ρLog as a Mathematica package [MAR 06]. The documented package is available at `www.score.score.cs.tsukuba.ac.jp/~mmarin/RhoLog/`.

4. Applications

The most obvious application of ρLog is in automated reasoning, where rule-based programs can be used to model inference rules, and strategies may enforce an efficient exploration of the space for solutions. We have used this approach for fast prototyping of unification algorithms with sequence variables in free, flat, and restricted flat theories [MAR 03], and of lazy narrowing calculi for theories presented by both unconditional and conditional term rewrite systems [MAR 04b]. Pattern matching with regular constraints makes ρLog useful for querying and manipulating data with regular (sub)structures, such as XML documents [KUT 05a], or genetic data.

In this section we illustrate the capabilities of ρLog by emphasizing the concise and declarative character of this programming style.

4.1. *Automated Deduction*

Proof derivations are sequences of inference steps. For several deductive systems, the inference rules can be modeled by program clauses for basic strategies, and proof derivations correspond to sequences of resolution steps produced by unfolding certain strategies. In these situations, rule-based programming provides a natural encoding of the deductive system and support for proof search: the resolution derivation obtained in this way constitutes a proof certificate. By tracing the derivations of ρLog, we can generate certificates which justify the correctness of proofs.

To illustrate, we consider Gentzen's system G' [GAL 85], which is a sequent calculus for the fragment of propositional logic with logical connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication) and $\neg$ (negation). The inference rules of system G' are shown in Table 1. The metavariables $\Gamma, \Delta, \Lambda, \Omega$ range over sequences of propositional formulas, L and R are lists of formulas, and A and B are propositional formulas.

System G' has a a straightforward encoding into ρLog program clauses:

$\{\__{\mathtt{s}}, A, \__{\mathtt{s}}\} \vdash \{\__{\mathtt{s}}, A, \__{\mathtt{s}}\} \rightarrow_{G'} \mathtt{true}.$
$\{\Gamma, A \wedge B, \Delta\} \vdash R \rightarrow_{G'} \mathtt{true} \Leftarrow (\{\Gamma, A, B, \Delta\} \vdash R \rightarrow_{G'} \mathtt{true}).$

$$\frac{}{\{\Gamma, A, \Delta\} \vdash \{\Lambda, A, \Omega\}}$$

$$\frac{\{\Gamma, A, B, \Delta\} \vdash R}{\{\Gamma, A \wedge B, \Delta\} \vdash R} \qquad \frac{L \vdash \{\Gamma, A, \Delta\} \quad L \vdash \{\Gamma, B, \Delta\}}{L \vdash \{\Gamma, A \wedge B, \Delta\}}$$

$$\frac{\{\Gamma, A, \Delta\} \vdash R \quad \{\Gamma, B, \Delta\} \vdash R}{\{\Gamma, A \vee B, \Delta\} \vdash R} \qquad \frac{L \vdash \{\Gamma, A, B, \Delta\}}{L \vdash \{\Gamma, A \vee B, \Delta\}}$$

$$\frac{\{\Gamma, \Delta\} \vdash \{A, \Lambda\} \quad \{B, \Gamma, \Delta\} \vdash \{\Lambda\}}{\{\Gamma, A \Rightarrow B, \Delta\} \vdash \{\Lambda\}} \qquad \frac{\{A, \Gamma\} \vdash \{B, \Delta, \Lambda\}}{\{\Gamma\} \vdash \{\Delta, A \Rightarrow B, \Lambda\}}$$

$$\frac{\{\Gamma, \Delta\} \vdash \{A, \Lambda\}}{\{\Gamma, \neg A, \Delta\} \vdash \{\Lambda\}} \qquad \frac{\{A, \Gamma\} \vdash \{\Delta, \Lambda\}}{\{\Gamma\} \vdash \{\Delta, \neg A, \Lambda\}}$$

Table 1. *System G' for propositional logic.*

$$\begin{array}{ll}
L \vdash \{\Gamma, A \wedge B, \Delta\} \rightarrow_{G'} \texttt{true} \Leftarrow & (L \vdash \{\Gamma, A, \Delta\} \rightarrow_{G'} \texttt{true}) \wedge \\
& (L \vdash \{\Gamma, B, \Delta\} \rightarrow_{G'} \texttt{true}). \\
\{\Gamma, A \vee B, \Delta\} \vdash R \rightarrow_{G'} \texttt{true} \Leftarrow & (\{\Gamma, A, \Delta\} \vdash R \rightarrow_{G'} \texttt{true}) \wedge \\
& (\{\Gamma, B, \Delta\} \vdash R \rightarrow_{G'} \texttt{true}). \\
L \vdash \{\Gamma, A \vee B, \Delta\} \rightarrow_{G'} \texttt{true} \Leftarrow & (L \vdash \{\Gamma, A, B, \Delta\} \rightarrow_{G'} \texttt{true}). \\
\{\Gamma, A \Rightarrow B, \Delta\} \vdash \{\Lambda\} \rightarrow_{G'} \texttt{true} \Leftarrow & (\{\Gamma, \Delta\} \vdash \{A, \Lambda\} \rightarrow_{G'} \texttt{true}) \wedge \\
& (\{B, \Gamma, \Delta\} \vdash \{\Lambda\} \rightarrow_{G'} \texttt{true}). \\
\{\Gamma\} \vdash \{\Delta, A \Rightarrow B, \Lambda\} \rightarrow_{G'} \texttt{true} \Leftarrow & (\{A, \Gamma\} \vdash \{B, \Delta, \Lambda\} \rightarrow_{G'} \texttt{true}). \\
\{\Gamma, \neg A, \Delta\} \vdash \{\Lambda\} \rightarrow_{G'} \texttt{true} \Leftarrow & (\{\Gamma, \Delta\} \vdash \{A, \Lambda\} \rightarrow_{G'} \texttt{true}). \\
\{\Gamma\} \vdash \{\Delta, \neg A, \Lambda\} \rightarrow_{G'} \texttt{true} \Leftarrow & (\{A, \Gamma\} \vdash \{\Delta, \Lambda\} \rightarrow_{G'} \texttt{true}).
\end{array}$$

where $\Gamma, \Delta, \Lambda \in \mathcal{V}_{\mathtt{s}}$, $A, B \in \mathcal{V}_{\mathtt{i}}$, $G' \in \mathcal{L}$, and `list`, `entails`, `implies`, `and`, `or`, `not`, `true` are function symbols for which we consider the following abbreviations:

$$\begin{array}{ll}
t_1 \vdash t_2 \text{ instead of } \texttt{entails}(t_1, t_2), & \{ts\} \text{ instead of } \texttt{list}(ts), \\
t_1 \wedge t_2 \text{ instead of } \texttt{and}(t_1, t_2), & t_1 \vee t_2 \text{ instead of } \texttt{or}(t_1, t_2), \\
t_1 \Rightarrow t_2 \text{ instead of } \texttt{implies}(t_1, t_2), & \text{and } \neg t \text{ instead of } \texttt{not}(t).
\end{array}$$

Let's check whether the formula $(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))$ is a tautology or not. This amounts to checking whether there is a ρLog refutation of the query

$$\{\} \vdash \{(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))\} \rightarrow_{G'} \texttt{true}.$$

We can refute this query as follows:

$$\begin{array}{l}
\{\} \vdash \{(P \Rightarrow Q) \Rightarrow ((\neg Q) \Rightarrow (\neg P))\} \rightarrow_{G'} \texttt{true} \rightsquigarrow_7 \\
(\{P \Rightarrow Q\} \vdash \{(\neg Q) \Rightarrow (\neg P)\} \rightarrow_{G'} \texttt{true}) \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \rightsquigarrow_7 \\
(\{\neg Q, P \Rightarrow Q\} \vdash \{\neg P\} \rightarrow_{G'} \texttt{true}) \ \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \rightsquigarrow_8 \\
(\{P \Rightarrow Q\} \vdash \{Q, \neg P\} \rightarrow_{G'} \texttt{true}) \ \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \\
\qquad\qquad \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \rightsquigarrow_9 \\
(\{P, P \Rightarrow Q\} \vdash \{Q\} \rightarrow_{G'} \texttt{true}) \ \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \\
\qquad\qquad \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \wedge (\texttt{true} \rightarrow_{\texttt{Id}} \texttt{true}) \rightsquigarrow_6
\end{array}$$

$$
\begin{array}{l}
(\{P\} \vdash \{P,Q\} \rightarrow_{G'} \mathtt{true}) \ \wedge(\{Q,P\} \vdash \{Q\} \rightarrow_{G'} \mathtt{true}) \\
\qquad\qquad\qquad\qquad \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \\
\qquad\qquad\qquad\qquad \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \rightsquigarrow_1 \\
(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \ \wedge(\{Q,P\} \vdash \{Q\} \rightarrow_{G'} \mathtt{true}) \\
\qquad\qquad\qquad\quad \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \\
\qquad\qquad\qquad\quad \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \rightsquigarrow_{\mathtt{Id}} \\
(\{Q,P\} \vdash \{Q\} \rightarrow_{G'} \mathtt{true}) \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \\
\qquad\qquad\qquad\qquad \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \rightsquigarrow_1 \\
(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \\
\qquad\qquad\qquad\quad \wedge(\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \wedge (\mathtt{true} \rightarrow_{\mathtt{Id}} \mathtt{true}) \rightsquigarrow_{\mathtt{Id}} \cdots \rightsquigarrow_{\mathtt{Id}} \Box.
\end{array}
$$

where $\rightsquigarrow_n$ indicates a ρLog resolution step with respect to the n-th program clause for system G', and $\rightsquigarrow_{\mathtt{Id}}$ is a ρLog resolution step when the applied strategy is `Id`.

4.2. *Bioinformatics*

Bioinformatics is an area of research where the analysis of long sequences of DNA or aminoacids reveals important genetic information. There are many situations when relevant genetic information can be detected by the identification of regular sequence patterns in DNA strands.

First, we illustrate the simplicity of encoding the translation of segments of RNA into proteins. The overall translation is achieved by translating successively triplets of nucleotides (*codons*) into aminoacids in accordance with the universal genetic code. There are 20 known aminoacids, and we will encode them as terms of the form $a()$ with a from the subset $\{A, V, L, I, G, C, F, W, M, P, S, T, Y, N, Q, D, E, K, R, H\}$ of $\mathcal{F}$. We have chosen the names of these function symbols to coincide with the standard one-letter abbreviations of the aminoacids. The RNA nucleotides are encoded as terms of the form $n()$ with $n \in \{\mathtt{U}, \mathtt{C}, \mathtt{A}, \mathtt{G}\} \subset \mathcal{F}$.

The genetic code has 64 translation rules, one for every possible codon, but we can encode them with 24 ρLog rules by using anonymous variables. There are 3 codons that do not actually translate into an aminoacid, but trigger the termination of the translation process. We encode this fact by translating those codons into a special term, $\mathtt{Ter}()$, where `Ter` is a special function symbol that indicates termination of translation. In ρLog, the genetic code can be encoded as shown in Figure 2. The translation of an RNA sequence can be easily encoded in two program clauses:

$$
\begin{array}{l}
\{\} \rightarrow_{\mathtt{transl}} \{\}. \\
\{x, \overline{x}, \overline{y}\} \rightarrow_{\mathtt{transl}} \{z, \overline{z}\} \Leftarrow (\{x, \overline{x}\} \rightarrow_{\mathtt{uc}} z) \wedge (\{\overline{y}\} \rightarrow_{\mathtt{transl}} \{\overline{z}\}).
\end{array}
$$

As a second example, we illustrate the usefulness of ρLog in the detection of genetic mutations. One well-known kind of mutation is the *trinucleotide repeat expansion*, discovered in humans in 1991, which denotes an increase in the number of

$\{\mathtt{U},\mathtt{U},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}\}} F.$
$\{\mathtt{U},\mathtt{U},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{A}|\mathtt{G}\}} L.$
$\{\mathtt{C},\mathtt{U},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} L.$
$\{\mathtt{A},\mathtt{U},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}|\mathtt{A}\}} I.$
$\{\mathtt{A},\mathtt{U},\mathtt{G}\} \rightarrow_{\mathtt{uc}} M.$
$\{\mathtt{G},\mathtt{U},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} V.$
$\{\mathtt{U},\mathtt{C},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} S,$
$\{\mathtt{C},\mathtt{C},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} P.$
$\{\mathtt{A},\mathtt{C},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} T.$
$\{\mathtt{G},\mathtt{C},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} A.$
$\{\mathtt{U},\mathtt{A},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}\}} Y.$
$\{\mathtt{U},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\ulcorner\mathtt{A},\mathtt{A}\urcorner|\ulcorner\mathtt{A},\mathtt{G}\urcorner|\ulcorner\mathtt{G},\mathtt{A}\urcorner\}} \mathtt{Ter}.$
$\{\mathtt{C},\mathtt{A},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}\}} H.$
$\{\mathtt{C},\mathtt{A},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{A}|\mathtt{G}\}} Q.$
$\{\mathtt{A},\mathtt{A},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}\}} N.$
$\{\mathtt{A},\mathtt{A},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{A}|\mathtt{G}\}} K.$
$\{\mathtt{G},\mathtt{A},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}\}} D.$
$\{\mathtt{G},\mathtt{A},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{A}|\mathtt{G}\}} E.$
$\{\mathtt{U},\mathtt{G},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}\}} C.$
$\{\mathtt{U},\mathtt{G},\mathtt{G}\} \rightarrow_{\mathtt{uc}} W.$
$\{\mathtt{C},\mathtt{G},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} R.$
$\{\mathtt{A},\mathtt{G},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{U}|\mathtt{C}\}} S.$
$\{\mathtt{A},\mathtt{G},\overline{x}\} \rightarrow_{\mathtt{uc},\{\overline{x}:\mathtt{A}|\mathtt{G}\}} R.$
$\{\mathtt{G},\_\mathtt{i},\_\mathtt{i}\} \rightarrow_{\mathtt{uc}} G.$

Figure 2. *ρLog encoding of the generic code.*

trinucleotide tandem repeats. The *fragile X syndrome* is the most thoroughly studied syndrome, responsible for over 10 genetic disorders. This syndrome is due to the expansion of a certain region in DNA where the trinucleotide CGG is repeated several times in tandem. People whose DNA region contains more than 50 repeats of this trinucleotide have high probability to show symptoms of genetic disorder [FAI 99].

With ρLog we can detect such abnormal CGG repeats by answering the query

$$G \rightarrow_{\mathtt{Id},\{\overline{y}:\ulcorner\mathtt{C},\mathtt{G},\mathtt{G}\urcorner^{50}\}} \{\ulcorner\overline{x},\overline{y},\_\mathtt{s}\urcorner\}$$

where G denotes the region of DNA that is prone to the fragile X syndrome, encoded as a list of terms of the form $n()$ with $n \in \{\mathtt{C},\mathtt{G},\mathtt{A},\mathtt{T}\} \subset \mathcal{F}$.

4.3. *Term Rewriting Strategies*

Labeled rules act on expressions by trying to transform the expression as a whole. Term rewriting is a transformation relation which is closed under contexts, i.e., it can act on a term by selecting and transforming certain subterms. Often, term rewriting relations are defined by starting with a finite set of transformation rules. Efficient computations can be achieved by defining rewriting strategies which constrain the selection of the subterms which are going to be transformed.

In ρLog, term rewriting strategies can be easily specified. To illustrate, let's start with a set $\mathcal{PA}$ of transformation rules which formalize the operations of addition and multiplication in Peano arithmetic:

$$0 \oplus x \rightarrow_{\mathtt{PA}} x, \quad s(x) \oplus y \rightarrow_{\mathtt{PA}} s(x \oplus y),$$
$$0 \otimes x \rightarrow_{\mathtt{PA}} 0, \quad s(x) \otimes y \rightarrow_{\mathtt{PA}} y \oplus (x \otimes y)$$

where $x, y \in \mathcal{V}_i$, $\mathtt{PA} \in \mathcal{L}$, and $0, s, \oplus, \otimes$ are function symbols. The rewrite relation induced by the term rewrite system $\mathcal{PA}$ is the reducibility relation induced by the labeled rule `rw-PA` defined by the program clause

$$x \to_{\texttt{rw-PA}} \overline{C}(x_2) \Leftarrow (x \to_{\texttt{Id},\{\overline{C}:\_\texttt{f}(\_\texttt{s},\diamond,\_\texttt{s})^*\}} \overline{C}(x_1)) \wedge (x_1 \to_{\texttt{PA}} x_2).$$

where $x, x_1, x_2 \in \mathcal{V}_\texttt{i}$ and $\overline{C} \in \mathcal{V}_\texttt{c}$. This program clause is a straightforward transliteration in ρLog of the definition of term rewriting. We can take advantage of the fact that the rewrite relation induced by PA is terminating, and define an evaluator for Peano arithmetic by adding the program clause

$$x \to_{\texttt{eval}} y \Leftarrow (x \to_{\texttt{NormalForm(rw-PA)}} y).$$

Rule rw-PA does not impose any restriction on the selection of the subterm which is rewritten. This is one of the reasons why the rewriting relation $\to_{\texttt{rw-PA}}$ is nondeterministic. Rewriting strategies restrict the selection of the subterm which is rewritten. We illustrate the ρLog implementation of two popular rewriting strategies: innermost rewriting and outermost rewriting. *Innermost rewriting* always selects an innermost subterm which can be transformed, and *outermost rewriting* always selects an outermost subterm which can be transformed. This kind of behavior can be specified with the following user-defined program clauses:

$$\begin{array}{l} x \to_{\texttt{I-rw-PA}} y \Leftarrow (x \to_{\texttt{OrElse(I-rw-PA-1, PA)}} y). \\ x \to_{\texttt{O-rw-PA}} y \Leftarrow (x \to_{\texttt{OrElse(PA, O-rw-PA-1)}} y). \\ F(\overline{x}, x, \overline{z}) \to_{\texttt{I-rw-PA-1}} F(\overline{x}, y, \overline{z}) \Leftarrow (x \to_{\texttt{I-rw-PA}} y). \\ F(\overline{x}, x, \overline{z}) \to_{\texttt{O-rw-PA-1}} F(\overline{x}, y, \overline{z}) \Leftarrow (x \to_{\texttt{O-rw-PA}} y). \end{array}$$

It is not hard to see that the reduction relation $\to_{\texttt{I-rw-PA}}$ encodes leftmost innermost term rewriting, and $\to_{\texttt{O-rw-PA}}$ encodes leftmost outermost term rewriting:

– Innermost subterm selection is achieved by giving priority to rule I-rw-PA-1 (which rewrites a subterm) over rule PA (which transforms the term as a whole).

– Outermost subterm selection is achieved by giving priority to rule PA (which transforms the term as a whole) over rule O-rw-PA-1 (which rewrites a subterm).

5. Conclusion and Future Work

The interplay between (1) matching with context and sequence variables and context variables, (2) regular constraints, and (3) strategic programming with labeled rules provides support for concise and natural implementations of several applications. Since the basic principles of matching and rule-based programming advocated by us are so pervasive in all areas of sciences and engineering, we believe that ρLog could become a very useful programming paradigm.

As future work, we intend to extend matching with allowing regular expressions inside terms and permitting in regular constraints the same variable to be constrained by several regular expressions. This would make the language more expressive, and would allow to relax some current restrictions on regular constraints.

Having ρLog implemented in Mathematica has several advantages that have been emphasized in [MAR 04d]. However, it has also a major drawback: Mathematica is not free. Therefore, we plan to reimplement ρLog on a free platform.

Acknowledgements

Mircea Marin has been supported by the JSPS Grant-in-Aid no. 17700025 for Scientific Research sponsored by the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT). Temur Kutsia has been supported by the Austrian Science Foundation FWF, under the SFB project F1302.

Part of this work was done while the second author was visiting the SCORE lab at University of Tsukuba.

6. References

[APT 94] APT K. R., BOL R. N., "Logic Programming and Negation: A Survey", *Journal of Logic Programming*, vol. 19/20, 1994, p. 9-71.

[BER 89] BERGSTRA J., HEERING J., KLINT P., "The Algebraic Specification formalism ASF", BERGSTRA J., HEERING J., KLINT P., Eds., *Algebraic Specification*, ACM Press in cooperation with Addison Wesley Frontier Series, 1989, p. 1-66.

[BER 04] BERTOLISSI C., BALDAN P., CIRSTEA H., KIRCHNER C., "A rewriting calculus for cyclic higher-order term graphs", *Wokshop on term graph rewriting*, 2004.

[BOC 90] BOCKMAYR A., "Beiträge zur Theorie des Logisch-Funktionalen Programmierens", PhD thesis, Universität Karlsruhe, 1990, In German.

[BOR 02] BOROVANSKÝ P., KIRCHNER C., KIRCHNER H., MOREAU P.-E., "ELAN from a rewriting logic point of view", *TCS*, vol. 285, num. 2, 2002, p. 155–185.

[CIR 03] CIRSTEA H., LIQUORI L., WACK B., "Rewriting Calculus with Fixpoints: Untyped and First-order Systems", *Post-proceedings of TYPES'03*, 2003.

[CIR 04] CIRSTEA H., FAURE G., KIRCHNER C., "A Rho-calculus of explicit constraint application", *Proceedings of the 5th workshop on rewriting logic and applications*, 2004.

[CLA 02] CLAVEL M., DURÁN F., EKER S., LINCOLN P., MARTÍ-OLIET N., MESEGUER J., QUESADA J., "Maude: Specification and Programming in Rewriting Logic", *TCS*, vol. 285, num. 2, 2002, p. 186–243.

[DIA 02] DIACONESCU R., FUTATSUGI K., "Logical foundations of CafeOBJ", *TCS*, vol. 285, num. 2, 2002, p. 289–318.

[FAI 99] FAIRBANKS D. J., ANDERSON W. R., *Genetics: the continuity of life*, Brooks/Cole Publishing Company • Wadsworth Publishing Company, 1999.

[GAL 85] GALLIER J., *Logic for computer science: foundations of automatic theorem proving*, Harper & Row Publishers, Inc., New York, NY, USA, 1985.

[KUT 05a] KUTSIA T., MARIN M., "Can Context Sequence Matching be Used in XML Querying?", VIGNERON L., Ed., *Proceedings of UNIF'05*, Nara, Japan, 2005, p. 77–95.

[KUT 05b] KUTSIA T., MARIN M., "Matching with Regular Constraints", SUTCLIFFE G., VORONKOV A., Eds., *Proceedings of LPAR'05*, vol. 3835 of *LNAI*, Springer, 2005, p. 215–229.

[MAR 03] MARIN M., KUTSIA T., "On the Implementation of a Rule-Based Programming System and Some of its Applications", KONEV B., SCHMIDT R., Eds., *Proceedings of the 4th International Workshop on the Implementation of Logics*, Almaty, Kazakhstan, 2003, p. 55–69.

[MAR 04a] MARIN M., KUTSIA T., "A Rule-based Approach to the Implementation of Evaluation Strategies", PETCU D., NEGRU V., ZAHARIE D., JEBELEAN T., Eds., *Proceedings of SYNASC 2004*, Timişoara, Romania, 2004, Mirton, p. 227-241.

[MAR 04b] MARIN M., MIDDELDORP A., "New Completeness Results for Lazy Conditional Narrowing", *Proceedings of the Sixth ACM-SIGPLAN Conference on Principles and Practice of Declarative Programming*, Verona, Italy, 2004, ACM Press, p. 120-131.

[MAR 04c] MARIN M., PIROI F., "Deduction and Presentation in ρLog", KAMAREDDINE F., Ed., *Mathematical Knowledge Management Symposium*, vol. 93 of *ENTCS*, Edinburgh, Scotland, 2004, Elsevier, p. 161-182.

[MAR 04d] MARIN M., PIROI F., "Rule-based Programming with Mathematica", *Electronic Proceedings of IMS 2004*, Banff, Alberta, Canada, 2004.

[MAR 06] MARIN M., IDA T., "Progress of ρLog, a rule-based programming system", *Mathematica in Education and Research*, vol. 11, num. 1, 2006, p. 50-66, *i*Journals.

[MES 92] MESEGUER J., "Conditional rewriting logic as a unified model of concurrency", *TCS*, vol. 1, num. 96, 1992, p. 73–155.

[OHL 01] OHLEBUSCH E., "Termination of Logic Programs: Transformational Methods Revisited", *AAECC*, vol. 12, 2001, p. 73–116.

[VIS 98] VISSER E., BENAISSA Z.-E.-A., "A core language for rewriting", KIRCHNER C., KIRCHNER H., Eds., *WRLA'98*, vol. 15 of *ENTCS*, 1998.

[VIS 04] VISSER E., "Program Transformation with Stratego/XT: Rules, Strategies, Tools, and Systems in StrategoXT-0.9", C. LENGAUER ET. AL, Ed., *Domain-Specific Program Generation*, vol. 3016 of *LNCS*, 2004, p. 216–238.