Model checking strategy-controlled rewriting systems

Extended version of the article presented at FSCD 2019

Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, Alberto Verdejo
{rubenrub,narciso,ipandreu,jalberto}@ucm.es

Technical report SIC 02/19
Dpto. de Sistemas Informáticos y Computación – Facultad de Informática – Universidad Complutense de Madrid

Abstract

Strategies are widespread in Computer Science. In the domain of reduction and rewriting systems, strategies are studied as recipes to restrict and control reduction steps and rule applications, which are intimately local, in a derivation-global sense. This idea has been exploited by various tools and rewriting-based specification languages, where strategies are an additional specification layer. Systems so described need to be analyzed too. This article discusses model checking of systems controlled by strategies and presents a working strategy-aware LTL model checker for the Maude specification language, based on rewriting logic, and its strategy language.

This extended version includes the proofs of the propositions in the conference paper, and a complete description of the small-step operational semantics used to define model checking for the Maude strategy language.

2012 ACM Subject Classification Software and its engineering → Model checking; Theory of computation → Equational logic and rewriting; Theory of computation → Rewrite systems

Keywords and phrases Model checking, strategies, Maude, rewriting logic

Related Version Extended version of [31].

Funding Research partially supported by MCIU Spanish project TRACES (TIN2015-67522-C3-3-R). *Rubén Rubio*: Partially supported by MCIU grant FPU17/02319.

Contents

1 Introduction

Strategies are meaningful for artificial intelligence, games, semantics of programming languages, automated reasoning, etc. Although their purposes and formalizations differ, they all honor the Greek etymology of the word, meaning *the office of a general*, who is in charge of the overall planning of the operations. In the modeling of concurrent systems using rewriting techniques, strategies are a useful resource to capture the global behavior of the intended models. Since rewriting consists of a successive, non-deterministic and somehow unrelated application of rules anywhere within a term, strategies have been studied in deep [34], and different definitions have been proposed [8, 20]. Strategies as a first-class object have been exploited in tools like Stratego [10], Tom [4], and the specification languages ELAN [7], and Maude [13].

Model checking [11] is a consolidated formal method, which still evolves in multiple directions. Its classical setting is transition systems, where the notion of strategy is naturally defined. This paper studies the satisfaction of temporal properties by models controlled by strategies and how it can be checked, and applies the method to the Maude strategy language by implementing a strategy-aware model checker. It is built as an extension of the existing Maude LTL model checker [17] for systems specified in rewriting logic, already applied to various interesting systems [6, 24, 29].

Strategies and model checking together have already been addressed in the literature, but with a different approach and objectives. In the context of multiplayer games, several logics have been proposed to *reason* about player strategies like ATL* [1] and *strategy logic* [28]. However, strategies are not provided as input but quantified in the formula, and they are not represented explicitly. Other logics take past actions into account to condition its requirements like mCTL* [22]. In our case, strategies are part of the model specification while the property logic remains unaltered. As well, strategies should not be confused with heuristics to guide the search in the model-checker algorithms [15].

After reviewing the model-checking framework and strategies as defined in the literature, this paper discusses model checking linear temporal properties on strategy-controlled systems and a model transformation is proposed to match the classical setting and allow using the standard algorithms. Following a short introduction to rewriting logic [27], Maude [12], and its strategy language [16], we propose a small-step operational semantics from which model checking is defined according to the previous approach. The strategy-aware LTL model checker we have implemented is then described and illustrated by an example. This document, the complete semantics of the Maude strategy language, and the model checker itself are all available at `http://maude.ucm.es/strategies`.

2 Preliminaries

2.1 Model checking

Model checking [11] is a well-established formal method to ensure or refute the correctness of a model according to a temporal specification. In the classical setting, models are based on transition systems, formally described by Kripke structures [21] $\mathcal{K} = (S, \to, I, AP, \ell)$ where

- S is the set of states,
- $(\to) \subseteq S \times S$ is a serial binary relation on S,
- $I \subseteq S$ is a finite set of initial states,
- AP is a finite set of *atomic propositions*, and
- $\ell : S \to \mathcal{P}(AP)$ labels each state with the propositions that hold on it.

In turn, the property is expressed by a formula φ in some temporal logic like CTL, CTL* or LTL, which describes the intended behavior in terms of the atomic properties $p, q, \ldots \in AP$ combined by different temporal operators. The model-checking problem, deciding whether the model satisfies the formula $\mathcal{K} \vDash \varphi$, is decidable for any of the previous logics, a decidable transition relation, and a finite S. However, for both CTL* and LTL, model checking is PSPACE-complete and the models of interest usually have a huge number of states. In various situations, the expectation of refuting correctness in reasonable time is good enough.

The actual model executions $\pi = (s_k)_{k=1}^{\infty}$ leave propositional traces $\ell(\pi) := (\ell(s_k))_{k=1}^{\infty}$, from which the satisfaction of the formula is decided.

$$\begin{array}{lccccccc} \pi & s_1 & \longrightarrow & s_2 & \longrightarrow \cdots \longrightarrow & s_n & \longrightarrow \cdots & \in S^\omega \\ \ell(\pi) & \{p\} & & \emptyset & \cdots & \{p,q\} & \cdots & \in \mathcal{P}(AP)^\omega \end{array}$$

Linear-time properties can always be characterized by a satisfaction relation $\ell(\pi) \vDash \varphi$ on propositional traces and an implicit universal quantification over all model paths π. Differently, branching-time properties combine universal and existential requirements on any state of the derivation, so that the execution should be seen as a tree.

2.2 Strategies

In rewriting systems, rules typically represent local transitions that are often not enough to describe complex computations. These intricacies are usually expressed at a higher level, describing how rules should be applied. This is the task of strategies, whose study goes back to the λ-calculus, as fixed criteria for selecting the next redex to be reduced [5]. Later, strategies were allowed to be aware of the derivation history and to be *explicit* [2, §11.5], expressed as programs that control the application of rules. We are interested in the latter kind of strategies, for which different abstract descriptions and practical representations have been proposed and implemented [8, 20].

Strategies are properly defined in the context of *abstract reduction systems* (ARS). An ARS [2] $\mathcal{A} = (S, \to)$ is a set of states S endowed with a binary relation $\to$. An element $(s, s') \in (\to)$ or $s \to s'$ is called a *reduction step*, and a finite or infinite sequence of connected reduction steps $s_0 \to s_1 \to \cdots \to s_n$ is a *derivation*. We denote by $\Gamma^\omega_{\mathcal{A}}$ the set of infinite derivations of $\mathcal{A}$ seen as words in S^ω,

$$\Gamma^\omega_{\mathcal{A}} \coloneqq \{s_0 s_1 \cdots s_n \cdots : s_k \in S \text{ and } s_k \to s_{k+1}, k \geq 0\},$$

$\Gamma^*_{\mathcal{A}}$ is the set of finite derivations as words in S^*, and $\Gamma_{\mathcal{A}} \coloneqq \Gamma^\omega_{\mathcal{A}} \cup \Gamma^*_{\mathcal{A}}$ the union of both in $S^\infty \coloneqq S^\omega \cup S^*$. Considering both finite and infinite derivations is tedious, but we are interested in modeling computations and proofs as well as reactive systems behavior, for which they are respectively relevant. We say that $\mathcal{A}$ is *finite* if S is finite, and $\mathcal{A}$ is *finitary* if for any $s \in S$ the states s' such that $s \to s'$ are finitely many.

Several definitions of strategies are reviewed in [8], but two general formalizations are specially discussed:

1. *Abstract* or *extensional strategies* are subsets of derivations $E \subseteq \Gamma_{\mathcal{A}}$, that is, languages in S^∞ whose words w are $\mathcal{A}$-derivations with $w_k \to w_{k+1}$.
2. *Intensional strategies* are defined as partial functions $\lambda : S^* \to \mathcal{P}(S \cup \{\top\})$ that decide the possible next steps according to the past of the derivation. They must satisfy that for all $s, s' \in S$ and $v \in S^*$, $s' \in \lambda(vs)$ must imply $s \to s'$. The symbol $\top$ indicates that the derivation can stop there. In case $\lambda(w) = \emptyset$, the derivation cannot stop or continue, so it is discarded. Hence, these strategies can attempt rewriting paths that may eventually fail.

Extensional strategies represent an abstract selection of ARS executions as a whole, while the more constructive intensional strategies determine the next reduction in each step. Unlike in [8], we have considered unlabeled transition systems to simplify the presentation. Since classical model checking only considers properties on the states, labels can be easily added without repercussions. Moreover, the definition of intensional strategies has been modified to include the $\top$ symbol. Otherwise, derivations could stop at any step, which is inconvenient in practice. These definitions fall into the class of *history aware strategies* of [34], and intensional strategies are similar to *non-deterministic strategies* in games, except that these may select the next player action instead of the next state.

▶ **Example 1.** Consider the ARS $(\{a, b\}, \{(a, a), (a, b), (b, a)\})$

A strategy allowing at most one stay in b is described extensionally as $\{a\}^*\{b, \varepsilon\}(\{a\}^* \cup \{a\}^\omega)$, and intensionally as $\lambda(v) = \{a, \top\}$ if v contains a b, and $\lambda(v) = \{a, b, \top\}$ otherwise.

An intensional strategy induces an extensional one

$$E(\lambda) \coloneqq \{w \in \Gamma_{\mathcal{A}} : w_i \in \lambda(w_0 \cdots w_{i-1}) \wedge (w \in S^\omega \vee \top \in \lambda(w))\}.$$

But the converse is not true, intensional strategies are less expressive than extensional ones. In the ARS of Example 1, the intensional strategy for $\{a\}^*$ has to be defined by $\lambda(v) = \{a, \top\}$ for any $v \in \{a\}^*$, since another a can always be added. And thus, the word a^ω would be included in $E(\lambda)$ by definition, so that $E(\lambda) \neq \{a\}^*$. Intuitively, intensional strategies decide on-the-fly while constructing the derivation, so they cannot decide on properties on the *infinity*. Languages recognized by automata with non-trivial acceptance conditions are examples of strategies that are necessarily extensional, but the more realistic devices or computations we are interested in modeling are very likely to be intensional. In any case, the extensional definition is simpler and will be useful.

Formally, intensional strategies are characterized as closed sets [8], which contain all words $w \in S^\omega$ whose finite prefixes are all prefixes of derivations within the strategy[1]. When discussing model checking, we will find an alternative characterization.

3 Strategy-aware model checking

Given a Kripke structure $\mathcal{K} = (S, \to, I, AP, \ell)$ and a strategy $E \subseteq \Gamma_\mathcal{K} := \Gamma_{(S,\to)} \cap IS^\infty$ starting from the initial states of $\mathcal{K}$, we want to give sense to model checking against a linear temporal formula φ and define the satisfaction relation $(\mathcal{K}, E) \vDash \varphi$. First, since temporal formulas are properly defined on infinite executions, we assimilate finite traces to infinite ones by repeating its last state forever. This is standard and fits with the idea of a finite machine that remains in its final state once stopped.

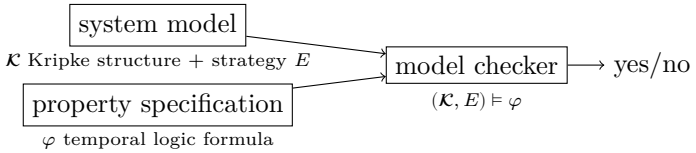

Figure 1 Model-checking procedure sketch.

Model checking a system controlled by a strategy against a linear-time property has an unavoidable and clear definition. As pointed in Section 2, the satisfaction of a linear property follows from a satisfaction relation $\rho \vDash \varphi$ on propositional traces $\rho \in \mathcal{P}(AP)^\omega$. Then,

▶ **Definition 2.** *Let φ be a linear formula, $(\mathcal{K}, E) \vDash \varphi$ if $\ell(\pi) \vDash \varphi$ for all $\pi \in E$.*

The set of allowed propositional traces $P = \{\, \rho \in \mathcal{P}(AP)^\omega : \rho \vDash \varphi \,\}$ completely defines the property, and the model-checking problem is the language containment problem $\ell(E) \subseteq P$, which is decidable and PSPACE-complete as long as $\ell(E)$ and P are ω-regular, like in the LTL case [30]. Although this is a clear starting point and it would provide an actual algorithm if strategies were given as Büchi automata, this is not usually the case since they are rather expressed in some intensional or syntactical form like the Maude strategy language described in Section 4.1. Thus, to effectively decide model checking with standard algorithms, we propose to encode the model controlled by strategies as another abstract reduction system. Any intensional strategy can be so encoded, but the actual procedure will depend on the strategy representation. In Section 4.2, we do it for the Maude strategy language by means of a small-step operational semantics.

▶ **Definition 3.** *Given an ARS $\mathcal{A} = (S, \to)$ and a strategy $E \subseteq \Gamma_\mathcal{A}$, the pair $(\mathcal{A}, E)$ is represented in the ARS $\mathcal{A}' = (X, R)$ with descent function $d : X \to S$ if*

$$d(\Gamma_{\mathcal{A}'} \cap J\, X^\infty \cap (X^\omega \cup X^*F)) = E$$

for some sets $J, F \subseteq X$ of initial and final states of the strategy.

The representation $\mathcal{A}'$ simulates the system $\mathcal{A}$ constrained by E, in the sense that the executions of $\mathcal{A}'$ are the traces selected by the strategy. Intuitively, the representation states include something else to guarantee

[1] In fact, X^ω can be given a *prefix* topology (and even a distance) such that *closed* can be understood in the topological sense and the points so described are limits.

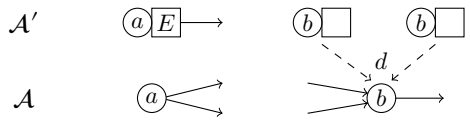

Figure 2 Strategy representation in an ARS (Definition 3).

that the strategy is respected, which can be stripped with the descent function d. The initial states for the strategy execution are the subset J, since other states in X may represent ongoing strategy executions. The final set F is only required to distinguish admitted finite traces from incomplete ones. Still, we do not want them for model checking: the trace extension of the first paragraph can be implemented in the above representation by adding self-loops in F. However, this cannot be done without care. For example, observe the following encoding for the strategy $\{a, ab\}$, where a and b are final.

The extended language is $\{a^\omega, ab^\omega\}$ according to our criterion. However, a loop in a will allow spurious derivations like aab^ω. This can be solved by adding an extra copy of the final states, like in the right figure. Now, we assume that the strategies are over infinite words and define the concept of model checking.

▶ **Proposition 4.** *Let $\mathcal{K} = (S, \to, I, AP, \ell)$ be a Kripke structure, $E \subseteq \Gamma_{\mathcal{K}}$ a strategy, and (X, R) a representation of $(\mathcal{A}, E)$ with descent function d and initial states J, $(\mathcal{K}, E) \models \varphi$ iff $\mathcal{K}' \models \varphi$ where $\mathcal{K}' = (X, R, J, AP, \ell \circ d)$.*

That is, model checking $\mathcal{K}$ controlled by E is model checking its representation $\mathcal{K}'$. The reason is that the propositional traces of any such $\mathcal{K}'$ are exactly those of E. This method can be applied to any intensional strategy, but model checking will only be decidable if the ARS representation is finite.

▶ **Proposition 5.** *A strategy $E \subseteq \Gamma_{\mathcal{A}}$ on a finitary ARS $\mathcal{A}$ can be represented in a finitary ARS iff E is intensional, i.e. there is an intensional strategy λ such that $E = E(\lambda)$. In that case, it can be represented in a finite ARS iff E is ω-regular.*

We can draw from this proposition that a strategy on a finite ARS must be intensional and ω-regular to match the classical model-checking framework and apply its algorithms, because otherwise it cannot be represented in a finite Kripke structure.

Notice that the discussion in this section is restricted to linear-time properties. The representation of Definition 3 is not adequate for branching-time properties, since branches need not be preserved while descending with d. A suitable model-checking definition for logics like CTL* passes by model checking a more restricted representation, any bisimulation of the ARS (S^+, R) where $v\,R\,vs$ if $v \in \lambda(v)$ and λ is an intensional strategy.

4 Maude and its strategy language

Maude is a specification language [13, 12] based on *rewriting logic* [27], a general framework for modeling concurrency proposed in 1992 by José Meseguer. Its specifications are organized in modules of different kinds:

1. *Functional* modules define *membership equational logic* theories, composed of an order-sorted signature Σ, equations E, and sort membership axioms to express that a term t belongs to a sort s. Equations and membership axioms can be conditional.

$$(\forall X) \qquad \begin{matrix} t = t' \\ t : s \end{matrix} \quad \text{if} \quad \bigwedge_i u_i = u_i' \ \wedge \ \bigwedge_j v_j : s_j$$

   For the specification to be executable, equations are oriented and functional modules must be confluent and terminating [12, §4.6]. Bidirectional relations, like commutativity, associativity and identity, are specified apart as *structural axioms*.

2. *System* modules specify rewriting theories $\mathcal{R} = (\Sigma, E, R)$ by adding rewriting rules R to a functional specification. Unlike equations, rewriting rules need be neither confluent nor terminating, so they are likely to express non-deterministic behavior.

$$(\forall X) \qquad t \Rightarrow t' \;\text{ if }\; \bigwedge_i u_i = u'_i \;\wedge\; \bigwedge_j v_j : s_j \;\wedge\; \bigwedge_k w_k \Rightarrow w'_k$$

However, rules are required to be coherent with equations and axioms [12, 27]. Conditional rules take a third type of conditions called rewriting conditions, that are satisfied if the term w_k can be rewritten to match w'_k.

The language syntax is a natural ASCII encoding of the mathematical notation above. Modules are a collection of declarations between `mod NAME is ... endm` (or `fmod`/`endfm` for functional modules). An operator $f : s_1 \times \cdots \times s_n \to s$ is defined as `op f : s1 .. sn -> s .` and structural axioms are inserted as attributes (`comm`, `assoc`, …) between brackets. Equations are introduced by the keyword `eq` and rules by `rl`, prefixed by `c` if conditional.

▶ **Example 6.** The classical problem of the dining philosophers [19] is specified in `PHILOSOPHER-DINNER` of Listing 1. A philosopher is represented as a triple describing both hands contents (a fork ψ or nothing `o`) and an identifier. Rules `left` and `right` allow them to take the forks at their sides if they are in the table. The `release` rule restores both forks to the table. Since a circular table is represented by a list, we adopt the convention that the fork between the last and first philosophers is on the right, ensure it by the equation, and add a second `left` rule to allow the first philosopher to take this fork.

Listing 1 Dining philosophers problem specified in Maude

```
fmod PHILOSOPHERS-TABLE is          *** functional module
  protecting NAT .                  *** import a module (natural n.)

  sorts Obj Phil List Table .       *** declare some sorts
  subsorts Obj Phil < List .        *** establish subsort relations

  op (_|_|_) : Obj Nat Obj -> Phil [ctor] .   *** constructor
  ops o ψ    : -> Obj [ctor] .
  op empty   : -> List [ctor] .
  op __      : List List -> List [ctor assoc id: empty] .
  op <_>     : List -> Table [ctor] .

  var L : List . var P : Phil .               *** declare a variable
  eq < ψ L P > = < L P ψ > .

  op initial : -> Table .
  eq initial = < (o | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) ψ > .
endfm

mod PHILOSOPHERS-DINNER is          *** system module
  protecting PHILOSOPHERS-TABLE .
  var Id : Nat .
  var X  : Obj .
  var L  : List .
  rl [left]    :       ψ (o | Id | X) => (ψ | Id | X) .
  rl [right]   :       (X | Id | o) ψ => (X | Id | ψ) .
  rl [left]    : < (o | Id | X) L ψ > => < (ψ | Id | X) L > .
  rl [release] :         (ψ | Id | ψ) => ψ (o | Id | o) ψ .
endm
```

Terms can be reduced equationally to a normal form using the `reduce` command. Rules can be applied using the `rewrite` and `frewrite` commands, which follow different fixed built-in strategies to choose which rule, which subterm, and which substitution to try first. Finally, `search` allows searching for any terms satisfying some given conditions in all possible rewriting paths from its argument [12, §5.4].

4.1 The strategy language

Effective manipulation of strategies requires expressing them in a convenient syntactical form. Since Maude is a reflective language, strategies have usually been expressed by explicitly applying rules at the metalevel. Because of its low-level, this is an awkward and error-prone method, so a strategy language was proposed [25, 16], conceived as an additional layer above functional and system specification. It has already been used in different contexts, among others [18, 26, 33, 35]. A strategy expression α can be executed to rewrite a term t using the `srewrite` t `using` α command. The language's basic component is rule application

$$ruleLabel\texttt{[}x_1\texttt{ <- }t_1\texttt{ , ... ,}x_n\texttt{ <- }t_n\texttt{]\{}\alpha_1\texttt{ , ... , }\alpha_m\texttt{\}}$$

where rules are selected by their labels, an optional initial substitution can instantiate both rule sides before matching, and strategies between curly brackets must be provided for rules with rewriting conditions. The other basic construct is the test `match` P `s.t.` C that checks if the subject term matches the pattern P and the condition C is satisfied. Tests come in three flavors: `match` that matches on top only, `xmatch` that can also match fragments by structural axioms, and `amatch` that matches anywhere within the term. These operators are combined by various constructs like

- concatenation $\alpha\texttt{;}\beta$, to execute α and then β on its results;
- alternation $\alpha\texttt{|}\beta$, which non-deterministically chooses α or β;
- iteration $\alpha\texttt{*}$, which executes α zero or more consecutive times;
- the constants `idle`, to do nothing, and `fail`, to discard the current execution path;
- the conditional α `?` β `:` γ, with condition α, positive β and negative γ branches, where γ is only executed if α does not produce any result. Otherwise, β is run after any of those.

Notice that concatenation, alternation, `idle`, and `fail` are the counterparts of regular expression constructors. Derived operators are available too, like $\alpha\texttt{+} \equiv \alpha\texttt{;}\alpha\texttt{*}$, `not(`$\alpha$`)` $\equiv \alpha$ `? fail : idle`, a normalization operator $\alpha\texttt{!} \equiv \alpha\texttt{* ; not(}\alpha\texttt{)}$, etc. To control *where* rules are applied, the language counts with the `top(`α`)` operator that restricts rule applications to the top symbol, and with a subterm rewriting operator

$$\texttt{matchrew }P(x_1,\dots,x_n)\texttt{ s.t. }C\texttt{ by }x_1\texttt{ using }\alpha_1\texttt{ , ... , }x_n\texttt{ using }\alpha_n$$

that matches the subject term against a pattern P, extracts the matched subterms, and rewrites them by means of substrategies α_1, ..., α_n in parallel. Like the `match` operator, three different versions of this operator exist. Finally, the language allows the declaration of named strategies with arguments in separate strategy modules `smod NAME is` ... `endsm`:

`strat` *slabel* [`:` *parameterTypes*] `@` *subjectType* `.`

Strategies are called *slabel*`(`t_1`,` ... `,`t_n`)` by citing its strategy name and providing the required arguments in a comma-separated list between parentheses, as a typical function invocation. They can be recursive and mutually recursive, and are defined in strategy modules with any number of (potentially conditional) definitions of the form

[`c`]**`sd`** *slabel*`(`*arguments*`)` `:=` *strategyExpression* [**`if`** C] `.`

where the strategy expression can use the variables in the left-hand side and in the condition. All strategy definitions whose left-hand side matches the call term are executed. An example is shown in Listing 2.

Listing 2 Dining philosophers strategy module

```
smod DINNER-STRAT is
  protecting PHILOSOPHERS-DINNER .

  strats free parity @ Table .

  sd free := all ? free : idle .

  sd parity := (release
```

```
    *** The even take the left ψ first
    | (amatchrew L s.t. ψ (o | Id | o) := L /\ 2 divides Id
      by L using left)
    | left[Id <- 0]
    *** The odd take the right ψ first
    | (amatchrew L s.t. (o | Id | o) ψ := L
      /\ not (2 divides Id) by L using right)
    *** When they already have one, they take the other ψ
    | (amatchrew L s.t. (ψ | Id | o) ψ := L by L using right)
    | (matchrew M s.t. < L (o | Id | ψ) L' > := M
      by M using left[Id <- Id])
    ) ? parity : idle .
endsm
```

The `DINNER-STRAT` module imports the module `PHILOSOPHERS-DINNER` (Example 6) that it will control, and defines two recursive strategies. The first one, `free`, is the recursive application of any rule (`all` allows the application of any available rule) until it cannot be further applied. It behaves like the built-in rewrite strategy. The other strategy, `parity`, forces a particular order in which to take the forks, which is alternative for evens and odds, that is, for neighbors. In Section 5.1, we will see properties that are satisfied with `parity` but not with `free`.

More details on the language syntax and semantics are provided in [16, 9] and the companion web page, where the complete implementation of the strategy language is available for download.

4.2 An operational semantics for model checking

This section provides the basis to model check systems specified in Maude and controlled by its strategy language. In this situation, it is essential to know which rewriting paths are allowed by the strategy. However, the semantics of a strategy expression applied to a term has usually been given as a set of result terms [16], so that the intermediate execution states remain unknown. A small-step operational semantics is required to observe them all. One has already been given in [9] by means of a rewrite theory transformation. Still and all, it specifies a global strategic search where multiple execution paths advance in parallel, in a way that they cannot be easily isolated. Based on these, we propose a nondeterministic operational semantics whose derivations clearly denote the full rewriting paths allowed by the strategy. Some technical details have been omitted, but are available in the appendices.

First, we define the *strategy execution states* on which the semantics is defined. Essentially, states consist of a term t in some rewrite theory $\mathcal{R} = (\Sigma, E, R)$ and a stack s of pending strategies and variable contexts, represented by substitutions $\sigma : X \to T_{\Sigma/E}(X)$. However, the subterm rewriting operator and rewriting conditions require executing strategies in nested contexts that are represented by the "subterm" and "rewc" symbols. In summary, execution states are generated by the following grammar where x is a variable, t is a term in $\mathcal{R}$, α is a strategy expression, and ε is the empty word.

$$s ::= \varepsilon \mid \sigma s \mid \alpha s$$
$$p ::= t \mid \text{subterm}(x : q, \dots, x : q; t) \mid \text{rewc}(p : q, \sigma, C, \alpha \cdots \alpha, \sigma, t, t; t)$$
$$q ::= p \mathbin{@} s$$

Any execution state can be projected to a term by the recursive function $\text{cterm}(t \mathbin{@} s) = t$, $\text{cterm}(\text{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} s) = t[x_1/\text{cterm}(q_1), \dots, x_n/\text{cterm}(q_n)]$ and finally $\text{cterm}(\text{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta, r, c; t) \mathbin{@} s) = t$. Moreover, every stack designates a variable context by looking at the top-most substitution, $\text{vctx}(\varepsilon) = \text{id}$, $\text{vctx}(\theta s) = \theta$ and $\text{vctx}(\alpha s) = \text{vctx}(s)$ where id is the identity function. States of the form $t \mathbin{@} \varepsilon$ are called *solutions* and represent successful strategy executions.

The semantics is defined by two distinct transition relations: control $\to_c$ and system $\to_s$ steps. The latter represents real transitions in the underlying system, i.e. rule rewrites, while the first does the auxiliary work to make strategies run. Among control transitions, some are devoted to handle alternation and iteration by taking non-deterministic choices,

$$t \mathbin{@} \alpha \,|\, \beta \to_c t \mathbin{@} \alpha \qquad t \mathbin{@} \alpha \,|\, \beta \to_c t \mathbin{@} \beta \qquad\qquad t \mathbin{@} \alpha* \to_c t \mathbin{@} \varepsilon \qquad t \mathbin{@} \alpha* \to_c t \mathbin{@} \alpha\alpha*$$

Concatenation is reduced by a rule $t \,@\, \alpha\,;\beta \to_c t \,@\, \alpha\beta$ that pushes α on top of β in the pending strategies stack. The rule for tests simply pops the operator on success

$$t \,@\, (\texttt{match}\ P\ \texttt{s.t.}\ C)\,\theta \to_c t \,@\, \theta \qquad \text{if there is } \sigma \text{ s.t. } \sigma(\theta(P)) = t \text{ and } \sigma(\theta(C)) \text{ holds}$$

where substitutions σ are extended to substitute variables within terms and conditions. The other test flavors have similar rules. There is no rule for the negative case: the execution path arrives to a deadlock state and will later be discarded. The positive case behaves like an `idle`, whose rule is $t \,@\, \texttt{idle} \to_c t \,@\, \varepsilon$. The semantics of conditionals is given by two rules

$$t \,@\, \alpha\ \texttt{?}\ \beta\ \texttt{:}\ \gamma \to_c t \,@\, \alpha\beta \qquad \frac{\text{the derivations from } t \,@\, \alpha\,\theta \text{ are finite and no solution is reached}}{t \,@\, (\alpha\ \texttt{?}\ \beta\ \texttt{:}\ \gamma)\,\theta \to_c t \,@\, \gamma\,\theta}$$

The first rule simply tries the condition followed by the positive branch: in case α does not produce any result, β will not be executed. Then, the second strategy must be triggered to run γ. Notice that the $\to_c$ rule is undecidable in general since the negative branch condition implies deciding whether the derivations from $t \,@\, \alpha\theta$ are all terminating.

Strategy calls are handled with rule instances of the form

$$t \,@\, sl(p_1, \dots, p_n)\,\theta \to_c t \,@\, \delta\sigma\theta \qquad \text{for any matching } \sigma \text{ of } (t_i)_{i=1}^n \text{ in } (p_i)_{i=1}^n \text{ s.t. } \sigma(C) \text{ holds}$$

for each strategy definition `csd` sl`(`t_1`, ..., `t_n`) :=` δ `if` C (or its unconditional version). When a strategy call finishes, its variable context is popped $t \,@\, \theta \to_c t \,@\, \varepsilon$.

The mission of the execution stack is holding pending strategies and also active call contexts. Only the top element determines the possible next steps, but the current variable context may be determined by a substitution buried by multiple strategies inside the stack. Rules have been defined for states with almost empty stacks, but they can be easily extended from the bottom as long as the variable context is preserved.

$$t \,@\, s = t \,@\, s\,\mathrm{id} \qquad \frac{t \,@\, s\,\theta \to_\bullet t' \,@\, s'\,\theta}{t \,@\, s\,s_0 \to_\bullet t \,@\, s's_0} \text{ if } \mathrm{vctx}(s_0) = \theta$$

where $\to_\bullet$ can be replaced by both $\to_s$ and $\to_c$.

The `matchrew` rewrites matched subterms independently via the subterm execution states,

$$\frac{q_i \to_s^c q_i'}{\mathrm{subterm}(\dots, x_i : q_i, \dots; t) \,@\, s \to_s^c \mathrm{subterm}(\dots, x_i : q_i', \dots; t) \,@\, s}$$

These structured states are created with the rule

$$\begin{aligned} t \,@\, (\texttt{matchrew}\ P(x_1, \dots, x_n)\ \texttt{s.t.}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1, \dots, x_n\ \texttt{using}\ \alpha_n)\,\theta \\ \to_c \mathrm{subterm}(x_1 : \sigma(x_1) \,@\, \alpha_1\,\rho, \dots, x_n : \sigma(x_n) \,@\, \alpha_n\,\rho, \dots) \,@\, \theta \end{aligned}$$

for any matching σ of $\theta(P)$ in t such that $\sigma(\theta(C))$ holds, and where $\rho(x) = \sigma(x)$ if $\sigma(x) \neq x$ and $\theta(x)$ otherwise. And they are resolved, once its subterms have arrived to solutions, with

$$\mathrm{subterm}(x_1 : t_1 \,@\, \varepsilon, \dots, x_n : t_n \,@\, \varepsilon; t) \,@\, s \to_c t[x_1/t_1, \dots, x_n/t_n] \,@\, s$$

Finally, system transitions $\to_s$ are generated by rule applications. The execution of a maybe conditional rule without rewriting fragments is

$$t \,@\, rl[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]\,\theta \to_s t[p/\sigma(\rho(r))] \,@\, \theta$$

if for a position p within t and for a rule $l \to r$, there is a matching σ such that $t_p = \sigma(\rho(l))$ and $\sigma(\rho(C))$ holds, where ρ is the initial substitution that maps x_i to $\theta(t_i)$. If the rule application is surrounded by the `top` modifier, it is only applied on top. For rules with rewriting conditions l `=>` r, substrategies must be provided between curly brackets and these must be used to rewrite its condition fragments. This is achieved using a structured

execution state similar to the subterm construct, where the nested state q executes the corresponding strategy in the rewriting fragment initial term.

$$t \,@\, rl[\ldots, x_i \leftarrow t_i, \ldots]\{\alpha_1, \ldots, \alpha_m\}\,\theta \to_c \mathrm{rewc}(t_r : \sigma(t_l) \,@\, \alpha_1\theta, \sigma, C, \alpha_2 \cdots \alpha_m, \theta, r, c; t)$$

for any rule rl with condition $C_0 \wedge t_l \Rightarrow t_r \wedge C$ where C_0 is an equational condition, and any matching substitution σ and matching context c such that $\sigma(C_0)$ holds. Like in the previous case, the rule is first instantiated with the given initial substitution. The right-hand side of the rule r is kept to do the actual rewriting once the conditions have been checked:

$$\mathrm{rewc}(p : t' \,@\, \varepsilon, \sigma, C_0, \varepsilon, \theta, r, c; t) \to_s c(\sigma'(r))$$

where the substitution σ' extends σ by matching t' against $\sigma(p)$ and satisfies the equational condition $\sigma'(C_0)$. If the remaining condition contains more rewriting fragments, these are tried one after another accumulating variable bindings:

$$\mathrm{rewc}(p : t' \,@\, \varepsilon, \sigma, C_0 \wedge t_l \Rightarrow t_r \wedge C, \alpha\vec{\alpha}, \theta, r, c; t) \to_c \mathrm{rewc}(t_r : \sigma'(t_l) \,@\, \alpha\,\theta, \sigma', C, \vec{\alpha}, \theta, r, c; t)$$

The rewc state follows the transitions of the inner state,

$$\frac{q \to_\bullet q'}{\mathrm{rewc}(p : q, \sigma, C, \vec{\alpha}, \theta, r, c; t) \to_c \mathrm{rewc}(p : q', \sigma, C, \vec{\alpha}, \theta, r, c; t)}$$

However, transitions inside this nested state are always control transitions in the outer one, because they are not applied to the subject term.

Finally, this semantics can be used to define the translation of the strategy expression α to an abstract strategy $E(\alpha)$ in $(T_{\Sigma/E}, \to^1_{\mathcal{R}})$ where $T_{\Sigma/E}$ is the initial term algebra and $\to^1_{\mathcal{R}}$ the one-step rewrite relation of the rewrite theory $\mathcal{R}$. The derived relation $\twoheadrightarrow = \to_c^* \circ \to_s$, a single system transition preceded with all the necessary strategic preparation, has the property that $q \twoheadrightarrow q'$ implies $\mathrm{cterm}(q) \to^1_{\mathcal{R}} \mathrm{cterm}(q')$.

▶ **Definition 7.** *For a strategy expression α and a set of initial states I, we define*

$$\begin{aligned} E(\alpha) :=& \{t\,\mathrm{cterm}(q_1)\cdots\mathrm{cterm}(q_n)\cdots : t \,@\, \alpha \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots, t \in I\} \\ & \cup\ \{t\,\mathrm{cterm}(q_1)\cdots\mathrm{cterm}(q_n) : t \,@\, \alpha \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \to_c^* t_n \,@\, \varepsilon, t \in I\} \end{aligned}$$

Observe that finite derivations end with solutions, perhaps modulo some control transitions. The strategy $E(\alpha)$ is intensional and can be encoded in the ARS $\mathcal{A}' = (\mathcal{XS}, \twoheadrightarrow, \{t \,@\, \alpha : t \in I\})$ with final states $F = \{q \in \mathcal{XS} : \exists\, t \in T_{\Sigma/E} \quad q \to_c^* t \,@\, \varepsilon\}$, and $d = \mathrm{cterm}$ the descent function. $\mathcal{XS}$, the set of execution states, is always an infinite set, but we can restrict to the reachable execution states from the initial ones. For model checking to be decidable, the ARS needs to be finite. Some sufficient conditions can be established:

▶ **Proposition 8.** *Given a term $t \in T_{\Sigma/E}$ and a strategy expression α, if the reachable terms from $t \,@\, \alpha$ are finitely many, and the recursive strategy calls are tail recursive and their call arguments only take a finite number of values, $\twoheadrightarrow$ is decidable and the set of reachable execution states is finite.*

Both premises are reasonable, since they bound the number of state and strategy combinations that may appear during execution. This also implies the decidability of the $\twoheadrightarrow$ relation, whose threats are the negative branch of the conditional combinator and rewriting conditions. We understand by *reachable terms* all the elements of $T_{\Sigma/E}$ that occur while executing the strategy, while rewriting both the state and the rewriting conditions of rules. It is easy to observe that expressions without iterations and recursive calls never produce an infinite number of execution states, but they are not usually interesting.

Often, this sufficient condition holds and is checked easily, like in the example strategies of Listing 2. In the `free` and `parity` definitions, all strategy calls are tail recursive and do not take parameters. The reachable terms from `initial` are finitely many since, even by unrestricted rewriting, only 3^3 *tables* are reachable, as shown by counting the possible positions of the forks.

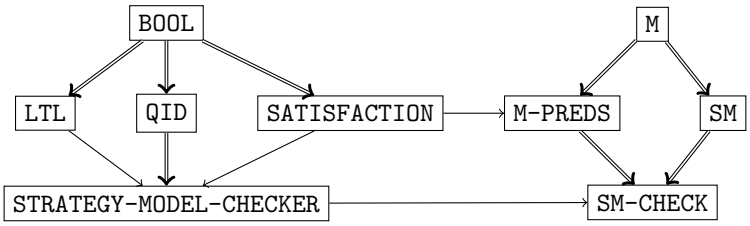

Figure 3 Structure of the strategy model checker modules.

5 The Maude strategy-aware model checker

Following the principles of Section 3 and the strategy language semantics, the new Maude strategy-aware model checker was programmed in C++ as an extension of the already existing explicit-state LTL model checker [17]. Both have a similar interface [12, §10] and are accessed using separate special operators declared in Maude itself.

The built-in modules of the model checker appear on the left of Figure 3. All but `STRATEGY-MODEL-CHECKER` are shared with the standard model checker. The right side of the figure shows the typical structure of the user specification of the model and properties. The user must:

1. Specify the model in a system module `M`, and define strategies to control `M` in a strategy module `SM`.
2. In a protecting[2] extension of `M`, say `M-PREDS`, choose the sort of the model states, making it a subsort of the `State` sort declared in `SATISFACTION`, declare atomic propositions as operators of type `Prop`, and define the satisfaction relation `|=` for all of them.

```
fmod SATISFACTION is
 protecting BOOL . sorts State Prop .
  op _|=_ : State Prop -> Bool .
endfm
```

```
mod M-PREDS is
  protecting M .
  including SATISFACTION .
  subsort Foo < State .
 op p : -> Prop .
 eq F:Foo |= p = ... .
endm
```

3. Declare a strategy module, say `SM-CHECK`, to combine the model `M` with the property specification in `M-PREDS` and the strategies `SM`. Import `STRATEGY-MODEL-CHECKER` too.

```
smod SM-CHECK is
  protecting M-PREDS .
  protecting SM .
  including STRATEGY-MODEL-CHECKER .
endsm
```

Once this is done, model checking is invoked using the operator

```
op modelCheck : State Formula Qid QidList
  ~> ModelCheckerResult [special (...)] .
```

which receives an initial state, an LTL formula as defined in the `LTL` module [12, §10], and a strategy identifier, which must correspond to a strategy without parameters defined in the module. The last argument is an optional list of *opaque* strategy names: when a strategy in this list is called, instead of the transitions occurring during

[2] Protecting means that it does not alter the signature, equations and rules of the types defined in `M`.

the strategy execution, the model checker will see direct transitions to its results. This optional feature produces traces that do not fit in the base `M` model, but allows model checking coarse-grain and fine-grain models with little changes.

The result is either `true` if the model satisfies the specification, or a counterexample, expressed as a cycle of rewriting steps and a path to it, if it does not. Additionally, the model checker optionally outputs an extended dump, from which graphical representations of the system automaton and counterexamples can be generated using an auxiliary program. The model checker, the auxiliary program, additional documentation, and various examples can be downloaded at `http://maude.ucm.es/strategies`.

The fundamentals of the strategy-aware model checker are given by the small-step operational semantics of Section 4.2 and the fundamentals of the original model checker. The model controlled by a strategy α from an initial term $t \in T_{\Sigma/E}$ can be encoded in the Kripke structure

$$\mathcal{K} = (\mathcal{XS}, \twoheadrightarrow, \{t \,@\, \alpha\}, AP_\Pi, L_\Pi \circ \mathrm{cterm}),$$

where atomic propositions are defined as in the standard model checker, for Π the signature of `M-PREDS` and D its set of equations,

$$AP_\Pi := \{\ \theta(p(x_1, \dots, x_n)) \mid p \in \Pi, \theta \text{ ground substitution }\}.$$

The labeling function $L_\Pi : T_{\Sigma/E} \to \mathcal{P}(AP_\Pi)$ is given by

$$L_\Pi([t]) := \{\theta(p(x_1, \dots, x_n)) \in AP_\Pi \mid (E \cup D) \vdash\ t \vDash \theta(p(x_1, \dots, x_n)) = true\}.$$

For the model checking to be decidable the conditions in [12, §10.3] for the standard one must be fulfilled, and additionally the reachable execution states must be finitely many.

5.1 Example: dining philosophers

In this section, we resume the dining philosophers problem (Example 6) to model check some properties with different strategies. Although the example is presented with three philosophers, it can be instantiated with many more. We already know that some unwanted situations may appear during the dinner: a philosopher may starve or, even worse, none of them could be able to eat. First, we express the collection of properties "the philosopher n eats" as atomic propositions and prepare the model checker input data.

Listing 3 Atomic proposition for the dining philosophers example

```
mod DINNER-PREDS is
  protecting PHILOSOPHERS-DINNER .    *** From Example 6 (Listing 1)
  including SATISFACTION .

  subsort Table < State .
  op eats : Nat -> Prop [ctor] .

  var  Id  : Nat .
  vars L M : List .

  eq < L (ψ | Id | ψ) M > |= eats(Id) = true .
  eq < L > |= eats(Id) = false [owise] .        *** otherwise
endm
```

Then, we put together and include the built-in model checker module.

```
smod DINNER-CHECK is
  protecting DINNER-PREDS .
  protecting DINNER-STRAT .        *** From Listing 2
  including STRATEGY-MODEL-CHECKER .
  including MODEL-CHECKER .        *** the standard model checker too
endsm
```

Now, we can check that the property $\Box \Diamond (eats(0) \vee eats(1) \vee eats(2))$ (no *deadlock*) does not hold for free rewriting, either using the standard model checking or the strategy-aware one with the `free` strategy, but it does hold when using the `parity` strategy.

```
Maude> red modelCheck(initial,
             [] <> (eats(0) \/ eats(1) \/ eats(2))) .
ModelCheckerSymbol: Examined 4 system states.
rewrites: 43 in 4ms cpu (0ms real) (10750 rewrites/second)
result ModelCheckResult: counterexample(
  {< (o | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) ψ >,'left}
  {< (ψ | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) >, 'left}
  {< (ψ | 0 | o) (ψ | 1 | o) ψ (o | 2 | o) >,'left},
  {< (ψ | 0 | o) (ψ | 1 | o) (ψ | 2 | o) >,deadlock})

Maude> red modelCheck(initial,
             [] <> (eats(0) \/ eats(1) \/ eats(2)), 'parity) .
StrategyModelCheckerSymbol: Examined 12 system states.
rewrites: 159 in 0ms cpu (2ms real) (~ rewrites/second)
result Bool: true
```

However, `parity` does not guarantee that all of them eat, because the property $\Diamond\, eats(0)$ does not hold. The model checker presents a counterexample, where the philosopher number two takes both forks and releases them in a loop, while the rest keep inactive:

```
Maude> red modelCheck(initial, <> eats(0), 'parity) .
rewrites: 55 in 0ms cpu (0ms real) (~ rewrites/second)
result ModelCheckResult: counterexample(nil,
    {< (o | 0 | o) ψ (o | 1 | o) ψ (o | 2 | o) ψ >,'left}
    {< (o | 0 | o) ψ (o | 1 | o) (ψ | 2 | o) ψ >,'right}
    {< (o | 0 | o) ψ (o | 1 | o) (ψ | 2 | ψ) >,'release})
```

In order to ensure that all of them eat, the strategy should act as a referee. A succinct and direct solution is fixing turns, like in the following strategy:

```
sd turns(K, N) := left[Id <- K] ; right[Id <- K] ; release ;
            turns(s(K) rem N, N) .
sd turns := turns(0, 3) .
```

If the number of diners is greater, a more parallel version can be written allowing n div 2 philosophers to eat at the same time. By model checking with `turns`, we obtain

```
Maude> red modelCheck(initial,
  [] (<> eats(0) /\ <> eats(1) /\ <> eats(2)), 'turns) .
rewrites: 131 in 0ms cpu (1ms real) (~ rewrites/second)
result Bool: true
```

5.2 Implementation notes

We have programmed the new model checker in C++ as part of a modified version of the Maude interpreter that includes full strategy language support. The implementation reuses both the existing explicit-state LTL model checker and the existing infrastructure for strategic execution [16], which we have completed to support strategy modules and the `matchrew` operator. This infrastructure is based on a collection of *tasks*, which reflect continuations and call frames, and *processes* that are in charge of finding matches, applying rules, etc. The already existing model checker follows the automata-theoretic approach [11, §4] based on testing the emptiness of the language recognized by the synchronous product of the model and the negation of the linear property as Büchi automata. The model automaton is built specifically for the system controlled by the strategy, while the LTL to Büchi automaton translation and the nested depth-first algorithm are reused.

Each model state corresponds to a strategy execution state in the proposed semantics, and stores some identifying information and a list of processes from which successors are obtained on-the-fly when requested. Control operations $\to_c$ are handled as in usual execution, but rule rewrites trigger the commitment of a new state. Different techniques are used to detect cycles and already visited states in order to reuse previous work.

6 Conclusions and future work

Strategies and languages to express them are useful resources to specify restrictions and global control in rewriting systems, following the *separation of concerns* principle. This approach has already been used to specify deduction procedures, semantics of programming languages, chemical and biological processes, ... Such models need to be formally verified and analyzed. In this paper, we show that model checking has a natural definition in this context, and we tell how to effectively model check specifications for the Maude strategy language, by means of a small-step operational semantics that emphasizes rewriting sequences. This procedure can be applied to other strategy representations. A model checker for systems specified in Maude and controlled by its strategy language has been implemented in C++ as an extension of the existing explicit-state LTL model checker, which can be downloaded from `http://maude.ucm.es/strategies`. It has already been tested with classical examples and its performance is comparable to the original model checker.

The ongoing work comprises the study of branching-time properties and other theoretical aspects, as well as the development of examples to exploit and test the performance of the tool. The model checker can also be improved by providing clearer counterexamples with more information on the strategy execution, and updating the inherited LTL-to-automaton algorithm to more recent and efficient proposals [3, 23].

A Proofs

A.3 Strategy-aware model checking

Throughout this section, let $\mathcal{A} = (S, \to)$ be an abstract reduction system (ARS), $\mathcal{K} = (S, \to, I, AP, \ell)$ a Kripke structure, and E an extensional strategy in $\mathcal{A}$.

▶ **Proposition 4.** *Let E be a strategy on $\mathcal{A} = (S, \to)$ and (X, R) a representation of $(\mathcal{A}, E)$ with descent function d and initial states J, $(\mathcal{K}, E) \vDash \varphi$ iff $\mathcal{K}' \vDash \varphi$ where $\mathcal{K}' = (X, R, J, AP, \ell \circ d)$.*

Proof. The proof is a straightforward equivalence:

$$\mathcal{K}' \vDash \varphi \overset{\text{(linear-time)}}{\iff} (\ell \circ d)(\pi) \vDash \varphi \quad \forall \pi \in \Gamma^\omega_{\mathcal{K}'} \iff \ell(\pi') \vDash \varphi \quad \forall \pi' \in E \overset{\text{(Definition 2)}}{\iff} (K, E) \vDash \varphi$$

where the equivalence in the middle holds because (X, R) is a representation of $\mathcal{A}$ and so $d(\Gamma^\omega_{\mathcal{K}'}) = E$ where $\Gamma^\omega_{\mathcal{K}'} = \Gamma^\omega_{(X,R)} \cap J\, X^\omega$. ◀

▶ **Lemma 1.** *The strategy E in the finitary $\mathcal{A}$ is intensional iff E can be represented in a finitary ARS.*

Proof. The proof of the rightward implication is the representation $\mathcal{A}' = (S^+, R)$ where $v\, R\, ws$ iff $v = w$ and $s \in \lambda(v)$ for all $v \in S^+$ and $w \in S^*$, with descent function is $d(vs) = s$ for all $v \in S^*$, initial states S, and final states are $F = \{v \in S^+ : \top \in \lambda(v)\}$. To be a representation, $d(\Gamma^\omega_{\mathcal{A}'} \cup \{v_1 \cdots v_n \in \Gamma^*_{\mathcal{A}'} : \top \in \lambda(v_n)\}) = E(\lambda)$ must hold, but it does since all the derivations in $\mathcal{A}'$ have the form $v \in (S^+)^\infty$ with $v_n = w_0 \cdots w_n$ for some word $w \in \Gamma_{\mathcal{A}}$ because of the definition of R, which additionally implies that $w_n \in \lambda(w_0 \cdots w_{n-1})$. This finally yields the definition of $E(\lambda)$.

For the leftwards direction, define $\lambda : S^* \to \mathcal{P}(S \cup \{\top\})$ as follows

$$\lambda(v) = \{s \in S : \exists\, w \in X^\infty\ vsw \in E\} \cup \{\top : v \in E\} \quad \text{for all } v \in S^*. \tag{1}$$

This is an intensional strategy, since for all $s' \in \lambda(vs)$ the relation $s \to s'$ holds as there must be an $w \in S^\infty$ such that $vss'w \in E \subseteq \Gamma_{\mathcal{A}}$ by (1). Notice that the definition did not use the premise and the property $E \subseteq E(\lambda)$ does not require it either. It holds because for all $w \in E$ and $0 < n < |w|$, $w_n \in \lambda(w_0 \cdots w_{n-1})$ since $w_0 \cdots w_{n-1} w_n w^{n+1} \in E$. We need to prove the opposite inclusion to conclude $E(\lambda) = E$, so that E is intensional.

The inclusion $E(\lambda) \subseteq E$ requires the fact that E can be represented in a ARS, say $\mathcal{A}' = (X, R)$ with initial states J, final states F, and descent function $d : X \to S$. Take any $w \in E(\lambda)$. If w is finite, then $\lambda(w) = \top$ and this implies $w \in E$ by (1). If w is infinite, we consider the tree of all $\pi \in \Gamma_{A'}$ such that $d(\pi)$ is a prefix of w. The tree is infinite given that for all $n \geq 0$ there is a path π such that $d(\pi) = w_0 \cdots w_n$. In effect, $w_n \in \lambda(w_0 \cdots w_{n-1})$ because $w \in E(\lambda)$, and by (1), there is a $v \in S^\omega$ such that $w_0 \cdots w_n v \in E$. Considering that the infinite words part of E is $d(\Gamma^\omega_{\mathcal{A}'} \cap J\, X^\omega)$, there is a $\rho \in X^\omega$ such that $d(\rho) = w_0 \cdots w_n v$. The path $\pi = \rho_0 \cdots \rho_n$ satisfies $d(\pi) = w_0 \cdots w_n$ and is in the tree, as we wanted. Since the ARS is finitary, the tree is finitary, and by Kőnig's lemma there is an infinite path π inside it. This path satisfies $d(\pi) = w$ because the images by d of all its finite prefixes are prefixes of w. Then $w \in d(\Gamma^\omega_{\mathcal{A}'} \cap J\, X^\omega) \subseteq E$. ◀

For the rest of the section, we will assume that all words in E are infinite. This simplifies the exposition and is enough for model checking purposes. The same (or simpler) proofs can be done for the finite words part of a strategy.

▶ **Lemma 2.** *If E is intensional and ω-regular, it is recognized by a Büchi automaton with trivial acceptance conditions.*

Proof. Since E is intensional, there is a partial function $\lambda : S^* \to \mathcal{P}(S \cup \{\bot\})$ such that $E = E(\lambda)$. And being E regular, there is an automaton $B = (Q, S, \delta, Q_0, F)$ such that $E = L(B)$. We claim that $B' = (Q, S, \delta, Q_0, Q)$ satisfies $L(B') = E$. Without loss of generality, we assume that all states in Q are reachable from Q_0 and that the language from each state is non-empty. Otherwise, we can simply remove these states.

It is clear that $E = L(B) \subseteq L(B')$ because more runs are accepted. To prove $L(B') \subseteq L(B) = E(\lambda)$, let $w \in L(B')$. There must be a trivially accepted run $\pi \in Q^\omega$ for the word in B'. For all $n \geq 0$, π_n is a state in B whose language is non-empty, so there is a run $\rho \in Q^\omega$ such that $\pi_0 \cdots \pi_n\, \rho$ is accepted in B. Then, a word $w_0 \cdots w_n v \in E = E(\lambda)$ for some $v \in S^\omega$ must be accepted by the run. Thus, $w_k \in \lambda(w_0 \cdots w_{k-1})$ for all $1 \leq k \leq n$ by definition of $E(\lambda)$. As this is true for all $n \geq 0$, then $w_k \in \lambda(w_0 \cdots w_{k-1})$ for all $k \geq 0$ and $w \in E(\lambda)$. ◀

▶ **Lemma 3.** *The strategy E can be represented in a finite ARS iff E is intensional and ω-regular.*

Proof. Notice that E must be defined in a finite alphabet S to be an ω-regular language. We already know by Lemma 1 that E can be represented in an ARS iff E is intensional. For the $\Rightarrow$ implication, assume that E can be represented in a finite ARS $\mathcal{A}' = (X, R, J)$ with descent function d. We will construct a Büchi automaton imitating $\mathcal{A}'$. Let such automaton be $B = (\{init\} \cup X, S, \delta, \{init\}, \{init\} \cup X)$ where $\delta(init, s) = \{x \in J : d(x) = s\}$ and $\delta(x, s) = \{y \in X : xRy \wedge d(y) = s\}$. The language recognized by B must be exactly E. First, it is clear that any run in B has the form $init\, \pi$ with $\pi \in X^\omega$ since $init$ is the only initial state and no transition returns to it. Suppose $w \in S^\omega$ is a word recognized by this run: $d(\pi_0) = w_0$ since $\pi_0 \in \delta(init, w_0)$, and as $\pi_{k+1} \in \delta(\pi_k, w_{k+1})$ and by the definition of δ, $\pi_k\, R\, \pi_{k+1}$ and $d(\pi_{k+1}) = w_{k+1}$. In conclusion, π is a path in $\mathcal{A}'$ and $d(\pi) = w$. Then, it is easy to prove:

- $L(B) \subseteq E$. For $w \in L(B)$ there is a run $init\, \pi$ such that π is a derivation in $\mathcal{A}'$ and $d(\pi) = w$. Then $w \in E$ since $d(\Gamma^\omega_{\mathcal{A}'}) = E$.
- $E \subseteq L(B)$. Given $w \in E$, there is a derivation $\pi \in X^\omega$ such that $d(\pi) = w$ because $d(\Gamma^\omega_{\mathcal{A}'}) = E$. The word $init\, \pi$ is a valid run in B, and it is accepting because the Büchi conditions are trivial. So $w \in L(B)$.

For the rightward implication, being E ω-regular and intensional, there is an automaton $B = (Q, S, \delta, Q_0, Q)$ that accepts all runs, according to Lemma 2. We can represent the strategy in the finite ARS $\mathcal{A}' = (S \times Q, R)$ with initial states $S \times Q_0$ and where the descent function d is the projection on the first component. The transition relation is defined by $(s, q)\, R\, (s', q')$ if $q' \in \delta(q, s)$. We must prove $T := d(\Gamma^\omega_{\mathcal{A}'} \cap (S \times Q_0)(S \times Q)^\omega) \stackrel{?}{=} E = L(B)$:

- $T \subseteq E$. Take a run $(s_n, q_n)_{n \geq 0}$ in $\mathcal{A}'$. The derivation $(q_n)_{n \geq 0}$ satisfies $q_{n+1} \in \delta(q_n, s_n)$ by definition of R, so $(q_n)_{n \geq 0}$ is a run for $(s_n)_{n \geq 0}$ in B. Since B accepts all runs and $L(B) = E$, $d((s_n, q_n)_{n \geq 0}) = (s_n)_{n \geq 0} \in E$.
- $E \subseteq T$. Take any word $w \in E$. Since B accepts w, there must exist a run $\pi \in Q^\omega$ for w. Then, the combined $(w_n, \pi_n)_{n \geq 0}$ is a path in $\mathcal{A}'$ because $w_0 \in I$, $\pi_0 \in Q_0$ and $\pi_{n+1} \in \delta(\pi_n, w_n)$. So $w = d((w_n, \pi_n)_{n \geq 0})$ is in T.

◀

▶ **Proposition 5.** *A strategy $E \subseteq \Gamma_{\mathcal{A}}$ in a finitary ARS $\mathcal{A}$ can be represented in a finitary ARS iff E is intensional, i.e. there is an intensional strategy λ such that $E = E(\lambda)$. In that case, it can be represented in a finite ARS iff E is ω-regular.*

Proof. It is the combination of Lemma 1 and Lemma 3. ◀

A.4 Maude and its strategy language

A.4.2 An operational semantics for model checking

▶ **Lemma 6.** *The ARS $(\mathcal{XS}, \twoheadrightarrow)$ with descent function* cterm*, initial states $J = \{t\,@\,\alpha : t \in I\}$ and final states $F = \{q \in \mathcal{XS} : q \to_c^* \mathrm{cterm}(q)\,@\,\varepsilon\}$ is a representation of the strategy $E(\alpha)$ on $(T_{\Sigma/E}, \to_{\mathcal{R}}^1)$ where $I = \{t \in T_{\Sigma/E} : tw \in E(\alpha), w \in T^\infty_{\Sigma/E}\}$.*

Proof. Let $\mathcal{A}$ be $(T_{\Sigma/E}, \to_{\mathcal{R}}^1)$ and $\mathcal{A}'$ the proposed $E(\alpha)$ representation. To see that $\mathcal{A}'$ is a genuine representation according to Definition 3, we must prove $\mathrm{cterm}(\Gamma^\infty_{\mathcal{A}'} \cap J\mathcal{XS}^\infty \cap (\mathcal{XS}^\omega \cup \mathcal{XS}^* F)) = E(\alpha)$. This is straightforward, since the argument of cterm in the left-hand side is

$$\begin{aligned}\Gamma^\infty_{\mathcal{A}'} \cap J\mathcal{XS}^\infty \cap (\mathcal{XS}^\omega \cup \mathcal{XS}^* F) &= (\Gamma^\infty_{\mathcal{A}'} \cap J\mathcal{XS}^\infty \cap \mathcal{XS}^\omega) \cup (\Gamma^\infty_{\mathcal{A}'} \cap J\mathcal{XS}^\infty \cap \mathcal{XS}^* F)) \\ &= \{(t\,@\,\alpha)\,q_1 \cdots q_n \cdots \;:\; t\,@\,\alpha \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots, t \in I\} \\ &\quad \cup \{(t\,@\,\alpha)\,q_1 \cdots q_n \;:\; t\,@\,\alpha \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \in F, t \in I\}\end{aligned}$$

because these are all the derivations of $\mathcal{A}'$ with initial states of the form $t\,@\,\alpha$ for $t \in I$. Then, we take the image of the set by cterm, which convert $(t\,@\,\alpha)\,q_1 \cdots q_n$ in $t\,\mathrm{cterm}(q_1) \cdots \mathrm{cterm}(q_n)$. The definition of $E(\alpha)$ is directly obtained, provided the final states $q_n \in F$ satisfy $q_n \to_c^* t_n\,@\,\varepsilon$ for some term t_n by definition of F. ◀

▶ **Definition 7.** *The following strategy expressions* only contain tail recursive calls*:*

- *`idle`, `fail`, tests, and strategy call expressions.*
- *$\alpha \,|\, \beta$ if α and β only contain tail recursive calls.*
- *$\alpha\,;\beta$ if α does not contain recursive calls and β only contains tail recursive calls.*
- *α `?` β `:` γ if α does not contain recursive calls, and β and γ only contain tail recursive calls.*
- *Subterm rewriting and rule application expressions if all its substrategies only contain tail recursive calls.*

▶ **Proposition 8.** *Given a term $t \in T_{\Sigma/E}$ and a strategy expression α, if the reachable terms from $t\,@\,\alpha$ are finitely many, and the recursive calls are tail recursive and only take a finite number of argument values, $\twoheadrightarrow$ is decidable and the set of reachable execution states is finite.*

Proof. This follows from Proposition 17. The idea is that the semantic rules always make the stack decrease but in two cases: the iteration and strategy calls. For these cases, we inductively count a finite number of strategy states. ◀

B Complete operational semantics

This is a more detailed description of the operational semantics for the Maude strategy language, extracted from [32] where additional discussion and comparison are addressed. The following notation will be used:

- $\mathrm{Strat}_{\mathcal{R}}$ is the set of strategy expression for a fixed strategy module.
- $\mathsf{VEnv}_{\mathcal{R}}$ is the set of variable environments (substitution) in the rewriting theory $\mathcal{R}$.
- $\mathsf{mcheck}(P, t, C, \theta)$ is the set of all substitutions σ that make P match on t and satisfy the condition C, while preserving the bindings of θ.
- vmatch is the same as the previous function but for many patterns and as many terms.
- $\mathsf{ruleApply}(t, rlabel, \sigma)$ is the set of all one-step rewrites from t using any rule with label $rlabel$, and applying the initial substitution σ to the rule before matching .

First, as we did in the main matter of the paper, we present the more complex execution states that gather the underlying state and the strategy execution state:

▶ **Definition 9.** *A* context stack *is a word in* $\mathrm{Stack} = (\mathrm{Strat}_{\mathcal{R}} \cup \mathsf{VEnv}_{\mathcal{R}})^*$*. An* execution state $\mathcal{XS}_{\mathcal{R}}$ *is:*

1. *A pair* $t \mathbin{@} s$ *composed of a term* $t \in T_\Sigma(X)$ *and a context stack* $s \in \mathrm{Stack}$.
2. *A collection of the form*

$$\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} s$$

where $x_1, \dots, x_n \in X$, $q_1, \dots, q_n \in \mathcal{XS}_\mathcal{R}$, $t \in T_\Sigma(X)$ *and* $s \in \mathrm{Stack}$.
3. *A collection of the form*

$$\mathrm{rewc}(p : q, \sigma, C, \bar{\alpha}, \theta_s, r, c; t) \mathbin{@} s$$

where $p, t, r \in T_\Sigma(X)$, $q \in \mathcal{XS}_\mathcal{R}$, $\bar{\alpha} \in \mathrm{Strat}^*_\mathcal{R}$, *a context* c, $\sigma, \theta_s \in \mathsf{VEnv}_\mathcal{R}$, C *a rule condition, and* $s \in \mathrm{Stack}$.

The stack will be used to store the pending strategies and the nested variable contexts. Item 2 represents parallel `matchrew` executions, and item 3 condition fragment rewriting when applying rules. States of the form $t \mathbin{@} \varepsilon$ are called *solutions*. All these states host information both about the subject term being rewritten and the progress of the strategy execution. In particular, they are uniquely associated to a term:

▶ **Definition 10.** *The* current term *of an execution state* $\mathrm{cterm} : \mathcal{XS}_\mathcal{R} \to T_\Sigma(X)$ *is*
1. $\mathrm{cterm}(t \mathbin{@} s) = t$.
2. $\mathrm{cterm}(\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} s) = t[x_1/\mathrm{cterm}(q_1), \dots, x_n/\mathrm{cterm}(q_n)]$.
3. $\mathrm{cterm}(\mathrm{rewc}(p : q, \sigma, C, \bar{\alpha}, \theta_s, r, c; t) \mathbin{@} s) = t$.

▶ **Definition 11.** *The* current variable context *of a context stack* $\mathrm{vctx} : \mathrm{Stack} \to \mathsf{VEnv}_\mathcal{R}$ *is*

$$\mathrm{vctx}(\varepsilon) = \mathrm{id} \qquad \mathrm{vctx}(\alpha\, s) = \mathrm{vctx}(s) \qquad \mathrm{vctx}(\sigma\, s) = \sigma$$

for $\alpha \in \mathrm{Strat}_\mathcal{R}$ *and* $\sigma \in \mathsf{VEnv}_\mathcal{R}$.

Two small-step relations are defined for two distinct levels: system behavior (rewriting with rules) and strategy control (auxiliary tasks to manage strategy execution). In the following, $\to_s$ is the system relation and $\to_c$ the control relation. When the distinction is not necessary, we use the union of both $\to_{s,c} = \to_s \cup \to_c$. Another relation $\twoheadrightarrow = \to_c^* \circ \to_s$ includes a single rule rewriting and all the previous work needed to apply it. This relation is particularly meaningful because whenever $q_1 \twoheadrightarrow q_2 \twoheadrightarrow \dots \twoheadrightarrow q_n \twoheadrightarrow \dots$ the rewriting system controlled by the strategy evolves

$$\mathrm{cterm}(q_1) \to_\mathcal{R}^1 \mathrm{cterm}(q_2) \to_\mathcal{R}^1 \dots \to_\mathcal{R}^1 \mathrm{cterm}(q_n) \to_\mathcal{R}^1 \dots$$

The operational semantics is given in Figures 4 and 5. There, $\alpha, \beta, \gamma \in \mathrm{Strat}_\mathcal{R}$, $s \in \mathrm{Stack}$, C, C' are rule conditions and C_0 an equational condition. $\theta \in \mathsf{VEnv}_\mathcal{R}$ always refer to $\mathrm{vctx}(s)$. The rules for the `a` and `x` variants of `match` and `matchrew` are omitted as they are too similar. Only note that in the first `matchrew` rule, the context c must be applied to the last entry of the subterm state, $c(\sigma_{\{x_1,\dots,x_n\}}(P))$.

Both $\to_c$ and $\to_s$ are not deterministic: they may decide between different alternatives and lose solutions on each step. Executions can be seen in different ways: as finite or infinite *sequences* of states or derivations like

$$t \mathbin{@} (rl_1 \,|\, rl_2)\, s \to_c t \mathbin{@} rl_1\, s \to_s t_1 \mathbin{@} s$$

Executions may get stuck or arrive to a solution $t \mathbin{@} \varepsilon$. Strategy execution is non-deterministic and the previous derivation is only one of the possibilities. When considering all of them, we refer to an *execution tree*

$$t \mathbin{@} (rl_1 \,|\, rl_2)\, s \;\begin{cases} \xrightarrow{\mathrm{c}} t \mathbin{@} rl_1\, s \xrightarrow{\mathrm{s}} t_1 \mathbin{@} s \\ \xrightarrow{\mathrm{c}} t \mathbin{@} rl_2\, s \xrightarrow{\mathrm{s}} t_2 \mathbin{@} s \end{cases}$$

But there is another tree in the picture: the *proof tree* for a single step of the semantics. For example, when rewriting subterms

$$\begin{array}{rcl} t\,@\,(\alpha\,;\beta)\,s & \to_c & t\,@\,\alpha\beta\,s \\ t\,@\,(\alpha\,|\,\beta)\,s & \to_c & t\,@\,\alpha\,s \\ t\,@\,\alpha^*\,s & \to_c & t\,@\,s \\ t\,@\,(\alpha\ \texttt{?}\ \beta\ \texttt{:}\ \gamma)\,s & \to_c & t\,@\,\alpha\beta\,s \end{array} \qquad \begin{array}{rcl} t\,@\,\theta\,s & \to_c & t\,@\,s \\ t\,@\,(\alpha\,|\,\beta)\,s & \to_c & t\,@\,\beta\,s \\ t\,@\,\alpha^*\,s & \to_c & t\,@\,\alpha\alpha^*\,s \\ t\,@\,\texttt{idle}\,s & \to_c & t\,@\,s \end{array}$$

$$t\,@\,(\texttt{match}\ P\ \texttt{s.t}\ C)\,s \to_c t\,@\,s \qquad \text{if } \mathsf{mcheck}(P,t,C,\theta_{-PC}) \neq \emptyset$$

$$t\,@\,sl(t_1,\ldots,t_n)\,s \to_c t\,@\,\delta_{sl,i}\,\sigma\,s \qquad \text{if } \sigma \in \mathsf{vmatch}(sdp_{sl,i},(\theta(t_1),\ldots,\theta(t_n)),C_{sl,i})$$

$$[\text{else}]\ \frac{}{t\,@\,\alpha\ \texttt{?}\ \beta\ \texttt{:}\ \gamma\,s \to_c t\,@\,\gamma\,s}\ \text{if the derivation tree for } t\,@\,\alpha\,\theta \text{ is finite without solutions}$$

Rewriting of subterms

$$\begin{array}{l} t\,@\,(\texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \ldots,x_n\ \texttt{using}\ \alpha_n)\,s \\ \quad \to_c \mathrm{subterm}(x_1 : \sigma(x_1)\,@\,\alpha_1\,\sigma,\ldots,x_n : \sigma(x_n)\,@\,\alpha_n\,\sigma;\sigma_{-\{x_1,\ldots,x_n\}}(P))\,@\,s \\ \qquad \text{if } \sigma \in \mathsf{mcheck}(P,t,C,\theta_{-PC}) \end{array}$$

$$\mathrm{subterm}(x_1 : t_1\,@\,\varepsilon,\ldots,x_n : t_n\,@\,\varepsilon;t)\,@\,s \to_c t[x_1/t_1,\ldots,x_n/t_n]\,@\,s$$

$$[\mathrm{prl}_c]\ \frac{q_i \to_c q_i'}{\mathrm{subterm}(\ldots,x_i : q_i,\ldots;t)\,@\,s \to_c \mathrm{subterm}(\ldots,x_i : q_i',\ldots;t)\,@\,s}$$

Rewriting conditions

$$[\mathrm{rewc}_c]\ \frac{q \to_c q'}{\mathrm{rewc}(p : q,\sigma,C,\bar{\alpha},\theta_s,r,c;t)\,@\,s \to_c \mathrm{rewc}(p : q',\sigma,C,\bar{\alpha},\theta_s,r,c;t)\,@\,s}$$

$$[\mathrm{rewc}_s]\ \frac{q \to_s q'}{\mathrm{rewc}(p : q,\sigma,C,\bar{\alpha},\theta_s,r,c;t)\,@\,s \to_c \mathrm{rewc}(p : q',\sigma,C,\bar{\alpha},\theta_s,r,c;t)\,@\,s}$$

$$\mathrm{rewc}(p : t'\,@\,\varepsilon,\sigma,C_0 \wedge l \texttt{ => } p' \wedge C,\alpha\bar{\alpha},\theta_s,r,c;t)\,@\,s \to_c \mathrm{rewc}(p' : \sigma'(l)\,@\,\alpha\,\theta_s,\sigma',C,\bar{\alpha},\theta_s,r,c;t)\,@\,s$$

$$\text{if } \sigma' \in \mathsf{mcheck}(p,t',C_0,\sigma)$$

Figure 4 Operational semantics for the Maude strategy language (control).

$$t \,@\, rl\,[x_1 \texttt{<-} t_1, \dots, x_n \texttt{<-} t_n]\; s \to_s t' \,@\, s \qquad \text{if } t' \in \mathsf{ruleApply}(t, rl, \mathrm{id}[x_1/\theta(t_1), \dots, x_n/\theta(t_n)])$$

$$t \,@\, rl\,[x_1 \texttt{<-} t_1, \dots, x_n \texttt{<-} t_n]\,\{\alpha_1, \dots, \alpha_k\}\; s \to_c \mathrm{rewc}(p : \sigma(t_0) \,@\, \alpha_1\theta_s, \sigma, C', \alpha_2 \cdots \alpha_k, \theta, r, c; t) \,@\, s$$
$$\text{if } (rl, l, r, C) \in R, \mathrm{nrewf}(C) = k, C = C_0 \wedge t_0 \texttt{ => } p \wedge C', \theta = \mathrm{vctx}(s),$$
$$\text{and } (\sigma, c) \in \mathsf{amcheck}(l, t, C_0, \mathrm{id}[x_1/\theta(t_1), \dots, x_n/\theta(t_n)])$$

$$[\mathrm{prl}_s]\ \frac{q_i \to_s q_i'}{\mathrm{subterm}(\dots, x_i : q_i, \dots; t) \,@\, s \to_s \mathrm{subterm}(\dots, x_i : q_i', \dots; t) \,@\, s}$$

$$\mathrm{rewc}(p : t' \,@\, \varepsilon, \sigma, C_0, \bar{\alpha}, r, c; t) \,@\, s \to_s c(\sigma'(r)) \,@\, s$$
$$\text{if } \sigma' \in \mathsf{mcheck}(p, t', C_0, \sigma)$$

Figure 5 Operational semantics for the Maude strategy language (system).

$$\frac{t \,@\, (rl_1 \,|\, rl_2)\; s \to_c t \,@\, rl_1}{\mathrm{subterm}(x_1 : t \,@\, (rl_1 \,|\, rl_2)\; s; t) \,@\, s' \to_c \mathrm{subterm}(x_1 : t \,@\, rl_1\; s; t) \,@\, s'} \text{ using } [\mathrm{prl}_c]$$

The condition for the [else] rule refers to the execution tree: it requires the tree from $t \,@\, \alpha\,\theta$ to be finite and not to contain solutions, i.e. the translation of $[\![\alpha]\!](\theta, t) = \emptyset$ for the denotational semantics. Note that the evaluation of this condition may not finish. Thus, $\to_c$ is undecidable and so $\to_{s,c}$ and $\twoheadrightarrow$ are.

While derivations are linear, proofs almost never are. The main rules are usually applied to substates by using $[\mathrm{prl}_c]$, $[\mathrm{prl}_s]$, $[\mathrm{rewc}_c]$ or $[\mathrm{rewc}_s]$. In the case of [else], all derivations from the condition must be proved to finish and fail. However, a practical execution of these rules is easier. For example the different $t \,@\, \alpha \texttt{ ? } \beta \texttt{ : } \gamma \to_c t \,@\, \alpha\beta$ may share the computation of $t \,@\, \alpha$ and [else] be triggered only if all attempts from $t \,@\, \alpha$ have failed.

B.1 Some properties

▶ **Definition 12.** *For any* $q_0 \in \mathcal{XS}_{\mathcal{R}}$

1. *The* reachable states *from* q_0 *are* $\{q : q_0 \to_{s,c}^* q\}$.
2. *The* reachable subject terms *from* q_0 *are* $\{\mathrm{cterm}(q) : q_0 \to_{s,c}^* q\}$.
3. *The* reachable terms *from* q_0 *are* $\{t : q_0 \to_{s,c}^* q$ *and* $t \,@\, s$ *is a subterm of* $q\}$.

▶ **Lemma 13** (Stack concatenation). *For any* $s_1, s_2 \in \mathrm{Stack}$, $q_0, q' \in \mathcal{XS}$ *and* $t \in T_{\Sigma/E}$,

1. *If* $t \,@\, s_1\,\theta \to_{s,c}^* t_m \,@\, \varepsilon$ *and* $t_m \,@\, s_2 \to_{s,c}^* q$ *then* $t \,@\, s_1 s_2 \to_{s,c}^* q$ *where* $\theta = \mathrm{vctx}(s_2)$.
   Moreover, the length of the resulting execution is the sum of the lengths of the original ones.
2. *If* $q_0 \,@\, s_1 s_2 \to_{s,c}^* q'$ *and* $s_1 \neq \varepsilon$ *then*

$$q' \in \{q \,@\, s' s_2 : q_0 \,@\, s_1\theta \to_{s,c}^* q \,@\, s'\} \cup \{q : \exists\, t_m \quad q_0 \,@\, s_1\theta \to_{s,c}^* t_m \,@\, \varepsilon \;\wedge\; t_m \,@\, s_2 \to_{s,c}^* q\}$$

3. *If* $t \,@\, s_1 s_2 \to_{s,c}^* t' \,@\, \varepsilon$ *then there is a term* t_m *such that* $t \,@\, s_1\theta \to_{s,c}^* t_m \,@\, \varepsilon$ *and* $t_m \,@\, s_2 \to_{s,c}^* t' \,@\, \varepsilon$.

Proof. Some common facts follow easily from the inspection of the rules and axioms of the semantics definition:

- The global stack s in $q \,@\, s$, only changes by a $\to_{s,c}$ reduction if $q = t \,@\, s$ for some term t.
- Only the top of the stack can be popped by a reduction, $t \,@\, \alpha\, s \to_{s,c} q \,@\, s_\alpha\, s$ for some stack s_α.
- Reductions only depend on the top of the stack and the variable environment. Thus, the stack can be extended from below without effect if the latter is preserved, i.e. $q \,@\, s\,\theta \to_{s,c} q' \,@\, s'$ implies $q \,@\, s\, s_0 \to_{s,c} q \,@\, s' s_0$ where $\theta = \mathrm{vctx}(s_0)$, which may be omitted if it is id.

Using these basic facts, we will prove the statements:

1. Replace θ by s_2 in the stacks' bottoms of the first execution. Then we obtain $t \mathbin{@} s_1 s_2 \to^*_{s,c} t_m \mathbin{@} s_2$ and joining it with the second execution, the statement holds.
2. The proof will be carried out by induction on the length k of the derivation $q \mathbin{@} s_1 s_2 \to^*_{s,c} q'$
   - Base case ($k = 0$): then $q_0 \mathbin{@} s_1 s_2 = q'$, so q' is in the first set with the 0-length execution and $s' = s_1$.
   - Inductive case: we have $q_0 \mathbin{@} s_1 s_2 \to_{s,c} q_1 \to^k_{s,c} q'$. If q_0 is a subterm or rewc state, the stack remains unchanged, so we can apply induction hypothesis on $q_1 \mathbin{@} s_1 s_2 \to^k_{s,c} q'$ to conclude. Otherwise, $q_0 \mathbin{@} s_1 s_2 = t \mathbin{@} s_1 s_2$ for some term t. And since the stack s_1 is not empty, it can be $\sigma s_1'$ or $\alpha\, s_1'$. In the first case $t \mathbin{@} \sigma s_1' s_2 \to_c t \mathbin{@} s_1' s_2$: if $s_1' = \varepsilon$ then q' is in the second set with $t_m = t$, otherwise, we apply induction hypothesis to the rest of the derivation $t \mathbin{@} s_1' s_2 \to^k_{s,c} q'$. Here, we have two possibilities
     - There is a term t_m such that $t \mathbin{@} s_1' s_2 \to^*_{s,c} t_m \mathbin{@} s_2$ and $t_m \mathbin{@} s_2 \to^*_{s,c} q'$. Hence, our q' is in the second set with witnesses the second execution one line above, and the extended first derivation $t \mathbin{@} s_1 \theta \to_c t \mathbin{@} s_1' \theta \to^*_{s,c} t_m \mathbin{@} \theta \to_c t_m \mathbin{@} \varepsilon$.
     - $t \mathbin{@} s_1' \theta \to^*_{s,c} q \mathbin{@} s'$ for some state q and $q' = q \mathbin{@} s' s_2$. Thus, our q' is in the first set with $t \mathbin{@} s_1 \theta = t \mathbin{@} \sigma s_1' \theta \to_c t \mathbin{@} s_1' \theta \to^*_{s,c} q \mathbin{@} s'$.

     The other case $t \mathbin{@} \alpha s_1' s_2 \to_{s,c} q_1 \mathbin{@} s_\alpha s_1' s_2$ is almost identical. However, when $s_\alpha s_1'$ is empty, $q_1 \mathbin{@} \alpha s_1' s_2$ may also be a subterm or rewc state. Then, or the stack remains unchanged in the full execution, so that $q' = q' \mathbin{@} s_\alpha s_1' s_2$ and q' is in the first set; or the state reduces after some steps to a term state. In this case, we apply the induction hypothesis to conclude.
3. We only have to apply (2) with $q_0 \mathbin{@} s_1 s_2 = t \mathbin{@} s_1 s_2$ and $q' = t' \mathbin{@} \varepsilon$ if $s_1 \neq \varepsilon$ (otherwise is trivial).

◀

▶ **Proposition 14** (Subterm states). *For any states $q_1, \dots, q_n \in \mathcal{XS}$, and terms $t, t', t_1, \dots, t_n \in T_{\Sigma/E}$,*

1. $\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} \mathrm{subterm}(x_1 : q_1', \dots, x_n : q_n'; t) \mathbin{@} \varepsilon$ *if* $q_i \to^*_{s,c} q_i'$ *for all* i.
2. *If* $\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} \mathrm{subterm}(x_1 : q_1', \dots, x_n : q_n'; t) \mathbin{@} \varepsilon$ *then* $q_i \to^*_{s,c} q_i'$ *for all* i.
3. *If* $q_i \to^*_{s,c} t_i \mathbin{@} \varepsilon$ *for all* $1 \leq i \leq n$ *then* $\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} t[x_i/t_i]_{i=1}^n \mathbin{@} \varepsilon$.
4. *If* $\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} \varepsilon \to^*_{s,c} t' \mathbin{@} \varepsilon$ *then there are terms* $t_1, \dots, t_n$ *such that* $q_i \to^*_{s,c} t_i \mathbin{@} \varepsilon$ *for all* $1 \leq i \leq n$ *and* $t' = t[x_i/t_i]_{i=1}^n$.

Proof. Statements (3) and (4) are a corollary of statements (1) and (2). The two first statements will be proven by induction.

1. The proof will be done by induction on the lengths of the executions from q_i. Suppose all lengths are 0 then $q_i = q_i'$ and the empty execution solves the statement. Otherwise, there is at least a positive number in the n-tuple of execution lengths. We assume the statement is true for any collection of derivation of lower or equal length with at least one strictly lower coordinate. Suppose, without loss of generality, that $q_1 \to_{s,c} q_{1,1} \to^*_{s,c} q_1'$. Then by $[\mathrm{prl}_s]$ or $[\mathrm{prl}_c]$ we can assert

   $$\mathrm{subterm}(x_1 : q_1, \dots, x_n : q_n; t) \mathbin{@} \varepsilon \to_{s,c} \mathrm{subterm}(x_1 : q_{1,1}, \dots, x_n : q_n; t) \mathbin{@} \varepsilon$$

   And the children executions for the last states are strictly shorter than the original ones. By induction hypothesis, we can complete the execution and conclude the statement.
2. Now, induction is on the length of the execution in the premise. Suppose the length is 0, then $q_i = q_i'$ and the empty execution from all q_i make the statement true. If the length is positive, there must be an initial $[\mathrm{prl}_c]$ or $[\mathrm{prl}_s]$ transition

   $$\mathrm{subterm}(x_1 : q_1, \dots, x_i : q_i, \dots, x_n : q_n; t) \mathbin{@} \varepsilon \to_{s,c} \mathrm{subterm}(x_1, q_1, \dots, x_i : q_{i,1}, \dots, x_n : q_n; t) \mathbin{@} \varepsilon$$

   and for this to be true $q_i \to_{s,c} q_{i,1}$ must hold. By induction hypothesis on the execution from the right-hand side, $q_j \to^*_{s,c} q_j'$ for all $j \neq i$ and $q_i \to_{s,c} q_{i,1} \to^*_{s,c} q_i'$. So we have found the desired executions.
3. It is enough to apply (1) and then $\mathrm{subterm}(x_1 : t_1 \mathbin{@} \varepsilon, \dots, x_n : t_n \mathbin{@} \varepsilon; t) \mathbin{@} \varepsilon \to_c t[x_i/t_i]_{i=1}^n \mathbin{@} \varepsilon$.
4. $t[x_i/t_i]_{i=1}^n \mathbin{@} \varepsilon$ must be preceded by $\mathrm{subterm}(x_1 : t_1 \mathbin{@} \varepsilon, \dots, x_n : t_n \mathbin{@} \varepsilon; t) \mathbin{@} \varepsilon$ so that we can apply (2) to conclude.

◀

The *static strategy call graph* reflects the control flow relations among the named strategies available in a strategy module. Its vertices are the strategies identified by their labels and parameter kinds, and there is an edge from one strategy to another if any definition of the first contains a strategy call expression to the second.

▶ **Definition 15** (Recursive strategies). *A named strategy is* recursive *if a cycle passes through it in the static call graph. Moreover, the following strategy expression are said to* only contain tail recursive calls*:*

- `idle`, `fail`, *tests, and strategy call expressions.*
- $\alpha \,|\, \beta$ *if* α *and* β *only contain tail recursive calls.*
- $\alpha \,;\, \beta$ *if* α *does not contain recursive calls and* β *only contains tail recursive calls.*
- $\alpha \;?\; \beta \;:\; \gamma$ *if* α *does not contain recursive calls, and* β *and* γ *only contains tail recursive calls.*
- *Subterm rewriting and rule application expressions if all its substrategies does not contain recursive calls.*

▶ **Lemma 16.** *Given a finitary graph* $G = (V, E)$ *and a vertex* $v_0 \in V$*, if every infinite path* $\pi \in V^\omega$ *from* v_0 *has a loop (there exists* $0 \leq i < j \in \mathbb{N}$ *such that* $\pi_i = \pi_j$*), the reachable vertices from* v_0 *are finitely many.*

Proof. Let V_0 be the set of reachable states from v_0. For each vertex $v \in V_0$, we consider the distance from v_0 to v as the minimum length of a path that goes from the first to the second. Then, we consider $G' = (V_0, E')$ where $(v, w) \in E'$ if $(v, w) \in E$ and the distance to v is strictly lower than the distance to w. This ensures that any path in G' visits vertices in strictly increasing distance and does not visit any vertex twice. Moreover, all vertices are still reachable since the edges in the minimum path from v_0 to any vertex v should have been preserved due to the optimality of the prefix paths. In fact, G' is a tree.

Suppose V_0 is infinite, we should prove there is an infinite path in G without repeated nodes. Under this assumption, G' is a finitary but infinite tree so, by the Kőnig's lemma, there is an infinite length path within it. This path is a path in G too, and it does not have repeated nodes. We have finished. ◀

▶ **Proposition 17** (Reachable execution states). *For any strategy* $\alpha \in \mathrm{Strat}_{\mathcal{R}}$*,*

1. *If* α *and the definition of the strategies called by* α *do not contain iteration or recursive function calls, the reachable states from* $t \mathbin{@} \alpha\, \theta$ *are finitely many for any* $t \in T_{\Sigma/E}$ *and* $\theta \in \mathsf{VEnv}_{\mathcal{R}}$*.*
2. *If the reachable terms from* $t \mathbin{@} \alpha$ *are finitely many, strategies are only called with a finite number of distinct arguments, and strategy definitions only contain tail recursive calls, the reachable execution states from* $t \mathbin{@} \alpha$ *are finitely many and* $\twoheadrightarrow$ *restricted to the reachable states is decidable.*
   We must also consider that variable environments are replaced instead of pushed to the stack when other substitution is on top. This optimization *do not affect the semantics.*

   In both cases, $\twoheadrightarrow$ *restricted to the reachable states is decidable.*

Proof. First, we should first clarify that the optimization mentioned in the second statement does not affect the semantics because $\mathrm{vctx}(\sigma_1\sigma_2 s) = \sigma_1$ whatever σ_2 is and the only allowed execution from $t \mathbin{@} \sigma_1\sigma_2\, s$ starts with $t \mathbin{@} \sigma_1\sigma_2\, s \to_c t \mathbin{@} \sigma_2\, s \to_c t \mathbin{@} s$. Thus, σ_2 does not have any effect and can be safely removed.

Second, we will prove that the direct successors of any execution are finitely many and that they are lower in the following lexicographic order on the natural numbers:

$$m(q) := (\text{number of strategy constructors in } q, \text{number of substitutions in } q,$$
$$\text{number of nested } \mathcal{XS}_{\mathcal{R}} \text{ constructors}, \text{number of condition fragments in rewc states})$$

However, there are some exceptions. The direct successors of `all` strategy might be infinitely many because of conditional rules with rewriting fragments, but this case is not possible in the second statement due to finiteness of the reachable terms. The iteration and strategy calls transitions are not m-decreasing, so they will be treated apart.

- For $t \mathbin{@} \texttt{idle}\, \theta$, $t \mathbin{@} \texttt{fail}\, \theta$, $t \mathbin{@} \texttt{match}\ P\ \texttt{s.t}\ C\, \theta$ at most $t \mathbin{@} \theta$ can be reached by $\to_{s,c}$ transitions, and in all cases the number of strategy constructor decreases.
- $t \mathbin{@} \alpha \,|\, \beta\, \theta$ can only be followed by $t \mathbin{@} \alpha\, \theta$ and $t \mathbin{@} \beta\, \theta$, only two possibilities and with fewer constructors.
- $t \mathbin{@} \theta\, s$ is only followed by $t \mathbin{@} s$.
- $t \mathbin{@} \alpha \,;\, \beta\, \theta$. The only direct successor is $t \mathbin{@} \alpha\beta\, \theta$, which removes the concatenation constructor.

- $t \mathbin{@} rl[Subs]\,\theta$ has a finite number of successors, as a finite number of rules (with label rl) can be applied in a finite number of positions, with a finite number of matches. The application of a rule with rewriting conditions has a finite number of successors too, since the only allowed successors are

  $$t \mathbin{@} rl[Subs]\{\alpha_1, \dots, \alpha_n\}\theta \to_c \mathrm{rewc}(p : \sigma(l) \mathbin{@} \alpha_1\theta, \sigma, C, \alpha_1 \cdots \alpha_n, \theta_s, r, c; t) \mathbin{@} \theta$$

  for some finite choices of σ, θ_s, r, l, p and c. These execution states have a strategy constructor less.
- The only $\to_{s,c}$ successors of $t \mathbin{@} \alpha \mathbin{?} \beta : \gamma\,\theta$ are $t \mathbin{@} \gamma\,\theta$ and $t \mathbin{@} \alpha\beta\,\theta$, with less strategy combinators each.
- $t \mathbin{@} \texttt{matchrew}\ P\ \texttt{s.t}\ C\ \texttt{by}\ x_1\ \texttt{using}\ \alpha_1,\ \dots, x_n\ \texttt{using}\ \alpha_n\,\theta \to_c \mathrm{subterm}(\dots, x_i : t \mathbin{@} \alpha_i\,\theta \circ \sigma, \dots; t) \mathbin{@} \theta$ for all possible matches, which are a finite set. There is a strategy constructor less in the right-hand side.
- For $\mathrm{subterm}(x_1 : q_1, \dots, x_n, q_n; t) \mathbin{@} s$, if the number of q_i successors is finite, the number of successors of the compound term is the sum of those. Whenever m decreases in the subterms, it does so in the whole. If the substates are solutions, there is a single successor with an execution state constructor less $\mathrm{subterm}(x_1 : t_1 \mathbin{@} \varepsilon, \dots, x_n : t_n \mathbin{@} \varepsilon; t) \mathbin{@} s \to_c t[x_i/t_i]_{i=1}^n \mathbin{@} s$.
- For $\mathrm{rewc}(p : q, \sigma, C, \alpha_1 \cdots \alpha_n, \theta_s, r, c; t) \mathbin{@} s$, their successors are those of q by the [rewc] rules, or the successors of the solutions $\mathrm{rewc}(p : t_m \mathbin{@} \varepsilon, \sigma, C, \alpha_1 \cdots \alpha_n, \theta_s, r, c; t) \mathbin{@} s$, which are finitely many, one for each possible match. The order decreases in the second case because either an execution strategy constructor is removed or a rewriting condition fragment is taken from C.

In the following cases only the first property is respected:

- For the strategy calls, the derivation starts with $t \mathbin{@} sl(t_1, \dots, t_n)\theta \to_c t \mathbin{@} \delta\sigma\,\theta$ for every strategy definition (sl, sdp, C, δ) and any $\sigma \in \mathsf{vmatch}(sdp_i, (t_1, \dots, t_n), C)$, which are finitely many. However, the number of strategy constructors may have increased with the addition of δ.
- The successors of a $t \mathbin{@} \alpha^* s$ state are $t \mathbin{@} \alpha\alpha^* s$ and $t \mathbin{@} s$ following $\to_c$ transitions, so they are finitely many. In the first case the number of strategy constructors increases. Notice that the iteration body cannot contain recursive calls since they would not be tail recursive.

Since we have proven that the number of direct successors is finite, if there were infinitely many execution states, there must be a non-terminating execution by the Kőnig's lemma. In such an execution, all steps except iterations and calls are decreasing in the lexicographic well-founded order of $\mathbb{N}^4$, so there must be infinitely many of those exceptions for the execution to be infinite.

For the first statement, no iterations can occur, so in any infinite path there must be infinitely many function calls. Notice that a complete strategy execution

$$t \mathbin{@} sl(t_1, \dots, t_n)\ s \to_c t \mathbin{@} \delta\,\sigma\,s \to^*_{s,c} t' \mathbin{@} \sigma\,s \to_c t' \mathbin{@} s$$

does decrease the m order, so there must be infinitely nested calls. However, since the number of strategies is finite and our premise say that there are not recursive calls, this is impossible. Hence, the assumption of the existence of infinitely many execution states is a contraction, there are finitely many.

For the second statement, we will prove that all infinite executions have a loop. Then, by Lemma 16, the reachable states are infinitely many. Now, infinite executions must contain an infinite number of strategy calls and/or an infinite number of iterations. Notice that complete iterations globally decrease the m metric,

$$t \mathbin{@} \alpha^*\,s \to_c t \mathbin{@} \alpha\,\alpha^*\,s \to^*_{s,c} t' \mathbin{@} \alpha^*\,s \to_c t' \mathbin{@} s$$

Hence, an iteration must iterate forever to generate an infinite execution (a recursive strategy might execute complete finite iterations infinitely many times, but then there must be infinite strategy calls, and this case is considered later). Thus, states of the form $t' \mathbin{@} \alpha^*\,s$ must occur infinitely often. Since only a finite number of terms can be reached by hypothesis, these states cannot be all distinct, so there is a loop in the execution.

Suppose there is infinitely many strategy calls. As we argue for the first statement, it must be infinitely nested calls and non-recursive calls must finish eventually. It can be proven by induction that if α only contains tail recursive calls, and we have $t \mathbin{@} \alpha\theta \to^*_{s,c} t' \mathbin{@} sl(t_1, \dots, t_n)s$ where all previous calls have finished, then $s = \theta$. Then, the strategy call with optimization will give $t' \mathbin{@} sl(t_1, \dots, t_n)\,\theta \to_c t' \mathbin{@} \delta\,\sigma$ where σ replaces θ. Repeating the argument for δ and σ and using the assumption that only a finite number of strategy arguments (hence a

finite number of σ) and a finite number of terms could appear, we have that some $t' @ \delta\sigma$ must appear twice in the execution, so that there is a loop. There is a loop in every infinite execution path, so the number of reachable states is finitely many.

◀

The following definitions will be used to fix the concept of *abstract strategy* for any strategy expression α, as a language of all allowed execution traces. Strategies are used both to express computations and to describe the behavior of reactive systems, for which finite length and infinite words are respectively relevant, so neither of them should be omitted. Moreover, not all finite executions are interesting: some lead to a failure or deadlock state, and do not correspond to solutions for the strategies.

▶ **Definition 18.** **1.** *The language of* infinite executions *from* q_0 *is*

$$\mathrm{Ex}_\omega(q_0) = \{q_0\, q_1 \cdots q_n \cdots \mid q_0 \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots\}$$

2. *The language of* finite (complete) executions *from* q_0 *is*

$$\mathrm{Ex}_{\mathrm{fin}}(q_0) = \{q_0\, q_1 \cdots q_n \mid q_0 \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \wedge q_n \textit{ is an end }\}$$

and q *is an end if* $q \to^*_{s,c} \mathrm{cterm}(q)\ @\varepsilon$ *or* $q \twoheadrightarrow q'$ *never holds for* $q' \in \mathcal{XS}_{\mathcal{R}}$.

3. *The language of* successful executions *from* q_0 *is*

$$\mathrm{Ex}_{\mathrm{succ}}(q_0) = \{q_0 \cdots q_n \mid q_0 \twoheadrightarrow \cdots \twoheadrightarrow q_n \to^*_c \mathrm{cterm}(q_n)\ @\varepsilon\}$$

However, execution states are only an artifice to maintain the required control data for the execution of a strategy. We are interested in the evolution of the underlying rewriting system, in terms of $T_{\Sigma/E}$ and $\to^1_{\mathcal{R}}$. Abusing of notation, a function $f : X \to Y$ may be extended to $f : X^* \to Y^*$ by $f(x_1 \cdots x_n) := f(x_1) \cdots f(x_n)$, and to $f : X^\omega \to Y^\omega$ similarly. The application of f to a subset of X is standard, the image of a set by a function.

▶ **Definition 19.** **1.** *The language of* infinite execution traces *from* q_0 *is* $T_\omega(q_0) = \mathrm{cterm}(\mathrm{Ex}_\omega(q_0))$.
2. *The language of* finite execution traces *from* q_0 *is* $T_{\mathrm{fin}}(q_0) = \mathrm{cterm}(\mathrm{Ex}_{\mathrm{fin}}(q_0))$.
3. *The language of* successful execution traces *from* q_0 *is* $T_{\mathrm{succ}}(q_0) = \mathrm{cterm}(\mathrm{Ex}_{\mathrm{succ}}(q_0))$.

▶ **Definition 20.** *For any* $\alpha \in \mathrm{Strat}_{\mathcal{R}}$ *and* $I \subseteq T_\Sigma(X)$ *the* abstract strategy α *from states in* I *is*

$$E(\alpha, I) = \bigcup_{t \in I} T_\omega(t\ @\alpha) \cup T_{\mathrm{succ}}(t\ @\alpha)$$

For some applications, it is convenient to consider strategies as pure ω-languages. This can be done completing $\twoheadrightarrow$ to be total with $q \twoheadrightarrow \mathrm{cterm}(q)\,@\,\varepsilon$ if $q \to^*_c \mathrm{cterm}(q)\,@\,\varepsilon$, for all state q. In this case the definition is as simple as

$$E(\alpha, I) = \{\mathrm{cterm}(q_0)\,\mathrm{cterm}(q_1) \cdots \mathrm{cterm}(q_n) \cdots \mid q_0 \twoheadrightarrow q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow \cdots\}$$

This is equivalent to extend finite traces repeating the last state forever. The next propositions relate regular languages and reachable states cardinality:

▶ **Proposition 21.** *For any* $\alpha \in \mathrm{Strat}_{\mathcal{R}}$ *and* $I \subseteq T_\Sigma(X)$ *finite, if the reachable states from* $t\ @\alpha$ *are finite for* $t \in I$*, the extended abstract strategy* $E(\alpha, I)$ *is an* ω*-regular language, and a Büchi automaton for* $E(\alpha, I)$ *is* $\mathcal{A} = (q, \mathrm{cterm}(q), \delta, \{start\}, q)$ *where* $q = \{start\} \cup \{q \in \mathcal{XS}_{\mathcal{R}} \mid t \in I \ \wedge\ t\ @\alpha \twoheadrightarrow^* q\}$ *and*

$$\begin{aligned}
\delta(start, t) &= \{\ t\ @\alpha\ \} && \textit{if } t \in I\\
\delta(start, t) &= \emptyset && \textit{if } t \notin I\\
\delta(q, t) &= \{\ q' : q \twoheadrightarrow q' \ \wedge\ \mathrm{cterm}(q') = t\ \} && \textit{for } q \in q \backslash \{start\}\\
&\quad \cup \{\ t\ @\varepsilon : \textit{if } q \to^*_c t\ @\varepsilon\ \}
\end{aligned}$$

Nevertheless, the reciprocal does not hold, the language $E(\alpha, I)$ can be ω-regular and the reachable states from α be infinitely many. In a strategy module with `sd st(X) := fail | st(s(X))` consider $\alpha := \mathtt{st(0)}$. No matter the rewriting theory, $t\,@\,\alpha$ cannot be continued by $\twoheadrightarrow$. So $E(\alpha, I) = \emptyset$, which is ω-regular. But infinite states of the form $t\,@\,\mathtt{fail}[x \mapsto N] \ldots [x \mapsto 1][x \mapsto 0]$ are reachable from the origin.

Moreover, the automaton $\mathcal{A}$ for $E(\alpha, I)$ has always trivial Büchi conditions. Hence, abstract strategies whose generator span to a finite number of states are a restricted subclass of ω-regular languages.

References

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

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

[3] Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. LTL to Büchi automata translation: fast and more deterministic. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012.

[4] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking rewriting on Java. In Franz Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2007. ISBN: 978-3-540-73447-5.

[5] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 131. North Holland, 2nd edition, 2014.

[6] Massimo Bartoletti, Maurizio Murgia, Alceste Scalas, and Roberto Zunino. Modelling and verifying contract-oriented systems in Maude. In Santiago Escobar, editor, *Rewriting Logic and Its Applications - 10th International Workshop, WRLA 2014, Held as a Satellite Event of ETAPS, Grenoble, France, April 5-6, 2014, Revised Selected Papers*, volume 8663 of *Lecture Notes in Computer Science*, pages 130–146. Springer, 2014. ISBN: 978-3-319-12903-7.

[7] Peter Borovanský, Claude Kirchner, Hélène Kirchner, and Christophe Ringeissen. Rewriting with strategies in ELAN: A functional semantics. *Int. J. Found. Comput. Sci.*, 12(1):69–95, 2001.

[8] Tony Bourdier, Horatiu Cirstea, Daniel J. Dougherty, and Hélène Kirchner. Extensional and intensional strategies. In Maribel Fernández, editor, *Proceedings Ninth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2009, Brasilia, Brazil, 28th June 2009*, volume 15 of *EPTCS*, pages 1–19, 2009.

[9] Christiano Braga and Alberto Verdejo. Modular structural operational semantics with strategies. In Rob van Glabbeek and Peter D. Mosses, editors, *Proceedings of the Third Workshop on Structural Operational Semantics, SOS 2006, Bonn, Germany, August 26, 2006*, volume 175(1) of *Electronic Notes in Theoretical Computer Science*, pages 3–17. Elsevier, 2007.

[10] Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Visser. Stratego/XT 0.17. A language and toolset for program transformation. *Science of Computer Programming*, 72(1-2):52–70, 2008.

[11] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. ISBN: 978-3-319-10575-8.

[12] Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott. *Maude Manual v3.1*. October 2020.

[13] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007. ISBN: 978-3-540-71940-3.

[14] Grit Denker and Carolyn Talcott, editors. *Proceedings of the 6th International Workshop on Rewriting Logic and its Applications, WRLA 2006, Vienna, Austria, April 1-2, 2006*, volume 176 of number 4 in *Electronic Notes in Theoretical Computer Science*, 2007. Elsevier.

[15] Stefan Edelkamp, Stefan Leue, and Alberto Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. *Int. J. Softw. Tools Technol. Transf.*, 5(2-3):247–267, 2004.

[16] Steven Eker, Narciso Martí-Oliet, José Meseguer, and Alberto Verdejo. Deduction, strategies, and rewriting. In Myla Archer, Thierry Boy de la Tour, and César Muñoz, editors, *Proceedings of the 6th International Workshop on Strategies in Automated Deduction, STRATEGIES 2006, Seattle, WA, USA, August 16, 2006*, volume 174(11) of *Electronic Notes in Theoretical Computer Science*, pages 3–25. Elsevier, 2007.

[17] Steven Eker, José Meseguer, and Ambarish Sridharanarayanan. The Maude LTL model checker. In Fabio Gadducci and Ugo Montanari, editors, *Proceedings of the Fourth International Workshop on Rewriting Logic and its Applications, WRLA 2002, Pisa, Italy, September 19-21, 2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*, pages 162–187. Elsevier, 2004.

[18] Mercedes Hidalgo-Herrero, Alberto Verdejo, and Yolanda Ortega-Mallén. Using Maude and its strategies for defining a framework for analyzing Eden semantics. In Sergio Antoy, editor, *Proceedings of the Sixth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2006, Seattle, WA, USA, August 11, 2006*, volume 174(10) of *Electronic Notes in Theoretical Computer Science*, pages 119–137. Elsevier, 2007.

[19] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. ISBN: 0-13-153271-5.

[20] Hélène Kirchner. Rewriting strategies and strategic rewrite programs. In Narciso Martí-Oliet, Peter Csaba Ölveczky, and Carolyn L. Talcott, editors, *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2015. ISBN: 978-3-319-23164-8.

[21] Saul Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24(1):1–14, 1959.

[22] Orna Kupferman and Moshe Y. Vardi. Memoryful branching-time logic. In Rajeev Alur, editor, *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 265–274. IEEE Computer Society, 2006. ISBN: 0-7695-2631-4.

[23] Weiwei Li, Shuanglong Kan, and Zhiqiu Huang. A better translation from LTL to transition-based generalized Büchi automata. *IEEE Access*, 5:27081–27090, 2017.

[24] Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, and José Meseguer. Formal modeling and analysis of Cassandra in Maude. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2014. ISBN: 978-3-319-11736-2.

[25] Narciso Martí-Oliet, José Meseguer, and Alberto Verdejo. Towards a strategy language for Maude. In Narciso Martí-Oliet, editor, *Proceedings of the Fifth International Workshop on Rewriting Logic and its Applications, WRLA 2004, Barcelona, Spain, March 27-April 4, 2004*, volume 117 of *Electronic Notes in Theoretical Computer Science*, pages 417–441. Elsevier, 2004.

[26] Narciso Martí-Oliet, Miguel Palomino, and Alberto Verdejo. Strategies and simulations in a semantic framework. *Journal of Algorithms*, 62(3-4):95–116, 2007.

[27] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[28] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: on the model-checking problem. *ACM Trans. Comput. Log.*, 15(4):34:1–34:47, 2014.

[29] Martin R. Neuhäußer and Thomas Noll. Abstraction and model checking of Core Erlang programs in Maude. In Proceedings of the 6th International Workshop on Rewriting Logic and its Applications, WRLA 2006, Vienna, Austria, April 1-2, 2006 [14], pages 147–163.

[30] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[31] Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, and Alberto Verdejo. Model checking strategy-controlled rewriting systems. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, 34:1–34:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[32] Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, and Alberto Verdejo. *The semantics of the Maude strategy language*.

[33] Gustavo Santos-García and Miguel Palomino. Solving Sudoku puzzles with rewriting rules. In Proceedings of the 6th International Workshop on Rewriting Logic and its Applications, WRLA 2006, Vienna, Austria, April 1-2, 2006 [14], pages 79–93.

[34] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003. ISBN: 978-0521391153.

[35] Alberto Verdejo and Narciso Martí-Oliet. Basic completion strategies as another application of the Maude strategy language. In Santiago Escobar, editor, *Proceedings 10th International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2011, Novi Sad, Serbia, 29 May 2011*, volume 82 of *EPTCS*, pages 17–36, 2011.