A faithful encoding of programmable strategies into term rewriting systems

Horatiu Cirstea, Sergueï Lenglet, Pierre-Etienne Moreau

A faithful encoding of programmable strategies into term rewriting systems

Horatiu Cirstea, Sergueï Lenglet, and Pierre-Etienne Moreau

Université de Lorraine – LORIA
Campus scientifique – BP 239 – 54506 Vandœuvre-lès-Nancy, France
{firstname.lastname@loria.fr}

Abstract

Rewriting is a formalism widely used in computer science and mathematical logic. When using rewriting as a programming or modeling paradigm, the rewrite rules describe the transformations one wants to operate and declarative rewriting strategies are used to control their application. The operational semantics of these strategies are generally accepted and approaches for analyzing the termination of specific strategies have been studied. We propose in this paper a generic encoding of classic control and traversal strategies used in rewrite based languages such as Maude, Stratego and Tom into a plain term rewriting system. The encoding is proven sound and complete and, as a direct consequence, established termination methods used for term rewriting systems can be applied to analyze the termination of strategy controlled term rewriting systems. The corresponding implementation in Tom generates term rewriting systems compatible with the syntax of termination tools such as AProVE and TTT2, tools which turned out to be very effective in (dis)proving the termination of the generated term rewriting systems. The approach can also be seen as a generic strategy compiler which can be integrated into languages providing pattern matching primitives; this has been experimented for Tom and performances comparable to the native Tom strategies have been observed.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases Programmable strategies, termination, term rewriting systems

Digital Object Identifier 10.4230/LIPIcs.RTA.2015.74

1 Introduction

Rewriting is a very powerful tool used in theoretical studies as well as for practical implementations. It is used, for example, in semantics in order to describe the meaning of programming languages [24], but also in automated reasoning when describing by inference rules a logic, a theorem prover or a constraint solver [20]. It is also used to compute in systems making the notion of rule an explicit and first class object, like Mathematica, Maude [8], or Tom [25, 4]. Rewrite rules, the core concept in rewriting, consist of a pattern that describes a schematic situation and the transformation that should be applied in that particular case. The pattern expresses a potentially infinite number of instances and the application of the rewrite rule is decided locally using a (matching) algorithm which only depends on the pattern and its subject. Rewrite rules are thus very convenient for describing schematically and locally the transformations one wants to operate.

In many situations, the application of a set of rewrite rules to a subject eventually leads to the same final result independently on the way the rules are applied, and in such cases we say that the rewrite rules are *confluent* and *terminating*. When using rewriting as a programming or modeling paradigm it is nevertheless common to consider term rewriting systems (TRS) that are non-confluent or non-terminating. In order to make the concept operational when

26th International Conference on Rewriting Techniques and Applications (RTA'15).
Editor: Maribel Fernández; pp. 74–88

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

such rules are employed we need an additional ingredient allowing one to specify which rules should be applied and where (in the subject). This fundamental mechanism allowing the control of rule application is a *rewriting strategy*.

Rule-based languages like ELAN [6], Maude, Stratego [30] or Tom have introduced the notion of programmable strategies to express rule application control in a declarative way. All these languages provide means to define term representations for the objects we want to transform as well as rewrite rules and strategies expressing the way the terms are transformed and offer thus, a clear separation between data structures, function logic and control. Similarly to plain TRS (*i.e.* TRS without strategy), it is interesting to guarantee that the strategy controlled TRS enjoy properties such as confluence and termination. Confluence holds as long as the rewrite rules are deterministic (*i.e.* the corresponding matching is unitary) and all strategy operators are deterministic (or a deterministic implementation is provided).

Termination is more delicate and the normalization under some specific strategy is usually guaranteed by imposing (sufficient) conditions on the corresponding set of rewrite rules. Such conditions have been proposed for the innermost [2, 13, 29, 17], outermost [9, 28, 17, 26], context-sensitive [14, 1, 17], or lazy [15] reduction strategies. Termination under programmable strategies has been studied for ELAN [11] and Stratego [21, 23]. In [11], the authors prove that a programmable strategy is terminating if the system formed with all the rewrite rules the strategy contains is terminating. This criterion is too coarse as it does not take into account how the strategy makes its arguments interact and consequently, the approach cannot be used to prove termination for many terminating strategies. In [21, 23], the termination of some traversal strategies (such as top-down, bottom-up, innermost) is proven, assuming the rewrite rules are measure decreasing, for a notion of measure that combines the depth, and the number of occurrences of a specific constructor in a term.

Contributions. In this paper we propose a more general approach consisting in translating programmable strategies into plain TRS. The interest of this encoding that we show sound and complete is twofold. First, termination analysis techniques [2, 19, 16] and corresponding tools like AProVE [12] and TTT2 [22] that have been successfully used for checking the termination of plain TRS can be used to verify termination in presence of rewriting strategies. Second, the translation can be seen as a generic strategy compiler and thus can be used as a portable implementation of strategies which could be easily integrated in any language providing rewrite rules (or at least pattern matching) primitives. The translation has been implemented in Tom and generates TRS which could be fed into TTT2/AProVE for termination analysis or executed efficiently by Tom.

The paper is organized as follows. The next section introduces the notions of rewriting system and rewriting strategy. Section 3 presents the translation of rewriting strategies into rewriting systems, and its properties are stated together with proof sketches in Section 4. In Section 5 we give some implementation details and present experimental results. We end with conclusions and further work.

2 Strategic rewriting

This section briefly recalls some basic notions of rewriting used in this paper; see [3, 27] for more details on first order terms and term rewriting systems, and [31, 5] for details on rewriting strategies and their implementation in rewrite based languages.

2.1 Term algebra and term rewriting systems

A *signature* Σ consists of an alphabet $\mathcal{F}$ of symbols together with a function ar which associates to any symbol f its *arity*. We write $\mathcal{F}^n$ for the subset of symbols of arity n, and $\mathcal{F}^+$ for the symbols of arity $n > 0$. Symbols in $\mathcal{F}^0$ are called *constants*. Given a countable set $\mathcal{X}$ of *variable* symbols, the set of *first-order terms* $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the smallest set containing $\mathcal{X}$ and such that $f(t_1, \ldots, t_n)$ is in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ whenever $f \in \mathcal{F}^n$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ for $i \in [1, n]$. We write $\mathcal{V}ar\,(t)$ for the set of variables occuring in $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If $\mathcal{V}ar\,(t)$ is empty, t is called a *ground* term; $\mathcal{T}_{\mathcal{F}}$ denotes the set of all ground terms. A *linear* term is a term where every variable occurs at most once. A *substitution* σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$ which is the identity except over a finite set of variables (its *domain*). A substitution extends as expected to an endomorphism of $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

A *position* of a term t is a finite sequence of positive integers describing the path from the root of t to the root of the sub-term at that position. We write ε for the empty sequence, which represents the root of t, $\mathcal{P}os(t)$ for the set of positions of t, and $t_{|\omega}$ for the sub-term of t at position ω. Finally, $t\,[s]_\omega$ is the term t with the sub-term at position ω replaced by s.

A *rewrite rule* (over Σ) is a pair $(l, r) \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$ (also denoted $l \twoheadrightarrow r$) such that $\mathcal{V}ar\,(r) \subseteq \mathcal{V}ar\,(l)$ and a TRS is a set of rewrite rules $\mathcal{R}$ inducing a *rewriting relation* over $\mathcal{T}_{\mathcal{F}}$, denoted by $\longrightarrow_{\mathcal{R}}$ and such that $t \longrightarrow_{\mathcal{R}} t'$ iff there exist $l \twoheadrightarrow r \in \mathcal{R}$, $\omega \in \mathcal{P}os(t)$, and a substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_\omega$. In this case, we say that l matches t and that σ is the solution of the corresponding matching problem. The reflexive and transitive closure of $\longrightarrow_{\mathcal{R}}$ is denoted by $\longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}}$. In what follows, we generally use the notation $\mathcal{R} \bullet t \longrightarrow t'$ to denote $t \longrightarrow\!\!\!\!\rightarrow_{\mathcal{R}} t'$. A TRS $\mathcal{R}$ is *terminating* if there exists no infinite rewriting sequence $t_1 \longrightarrow_{\mathcal{R}} t_2 \longrightarrow_{\mathcal{R}} \ldots \longrightarrow_{\mathcal{R}} t_n \longrightarrow_{\mathcal{R}} \ldots$

2.2 Rewriting strategies

Rewriting strategies allow one to specify how rules should be applied. We call the term to which the strategy is applied the subject. The application of a strategy to a subject may diverge, fail, or return a (unique) result.

Taking the same terminology as the one proposed by ELAN and Stratego, a rewrite rule is considered to be an elementary strategy, and a strategy is an expression built over a strategy language. We consider a strategy language over a signature Σ consisting of the main operators used in rewrite based languages like Tom and Stratego:

$$S \quad := \quad \mathit{Identity} \mid \mathit{Fail} \mid l \twoheadrightarrow r \mid S\,;S \mid S \leftarrow\!\!\!\!+ S \mid \mathit{One}(S) \mid \mathit{All}(S) \mid \mu X\,.\,S \mid X$$

with X any variable from the set $\mathcal{X}_{\mathcal{S}}$ of strategy variables and $l \twoheadrightarrow r$ any rewrite rule over Σ. In what follows, we use uppercase for strategy variables and lowercase for term variables.

Before formally defining the strategy semantics, we can already mention that the simplest strategies we can imagine are *Identity* and *Fail*. The *Identity* strategy can be applied to any term without changing it, and thus *Identity* never fails. Conversely, the strategy *Fail* always fails when applied to a term. As mentioned above, a rewrite rule is an elementary strategy which is applied to the root position of its subject. By combining elementary strategies, more complex strategies can be built. In particular, we can apply sequentially two strategies, make a choice between the application of two strategies, apply a strategy to one or to all the immediate sub-terms of the subject, and apply recursively a strategy.

The operational semantics presented in Figure 1 is defined *w.r.t.* a context of the form $X_1 : S_1 \ldots X_n : S_n$ which associates strategy expressions to strategy variables. Indeed, a reduction step is written $\Gamma \vdash S \circ t \Longrightarrow u$, where Γ is the context under which the current

Elementary strategies

$$\frac{}{\Gamma \vdash \mathit{Identity} \circ t \Longrightarrow t}\ (\mathbf{id}) \qquad \frac{}{\Gamma \vdash \mathit{Fail} \circ t \Longrightarrow \mathtt{Fail}}\ (\mathbf{fail})$$

$$\frac{\exists \sigma, \sigma(l) = t}{\Gamma \vdash l \to r \circ t \Longrightarrow \sigma(r)}\ (\mathbf{r_1}) \qquad \frac{\nexists \sigma, \sigma(l) = t}{\Gamma \vdash l \to r \circ t \Longrightarrow \mathtt{Fail}}\ (\mathbf{r_2})$$

Control combinators

$$\frac{\Gamma \vdash S_1 \circ t \Longrightarrow t'}{\Gamma \vdash (S_1 \nleftarrow S_2) \circ t \Longrightarrow t'}\ (\mathbf{choice_1}) \qquad \frac{\Gamma \vdash S_1 \circ t \Longrightarrow \mathtt{Fail} \qquad \Gamma \vdash S_2 \circ t \Longrightarrow u}{\Gamma \vdash (S_1 \nleftarrow S_2) \circ t \Longrightarrow u}\ (\mathbf{choice_2})$$

$$\frac{\Gamma \vdash S_1 \circ t \Longrightarrow t' \qquad \Gamma \vdash S_2 \circ t' \Longrightarrow u}{\Gamma \vdash (S_1 \,;\, S_2) \circ t \Longrightarrow u}\ (\mathbf{seq_1}) \qquad \frac{\Gamma \vdash S_1 \circ t \Longrightarrow \mathtt{Fail}}{\Gamma \vdash (S_1 \,;\, S_2) \circ t \Longrightarrow \mathtt{Fail}}\ (\mathbf{seq_2})$$

$$\frac{\Gamma; X : S \vdash S \circ t \Longrightarrow u}{\Gamma \vdash \mu X \,.\, S \circ t \Longrightarrow u}\ (\mathbf{mu}) \qquad \frac{\Gamma; X : S \vdash S \circ t \Longrightarrow u}{\Gamma; X : S \vdash X \circ t \Longrightarrow u}\ (\mathbf{muvar})$$

Traversal combinators

$$\frac{\exists i \in [1,n], \Gamma \vdash S \circ t_i \Longrightarrow t'_i \qquad \forall j \in [1, i-1], \Gamma \vdash S \circ t_j \Longrightarrow \mathtt{Fail}}{\Gamma \vdash \mathit{One}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow f(t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n)}\ (\mathbf{one_1})$$

$$\frac{\forall i \in [1,n], \Gamma \vdash S \circ t_i \Longrightarrow \mathtt{Fail}}{\Gamma \vdash \mathit{One}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow \mathtt{Fail}}\ (\mathbf{one_2})$$

$$\frac{\forall i \in [1,n], \Gamma \vdash S \circ t_i \Longrightarrow t'_i}{\Gamma \vdash \mathit{All}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow f(t'_1, \ldots, t'_n)}\ (\mathbf{all_1})$$

$$\frac{\exists i \in [1,n], \Gamma \vdash S \circ t_i \Longrightarrow \mathtt{Fail}}{\Gamma \vdash \mathit{All}(S) \circ f(t_1, \ldots, t_n) \Longrightarrow \mathtt{Fail}}\ (\mathbf{all_2})$$

Figure 1 Strategy semantics. The meta variable t denotes a term (which cannot be `Fail`), whereas the meta variable u denotes a result which can be either a well-formed term, or `Fail`.

strategy is applied, S is the strategy to apply, t is the subject, and u is the resulting term or `Fail` (the context may be omitted if empty). The variables in strategy expressions could be thus bound by the recursion operator or by a corresponding assignment in the context: we primarily use the context as an accumulator for the evaluation of recursion strategies, but it can also be used for the evaluation of strategies featuring free variables. As usual, we work modulo α-*conversion* and we adopt Barendregt's *"hygiene-convention"*, *i.e.* free- and bound-variables have different names.

We distinguish three kinds of operators in the strategy language:

- *elementary strategies* consisting of the *Identity* and *Fail* strategies, and rewrite rules which succeed or fail when applied at the root position of the subject.

- *control combinators* that compose strategies but are still applied at the root of the subject. The (left-)choice $S_1 \nleftarrow S_2$ tries to apply S_1 and considers S_2 only if S_1 fails. The sequential application $S_1 ; S_2$ succeeds if S_1 succeeds on the subject and S_2 succeeds on the subsequent term; it fails if one of the two strategy applications fails. The application of a strategy $\mu X \,.\, S$ (rule **mu**) evaluates S in a context where X is bound to S. If the strategy variable X is applied to a term (rule **muvar**), the strategy S is (re)evaluated, allowing thus recursion. This process can, of course, go on forever but eventually stops if at some point the evaluation of S does not involve X anymore.
- *traversal combinators* that modify the current application position. The operator $One(S)$ tries to apply S to a sub-term of the subject. We have chosen a deterministic semantics for *One* which looks for the left-most sub-term successfully transformed; the non-deterministic behavior can be easily obtained by removing the second condition in the premises of the inference rule $\mathbf{one_1}$. If S fails on all the sub-terms, then $One(S)$ also fails (rule $\mathbf{one_2}$). In contrast, $All(S)$ applies S to all the sub-terms of the subject (rule $\mathbf{all_1}$) and fails if S fails on one of them (rule $\mathbf{all_2}$). Note that $One(S)$ always fails when applied to a constant while $All(S)$ always succeeds in this case.

The combinators of the strategy language can be used to define more complex ones. For example, we can define a strategy named *Try*, parameterized by a strategy S, which tries to apply S and applies the identity if S fails: $Try(S) = S \nleftarrow Identity$. The $Repeat(S)$ strategy can be defined using recursion: $Repeat(S) = \mu X \,.\, Try(S ; X)$. In fact, most of the classic reduction strategies can also be defined using the generic traversal operators:

$$\begin{aligned} OnceBottomUp(S) &= \mu X \,.\, One(X) \nleftarrow S \\ BottomUp(S) &= \mu X \,.\, All(X) ; S \\ OnceTopDown(S) &= \mu X \,.\, S \nleftarrow One(X) \\ TopDown(S) &= \mu X \,.\, S ; All(X) \end{aligned}$$

The strategy *OnceBottomUp* (denoted *obu* in the following) tries to apply the strategy S once, starting from the leftmost-innermost leaves. *BottomUp* behaves almost like *obu* except that S is applied to all nodes, starting from the leaves. The strategy which applies S as many times as possible, starting from the leaves can be either defined naively as $Repeat(obu(S))$ or using a more efficient approach [31]: $Innermost(s) = \mu X \,.\, All(X) ; Try(S ; X)$. Given the rules $R_1, \ldots, R_n$, the strategy $R_1 \nleftarrow \cdots \nleftarrow R_n$ can be used to express an order on the rules.

▶ **Example 1.** Consider the rewrite rules $\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x$ and $\mathtt{+}(\mathtt{S}(x), y) \twoheadrightarrow \mathtt{S}(\mathtt{+}(x, y))$ on the signature Σ with $\mathcal{F}^0 = \{\mathtt{Z}\}$, $\mathcal{F}^1 = \{\mathtt{S}\}$, $\mathcal{F}^2 = \{\mathtt{+}, \mathtt{*}\}$.
$\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{Z} \Longrightarrow \mathtt{Fail}$
$\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z})) \Longrightarrow \mathtt{S}(\mathtt{Z})$
$\vdash \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x ; \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \circ \mathtt{+}(\mathtt{Z}, \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))) \Longrightarrow \mathtt{S}(\mathtt{Z})$
$\vdash One(\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x) \circ \mathtt{+}(\mathtt{S}(\mathtt{Z}), \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))) \Longrightarrow \mathtt{+}(\mathtt{S}(\mathtt{Z}), \mathtt{S}(\mathtt{Z}))$
$\vdash TopDown(\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \nleftarrow \mathtt{+}(\mathtt{S}(x), y) \twoheadrightarrow \mathtt{S}(\mathtt{+}(x, y))) \circ \mathtt{+}(\mathtt{S}(\mathtt{Z}), \mathtt{S}(\mathtt{S}(\mathtt{Z}))) \Longrightarrow \mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{Z})))$
$\vdash Innermost(\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \nleftarrow \mathtt{+}(\mathtt{S}(x), y) \twoheadrightarrow \mathtt{S}(\mathtt{+}(x, y))) \circ \mathtt{+}(\mathtt{S}(\mathtt{S}(\mathtt{Z})), \mathtt{S}(\mathtt{S}(\mathtt{Z}))) \Longrightarrow \mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{S}(\mathtt{Z}))))$

3 Encoding rewriting strategies with rewrite rules

The evaluation of the application of a strategy on a subject consists in setting the "focus" on the active strategy *w.r.t.* the global strategy for control combinators (*e.g.* selecting $S1$ in $S_1 \nleftarrow S_2$ in the inference rule $\mathbf{choice_1}$), in setting the "focus" on the active term(s) *w.r.t.* the global term for traversal combinators (*e.g.* selecting t_i in $f(t_1, \ldots, t_n)$ in the inference rule $\mathbf{one_1}$), and eventually in applying elementary strategies.

The translation function presented in Figure 2 associates to each strategy a set of rewrite rules which encodes exactly this behaviour and preserves the original evaluation: $\mathcal{T}(S) \bullet \varphi_S(t) \longrightarrow u$ whenever $\vdash S \circ t \Longrightarrow u$ (the exact relationship between the strategy and its encoding is formally stated in Section 4). The set $\mathcal{T}(S)$ contains a number of rules whose left-hand sides are headed by φ_S and which encode the behaviour of the strategy S by using, in the right-hand sides, the symbols φ corresponding to the sub-strategies of S and potentially some auxiliary symbols. These φ_S and auxiliary φ symbols are supposed to be freshly generated and uniquely identified, *i.e.* there will be only one φ_S symbol for each encoded (sub-)strategy S and each auxiliary φ symbol can be identified by the strategy it has been generated for. For example, in the encoding $\mathcal{T}(S_1\,;S_2)$, the symbol $\varphi_;$ is just an abbreviation for $\varphi_;^{S_1;S_2}$, *i.e.* the specific $\varphi_;$ used for the encoding of the strategy $S_1\,;S_2$.

The left-hand and right-hand sides of the generated rules are built using the symbols of the original signature, the φ symbols mentioned previously as well as a particular symbol $\bot$ of arity 1 which encodes the failure and whose argument can be used to keep track of the origin of the failure. To keep the presentation of the translation compact and intuitive, we express it using rule schemas which use some special symbols to provide a concise representation of the rewrite rules. We start by introducing these special symbols and we then discuss the translation process.

First, we use the so-called *anti-terms*[1] of the form $!t$ with $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Intuitively, an anti-term $!t$ represents all the terms which do not match t and an anti-term $\varphi(!t)$ represents all the terms which do not match $\varphi(t)$; the way the finite representation of these terms is generated is detailed in Section 5. For example, if we consider the signature from Example 1, $!\mathtt{+}(\mathtt{Z}, x)$ denotes exactly the terms matched by $\mathtt{Z}$, $\mathtt{S}(x_1)$, $\mathtt{+}(\mathtt{S}(x_1), x_2)$ or $\mathtt{+}(\mathtt{+}(x_1, x_2), x_3)$. In this encoding, the semantics of $!t$ with $t \in \mathcal{T}_\mathcal{F}$ are considered *w.r.t.* the terms in $\mathcal{T}_\mathcal{F}$. For example, $!c$ for some constant c does not include $\bot(x)$ or terms of the form $\varphi(x_1, \ldots, x_n)$, because $\bot$ and φ symbols do not belong to the original signature. We can also complement failures but still *w.r.t.* the terms in $\mathcal{T}_\mathcal{F}$ and the pattern $!\bot(x)$ denotes thus all the ground terms $t \in \mathcal{T}_\mathcal{F}$ of the original signature. Terms in the left-hand sides of rules can be aliased, using the symbol "@", by variables which can be then conveniently used in the right-hand sides of the corresponding rules. Moreover, the variable symbol "_" can be used in the left-hand side of a rule to indicate a variable that does not appear in the right-hand side.

For example, the rule schema $\varphi(y \mathbin{@} !\mathtt{+}(\mathtt{Z}, \_)) \twoheadrightarrow \bot(y)$ denotes the set of rewrite rules consisting of $\varphi(\mathtt{Z}) \twoheadrightarrow \bot(\mathtt{Z})$, $\varphi(\mathtt{S}(y_1)) \twoheadrightarrow \bot(\mathtt{S}(y_1))$, $\varphi(\mathtt{+}(\mathtt{S}(y_1), y_2)) \twoheadrightarrow \bot(\mathtt{+}(\mathtt{S}(y_1), y_2))$ and $\varphi(\mathtt{+}(\mathtt{+}(y_1, y_2), y_3)) \twoheadrightarrow \bot(\mathtt{+}(\mathtt{+}(y_1, y_2), y_3))$.

The translation of the *Identity* strategy (**E1**) consists of a rule whose left-hand side matches any term in the signature[2] (contextualized by the corresponding φ symbol) and whose right-hand side is the initial term, and of a rule encoding strict propagation of failure. This latter rule guarantees a faithful encoding of the strategy guided evaluation and is in fact present, in different forms, in the translations of all the strategy operators. Similarly, the translation of the *Fail* strategy (**E2**) contains a failure propagation rule, and a rule whose left-hand side matches any term and whose right-hand side is a failure keeping track of this term.

A rewrite rule (which is an elementary strategy applicable at the root of the subject) is translated (**E3**) by two rules encoding the behaviour in case of respectively a matching success or a failure, together with a rule for failure propagation.

[1] We restrict here to a limited form of anti-terms; we refer to [7] for the complete semantics of anti-terms.
[2] The rule is in fact expanded into n rewrite rules with n the number of symbols in $\mathcal{F}$.

$$
\begin{array}{lrcl}
(\mathbf{E1}) & \mathcal{T}(\mathit{Identity}) & = & \{\ \varphi_{\mathit{Identity}}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow x,\quad \varphi_{\mathit{Identity}}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E2}) & \mathcal{T}(\mathit{Fail}) & = & \{\ \varphi_{\mathit{Fail}}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \bot(x),\quad \varphi_{\mathit{Fail}}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E3}) & \mathcal{T}(l \twoheadrightarrow r) & = & \{\ \varphi_{l\twoheadrightarrow r}(l) \twoheadrightarrow r, \\
 & & & \quad \varphi_{l\twoheadrightarrow r}(x \mathbin{@} !l) \twoheadrightarrow \bot(x),\quad \varphi_{l\twoheadrightarrow r}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
 & & & \\
(\mathbf{E4}) & \mathcal{T}(S_1\,;S_2) & = & \mathcal{T}(S_1) \cup \mathcal{T}(S_2) \\
 & & \bigcup & \{\ \varphi_{S_1;S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(x)), x),\quad \varphi_{S_1;S_2}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \varphi_{;}(x \mathbin{@} !\bot(\_), \_) \twoheadrightarrow x,\quad \varphi_{;}(\bot(\_), x) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E5}) & \mathcal{T}(S_1 \nleftrightarrow S_2) & = & \mathcal{T}(S_1) \cup \mathcal{T}(S_2) \\
 & & \bigcup & \{\ \varphi_{S_1\nleftrightarrow S_2}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{\nleftrightarrow}(\varphi_{S_1}(x)),\quad \varphi_{S_1\nleftrightarrow S_2}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \varphi_{\nleftrightarrow}(\bot(x)) \twoheadrightarrow \varphi_{S_2}(x),\quad \varphi_{\nleftrightarrow}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow x\ \} \\
(\mathbf{E6}) & \mathcal{T}(\mu X\,.\,S) & = & \mathcal{T}(S) \\
 & & \bigcup & \{\ \varphi_{\mu X.S}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_S(x),\quad \varphi_{\mu X.S}(\bot(x)) \twoheadrightarrow \bot(x), \\
 & & & \varphi_X(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_S(x),\quad \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E7}) & \mathcal{T}(X) & = & \emptyset \\
 & & & \\
(\mathbf{E8}) & \mathcal{T}(\mathit{All}(S)) & = & \mathcal{T}(S) \\
 & & \bigcup & \{\ \varphi_{\mathit{All}(S)}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
 & & \bigcup_{c\in\mathcal{F}^0} & \{\ \varphi_{\mathit{All}(S)}(c) \twoheadrightarrow c\ \} \\
 & & \bigcup_{f\in\mathcal{F}^+} & \{\ \varphi_{\mathit{All}(S)}(f(x_1,\ldots,x_n)) \twoheadrightarrow \varphi_f(\varphi_S(x_1),\ldots,\varphi_S(x_n), f(x_1,\ldots,x_n)), \\
 & & & \varphi_f(x_1 \mathbin{@} !\bot(\_),\ldots,x_n \mathbin{@} !\bot(\_),\_) \twoheadrightarrow f(x_1,\ldots,x_n), \\
 & & & \varphi_f(\bot(\_),\_,\ldots,\_,x) \twoheadrightarrow \bot(x), \\
 & & & \vdots \\
 & & & \varphi_f(\_,\ldots,\_,\bot(\_),x) \twoheadrightarrow \bot(x)\ \} \\
(\mathbf{E9}) & \mathcal{T}(\mathit{One}(S)) & = & \mathcal{T}(S) \\
 & & \bigcup & \{\ \varphi_{\mathit{One}(S)}(\bot(x)) \twoheadrightarrow \bot(x)\ \} \\
 & & \bigcup_{c\in\mathcal{F}^0} & \{\ \varphi_{\mathit{One}(S)}(c) \twoheadrightarrow \bot(c)\ \} \\
 & & \bigcup_{f\in\mathcal{F}^+} & \{\ \varphi_{\mathit{One}(S)}(f(x_1,\ldots,x_n)) \twoheadrightarrow \varphi_{f_1}(\varphi_S(x_1), x_2,\ldots,x_n)\ \} \\
 & \bigcup_{f\in\mathcal{F}^+}\ \bigcup_{1\le i\le ar(f)} & & \{\ \varphi_{f_i}(\bot(x_1),\ldots,\bot(x_{i-1}), x_i \mathbin{@} !\bot(\_), x_{i+1},\ldots,x_n) \twoheadrightarrow f(x_1,\ldots,x_n)\} \\
 & \bigcup_{f\in\mathcal{F}^+}\ \bigcup_{1\le i< ar(f)} & & \{\ \varphi_{f_i}(\bot(x_1),\ldots,\bot(x_i), x_{i+1},\ldots,x_n) \twoheadrightarrow \\
 & & & \quad \varphi_{f_{i+1}}(\bot(x_1),\ldots,\bot(x_i), \varphi_S(x_{i+1}), x_{i+2},\ldots,x_n)\ \} \\
 & & \bigcup_{f\in\mathcal{F}^+} & \{\ \varphi_{f_n}(\bot(x_1),\ldots,\bot(x_n)) \twoheadrightarrow \bot(f(x_1,\ldots,x_n))\ \} \\
 & & & \\
(\mathbf{E10}) & \mathcal{B}(\Gamma; X\colon S) & = & \mathcal{B}(\Gamma) \cup \mathcal{T}(S) \\
 & & \bigcup & \{\ \varphi_X(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_S(x),\quad \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \}
\end{array}
$$

Figure 2 Strategy translation.

▶ **Example 2.** The strategy $S_{pz} = \mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x$ consisting only of the rewrite rule of Example 1 is encoded by the following rules:

$$\begin{aligned}\mathcal{T}(S_{pz}) = \{\quad & \varphi_{pz}(\mathtt{+}(\mathtt{Z}, x)) \twoheadrightarrow x,\\ & \varphi_{pz}(y \mathbin{@} !\mathtt{+}(\mathtt{Z}, x)) \twoheadrightarrow \bot(y),\\ & \varphi_{pz}(\bot(x)) \twoheadrightarrow \bot(x)\quad\}\end{aligned}$$

which lead, when the anti-terms are expanded *w.r.t.* to the signature, to the TRS:

$$\begin{aligned}\mathcal{T}(S_{pz}) = \{\quad & \varphi_{pz}(\mathtt{+}(\mathtt{Z}, x)) \twoheadrightarrow x,\\ & \varphi_{pz}(\mathtt{Z}) \twoheadrightarrow \bot(\mathtt{Z}),\\ & \varphi_{pz}(\mathtt{S}(y_1)) \twoheadrightarrow \bot(\mathtt{S}(y_1)),\\ & \varphi_{pz}(\mathtt{+}(\mathtt{S}(y_1), y_2)) \twoheadrightarrow \bot(\mathtt{+}(\mathtt{S}(y_1), y_2)),\\ & \varphi_{pz}(\mathtt{+}(\mathtt{+}(y_1, y_2), y_3)) \twoheadrightarrow \bot(\mathtt{+}(\mathtt{+}(y_1, y_2), y_3)),\\ & \varphi_{pz}(\bot(x)) \twoheadrightarrow \bot(x)\quad\}\end{aligned}$$

The term $\varphi_{pz}(\mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z})))$ reduces *w.r.t.* this latter TRS to $\mathtt{S}(\mathtt{Z})$ and $\varphi_{pz}(\mathtt{Z})$ reduces to $\bot(\mathtt{Z})$.

The translation of the sequential application of two strategies (**E4**) includes the translation of the respective strategies and some specific rules. A term $\varphi_{S_1;S_2}(t)$ is reduced by the first rule into a term $\varphi_{;}(\varphi_{S_2}(\varphi_{S_1}(t)), t)$, which guarantees that the rules of the encoding of S_1 are applied before the ones of S_2. Indeed, a term of the form $\varphi(t)$ can be reduced only if $t \in \mathcal{T}_{\mathcal{F}}$ or $t = \bot(\_)$ and thus, the rules for φ_{S_2} can be applied to a term $\varphi_{S_2}(\varphi_{S_1}(\_))$ only after $\varphi_{S_1}(\_)$ is reduced to a term in $\mathcal{T}_{\mathcal{F}}$ (or failure). The original subject t is kept during the evaluation (of $\varphi_{;}$), so that $\bot(t)$ can be returned if the evaluation of S_1 or S_2 fails (*i.e.* produces a $\bot$) at some point. If $\varphi_{S_2}(\varphi_{S_1}(t))$ evaluates to a term $t' \in \mathcal{T}_{\mathcal{F}}$, then the evaluation of $\varphi_{S_1;S_2}(t)$ succeeds, and t' is the final result. In a similar manner, the translation for the choice operator (**E5**) uses a rule which triggers the application of the rules for S_1. If the corresponding evaluation results in a failure then the application of the rules for S_2 is triggered on the original subject; otherwise the result is returned.

The translation for a strategy $\mu X\,.\,S$ (**E6**) triggers the application of the rules for S at first, and then each time the symbol φ_X is encountered. As in all the other cases, failure is strictly propagated. There is no rewrite rule for the translation of a strategy variable (**E7**) but we should note that the corresponding φ_X symbol could be used when translating the strategy S (in $\mu X\,.\,S$), as we can see in the next example.

▶ **Example 3.** The strategy $S_{rpz} = \mu X.(\mathtt{+}(\mathtt{Z}, x) \twoheadrightarrow x \;;\; X) \leftarrow\!\!+ \mathit{Identity}$ which applies repeatedly (as long as possible) the rewrite rule from Example 2 is encoded by:

$$\begin{aligned}\mathcal{T}(S_{rpz}) = \quad & \{\ \varphi_{rpz}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{tpz}(x), && \varphi_{rpz}(\bot(x)) \twoheadrightarrow \bot(x)\ \}\\ \cup\quad & \{\ \varphi_X(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{tpz}(x), && \varphi_X(\bot(x)) \twoheadrightarrow \bot(x)\ \}\\ \cup\quad & \{\ \varphi_{tpz}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{\leftarrow\!\!+}(\varphi_{pzX}(x)), && \varphi_{tpz}(\bot(x)) \twoheadrightarrow \bot(x),\\ & \quad \varphi_{\leftarrow\!\!+}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow x, && \varphi_{\leftarrow\!\!+}(\bot(x)) \twoheadrightarrow \varphi_{\mathit{Identity}}(x)\ \}\\ \cup\quad & \{\ \varphi_{\mathit{Identity}}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow x, && \varphi_{\mathit{Identity}}(\bot(x)) \twoheadrightarrow \bot(x)\ \}\\ \cup\quad & \{\ \varphi_{pzX}(x \mathbin{@} !\bot(\_)) \twoheadrightarrow \varphi_{;}(\varphi_X(\varphi_{pz}(x)), x), && \varphi_{pzX}(\bot(x)) \twoheadrightarrow \bot(x),\\ & \quad \varphi_{;}(x \mathbin{@} !\bot(\_), \_) \twoheadrightarrow x, && \varphi_{;}(\bot(\_), x) \twoheadrightarrow \bot(x)\ \}\\ \cup\quad & \mathcal{T}(S_{pz})\end{aligned}$$

For presentation purposes, we separated the TRS in sub-sets of rules corresponding to the translation of each operator occurring in the initial strategy. Note that the symbol φ_X used in the rules for the inner sequence can be reduced with the rules generated to handle the recursion operator. The term $\varphi_{rpz}(\mathtt{+}(\mathtt{Z}, \mathtt{+}(\mathtt{Z}, \mathtt{S}(\mathtt{Z}))))$ reduces *w.r.t.* the TRS to $\mathtt{S}(\mathtt{Z})$.

The rules encoding the traversal operators follow the same principle – the rules corresponding to the translation of the argument strategy S are applied, depending on the traversal operator, to one or all the sub-terms of the subject. For the *All* operator (**E8**), if

the application of S to all the sub-terms succeeds (produces terms in $\mathcal{T}_\mathcal{F}$), then the final result is built using the results of each evaluation. If the evaluation of one of the sub-terms produces a $\bot$, a failure with the original subject as origin is returned as a result. Special rules encode the fact that *All* applied to a constant always succeeds; the same behaviour could have been obtained by instantiating the rules for non-constants with $n = 0$, but we preferred an explicit approach for uniformity and efficiency reasons. In the case of the *One* operator (**E9**), if the evaluation for one sub-term results in a failure, then the evaluation of the strategy S is triggered on the next one. If S fails on all sub-terms, a failure with the original subject as origin is returned. The failure in case of constants is necessarily encoded by specific rules.

Finally, each binding $X : S$ of a context (**E10**) is translated by two rules, including the one that propagates failure. The other rule operates as in the recursive case (rule **E6**): applying the strategy variable X to a subject t leads to the application of the rules encoding S to t.

4 Properties of the translation

The goal of the translation is twofold: use well-established methods and tools for plain TRS in order to prove properties of strategy controlled rewrite rules, and offer a generic compiler for user defined strategies. For both items, it is crucial to have a sound and complete translation, and this turns out to be true in our case.

▶ **Theorem 4** (Simulation). *Given a term $t \in \mathcal{T}_\mathcal{F}$, a strategy S and a context Γ*
1. $\Gamma \vdash S \circ t \Longrightarrow t'$ *iff* $\mathcal{T}(S) \cup \mathcal{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow t'$, $t' \in \mathcal{T}_\mathcal{F}$
2. $\Gamma \vdash S \circ t \Longrightarrow \texttt{Fail}$ *iff* $\mathcal{T}(S) \cup \mathcal{B}(\Gamma) \bullet \varphi_S(t) \longrightarrow \bot(t)$

Proof. The completeness is shown by induction on the height of the derivation tree and the soundness by induction on the length of the reduction. The base cases consisting of the strategies with a constant length reduction – *Identity*, *Fail*, and the rewrite rule – are straightforward to prove since, in particular, the translation of a rule explicitly encodes matching success and failure. Induction is applied for all the other cases and the corresponding proofs rely on some auxiliary properties.

First, the failure is strictly propagated: if $\mathcal{B}(\Gamma) \cup \mathcal{T}(S) \bullet \varphi_S(\bot(t)) \longrightarrow u$, then $u = \bot(t)$. This is essential, in particular, for the sequence case where a failure of the first strategy should be strictly propagated as the final result of the overall sequential strategy.

Second, we note that terms in the signature are in normal form *w.r.t.* the (rules in the) translation of any strategy and that contextualized terms of the form $\varphi_S(t)$ are head-rigid *w.r.t.* to (the translation of) strategies other than S, *i.e.*, they can be reduced at the head position only by the rules obtained for the translation of S and only if t is not contextualized but a term in the signature. More precisely, if for a strategy S' and a context Γ, $\mathcal{B}(\Gamma) \cup \mathcal{T}(S') \bullet \varphi_S(t) \longrightarrow u$ then $t \in \mathcal{T}_\mathcal{F}$ and $\mathcal{T}(S) \subseteq \mathcal{B}(\Gamma) \cup \mathcal{T}(S')$ (or $S = X$ and Γ binds X). This guarantees that the steps in the strategy derivation are encoded accurately by the evaluations *w.r.t.* the rules in the translation.

Finally, the origin of the failure is preserved in the sense that if for a $t \in \mathcal{T}_\mathcal{F}$, $\varphi_S(t)$ reduces to a failure, then the reduct is necessarily $\bot(t)$. This is crucial in particular for the choice strategy: if the (translation of the) first strategy fails, then the (translation of the) second one should be applied on the initial subject. ◀

As a direct consequence of this property, we obtain that (non-)termination of one system implies the (non-)termination of the other.

▶ **Corollary 5** (Termination). *Given a strategy S and a context Γ, the strategy application $\Gamma \vdash S \circ t$ has a finite derivation for any term $t \in \mathcal{T}_{\mathcal{F}}$ iff $\mathcal{T}(S) \cup \mathcal{B}(\Gamma)$ is terminating.*

The main goal is to prove the termination of some strategy guided system by proving the property for the plain TRS obtained by translation. When termination does not hold, non-terminating TRS reductions correspond to infinite strategy-controlled derivations.

5 Implementation and experimental results

The strategy translation presented in Section 3 has been implemented in a tool called StrategyAnalyser[3], written in Tom, a language that extends Java with high level constructs for pattern matching, rewrite rules and strategies. Given a set of rewrite rules guided by a strategy, the tool generates a plain TRS in AProVE/TTT2 syntax[4] or Tom syntax. In this section we illustrate our approach on two representative examples.

The first one comes from an optimizer for multi-core architectures, a project where abstract syntax trees are manipulated and transformations are expressed using rewrite rules and strategies, and consists of two rewrite rules identified as patterns occurring often in various forms in the project. First, the rewrite rule $g(f(x)) \to f(g(x))$ corresponds to the search for an operator g (which can have more than one parameter in the general case) which is pushed down under another operator f (again, this operator may have more than one parameter). This rule is important since the corresponding (innermost) reduction of a term of the form $t_{\text{gf}} = \overbrace{g(\underbrace{f(\cdots(f}_{n}(g(\underbrace{f(\cdots(f}_{n}(g(\underbrace{f(\cdots(f}_{n}(g(a)))\cdots)}^{m}$, with, for example, $n = 10$ and $m = 18$ occurrences of g, involves a lot of computations and could be a performance bottleneck. Second, the rewrite rule $h(x) \to g(h(x))$ corresponds to wrapping some parts of a program by some special constructs, like `try/catch` for example, and it is interesting since its uncontrolled application is obviously non-terminating.

At present, a strategy given as input to StrategyAnalyser is written in a simple functional style and a possible strategy for our example could be:

```
let S = signature {a:0, b:0, f:1, g:1, h:1} in
let gfx = { g(f(x)) -> f(g(x)) } in
let hx  = { h(x) -> g(h(x)) } in
let obu(S) = mu X.(one(X) <+ S) in   ## obu stands for OnceBottomUp
let try(S) = S <+ identity in
let repeat(S) = mu X.(try(S ; X)) in  ## naive definition of innermost to
repeat(obu(gfx))                      ## illustrate various possibilites
```

As a second example, we consider the following rewrite rules which implement the distributivity and factorization of symbolic expressions composed of `+` and `*` and their application under a specific strategy:

```
let S = signature { Z:0, S:1, +:2, *:2 } in
let dist = { *(x, +(y,z))      -> +(*(x,y),*(x,z)) } in
let fact = { +(*(x,y), *(x,z)) -> *(x,+(y,z)) } in
let innermost(S) = mu X.(all(X) ; ((S ; X) <+ identity)) in
    innermost(dist) ; innermost(fact)
```

[3] source code available at `http://tom.loria.fr/`, directory `jtom/applications`
[4] `http://aprove.informatik.rwth-aachen.de/`

This time the strategy involves rewrite rules which are either non left-linear or non right-linear and which are non-terminating if their application is not guided by a strategy.

The StrategyAnalyser tool is built in a modular way such that the compilation (*i.e.* the translation presented in Section 3) is performed at an abstract level and therefore, new concrete syntaxes and new backends can be easily added.

5.1 Generation of executable TRS

When run with the flag `-tom`, the StrategyAnalyser tool generates a TRS in Tom syntax which can be subsequently compiled into Java code and executed. Moreover, the tool can be configured to generate TRS that use the alias notation or not, and to use the notion of anti-term or not. An encoding using anti-terms and aliasing can be directly used in a Tom program but for languages and tools which do not offer such primitives, aliases and anti-terms have to be expanded into plain rewrite rules. We explain first how this expansion is realized and we discuss then the performances of the obtained executable TRS.

The rules given in Figure 2 can generate two kinds of rules which contain anti-terms. The first family is of the form $\varphi(\ldots, y_i \mathbin{@} !\bot(\_), \ldots) \twoheadrightarrow u$ with $y_i \in \mathcal{X}$, and with potentially several occurrences of $!\bot(\_)$. These rules can be easily expanded into a family of rules $\varphi(\ldots, y_i \mathbin{@} f(x_1, \ldots, x_n), \ldots) \twoheadrightarrow u$ with such a rule for all $f \in \mathcal{F}$, and with $x_1, \ldots, x_n \in \mathcal{X}$ and $n = ar(f)$. This expansion is performed recursively to eliminate all the instances of $!\bot(\_)$. The other rules containing anti-terms come from the translation of rewrite rules (see (E3)) and have the form: $\varphi(y \mathbin{@} !f(t_1, \ldots, t_n)) \twoheadrightarrow \bot(y)$, with $f \in \mathcal{F}^n$, $y \in \mathcal{X}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. If the term $f(t_1, \ldots, t_n)$ is linear, then the tool generates two families of rules:

- $\varphi(g(x_1, \ldots, x_m)) \twoheadrightarrow \bot(g(x_1, \ldots, x_m))$ for all $g \in \mathcal{F}$ s.t. $g \neq f$, $x_1, \ldots, x_m \in \mathcal{X}$, $m = ar(g)$,
- $\varphi(f(x_1, \ldots, x_{i-1}, x_i \mathbin{@} !t_i, x_{i+1}, \ldots, x_n)) \twoheadrightarrow \bot(f(x_1, \ldots, x_n))$ for all $i \in [1, n]$ and $t_i \notin \mathcal{X}$,

with the second family of rules recursively expanded, using the same algorithm, until there is no anti-term left. For example, if we consider the signature used for the rule gfx, the rule $\varphi(y \mathbin{@} !\bot(\_)) \twoheadrightarrow y$ is expanded into the set of rewrite rules $\{\, \varphi(a) \twoheadrightarrow a,\ \varphi(b) \twoheadrightarrow b,\ \varphi(f(x_1)) \twoheadrightarrow f(x_1),\ \varphi(g(x_1)) \twoheadrightarrow g(x_1),\ \varphi(h(x_1)) \twoheadrightarrow h(x_1) \,\}$ and the rule $\varphi(y @ !g(f(x)) \twoheadrightarrow \bot(y)$ is expanded into the set of rewrite rules $\{\ \varphi(a) \twoheadrightarrow \bot(a),\ \varphi(b) \twoheadrightarrow \bot(b),\ \varphi(f(x_1)) \twoheadrightarrow \bot(f(x_1)),\ \varphi(g(a)) \twoheadrightarrow \bot(g(a)),\ \varphi(g(b)) \twoheadrightarrow \bot(g(b)),\ \varphi(g(g(x_1))) \twoheadrightarrow \bot(g(g(x_1))),\ \varphi(g(h(x_1))) \twoheadrightarrow \bot(g(h(x_1))),\ \varphi(h(x_1)) \twoheadrightarrow \bot(h(x_1)) \,\}$.

This expansion mechanism is more difficult when we want to find a convenient (finite) encoding for non-linear anti-terms and in this case the expansion should be done, in fact, *w.r.t.* the entire translation of a rewrite rule. Given the rules $\varphi(l) \twoheadrightarrow r$ and $\varphi(y \mathbin{@} !l) \twoheadrightarrow \bot(y)$ with $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ a non linear term, we consider the linearized version of l, denoted l', with all the variables $x_i \in \mathcal{V}ar\,(l)$ appearing more than once (m_i times, with $m_i > 1$) renamed into $z_i^1, \ldots, z_i^{m_i - 1}$ (the first occurrence of x_i is not renamed). Then, these two rules can be translated into:

- $\varphi(y \mathbin{@} !l') \twoheadrightarrow \bot(y)$
- $\varphi(l') \twoheadrightarrow \varphi'(l', x_1 = z_1^1 \wedge \cdots \wedge x_1 = z_1^{m_1 - 1} \wedge \cdots \wedge x_n = z_n^1 \wedge \cdots \wedge x_n = z_n^{m_n - 1})$
- $\varphi'(l', \text{true}) \twoheadrightarrow r$
- $\varphi'(l', \text{false}) \twoheadrightarrow \bot(l')$

with the first rule containing now the linear anti-term $!l'$ expanded as previously. The rules generated for equality and conjunction are as expected.

When considering the rule $+(*(x, y), *(x, z)) \twoheadrightarrow *(x, +(y, z))$ the translation generates the rules $\varphi(+(*(x_1, x_2), *(x_1, x_3))) \twoheadrightarrow *(x_1, +(x_2, x_3))$ and $\varphi(y \mathbin{@} !+(*(x_1, x_2), *(x_1, x_3))) \twoheadrightarrow \bot(y)$,

which are expanded into the following plain TRS:

$$
\begin{array}{lcl}
\varphi(\mathsf{Z}) & \to & \bot(\mathsf{Z}),\\
\varphi(\mathsf{S}(x_1)) & \to & \bot(\mathsf{S}(x_1)),\\
\varphi(*(x_1,x_2)) & \to & \bot(*(x_1,x_2)),\\
\varphi(+(x_1,\mathsf{Z})) & \to & \bot(+(x_1,\mathsf{Z})),\\
\varphi(+(x_1,\mathsf{S}(x_2))) & \to & \bot(+(x_1,\mathsf{S}(x_2))),\\
\varphi(+(x_1,+(x_2,x_3))) & \to & \bot(+(x_1,+(x_2,x_3))),\\
\varphi(+(\mathsf{Z},x_2)) & \to & \bot(+(\mathsf{Z},x_2)),\\
\varphi(+(\mathsf{S}(x_1),x_2)) & \to & \bot(+(\mathsf{S}(x_1),x_2)),\\
\varphi(+(+(x_1,x_2),x_3)) & \to & \bot(+(+(x_1,x_2),x_3)),\\
\varphi(+(*(x_1,x_2),*(z_1^1,x_3))) & \to & \varphi'(+(*(x_1,x_2),*(z_1^1,x_3)),x_1=z_1^1),\\
\varphi'(+(*(x_1,x_2),*(z_1^1,x_3)),\mathrm{true}) & \to & *(x_1,+(x_2,x_3)),\\
\varphi'(+(*(x_1,x_2),*(z_1^1,x_3)),\mathrm{false}) & \to & \bot(+(*(x_1,x_2),*(z_1^1,x_3)))
\end{array}
$$

The number of generated rules for a strategy could thus be significant and it is interesting to see how this impacts the efficiency of the execution of such a system.

If we execute a Tom+Java program corresponding to the `repeat(obu(gfx))` strategy defined at the beginning of the section and designed using a classic built-in implementation of strategies where strategy failure is implemented by a Java exception, the normalization of the term t_{gf} takes 6.3 s[5] (Table 1, column Tom). When using an alternative built-in implementation with a special encoding of failure which avoids throwing Java exceptions, the computation time decreases to 0.4 s (Table 1, column Tom*). The strategy `repeat(obu(gfx))` is translated into an executable TRS containing 90 Tom plain rewrite rules and the normalization takes in this case 0.7 s! More benchmarks for the application of other strategies involving the rules gfx and hx on the same term t_{gf} as well as the application of strategies involving the rules dist and fact on terms containing more than 400 symbols[6] `+` and `*` are presented in Table 1.

We observe that, although the number of generated rules could be significant, the execution times of the resulting plain TRS are comparable to those obtained with the native implementation of Tom strategy. This might look somewhat surprising but can be explained when we take a closer look to the way rewriting rules and strategies are generally implemented:

- the implementation of a TRS can be done in an efficient way since the complexity of syntactic pattern matching depends only on the size of the term to be reduce and, thanks to many-to-one matching algorithms [18, 10], the number of rules has almost no impact.
- in Tom, each native strategy constructor is implemented by a Java class with a `visit` method which implements (*i.e.* interprets) the semantics of the corresponding operator. The evaluation of a strategy S on a term t is implemented thus by a call `S.visit(t)` and an exception (`VisitFailure`) is thrown when the application of a strategy fails.

In the generated TRS, the memory allocation involved in the construction of terms headed by the $\bot$ symbol encoding failure appears to be more efficient than the costly Java exception handling. This is reflected by better performances of the plain TRS implementation compared to the exception-based native implementation (especially when the strategy involves a lot of failures). We obtain performances with the generated TRS comparable to an exception-free native implementation of strategies (as we can see with the columns TRS and Tom* in

[5] on a MacPro 3GHz

[6] term of the form $+(t_7,\mathsf{Z})$, with $t_{i+1}=*(\mathsf{Z},+(t_i,t_i))$, and $t_0=+(\mathsf{Z},\mathsf{Z})$

Table 1 Benchmarks: the column #rules indicates the number of plain rewrite rules generated for the strategy, the column T indicates whether the termination of the rules have been (dis)proven by AProVE, the column TRS indicates the execution time in milliseconds for the executable TRS, the column Tom indicates the execution time of the Tom built-in exception-based implementation and the column Tom* indicates the execution time of the Tom built-in exception-free implementation.

Name	Strategy	#rules	T	TRS	Tom	Tom*
repeat(dist)	$\mu X\,.\,((\mathsf{dist}\,;X) \leftarrow\!\!+ \mathit{Identity})$	60	✓	<5	<5	<5
repeat(fact)	$\mu X\,.\,((\mathsf{fact}\,;X) \leftarrow\!\!+ \mathit{Identity})$	63	✓	<5	13	<5
repeat(dist ; fact)	$\mu X\,.\,(((\mathsf{dist}\,;\mathsf{fact})\,;X) \leftarrow\!\!+ \mathit{Identity})$	83	✗	–	–	–
td(dist)	$\mu X\,.\,((\mathsf{dist} \leftarrow\!\!+ \mathit{Identity})\,;\mathit{All}(X))$	125	✓	39	12	30
obu(fact)	$\mu X\,.\,(\mathit{One}(X) \leftarrow\!\!+ \mathsf{fact})$	73	✓	<5	<5	<5
repeat(obu(fact))	$\mu X\,.\,((\mathsf{obu}(\mathsf{fact})\,;X) \leftarrow\!\!+ \mathit{Identity})$	103	✓	220	2460	120
	td(dist) ; repeat(obu(fact))	218	✓	296	2601	150
rbufact	$\mu X\,.\,(\mathit{All}(X);$					
	$\quad((\mathsf{fact}\,;\mathit{All}(X)) \leftarrow\!\!+ \mathit{Identity}))$	202	✓	511	427	302
	td(dist) ; rbufact	318	✓	557	453	328
innermost(dist)	$\mu X\,.\,(\mathit{All}(X)\,;((\mathsf{dist}\,;X) \leftarrow\!\!+ \mathit{Identity}))$	135	✓	370	650	230
innermost(fact)	$\mu X\,.\,(\mathit{All}(X)\,;((\mathsf{fact}\,;X) \leftarrow\!\!+ \mathit{Identity}))$	138	✓	345	308	149
	innermost(dist) ; innermost(fact)	138	✓	866	960	340
repeat(td(dist))	$\mu X\,.\,((\mathsf{td}(\mathsf{dist})\,;X) \leftarrow\!\!+ \mathit{Identity})$	155	✗	–	–	–
bu(hx)	$\mu X\,.\,(\mathit{All}(X)\,;(\mathsf{hx} \leftarrow\!\!+ \mathit{Identity}))$	72	✓	5	6	5
td(hx)	$\mu X\,.\,((\mathsf{hx} \leftarrow\!\!+ \mathit{Identity})\,;\mathit{All}(X))$	72	✗	–	–	–
repeat(obu(gfx))	$\mu X\,.\,((\mathsf{obu}(\mathsf{gfx})\,;X) \leftarrow\!\!+ \mathit{Identity})$	90	✓	699	6300	414
innermost(gfx)	$\mu X\,.\,(\mathit{All}(X)\,;((\mathsf{gfx}\,;X) \leftarrow\!\!+ \mathit{Identity}))$	85	✓	565	4180	365
propagate	$\mu X\,.\,(\mathsf{gfx}\,;(\mathit{All}(X) \leftarrow\!\!+ \mathit{Identity}))$	75	✓	<5	<5	<5
bup	$\mu X\,.\,(\mathit{All}(X)\,;(\mathsf{propagate} \leftarrow\!\!+ \mathit{Identity}))$	121	✓	59	46	42

Table 1), because efficient normalization techniques can be used for the plain TRS, since its rewrite rules are not controlled by a programmable strategy.

5.2 Generation of TRS for termination analysis

When run with the flag `-aprove`, the StrategyAnalyser tool generates a TRS in AProVE/TTT2 syntax which can be analyzed by any tool accepting this syntax. In this case, aliases and anti-terms are always completely expanded leading generally to an important number of plain rewrite rules. Fortunately, the number of rules does not seem to be a problem for AProVE and, for example, the termination of the strategy repeat(obu(gfx)), which is translated into 90 rules, is proven in approximately 10 s (using the web interface). Similarly, the termination of the strategy td(dist) ; rbufact, whose definition is given in Table 1, is translated into 318 rules, which can be proven terminating in approximately 75 s.

The termination of some strategies like, for example, repeat(obu(gfx)) might look pretty easy to show for an expert, but termination is less obvious for more complex strategies like, for example, bup, which is a specialized version of repeat(obu(gfx)), or rbufact, which is a variant of bu(fact).

The approach was effective not only in proving termination of some strategies, but also in disproving it when necessary. Once again this might look obvious for some strategies like, for example, td(hx), which involves a non-terminating rewrite rule, but it is less clear for strategies combining terminating rewrite rules or strategies like, *e.g.*, repeat(dist ; fact).

6 Conclusions and further work

We have proposed a translation of programmable strategies into plain rewrite rules that we have proven sound and complete. Well-established termination methods can be thus used to (dis)prove the termination of the obtained TRS and we can deduce, as a direct consequence, the property for the corresponding strategy. Alternatively, the translation can be used as a strategy compiler for languages which do not implement natively such primitives.

The translation has been implemented in Tom and can generate, for the moment, plain TRS using either a Tom or an AProVE/TTT2 syntax. We have experimented with classic strategies and AProVE and TTT2 have been able to (dis)prove the termination even when the number of generated rules was significant. The performances for the generated executable TRS are comparable to the ones of the Tom built-in (exception-free) strategies.

The framework can be of course improved. We expect problems in (dis)proving termination when the number of generated rewrite rules is too big, and we are currently working on a meta-level representation of the strategy translation which abstracts over the signature and considerably decreases the size of the generated TRS compared to the approach of this paper. When termination is disproven and a counter-example can be exhibited, it is interesting to reproduce the corresponding infinite reductions in terms of strategy derivations. Since the TRS reductions corresponding to distinct (sub-)strategy derivations are not interleaved, we think that the back-translation of the counter-examples provided by the termination tools can be automatized.

As far as the executable TRS is concerned, we intend to develop new backends allowing the integration of programmable strategies in other languages than Tom.

References

1 Beatriz Alarcón, Raúl Gutiérrez, and Salvador Lucas. Context-sensitive dependency pairs. *Inf. Comput.*, 208(8):922–968, 2010.

2 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.

3 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

4 Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking rewriting on java. In *RTA'07*, volume 4533 of *LNCS*, pages 36–47. Springer-Verlag, 2007.

5 Emilie Balland, Pierre-Etienne Moreau, and Antoine Reilles. Effective strategic programming for Java developers. *Software: Practice and Experience*, 44(2):129–162, 2012.

6 P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An overview of ELAN. In *WRLA'98*, volume 15. ENTCS, 1998.

7 Horatiu Cirstea, Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-patterns for rule-based languages. *Journal of Symbolic Computation*, 45(5):523–550, 2010. Symbolic Computation in Software Science.

8 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The Maude 2.0 System. In *RTA'03*, volume 2706 of *LNCS*, pages 76–87. Springer-Verlag, 2003.

9 Jörg Endrullis and Dimitri Hendriks. From outermost to context-sensitive rewriting. In *RTA'09*, volume 5595 of *LNCS*, pages 305–319. Springer, 2009.

10 Fabrice Le Fessant and Luc Maranget. Optimizing pattern matching. In *ICFP'01*, pages 26–37. ACM Press, 2001.

11 Olivier Fissore, Isabelle Gnaedig, and Hélène Kirchner. Simplification and termination of strategies in rule-based languages. In *PPDP'03*, pages 124–135. ACM, 2003.

12 Carsten Fuhs, Jürgen Giesl, Michael Parting, Peter Schneider-Kamp, and Stephan Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 47(2):133–160, 2011.

13 Jürgen Giesl and Aart Middeldorp. Innermost termination of context-sensitive rewriting. In *DLT'02*, volume 2450 of *LNCS*, pages 231–244. Springer, 2002.

14 Jürgen Giesl and Aart Middeldorp. Transformation techniques for context-sensitive rewrite systems. *J. Funct. Program.*, 14(4):379–427, 2004.

15 Jürgen Giesl, Matthias Raffelsieper, Peter Schneider-Kamp, Stephan Swiderski, and René Thiemann. Automated termination proofs for haskell by term rewriting. *ACM Trans. Program. Lang. Syst.*, 33(2):7, 2011.

16 Jürgen Giesl, René Thiemann, and Stephan Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37:2006, 2006.

17 Isabelle Gnaedig and Hélène Kirchner. Termination of rewriting under strategies. *ACM Trans. Comput. Log.*, 10(2), 2009.

18 Albert Gräf. Left-to-right tree pattern matching. In *RTA'91*, volume 488 of *LNCS*, pages 323–334. Springer-Verlag, April 1991.

19 Nao Hirokawa and Aart Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1–2):172–199, 2005.

20 J.-P. Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, 1991.

21 Markus Kaiser and Ralf Lämmel. An Isabelle/HOL-based model of stratego-like traversal strategies. In *PPDP'09*, pages 93–104. ACM, 2009.

22 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In *RTA'09*, volume 5595 of *LNCS*, pages 295–304. Springer-Verlag, 2009.

23 Ralf Lämmel, Simon J. Thompson, and Markus Kaiser. Programming errors in traversal programs over structured data. *Sci. Comput. Program.*, 78(10):1770–1808, 2013.

24 José Meseguer and Christiano Braga. Modular rewriting semantics of programming languages. In *Algebraic Methodology and Software Technology*, volume 3116 of *LNCS*, pages 364–378. Springer Berlin Heidelberg, 2004.

25 Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A Pattern Matching Compiler for Multiple Target Languages. In *CC'03*, volume 2622 of *LNCS*, pages 61–76. Springer-Verlag, 2003.

26 Matthias Raffelsieper and Hans Zantema. A transformational approach to prove outermost termination automatically. *ENTCS*, 237:3–21, 2009.

27 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. M. Bezem, J. W. Klop and R. de Vrijer, eds.

28 René Thiemann. From outermost termination to innermost termination. In *SOFSEM'09*, volume 5404 of *LNCS*, pages 533–545. Springer, 2009.

29 René Thiemann and Aart Middeldorp. Innermost termination of rewrite systems by labeling. *ENTCS*, 204:3–19, 2008.

30 Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5. In *RTA'01*, volume 2051 of *LNCS*, pages 357–361. Springer-Verlag, 2001.

31 Eelco Visser, Zine-el-Abidine Benaissa, and Andrew Tolmach. Building program optimizers with rewriting strategies. In *ICFP'98*, pages 13–26. ACM Press, 1998.