A Rewrite Stack Machine for ROC!

Georgiana Caltais Eugen-Ioan Goriac Dorel Lucanu Gheorghe Grigoraş
Faculty of Computer Science
Alexandru Ioan Cuza University
Iaşi, Romania
{gcaltais,egoriac,dlucanu,grigoras}@info.uaic.ro

***Abstract*—ROC! is a deterministic rewrite strategy language which includes the rewrite rules as basic operators, and the deterministic choice and the repetition as high-level strategy operators. In this paper we present a method which, for a given term rewriting system (TRS) R, constructs a new TRS $\overline{\mathrm{R}}$ such that $\overline{\mathrm{R}}$-rewriting is equivalent (sound and complete) with R-rewriting constrained by ROC! . Since $\overline{\mathrm{R}}$ uses a stack, it is called a *rewrite stack machine*.**

I. Introduction

Rewriting strategies are expressions built over a strategy language used for controlling the rewrite rule application. Roughly speaking, a strategy expression describes which computations (rewrite sequences), among all the possible ones, are appropriate for a given purpose. A rewriting strategy language consists of expressions built using rewrite rules and strategy operators. Various approaches have been followed, yielding slightly different strategy languages such as ELAN [1], [2], Stratego [3], TOM [4], or Maude [5]. All these provide flexible and expressive strategy languages where high-level operators are defined by combining low level primitives, and they all share the concern to provide abstract ways to express control of rule applications. For instance, for ELAN or Stratego, strategies such as bottom-up, top-down or leftmost-innermost are higher-order features that describe how rewrite rules should be applied.

In this paper we consider a simple strategy language, called ROC! , where the primitives are given by the rewrite rules applied at top, and the high-level strategy operators are the deterministic choice, the sequential composition and the repetition. In spite of its simplicity, this language is powerful enough and very efficient for metalanguage applications built using the patterns presented in [6]. A particular application of this kind which uses ROC! is CIRC [7]. The strategy language ROC! is parametric in the term rewriting system (TRS) over which it is defined; let ROC! (R) denote the particular language built over R, i.e., the primitives are given by the rewrite rules in R. The idea of implementing a rewriting strategy language is different from the classical ones. A pair $(R, \text{ROC!}\,(R))$ is "compiled" into a new TRS $\overline{R}$, such that $\overline{R}$-rewriting is equivalent with R-rewriting constrained by ROC! (R). The equivalence is showed by proving that $\overline{R}$ is sound and complete with respect to the semantics of $(R, \text{ROC!}\,(R))$.

This work is partially supported by the PNII grant IDEI 393.

The organization of the paper is as follows. Section II presents a situation where using a strategy language is suited. Section III defines the concept of *rewriting stack machine*. ROC! strategy language is introduced in Section IV. Section V presents the specification of a stack machine that evolves according to the strategy language semantics. The soundness and completeness of the machine are proved in Section VI. Appendix A includes an example showing how the stack machine evolves according to a term rewriting system computing the Collatz's function.

II. Motivating example

CIRC [7] is a Maude metalanguage application which implements the circular coinduction algorithm in order to prove *properties* (goals expressed as equations) for a given *equational behavioral specification*. CIRC is built using the patterns described in [6]. The procedure for proving a property uses a set of *rewrite rules* that can be applied according to some *proof strategy*. Three of the rules implemented for the CIRC prover are:

- $[comm]$ processes a goal expressing the commutativity of an operator by adding a special operator and some equations in their frozen form to the specification. The rule fails when the current property to be proved is not a commutativity goal.
- $[eqRed]$ removes an equational goal whenever it can be proved using ordinary equational reduction. The rule fails when the left hand side of an equation is different from the right hand side.
- $[ccstep]$ implements the circularity principle: an equation's frozen normalized form is added to the specification and its derivatives are added to the set of goals. This rule fails whenever it finds a visible goal which cannot be proved using ordinary equational reduction.

The following steps describe a possible strategy for the prover:

1) Try to apply $[comm]$. If it fails, leave the system state unchanged.
2) Check whether the first goal is proved using $[eqRed]$. If the check fails try to apply $[ccstep]$.
3) Repeat step 2 for as many times as possible.

The first problem is how to specify this kind of strategies. There are many ways to define rewriting strategies [4], [2], [8],

[1], [3]. A previous version of CIRC uses regular strategies [9]. However, the union, concatenation and iteration operators of the regular expressions language are not proper for specifying a behavior like "apply a rule for as many times as possible" or "apply a rule only if some other rule fails".

The solution is to use strategy operators appropriate for specifying these kinds of behavior. We prefer to call the elements of this strategy language *actions*. A basic action is a rewrite rule. Generally, actions are combined by means of several operators: $\triangleright$ (orelse), $\circ$ (composition) and ! (repeat) resulting in other actions. For the given example, the action described by the steps 1 - 3 is:

$$([comm] \triangleright [id]) \circ ([eqRed] \triangleright [ccstep])!$$

where

- $[id]$ leaves the list of goals unchanged. This rule never fails.
- $\triangleright$ has the semantics of an *orelse*-like operator. The system tries to apply $[comm]$ in the first place, and only if the action does not succeed should it apply $[id]$. The same strategy is followed for the rules $[eqRed]$ and $[ccstep]$.
- $\circ$ has the semantics of a sequential composition operator. For the example above, the full action is successfully applied if both $([comm] \triangleright [id])$ and $([eqRed] \triangleright [ccstep])!$ are evaluated exactly in this order with success.
- ! imposes that an action is applied for as many times as possible. In our case, the system stops following the strategy only when both $[eqRed]$ and $[ccstep]$ fail.

It is obvious that once the set of rules is chosen and the semantics of the strategy operators is provided, complex proof strategies can easily be specified in the same manner previously described.

The second problem is how to implement a system able to execute a strategy specified using the new language. A solution like the one described in [8] is not suitable because applications like CIRC are metalanguage applications and the tool described in [8] is used at object level.

In this paper we propose a solution which uses a rewriting stack machine to supply an executable operational semantics of the language.

III. Rewriting stack machines

In this section we introduce a new way to implement strategy languages. The general idea is as follows: given a TRS R and a language L of strategies over the rules from R, a new TRS $\overline{R}$ is defined. The purpose of $\overline{R}$ is to apply the rewrite rules R according to the semantics of the strategy language L. Since $\overline{R}$ is defined using a stack, it is called a *rewriting stack machine*.

We assume that the stack data type is specified by the sorts `Stack` for stacks and `Elt` for stack elements, a constant `empty` for the stack with zero elements and an operation `_;_ : Elt Stack -> Stack`, which is associative and has the identity `empty`. The stack behavior is a list of transitions $s \xrightarrow{*} s'$, each transition being accomplished using the `push` and `pop` basic operators. The operations `push` and `pop` are implemented by rewrite rules: `push(E) : S -> E S` and `pop : E S -> S`. These operations will be embedded in the definition of the rewriting stack machine.

Let R be a TRS. Consider a sort `State` for the system states, so that if $r : u \to v$ `if` $c \in R$ then u and v are of sort `State`. Also let L be a strategy language like ROC! whose elements are called *actions*. We follow the line from [10] and we specify the semantics of an action a by the relation $(t_{0..n}, r_{1..n}) \models a$, meaning that the rewriting $t_0 \xrightarrow{r_1} t_1 \xrightarrow{r_2} \ldots \xrightarrow{r_n} t_n$ is accomplished according to action a.

For a given term t_0 and an action a, the property $(\nexists\, t_{1..j}, r_{1..j})\ (j \geq 1) \wedge (t_{0..j}, r_{1..j}) \models a$ will be abbreviated under the form $t_0\!\uparrow\, \not\models a$. In plain English, this means that there no rewriting sequence starting from t_0 can be applied according to action a.

A *rewriting stack machine* for (R, L) is a TRS $\overline{R}$ with rewriting rules of the form $\lambda : (t, s) \to (t', s')$ `if` c, where t, t' are of sort `State` and s, s' are of sort `Stack`.

For each action $a \in L$, there are defined an initial term $\texttt{init}(a)$ of sort $\texttt{Stack}^{\texttt{init}}$ and a predicate which is used to identify stacks representing final states (of sort $\texttt{Stack}^{\texttt{final}}$). A pair (t, s) with s of sort $\texttt{Stack}^{\texttt{final}}$ cannot be rewritten according to $\overline{R}$. Both $\texttt{Stack}^{\texttt{init}}$ and $\texttt{Stack}^{\texttt{final}}$ are subsorts of `Stack`.

The relationship between (R, L) and $\overline{R}$ is expressed by the following properties:

1) *soundness*: if $(t_0,\ \texttt{init}(a)) \to \ldots \to (t_n, s)$ with s of sort $\texttt{Stack}^{\texttt{final}}$ then $(t_{0..n}, r_{1..n}) \models a$
2) *completeness*: if $(t_{0..n}, r_{1..n}) \models a$ then there is a stack $s' : \texttt{Stack}^{\texttt{final}}$ so that $(t_0,\ \texttt{init}(a)) \xrightarrow{*} (t_n, s')$

IV. The language ROC!

We present the formal description of the strategy language used in the current implementation of CIRC, called ROC!. The basic actions are Rewriting rules and the strategy operators are Orelse, Composition and !.

The grammar giving the syntax of the ROC! language is expressed as follows:

$$act ::= r \mid act \triangleright act \mid act \circ act \mid act\,!$$

where r is the label of a rule from the initial TRS R.

The semantics of ROC! is given by the next definitions:

1) $sem_r : (t_{0..n}, r_{1..n}) \models r \overset{def}{\Leftrightarrow}$
   $n = 1 \wedge r_1 = r \wedge t_0 \xrightarrow{r_1} t_1$
2) $sem_\triangleright : (t_{0..n}, r_{1..n}) \models a_1 \triangleright a_2 \overset{def}{\Leftrightarrow}$
   $(t_{0..n}, r_{1..n}) \models a_1 \vee$
   $(t_0\!\uparrow\, \not\models a_1 \wedge (t_{0..n}, r_{1..n}) \models a_2)$
3) $sem_\circ : (t_{0..i..n}, r_{1..i..n}) \models a_1 \circ a_2 \overset{def}{\Leftrightarrow}$
   $(t_{0..i}, r_{1..i}) \models a_1 \wedge$
   $(t_{i..n}, r_{i+1..n}) \models a_2$
4) $sem_!^{ind} : (t_{0..i..n}, r_{1..i..n}) \models a! \overset{def}{\Leftrightarrow}$
   $(t_{0..i}, r_{1..i}) \models a \wedge$
   $(t_{i..n}, r_{i+1..n}) \models a!$
5) $sem_!^{base} : (t, \phi) \models a! \overset{def}{\Leftrightarrow} t\!\uparrow\, \not\models a$

where ϕ from the last definition denotes an empty list.

The following propositions are used for proving properties in Section VI.

Proposition 1: $t_0\!\uparrow\, \not\models a$ *iff* $t_0\!\uparrow\, \not\models a!$

Proof:

We use the proof by contradiction method for both implications:

1) "$\Rightarrow$" Assume that $t_0\!\uparrow\, \not\models a$. If $(t_{0..n}, r_{1..n}) \models a!$ then there is an i so that $(1 \leq i \leq n)$ and $(t_{0..i}, r_{1..i}) \models a$ according to $sem_!^{ind}$, which contradicts the hypothesis.
2) "$\Leftarrow$" Assume that $t_0 \uparrow\, \not\models a!$. If $(\exists i \geq 1)$ so that $(t_{0..i}, r_{1..i}) \models a$, then there must be an $n \geq i$ so that $(t_{i..n}, r_{i+1..n}) \models a!$ holds. Hence the statement $(\exists n \geq 1)(t_{0..n}, r_{1..n}) \models a!$ holds according to $sem_!^{ind}$, which contradicts the hypothesis.

■

Proposition 2: $(t_{0..n}, r_{1..n}) \models a!$ *iff* $(t_{0..n}, r_{1..n}) \models a!!$

Proof:

If $n = 0$ the proof is trivial:

$(t_0, \phi) \models a!$ *iff* $(t_0\!\uparrow\, \not\models a)$ *iff* $(t_0\!\uparrow\, \not\models a!)$ *iff* $(t_0, \phi) \models a!!$

If $n \geq 0$:

1) "$\Rightarrow$" Assume that $(t_{0..n}, r_{1..n}) \models a!$
   It follows from the hypothesis that $t_n\!\uparrow\, \not\models a$. *Proposition 1* implies that $t_n\!\uparrow\, \not\models a!$, hence $(t_n, \phi) \models a!!$. It follows from the definition $sem_!^{ind}$ that $(t_{0..n}, r_{1..n}) \models a!!$.
2) "$\Leftarrow$" Assume that $(t_{0..n}, r_{1..n}) \models a!!$
   There is an i so that $(1 \leq i \leq n)$ and $(t_{0..i}, r_{1..i}) \models a!$. It follows that $t_i\!\uparrow\, \not\models a$, which implies $t_i\!\uparrow\, \not\models a!$. Because $t_n\!\uparrow\, \not\models a!$ (from the hypothesis), we obtain $i = n$, *i.e.* $(t_{0..n}, r_{1..n}) \models a!$.

■

V. A STACK MACHINE FOR ROC!

The syntactical tree for an action can be specified by a list of equations. For example, the action exemplified in Section II, $a = ([comm] \triangleright [id]) \circ ([eqRed] \triangleright [ccstep])!$, is specified as follows:

$$\begin{aligned} a &= a_1 \circ a_2 \\ a_1 &= comm \triangleright id \\ a_2 &= a_3! \\ a_3 &= eqRed \triangleright ccstep \end{aligned}$$

Let *def* denote the operator defined as follows: given an action a, *def*(a) returns the right hand side of the equation which defines a. For example, *def*$(a) = a_1 \circ a_2$, *def*$(a_1) = [comm] \circ [id]$, and so on.

We use a stack whose elements are quadruples of the form $\langle term, action, remainder, trail\rangle$, where:

- $term$ is either the special term $null$ or the term of the system before starting the execution of the action;
- $action$ is the label of the action being processed;
- $remainder$ is the part of the action that remains to be processed or ε when there is nothing left to be processed;
- $trail$ is the list of rules that have successfully been applied so far from the current action (the empty list is represented by the symbol ϕ).

A stack is of sort $\mathtt{Stack}^{\mathtt{init}}$ if it has only one element which has the form $\mathtt{init}(a)$=$\langle null, a, \textit{def}(a), \phi\rangle$ and of sort $\mathtt{Stack}^{\mathtt{final}}$ if it has only one element which can be represented as $\langle null, a, \varepsilon, trl\rangle$. From a final stack we are able to conclude if the action a has been successfully executed based on the value trl. As trl is the list of rules that have been successfully applied according to a, if trl is empty (ϕ) then the action has not been executed, and if it is not empty then the action has been successfully executed.

The behavior of the rewriting stack machine for ROC! is given by two classes of rules. The first class contains rules compiled from the initial TRS R. Given a rule $r : u \to v$ $\mathtt{if}$ $c \in R$, we denote a condition c^{ko} which is semantically equivalent to the negation of c. For the uniformity of the notation, c will be referred to as c^{ok}. The next two rules belong to $\overline{R}$:

1) $\overline{r^{ok}} : (u, [\langle\tau, a, r, \phi\rangle; \mathsf{ST}]) \to$
   $(v, [\langle\tau, a, \varepsilon, r\rangle; \mathsf{ST}])$ $\mathtt{if}$ c^{ok}
2) $\overline{r^{ko}} : (u, [\langle\tau, a, r, \phi\rangle; \mathsf{ST}]) \to$
   $(v, [\langle\tau, a, \varepsilon, \phi\rangle; \mathsf{ST}])$ $\mathtt{if}$ c^{ko}

The second class of rules from $\overline{R}$ give the behavior of the stack machine according to the action currently being executed. For the uniformity of the notation, the rules from this class will be overlined:

1) the start of an *orelse* action
   $\overline{rl_{\triangleright}^{start}}$:
   $(t, [\langle\tau, a, a_1 \triangleright a_2, trl\rangle; \mathsf{ST}]) \to$
   $(t, [\langle\tau, a_1, \textit{def}(a_1), \phi\rangle; \langle\tau, a, a_1 \triangleright a_2, trl\rangle; \mathsf{ST}])$
2) success for the first choice of an *orelse* action
   $\overline{rl_{\triangleright}^{ok}}$:
   $(t, [\langle\tau, a_1, \varepsilon, trl'\rangle; \langle\tau, a, a_1 \triangleright a_2, trl\rangle; \mathsf{ST}]) \to$
   $(t, [\langle\tau, a, \varepsilon, trl\, trl'\rangle; \mathsf{ST}])$
3) failure for the first choice starts the second choice
   $\overline{rl_{\triangleright}^{ko}}$:
   $(t, [\langle\tau, a_1, \varepsilon, \phi\rangle; \langle\tau, a, a_1 \triangleright a_2, trl\rangle; \mathsf{ST}]) \to$
   $(t, [\langle\tau, a_2, \textit{def}(a_2), \phi\rangle; \langle\tau, a, \varepsilon, trl\rangle; \mathsf{ST}])$
4) the end of the second choice (success or failure)
   $\overline{rl_{\triangleright}^{merge}}$:
   $(t, [\langle\tau, a', \varepsilon, trl'\rangle; \langle\tau, a, \varepsilon, trl\rangle; \mathsf{ST}]) \to$
   $(t, [\langle\tau, a, \varepsilon, trl\, trl'\rangle; \mathsf{ST}])$
5) the start of a *sequential composition* action
   $\overline{rl_{\circ}^{start}}$:
   $(t, [\langle\tau, a, a_1 \circ a_2, trl\rangle; \mathsf{ST}]) \to$
   $(t, [\langle t, a_1, \textit{def}(a_1), \phi\rangle; \langle\tau, a, a_1 \circ a_2, trl\rangle; \mathsf{ST}])$
6) success for the first part of a *sequential composition*
   $\overline{rl_{\circ}^{ok_1}}$:
   $(t', [\langle\tau', a_1, \varepsilon, trl'\rangle; \langle\tau, a, a_1 \circ a_2, trl\rangle; \mathsf{ST}]) \to$
   $(t', [\langle\tau', a_2, \textit{def}(a_2), \phi\rangle; \langle\tau, a, a_1 \circ a_2, trl\, trl'\rangle; \mathsf{ST}])$
7) failure for the first part of a *sequential composition*
   $\overline{rl_{\circ}^{ko_1}}$:
   $(t', [\langle\tau', a_1, \varepsilon, \phi\rangle; \langle\tau, a, a_1 \circ a_2, trl\rangle; \mathsf{ST}]) \to$
   $(\tau', [\langle\tau, a, \varepsilon, trl\rangle; \mathsf{ST}])$
8) success for the second part of a *sequential composition*
   $\overline{rl_{\circ}^{ok_2}}$:

$(t', [\langle \tau', a_2, \varepsilon, trl' \rangle;\ \langle \tau, a, a_1 \circ a_2, trl \rangle; \mathsf{ST}]) \rightarrow$
$(t', [\tau, a, \varepsilon, trl\, trl' \rangle; \mathsf{ST}]); j \geq 1$

9) failure for the second part of a *sequential composition* $\overline{rl_{\circ}^{ko_2}}$:
$(t', [\langle \tau', a_2, \varepsilon, \phi \rangle; \langle \tau, a, a_1 \circ a_2, trl \rangle; \mathsf{ST}]) \rightarrow$
$(\tau', [\langle \tau, a, \varepsilon, \phi \rangle; \mathsf{ST}])$

10) the start of a *repeat* action $\overline{rl_{!}^{start}}$:
$(t, [\langle \tau, a, a_1!, trl \rangle; \mathsf{ST}]) \rightarrow$
$(t, [\langle \tau, a_1, def(a_1), \phi \rangle; \langle \tau, a, a_1!, trl \rangle; \mathsf{ST}])$

11) success for the *repeat* action $\overline{rl_{!}^{ok}}$:
$(t, [\langle \tau, a_1, \varepsilon, trl' \rangle; \langle \tau, a, a_1!, trl \rangle; \mathsf{ST}]) \rightarrow$
$(t, [\langle \tau, a, a_1!, trl\, trl' \rangle; \mathsf{ST}])$

12) failure for the *repeat* action $\overline{rl_{!}^{ko}}$:
$(t, [\langle \tau, a_1, \varepsilon, \phi \rangle; \langle \tau, a, a_1!, trl \rangle; \mathsf{ST}]) \rightarrow$
$(t, [\langle \tau, a, \varepsilon, trl \rangle; \mathsf{ST}])$

VI. Soundness and Completeness

Generally, a system is *sound* if when proving that something is true, it really is true and is *complete* if when something is true, the system is capable of proving it.

For the ROC! stack machine, the soundness and completeness properties are expressed by the following equivalence:

$$\begin{array}{ll}(\forall n \geq 1) & (t_0, [\langle \tau, a, def(a), \phi \rangle; \mathsf{ST}]) \rightarrow \ldots \rightarrow \\ & (t_n, [\langle \tau, a, \varepsilon, r_{1..n} \rangle; \mathsf{ST}]) \\ & \text{iff} \\ & (t_{0..n}, r_{1..n}) \models a.\end{array}$$

A. Soundness

In order to prove that our stack machine system is sound, we have to prove that the following theorem holds:

Theorem 1: Let a be a ROC! action.

1) If

$$(t_0, [\langle \tau, a, def(a), \phi \rangle; \mathsf{ST}]) \rightarrow \ldots \rightarrow (t_n, [\langle \tau, a, \varepsilon, r_{1..n} \rangle; \mathsf{ST}])$$

for some $n \geq 1$, then $(t_{0..n}, r_{1..n}) \models a$.

2) If

$$(t, [\langle \tau, a, def(a), \phi \rangle; \mathsf{ST}]) \xrightarrow{*} (t', [\langle \tau, a, \varepsilon, \phi \rangle; \mathsf{ST}])$$

then $t = t'$ and $t\!\uparrow\, \not\models a$.

Proof: Let $P_1(def(a))$ denote the implication 1) and $P_2(def(a))$ denote the implication 2), where all the other variables are universally quantified. We prove $(\forall a) P_1(a) \wedge P_2(a)$ by structural induction on a, where a ranges over the expressions occurring in the description of the syntactical tree.

Base step. We assume $a = r$.

$P_1(r)$: The only rule that can be applied for the initial configuration is $\overline{r^{ok}}$. The configuration that follows after $(t_0, [\langle \tau, r, r, \phi \rangle; \mathsf{ST}])$ is $(t_1, [\langle \tau, r, \varepsilon, r \rangle; \mathsf{ST}])$. The latter is the final configuration ($n = 1$). Therefore $t_0 \xrightarrow{r} t_1$, which means that $(t_0 t_1, r) \models r$. Basically we have shown that $P_1(r)$ holds.

$P_2(r)$: In order to satisfy the hypothesis, the only possible transition is:

$$(t, [\langle \tau, r, r, \phi \rangle; \mathsf{ST}]) \xrightarrow{\overline{r^{ko}}} (t, [\langle \tau, r, \varepsilon, \phi \rangle; \mathsf{ST}])$$

It is obvious that $t' = t$ and that we cannot find a transition $t \xrightarrow{r} t'$. According to sem_r, $t\!\uparrow\, \not\models r$ holds, hence $P_2(r)$ holds.

Induction step. We consider only the case $a = a_1 \triangleright a_2$; the other cases are proved in a similar way. We prove that $P_1(a_1 \triangleright a_2)$ and $P_2(a_1 \triangleright a_2)$ hold, assuming that $P_1(a_1)$, $P_1(a_2)$, $P_2(a_1)$ and $P_2(a_2)$ hold. Since the syntactical tree includes the equations $a_i = def(a_i)$, $P_i(def(a_j))$ also holds for $i, j \in \{1, 2\}$.

$P_1(a_1 \triangleright a_2)$: We have to prove that if $(t_0, [\langle \tau, a, a_1 \triangleright a_2, \phi \rangle;\ \mathsf{ST}]) \rightarrow \ldots \rightarrow (t_n, [\langle \tau, a, \varepsilon, r_{1..n} \rangle; \mathsf{ST}])$, then $(t_{0..n}, r_{1..n}) \models a$ for some $n \geq 1$.

From the initial configuration, the first transition in the stack evolution is:

$$(t_0, [\langle \tau, a, a_1 \triangleright a_2, \phi \rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$$
$$(t_0, [\langle \tau, a_1, def(a_1), \phi \rangle; \langle \tau, a, a_1 \triangleright a_2, \phi \rangle; \mathsf{ST}])$$

The stack evolution for $a_1 \triangleright a_2$ is:

$$(t_0, [\langle \tau, a_1, def(a_1), \phi \rangle; \langle \tau, a, a_1 \triangleright a_2, \phi \rangle; \mathsf{ST}]) \xrightarrow{*}$$
$$(t', [\langle \tau, a_1, \varepsilon, trl' \rangle; \langle \tau, a, a_1 \triangleright a_2, \phi \rangle; \mathsf{ST}])$$

Two situations are identified:

1) $trl' = \phi$
In this case we apply $P_2(def(a_1))$ and obtain $t' = t_0$ and $t_0\!\uparrow\, \not\models a_1$. Given the current stack configuration, the only possible transition is the one corresponding to $\overline{rl_{\triangleright}^{ko}}$:

$$(t_0, [\langle \tau, a_1, \varepsilon, \phi \rangle; \langle \tau, a, a_1 \triangleright a_2, \phi \rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{ko}}}$$
$$(t_0, [\langle \tau, a_2, def(a_2), \phi \rangle; \langle \tau, a, \varepsilon, \phi \rangle; \mathsf{ST}])$$

Then the rewriting continues as follows:

$$(t_0, [\langle \tau, a_2, def(a_2), \phi \rangle; \langle \tau, a, \varepsilon, \phi \rangle; \mathsf{ST}]) \xrightarrow{*}$$
$$(t'', [\langle \tau, a_2, \varepsilon, trl'' \rangle; \langle \tau, a, \varepsilon, \phi \rangle; \mathsf{ST}])$$

The only possible next transition is:

$$(t'', [\langle \tau, a_1, \varepsilon, trl'' \rangle; \langle \tau, a, \varepsilon, \phi \rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{merge}}}$$
$$(t'', [\langle \tau, a, \varepsilon, trl'' \rangle; \mathsf{ST}])$$

From the hypothesis of $P_1(a_1 \triangleright a_2)$ it follows that $t'' = t_n$ and $trl'' = r_{1..n}$. Combining the previous two rewritings, we obtain:

$$(t_0, [\langle \tau, a_2, def(a_2), \phi \rangle; \mathsf{ST}']) \xrightarrow{*}$$
$$(t_n, [\langle \tau, a_2, \varepsilon, r_{1..n} \rangle; \mathsf{ST}'])$$

According to $P_1(def(a_2))$, we deduce that $(t_{0..n}, r_{1..n}) \models a_2$. This property, along with the true statement $t_0\!\uparrow\, \not\models a_1$ leads to $(t_{0..n}, r_{1..n}) \models a_1 \triangleright a_2$.

2) $trl' = r_{1..i}$
The only transition that can be applied is the one corresponding to $\overline{rl_{\triangleright}^{merge}}$:

$(t', [\langle \tau, a_1, \varepsilon, r_{1..i}\rangle; \langle \tau, a, \varepsilon, \phi\rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{merge}}}$
$(t', [\langle \tau, a, \varepsilon, r_{1..i}\rangle; \mathsf{ST}])$

From the hypothesis of $P_1(a_1 \triangleright a_2)$ it follows that $t' = t_n$ and $i = n$. Now, the following rewriting is obtained:

$(t_0, [\langle \tau, a_1, \mathit{def}(a_1), \phi\rangle; \mathsf{ST}']) \overset{*}{\to}$
$(t_n, [\langle \tau, a_1, \varepsilon, r_{1..n}\rangle; \mathsf{ST}'])$

It follows that $(t_{0..n}, r_{1..n}) \models a_1$ holds by $P_1(def(a_1))$, which implies $(t_{0..n}, r_{1..n}) \models a_1 \triangleright a_2$.

The proof of $P_2(a_1 \triangleright a_2)$ is realized in a similar way. ■

B. Completeness

In order to prove the completeness of the stack machine system we have to prove that the following theorem holds:

Theorem 2: Let a be a ROC! action.

1) If $(t_{0..n}, r_{1..n}) \models a$ then

$$(t_0, [\langle \tau, a, \mathit{def}(a), \phi\rangle; \mathsf{ST}]) \to \ldots \to (t_n, [\langle \tau, a, \varepsilon, r_{1..n}\rangle; \mathsf{ST}]).$$

2) If $t\!\uparrow\, \not\models a$ then

$$(t, [\langle \tau, a, \mathit{def}(a), \phi\rangle; \mathsf{ST}]) \overset{*}{\to} (t, [\langle \tau, a, \varepsilon, \phi\rangle; \mathsf{ST}]).$$

Proof: Let $Q_1(def(a))$ denote the implication 1) and $Q_2(def(a))$ denote the implication 2), where all the other variables are universally quantified. We prove $(\forall a)Q_1(a) \wedge Q_2(a)$ by structural induction on a, where a ranges over the expressions occurring in the description of the syntactical tree.

Base step. We assume $a = r$.

$Q_1(r)$: According to sem_r it follows that $n = 1$ and $r_1 = r$. Since $(t_0 t_1) \models r$ means that t_1 is obtained from t_0 by applying the rule r, the only possible transition is:

$$(t_0, [\langle \tau, r, r, \phi\rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_r^{ok}}} (t_1, [\langle \tau, r, \varepsilon, r\rangle; \mathsf{ST}])$$

Consequently we obtain that $Q_1(r)$ holds.

$Q_2(r)$: If $t\uparrow\, \not\models r$ then we cannot identify t' such that $(tt', r) \models r$, therefore a transition $t \xrightarrow{r} t'$ is never possible. The only feasible transition is:

$$(t, [\langle \tau, r, r, \phi\rangle; \mathsf{ST}]) \xrightarrow{\overline{r^{ko}}} (t, [\langle \tau, r, \varepsilon, \phi\rangle; \mathsf{ST}])$$

Therefore $Q_2(r)$ holds.

Induction step. We consider again only the case $a = a_1 \triangleright a_2$; the other cases are proved in a similar way. We prove that $Q_1(a_1 \triangleright a_2)$ and $Q_2(a_1 \triangleright a_2)$ hold, assuming that $Q_1(a_1)$, $Q_1(a_2)$, $Q_2(a_1)$ and $Q_2(a_2)$ hold. We recall that $Q_i(a_j)$ implies $Q_i(def(a_j))$ for $i, j \in \{1, 2\}$.

$Q_1(a_1 \triangleright a_2)$: We have to prove that if $(t_{0..n}, r_{1..n}) \models a$ then

$$(t_0, [\langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}]) \to \ldots \to (t_n, [\langle \tau, a, \varepsilon, r_{1..n}\rangle; \mathsf{ST}]).$$

The first transition is:

$(t_0, [\langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$
$(t_0, [\langle \tau, a_1, \mathit{def}(a_1), \phi\rangle; \langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}])$

By the semantics of $\triangleright$, two situations are identified:

1) $(t_{0..n}, r_{1..n}) \models a_1$
$Q_1(def(a_1))$ holds, so the rewriting that follows is:

$(t_0, [\langle \tau, a_1, \mathit{def}(a_1), \phi\rangle; \langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}]) \overset{*}{\to}$
$(t_n, [\langle \tau, a_1, \varepsilon, r_{1..n}\rangle; \langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}])$

The only possible transition is the one corresponding to $\overline{rl_{\triangleright}^{ok}}$:

$(t_n, [\langle \tau, a_1, \varepsilon, r_{1..n}\rangle; \langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{ok}}}$
$(t_n, \langle \tau, a, \varepsilon, r_{1..n}\rangle; \mathsf{ST}])$

2) $t_0\!\uparrow\, \not\models a_1$ and $(t_{0..n}, r_{1..n}) \models a_2$
By $Q_2(def(a_1))$, $(t_0, [\langle a_1, \mathit{def}(a_1), \phi\rangle; \mathsf{ST}'])$ evolves to $(t_0, [\langle a_1, \varepsilon, \phi\rangle; \mathsf{ST}'])$. The rewriting continues as follows:

$(t_0, [\langle \tau, a_1, \mathit{def}(a_1), \phi\rangle; \langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}]) \overset{*}{\to}$

$(t_0, [\langle \tau, a_1, \varepsilon, \phi\rangle; \langle \tau, a, a_1 \triangleright a_2, \phi\rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{ko}}}$
$(t_0, [\langle \tau, a_2, \mathit{def}(a_2), \phi\rangle; \langle \tau, a, \varepsilon, \phi\rangle; \mathsf{ST}])$

$Q_1(\mathit{def}(a_2))$ holds, therefore:

$(t_0, [\langle \tau, a_2, \mathit{def}(a_2), \phi\rangle; \langle \tau, a, \varepsilon, \phi\rangle; \mathsf{ST}]) \overset{*}{\to}$
$(t_n, [\langle \tau, a_2, \varepsilon, r_{1..n}\rangle; \langle \tau, a, \varepsilon, \phi\rangle; \mathsf{ST}])$

Now, the only possible transition is:

$(t_n, [\langle \tau, a_2, \varepsilon, r_{1..n}\rangle; \langle \tau, a, \varepsilon, \phi\rangle; \mathsf{ST}]) \xrightarrow{\overline{rl_{\triangleright}^{merge}}}$
$(t_n, [\langle \tau, a, \varepsilon, r_{1..n}\rangle; \mathsf{ST}])$

Combining all the rewritings above, we obtain $Q_1(a_1 \triangleright a_2)$. In the same manner it can be shown that $Q_2(a_1 \triangleright a_2)$ holds. ■

VII. Conclusion

This paper introduces a strategy language for specifying three kinds of behavior for a TRS:

- perform an action only if some other action fails
- perform two actions sequentially
- perform an action for as many times as possible

where an atomic action is the application of a rule at the top. The language is implemented using a stack machine system which is proved to be both sound and complete.

We have successfully managed to specify proof strategies in this new manner for CIRC after adding the stack machine implementation to the prover source code.

It is worth mentioning that if the sequential composition operator is not required, the rewriting stack machine is simplified. In this case, the structure of the stack elements is simplified from a quadruple to a triple, by eliminating the $term$ component. This component is no longer required because during the evolution of the system there is no need to turn back to a previous term (this is needed only when trying to execute an action of the form $a_1 \circ a_2$). An example emphasizing this situation is provided in Appendix A.

References

[1] C. Kirchner, H. Kirchner, and M. Vittek, "Designing Constraint Logic Programming Languages using Computational Systems," in *Principles and Practice of Constraint Programming. The Newport Papers.*, P. Van Hentenryck and V. Saraswat, Eds. MIT Press, 1995, ch. 8, pp. 131–158.

[2] P. Borovanský, C. Kirchner, H. Kirchner, and C. Ringeissen, "Rewriting with Strategies in ELAN: A Functional Semantics," *Int. J. Found. Comput. Sci.*, vol. 12, no. 1, pp. 69–95, 2001.

[3] E. Visser, "Stratego: A Language for Program Transformation based on Rewriting Strategies. System Description of Stratego 0.5," in *RTA*, ser. Lecture Notes in Computer Science, A. Middeldorp, Ed., vol. 2051. Springer-Verlag, 2001, pp. 357–361.

[4] E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles, "Tom: Piggybacking Rewriting on Java," in *RTA*, ser. Lecture Notes in Computer Science, F. Baader, Ed., vol. 4533. Springer-Verlag, 2007, pp. 36–47.

[5] N. Martí-Oliet, J. Meseguer, and A. Verdejo, "Towards a Strategy Language for Maude," *Electronic Notes in Theoretical Computer Science*, vol. 117, pp. 417–441, 2005.

[6] E. Goriac, G. Caltais, D. Lucanu, O. Andrei, and G. Grigoraş, "Patterns for Maude Metalanguage Applications," in *Proceedings of WRLA*, 2008.

[7] D. Lucanu and G. Roşu, "Circ: A Circular Coinductive Prover," in *2nd Conference on Algebra and Coalgebra in Computer Science (CALCO 2007)*, ser. Lecture Notes in Computer Science, T. Mossakowski and et al., Eds., vol. 4624. Springer, 2007, pp. 372–378.

[8] S. Eker, N. Martí-Oliet, J. Meseguer, and A. Verdejo, "Deduction, strategies, and rewriting," *Electron. Notes Theor. Comput. Sci.*, vol. 174, no. 11, pp. 3–25, 2007.

[9] D. Lucanu, G. Roşu, and G. Grigoraş, "Regular strategies as proof tactics for circ," *Electron. Notes Theor. Comput. Sci.*, vol. 204, pp. 83–98, 2008.

[10] J. Meseguer, "The temporal logic of rewriting: A gentle introduction," in *Concurrency, Graphs and Models*, 2008, pp. 354–382.

APPENDIX

We consider a term rewriting system inspired from Collatz's function:

$r_1 : n \to n/2 \quad \texttt{if} \ n \equiv 0 \ (\mathit{mod}\ 2) \ \wedge \ n \neq 1$

$r_2 : n \to 3n+1 \ \ \texttt{if} \ n \neq 1$

Let us analyze how the stack machine evolves for the term $t_0 = 5$ and the strategy $(r_1 \triangleright r_2)!$. The actions specifying the strategy are: $a = b!$ and $b = r_1 \triangleright r_2$.

$$\left(5, \begin{array}{|c|c|c|}\hline a & b! & \phi \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{start}}}$$

$$\left(5, \begin{array}{|c|c|c|}\hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & \phi \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$$

$$\left(5, \begin{array}{|c|c|c|}\hline r_1 & r_1 & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & \phi \\ \hline\end{array}\right) \xrightarrow{\overline{r_1^{ko}}}$$

$$\left(5, \begin{array}{|c|c|c|}\hline r_1 & \varepsilon & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & \phi \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{ko}}}$$

$$\left(5, \begin{array}{|c|c|c|}\hline r_2 & r_2 & \phi \\ \hline b & \varepsilon & \phi \\ \hline a & b! & \phi \\ \hline\end{array}\right) \xrightarrow{\overline{r_2^{ok}}}$$

$$\left(16, \begin{array}{|c|c|c|}\hline r_2 & \varepsilon & r_2 \\ \hline b & \varepsilon & \phi \\ \hline a & b! & \phi \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{merge}}}$$

$$\left(16, \begin{array}{|c|c|c|}\hline b & \varepsilon & r_2 \\ \hline a & b! & \phi \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{ok}}}$$

$$\left(16, \begin{array}{|c|c|c|}\hline a & b! & r_2 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{start}}}$$

$$\left(16, \begin{array}{|c|c|c|}\hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$$

$$\left(16, \begin{array}{|c|c|c|}\hline r_1 & r_1 & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 \\ \hline\end{array}\right) \xrightarrow{\overline{r_1^{ok}}}$$

$$\left(8, \begin{array}{|c|c|c|}\hline r_1 & \varepsilon & r_1 \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{ok}}}$$

$$\left(8, \begin{array}{|c|c|c|}\hline b & \varepsilon & r_1 \\ \hline a & b! & r_2 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{ok}}}$$

$$\left(8, \begin{array}{|c|c|c|}\hline a & b! & r_2 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{start}}}$$

$$\left(8, \begin{array}{|c|c|c|}\hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$$

$$\left(8, \begin{array}{|c|c|c|}\hline r_1 & r_1 & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{r_1^{ok}}}$$

$$\left(4, \begin{array}{|c|c|c|}\hline r_1 & \varepsilon & r_1 \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{ok}}}$$

$$\left(4, \begin{array}{|c|c|c|}\hline b & \varepsilon & r_1 \\ \hline a & b! & r_2 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{ok}}}$$

$$\left(4, \begin{array}{|c|c|c|}\hline a & b! & r_2 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{start}}}$$

$$\left(4, \begin{array}{|c|c|c|}\hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$$

$$\left(4, \begin{array}{|c|c|c|}\hline r_1 & r_1 & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{r_1^{ok}}}$$

$$\left(2, \begin{array}{|c|c|c|}\hline r_1 & \varepsilon & r_1 \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{ok}}}$$

$$\left(2, \begin{array}{|c|c|c|}\hline b & \varepsilon & r_1 \\ \hline a & b! & r_2 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{ok}}}$$

$$\left(2, \begin{array}{|c|c|c|}\hline a & b! & r_2 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{start}}}$$

$$\left(2, \begin{array}{|c|c|c|}\hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$$

$$\left(2, \begin{array}{|c|c|c|}\hline r_1 & r_1 & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{r_1^{ok}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline r_1 & \varepsilon & r_1 \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{ok}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline b & \varepsilon & r_1 \\ \hline a & b! & r_2 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{ok}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline a & b! & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{start}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{start}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline r_1 & r_1 & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{r_1^{ko}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline r_1 & \varepsilon & \phi \\ \hline b & r_1 \triangleright r_2 & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{ko}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline r_2 & r_2 & \phi \\ \hline b & \varepsilon & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{r_2^{ko}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline r_2 & \varepsilon & \phi \\ \hline b & \varepsilon & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{\triangleright}^{merge}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline b & \varepsilon & \phi \\ \hline a & b! & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right) \xrightarrow{\overline{rl_{!}^{ko}}}$$

$$\left(1, \begin{array}{|c|c|c|}\hline a & \varepsilon & r_2 r_1 r_1 r_1 r_1 \\ \hline\end{array}\right)$$

According to the machine evolution, the following statement holds: $(5\ 16\ 8\ 4\ 2\ 1, r_2 r_1 r_1 r_1 r_1) \models (r_1 \triangleright r_2)!$