Strategies, model checking and branching-time properties in Maude⋆ (extended version)

Rubén Rubio, Narciso Martí, Isabel Pita, Alberto Verdejo

Universidad Complutense de Madrid, Spain
{rubenrub,narciso,ipandreu,jalberto}@ucm.es

Abstract. Maude 3.0 includes as a new feature an object-level strategy language. Rewriting strategies can now be used to easily specify how rules should be applied and restrict the rewriting systems behavior. This new specification layer would not be useful if there were no tools to execute, analyze and verify its creatures. For that reason, we extended the Maude LTL model checker to systems controlled by strategies, after studying their model-checking problem. Now, we widen the range of properties that can be checked in Maude models, both strategy-aware and strategy-free, by implementing a module for the language-independent model checker LTSmin that supports logics like CTL* and μ-calculus.

1 Introduction

The Maude [9] specification language has recently reached its 3.0 version, integrating new features developed during the last years and including a full implementation of the Maude strategy language. Although rewriting logic owes its natural representation of concurrency to the possibility that different rules can be executed in different positions at each step of the rewriting process, there are situations in which it is convenient to control such nondeterminism. This is the purpose of strategies, which have traditionally been expressed in Maude at the metalevel by means of its reflective features [11, 10, 25], but since the complexity and learning curve of programming metalevel computations is hard, an object-level strategy language design was proposed [19, 14], exercised with different examples [28, 24, 20, 26, ...], and finally added to the Core Maude functionality. Strategies can be described compositionally using strategy modules on top of system modules, and different commands are provided to rewrite a term following a strategy.

However, this new feature would be worthless without convenient tools to analyze the specifications using it. One of the most useful tools for verifying regular Maude modules is its LTL model checker [15]. In a previous work [23], we have studied the model-checking problem for rewriting systems controlled by strategies and presented an extension of the model checker to deal with them. However, since the original Maude model checker is limited to LTL properties, these are the only ones that our extension can handle and the discussion was mainly centered on linear-time properties. In this paper, we further discuss branching-time properties and show an implementation of a language plugin for the language-independent model checker LTSmin [16] that widens the range of logics in which properties can be expressed to CTL* and μ-calculus, for both the strategy-aware specifications and the regular ones. It can be downloaded from `http://maude.ucm.es/strategies`.

In the following sections, we briefly introduce the strategy language, the model-checking problem in this context, and the plugin we have developed. But let us first introduce a motivational

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

example: the *river crossing* puzzle. In this classical game, a shepherd needs to cross a river carrying a wolf, a goat and a cabbage. The only way to cross it is using a boat that only the shepherd can operate and with room for only one more being. The shepherd could ship their companions to the other side one by one, but the wolf would eat the goat and the goat would eat the cabbage as soon as the shepherd is not present to impede it. The Maude signature of the problem is specified in a functional module:

```
fmod RIVER is
  sorts River Side Group .
  subsort Side < Group .

  op  _|_ : Group Group -> River [ctor comm] .
  ops left right : -> Side [ctor] .
  ops shepherd wolf goat cabbage : -> Group [ctor] .
  ops __ : Group Group -> Group [ctor assoc comm] .

  op initial : -> River .
  eq initial = left shepherd wolf goat cabbage | right .
endfm
```

The system module `RIVER-CROSSING` completes the equational specification with rules: `alone`, `wolf`, `goat` and `cabbage` cause the shepherd to cross the river with the mentioned passenger, while `wolf-eats` and `goat-eats` make such animal eat its prey, which vanishes from the scene.

```
mod RIVER-CROSSING is
  protecting RIVER .

  vars G G' : Group .

  rl [wolf-eats] : goat wolf G | G' shepherd =>
                   wolf G | G' shepherd .
  rl [goat-eats] : cabbage goat G | G' shepherd =>
                      goat G | G' shepherd .

  rl [alone] : shepherd G | G' =>
                   G | G' shepherd .
  rl [wolf] : shepherd wolf G | G' =>
                       G | G' shepherd wolf .
  rl [goat] : shepherd goat G | G' =>
                       G | G' shepherd goat .
  rl [cabbage] : shepherd cabbage G | G' =>
                    G | G' shepherd cabbage .
endm
```

The rules of the game tell that the predator will not miss the chance to claim their prey, so the eating rules must be applied before any other crossing if possible. This is not guaranteed in the system module, but expressing this restriction using strategies is easy, and we will see how in the following section.

In a previous specification of this problem in Maude [22], the eating rules were written as equations. While this alternative also ensures the discussed property according to the operational semantics of the Maude rewriting engine, it yields a rewrite theory where rules and equations are not coherent[1].

[1] A rewrite theory is *coherent* if for all term t rewritten by a rule to a term t', its canonical form u modulo equations and axioms can be rewritten to a term u' that is equationally equivalent to t', see [9,

2 The Maude strategy language

As we have said in the Introduction, the Maude strategy language was born to allow expressing rewriting strategies without the difficulties of the metalevel. Its design is based on the experience with reflective computations, and on earlier strategy languages like ELAN [4] and Stratego [6].

A strategy α can be seen, if we look at its results, as a transformation from a term t into a set of terms, since the rewriting process controlled by α may still be nondeterministic. These results can be obtained within the interpreter using the **srewrite** t **using** α command. The most elementary strategy is rule application

$$\textbf{top}(\mathit{label}[x_1 \texttt{<-} t_1, \dots, x_n \texttt{<-} t_n]\{\alpha_1, \dots, \alpha_m\}),$$

that executes any available rules with label *label* on any subterm of the subject term. An optional substitution can be specified between brackets to instantiate any occurrence of the variables x_k in the rule and its condition with t_k before matching, and to apply rules with rewriting conditions, strategies α_l must be provided to control each rewriting condition fragment. To restrict the application of the rule to the top of the subject term, top is available. A more powerful tool for selecting to which subterm a strategy is applied is the matchrew operator

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

It matches the pattern P on top of the subject term, and for each match satisfying the condition C, the subterms corresponding to the variables $x_1, \dots, x_n$ are rewritten using the strategies $\alpha_1, \dots, \alpha_n$, and reassembled again. The `matchrew` keyword can be prefixed by **a** to match anywhere within the term or **x** to match modulo structural axioms. The same variants exist for the tests **match** P **s.t.** C, to check if P matches the subject term and satisfies C. Regular expressions are included in the strategy language by means of the alternation $\alpha\,|\,\beta$, the concatenation $\alpha\,;\beta$, the Kleene star α^*, and the constants **idle** and **fail**. A conditional strategy $\alpha\,?\,\beta : \gamma$ is also available. It executes α and then β on its results, but if α does not produce any, it applies γ to the initial term. The language includes some other derived operators like α **or−else** β defined as α `? idle :` β or `not(`α`)` as α `? fail : idle`.

Using these combinators, we can guarantee that eating happens eagerly before traveling in the river crossing puzzle with the following strategy:

```
((wolf-eats | goat-eats) or-else (alone | cabbage | goat | wolf)) *
```

In each step of the iteration, which can stop nondeterministically at any time, the **or−else** combinator ensures that the crossing rules of its second argument are tried only if the eating rules in its first argument do not succeed. However, when strategies become more complex, writing long self-contained expressions is not practical. For example, the previous will be easier to understand if we name the first union of the expression as `eating` and the second as `oneCrossing`, `(eating` **or−else** `oneCrossing) *`. Strategy modules allow defining strategies, which can take parameters and call themselves recursively, extending the expressive power of the language. They are introduced by the `smod` keyword and may contain strategy declarations **strat** `sname : T1 ... Tn @ T` specifying its name and signature, and (possibly conditional) strategy definitions like **sd** `sname(`$t_1, \dots, t_n$`) :=` α. A strategy call will execute all strategy definitions whose left-hand side matches the call term, instantiating the right-hand side expression with the variables bound in the left-hand side and the optional condition.

The following strategy module gives some strategy definitions for the river crossing problem:

§5.3]. Coherence is assumed by Maude, which reduces terms to their canonical forms before applying a rule, not to miss any rewrite.

```
smod RIVER-CROSSING-STRAT is
  protecting RIVER-CROSSING .

  var G : Group .

  strats oneCrossing eating @ River .
  sd oneCrossing := alone | wolf | goat | cabbage .
  sd eating := wolf-eats | goat-eats .

  strats solution eagerEating safe @ River .

  sd solution := goat ; alone ; cabbage ; goat ;
                 wolf ; alone ; goat .

  sd eagerEating := match left | G cabbage goat ? idle
      : ((eating or-else oneCrossing) ; eagerEating) .
  sd safe := match left | G ? idle
      : (oneCrossing ; not(eating) ; safe) .
endsm
```

In addition to the `oneCrossing` and `eating` strategies described before, there is also a deterministic strategy `solution` that simply applies a choice of steps that are known to solve the problem. The `eagerEating` strategy recursively executes a rule respecting their precedence, indefinitely or until a solution is found. Observe that the definition is recursive and nonterminating. This will not pose a problem since the execution engine and the model checker will be able to detect this loop and finish, and it is a useful resource to specify the behavior of reactive systems. The last strategy `safe` discards all rewriting paths where some being can be swallowed by concatenating the `not(eating)` strategy that fails whenever `eating` succeeds. Note that the stop condition only checks whether the left side of the river is empty, which is enough provided no one dies, while in `eagerEating` it is necessary to check that the goat and cabbage are still alive. We can execute the strategy to see how the solution is reached:

```
Maude> srew initial using safe .

Solution 1
rewrites: 33
result River: left | right shepherd wolf goat cabbage

No more solutions.
rewrites: 33
```

More details about the strategy language and examples can be found in its chapter in the Maude manual [9], in [12], and the strategy language website [13].

3 Model checking

Model checking [7, 8] is an automated verification technique based on the exhaustive exploration of a system model to check a property describing aspects of its intended behavior. Multiple variants and algorithms exist, but traditionally the model is represented as a state and transition system, and the property in some temporal logic.

A *transition system* or *abstract reduction system* is a set of states S endowed with a binary transition relation $(\to) \subseteq S \times S$. It is usually required that every state has at least a successor to avoid dealing with finite executions. In the abstract context of an $\mathcal{A} = (S, \to)$, strategies can be seen as subsets E of the set of all execution paths $\Gamma^\omega_\mathcal{A} = \{(s_n)_{n=0}^\infty : s_n \to s_{n+1}\}$ of the system. In the following, we will write $\Gamma^\omega_{\mathcal{A},s}$ for the set of executions starting at $s \in S$ and $\Gamma^*_\mathcal{A}$ for the set of finite executions. This definition of strategy is sometimes called *abstract* or *extensional* [5] in contrast with an *intensional* characterization in terms of partial functions $\lambda : \Gamma^*_\mathcal{A} \to \mathcal{P}(S)$ that limits the possible next steps for a given execution prefix. Although these two definitions are not equivalent [5, 23], most common strategies can be expressed intensionally.

The properties about the system are expressed in terms of some tags declared for each state. This yields a Kripke structure $\mathcal{K} = (S, \to, AP, I, \ell)$ with a finite set of such atomic propositions AP, a finite set of initial states $I \subseteq S$, and a labeling function $\ell : S \to \mathcal{P}(AP)$. Temporal logics combine these properties with operators that describe how they occur in time. Well-known examples of such logics are CTL* and its sublogics LTL (Linear Temporal Logic) and CTL (Computational Tree Logic).

However, some other logics like the μ-calculus do not only refer to state properties but also to the transitions. The abstract setting needs then to be enriched with labels for them: *labeled transition systems* (LTS) are defined as triples (S, A, R) where A is a set of edge labels or actions and $R \subseteq S \times A \times S$ is a tagged relation. Strategies and executions are defined similarly, but in this case interleaving states with edge labels, i.e., $\Gamma^\omega_{\mathcal{A},s_0} = \{s_0(a_n s_n)_{n=1}^\infty : s_n \to^{a_{n+1}} s_{n+1}\}$.

Maude supports on-the-fly LTL model checking since its 2.0 version [15]. The mapping of a rewriting system to the model-checking framework is natural: its states are its terms and its transitions are rule applications. All executions are assumed to be infinite, by repeating the last state of finite executions, adding a loop transition to deadlock states, like in Spin and other verification tools. In order to prepare a Maude module for model checking, users need to extend it including the predefined `SATISFACTION` module, declaring the state sort, and the atomic propositions as regular Maude operators of sort `Prop`, and defining them equationally for all terms using the satisfaction relation symbol `_|=_`. Here is an example for the river crossing puzzle:

```
mod RIVER-CROSSING-PREDS is
  protecting RIVER-CROSSING .
  including SATISFACTION .

  subsort River < State .

  ops goal death bad : -> Prop [ctor] .

  var  R    : River .
  vars G G' : Group .

  eq left | G goat cabbage |= goal = true .
  eq R |= goal = false [owise] .

  eq cabbage G | G' goat |= death = false .
  eq cabbage goat G | G' |= death = false .
  eq R |= death = true [owise] .

  eq wolf goat G | G' shepherd |= bad = true .
  eq goat cabbage G | G' shepherd |= bad = true .
  eq R |= bad = false [owise] .
endm
```

Three properties are defined: `goal` that is only satisfied by the puzzle solution, `death` that tags states where someone has already been eaten, and `bad` that signals states in which eating is possible but not yet accomplished. Finally, the user should import the predefined `MODEL-CHECKER` module giving access to a special operator `modelCheck` that reduces to the verification result, assuming some decidability requirements [15].

```
Maude> red modelCheck(initial, [] (bad -> <> death)) .
rewrites: 44
result ModelCheckResult: counterexample(
   {right | left shepherd wolf goat cabbage,'alone}
    ...
   {left shepherd cabbage | right wolf goat,'cabbage},
   {left | right shepherd wolf goat cabbage,'alone}
   {left shepherd | right wolf goat cabbage,'alone})
```

In this case, the property is not satisfied and a counterexample execution is obtained, described by a cycle and a path to it.

Recently, we have extended the model checker to rewrite theories controlled by strategies [23]. From an abstract point of view, a system $\mathcal{K}$ controlled by a strategy $E \subseteq \Gamma^{\omega}_{\mathcal{K}}$ is said to satisfy a linear property φ if $\mathcal{K}, \pi \vDash \varphi$ for all $\pi \in E$. This definition is natural and almost unavoidable, since linear-time properties refer to individual executions quantified universally. The fundamental question is which are the executions E allowed by a Maude strategy language expression α.

This question has been answered by defining a nondeterministic structural operational semantics for the strategy language. Its execution states $q \in \mathcal{XS}$ are terms augmented with a continuation for the strategy execution, and its step $q \twoheadrightarrow q'$ correspond to single rule rewrites $\mathrm{cterm}(q) \to^1_R \mathrm{cterm}(q')$ on the underlying terms, denoted by $\mathrm{cterm}(q)$. States are usually of the form $t \,@\, s$ where s is a stack of strategy expressions whose execution is pending and substitutions defining the variable contexts of the active strategy calls, but more complex constructs are required for operators involving subsearches. Projecting the term part of the semantics executions leads to well-defined abstract strategies for the underlying system,

$$E(\alpha, t) = \{(\mathrm{cterm}(q_n))_{n=0}^{\infty} : q_0 = t \,@\, \alpha,\ q_n \twoheadrightarrow q_{n+1}\}.$$

Moreover, the abstract definition of model checking for this $E(\alpha, t)$ is equivalent to model checking the Kripke structure given by the semantics graph

$$\mathcal{B} := (\mathcal{XS}, \twoheadrightarrow, \{t \,@\, \alpha\}, AP, \ell \circ \mathrm{cterm})$$

under some decidability assumptions [23].

The strategy-aware model checker shares a great part of its infrastructure with the strategy execution engine and the original model checker. Their usage is similar, but in this case the `STRATEGY-MODEL-CHECKER` module, whose `modelCheck` symbol receives an additional argument to indicate the name of the strategy that controls the system, should be imported instead.

```
Maude> red modelCheck(initial, [] ~ bad, 'safe) .
rewrites: 54
result Bool: true

Maude> red modelCheck(initial, [] (bad -> O death),
                      'eagerEating) .
rewrites: 121
result Bool: true
```

A version of Maude including this model checker, its source code, and detailed documentation can be downloaded from [13].

4 Model checking branching-time properties

While the abstract definition for checking linear-time properties using strategies was very simple, the case of branching-time properties is not so clear. The main difficulty can be observed in the following example of a vending machine, which admits one-euro coins `e` and sells apples `a` and cakes `c` for one and two euros respectively. According to the semantics, the execution tree of the strategy $\alpha \equiv$ `(put1 ; apple) | (put1 ; put1 ; cake)` from the term `e e [empty]` is:

$$\begin{tikzcd}
 & \texttt{e [e]} @ \texttt{apple} \arrow[r] & \texttt{e a [empty]} @ \varepsilon \arrow[loop right] & \\
\texttt{e e [empty]} @ \alpha \arrow[ur] \arrow[dr] \arrow[ddd, "\mathrm{cterm}"'] & & & \\
 & \texttt{e [e]} @ \texttt{put1 ; cake} \arrow[d] & & \\
 & \texttt{empty [e e]} @ \texttt{cake} \arrow[r] & \texttt{c [empty]} @ \varepsilon \arrow[loop right] & \\
\texttt{e e [empty]} \arrow[r] & \texttt{e [e]} \arrow[r] \arrow[dr] & \texttt{e a [empty]} \arrow[loop right] & \\
 & & \texttt{empty [e e]} \arrow[r] & \texttt{c [empty]} \arrow[loop right]
\end{tikzcd}$$

We can see that the tree structure is not preserved by the projection. Its effects are observed in the satisfaction of the CTL property $\mathbf{A} \bigcirc \mathbf{E} \diamond \mathit{hasApple}$ where $\mathit{hasApple}$ is only satisfied if an apple has been bought. Moreover, expressions denoting the same abstract strategy, like `put1 ; (apple | put1 ; cake)` and the previous one, would satisfy different properties. Fortunately, the problem can be solved by simply merging successor states whose terms coincide, like `e [e]` @ `apple` and `e [e]` @ `put1 ; cake` in the example above.

In abstract terms, we suggested in [23] that the satisfaction of a branching-time property φ on a system $\mathcal{A}$ controlled by a strategy can be understood as the satisfaction of φ in its *unwinding*, the transition system whose states are the finite executions $\Gamma^*_{\mathcal{A},s}$ of the model and whose transitions are those allowed by the (intensional) strategy. However, since this construct is not finite, the practical usage of this definition goes through finding a bisimilar finite transition system. For the Maude strategy language, this can be the one derived from its nondeterministic semantics ($\mathcal{B}$) after merging states. We define it formally as $\mathcal{M} := (\mathcal{P}(\mathcal{XS}), [\twoheadrightarrow], \{\{t_0 \,@\, \alpha\}\}, AP, \ell \circ \mathrm{cterm})$ where

$$Q[\twoheadrightarrow]Q' \iff \exists\, t \in T_\Sigma \quad Q' = \{q' : q \twoheadrightarrow q', \mathrm{cterm}(q') = t, q \in Q\}.$$

When considering the labeled transition system, a slightly different Kripke structure $\mathcal{M}'$ should be defined, in which only the successors for a given rule label are included in each Q' state. The following proposition states that $\mathcal{M}$ is bisimilar to the strategy expansion:

Notation: Let $\mathcal{K} = (S, \to, I, AP, \ell)$ be a Kripke structure. $\Gamma^\omega_{\mathcal{K},s} := \{(s_n)_{n\in\mathbb{N}} : s_0 = s, s_n \to s_{n+1}\}$ is the set of non-terminating executions on $\mathcal{K}$ starting at s, and $\Gamma^\omega_{\mathcal{K}} = \bigcup_{s\in S} \Gamma^\omega_{\mathcal{K},s}$ is the set of all non-terminating executions. $\Gamma^*_{\mathcal{K},s}$ and $\Gamma^*_{\mathcal{K}}$ are defined similarly for finite executions, and the union of both are $\Gamma_{\mathcal{K},s}$ and $\Gamma_{\mathcal{K}}$. An abstract strategy E is a subset of $\Gamma_{\mathcal{K}}$. For any strategy E, a function can be defined $\lambda : S^+ \to \mathcal{P}(S)$ as $\lambda(w) := \{s \in S : \exists w' \in S^\infty \; wsw' \in E\}$. A strategy is intensional[2] if $E = E(\lambda)$ where $E(\lambda) := \{(s_k)_{k=1}^\infty : s_{k+1} \in \lambda(s_k)\}$. In general, we assume that all strategies we deal with are intensional.

Definition 1. *Given a Kripke structure $\mathcal{K} = (S, \to, I, AP, \ell)$ and a strategy $E = E(\lambda)$, we define its* unrolling *$\mathcal{U}(E) = (S^+, U, I, AP, \ell_{\mathrm{last}})$ where $(w, ws) \in U$ if $s \in \lambda(w)$ for all $w \in S^+$ and $\ell_{\mathrm{last}}(ws) = \ell(s)$ for all $w \in S^*$.*

[2] The abstract strategy $E(\lambda)$ generated by an intensional one λ is defined in a more complicated way in previous papers to represent finite executions faithfully. Here, we neglect this without consequences, since we assume that all executions are infinite for CTL* and μ-calculus does not distinguish them.

Notice that the executions in $\mathcal{U}(E)$ are of the form $(ws_0)(ws_0s_1)(ws_0s_1s_2)\cdots$. We will be interested in flattening them to executions of the underlying system $\mathcal{K}$ (in fact, in the strategy E). Hence, we write

$$\mathrm{flat}((ws_0)(ws_0s_1)(ws_0s_1s_2)\cdots) \coloneqq s_0s_1s_2\cdots$$

Lemma 1. *For every $w \in S^+$ prefix in E, $E \upharpoonright w = \{\mathrm{flat}(\pi) : \pi \in \Gamma_{\mathcal{U}(E),w}\}$.*

Proof. For arbitrary $\Gamma_{\mathcal{U}(E),ws_0} \ni \pi = (ws_0)(ws_0s_1)(ws_0s_1s_2)\cdots$, $\mathrm{flat}(\pi) = s_0s_1s_2\cdots$ and $ws_0s_1s_2\cdots \in E$ since $s_{n+1} \in \lambda(s_n)$. Hence $\mathrm{flat}(\pi) = s_0s_1\cdots \in E \upharpoonright w$ by definition. Reciprocally, for arbitrary $s_0s_1\cdots \in E \upharpoonright w$, $ws_0s_1\cdots \in E$, so $\pi = (ws_0)(ws_0s_1)\cdots \in \Gamma_{\mathcal{U}(E),w}$ and $\mathrm{flat}(\pi) = s_0s_1\cdots$.

Usually we are only interested in a subgraph of a Kripke structure that is reachable from a given state. For $\mathcal{K}$ and $s \in S$, we write $\mathcal{K}_s = (R, U|_R, \{s\}, \ell|_R)$ where $R = \{s' \in S : s \to^* s'\}$.

Proposition 1. *Given an expression α of the Maude strategy language, there is a bisimulation R between $\mathcal{U}(E(\alpha))_t$ and $\mathcal{M}_{t@\alpha}$. Moreover, if $(w, Q) \in R$ then $\ell_{\mathrm{last}}(w) = \ell(\mathrm{cterm}(Q))$.*

Proof. Let $f : S^+ \to \mathcal{P}(\mathcal{XS})$ be $f(s_1\cdots s_n) = \{q_n \in \mathcal{XS} : q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n, \mathrm{cterm}(q_k) = s_k\}$. Our goal is proving that the graph of this function, $(w, Q) \in R$ iff $f(w) = Q$, is the required bisimulation. Clearly, $\ell_{\mathrm{last}}(ws) = \ell(s) = \ell(\mathrm{cterm}(Q))$ if $(ws, Q) \in R$.

First, we claim that $f(w)\,[\twoheadrightarrow]\,f(ws)$ for all $w \in S^+$ and $s \in S$ if $f(w) \neq \emptyset \neq f(ws)$. In fact, for $w = s_1\cdots s_n$,

$$\begin{aligned} f(ws) &= \{q \in \mathcal{XS} : q_1 \twoheadrightarrow \cdots \twoheadrightarrow q_n \twoheadrightarrow q, \mathrm{cterm}(q) = s\} \\ &= \{q \in \mathcal{XS} : q_n \twoheadrightarrow q, q_n \in f(w), \mathrm{cterm}(q) = s\} \end{aligned}$$

and this is the definition of $f(w)\,[\twoheadrightarrow]\,f(ws)$ whenever $f(ws) \neq \emptyset$. Let M be the set of states of $\mathcal{M}_{t@\alpha}$ and $E_* = \{w : \exists w' \in S^\omega\ ww' \in E\}$ the finite prefixes of the executions in E, then we known $f(E_*) \subseteq M$. In order to prove $M \subseteq f(E_*)$, suppose $Q_1 = \{t\,@\,\alpha\}\,[\twoheadrightarrow]\cdots[\twoheadrightarrow]Q_n$, we will see $Q_n = f(\mathrm{cterm}(Q_1)\cdots\mathrm{cterm}(Q_n))$ by induction. In the base case $n = 1$, $Q_n = \{t\,@\,\alpha\}$ and coincides with $f(t)$. Otherwise, we have $Q_{n-1}\,[\twoheadrightarrow]\,Q_n$ and by induction hypothesis $Q_{n-1} = f(\mathrm{cterm}(Q_1)\cdots\mathrm{cterm}(Q_{n-1}))$. By $[\twoheadrightarrow]$, there is a term t such that $Q_n = \{q' \in \mathcal{XS} : q \twoheadrightarrow q', q \in Q_{n-1}, \mathrm{cterm}(q') = t\}$, but from the equality above, this is $f(\mathrm{cterm}(Q_1)\cdots\mathrm{cterm}(Q_{n-1})t)$ and $t = \mathrm{cterm}(Q_n)$.

Since the previous paragraph states that $f(E_*) = M$, the relation R given by its graph is well-defined in $E_* \times M$. Moreover, if the successors of $w \in E_*$ are $ws \in E_*$ for some $s \in S$, the successors of $f(w) \in \mathcal{P}(\mathcal{XS})$ are $f(ws)$ for the same states s. In fact, we know $f(w)\,[\twoheadrightarrow]\,f(ws)$ and that every Q such that $f(w)\,[\twoheadrightarrow]\,Q$ is $f(ws)$ for some $s \in S$. Thus, R is a bisimulation.

Hence, to model check state-based properties of system controlled by strategies, we propose applying standard algorithms on $\mathcal{M}$. A similar proposition holds for the labeled variant $\mathcal{M}'$ and $\mathcal{U}' = ((S \cup A)^*, U')$ where $(wt)\,U'\,(wtat')$ iff $\exists w' \in (S \cup A)^\omega\ wtat'w' \in E_{\mathrm{labeled}}(\alpha, s)$. For action-based or doubly-labeled logics, we propose using $\mathcal{M}'$ instead. In the following sections, to justify that the proposed procedure is meaningful, we give reasonable generalizations of two specific branching-time logics for strategy-controlled systems, and show how their notions of satisfaction coincide. CTL* and μ-calculus are chosen because they are well-known and because the tool used only handles those, but the procedure is general and can be applied to other logics.

4.1 CTL*

The proposed generalized definition is similar to those we can find in most reference textbooks [7, 8] and coincides with a previous definition for trees [27]. We identify abstract strategies with trees since they are in univocal relation as long as trees only branch to distinct children, as it is the case. For an execution $\pi = (\pi_n)_{n=0}^{\infty}$, we denote the suffix that starts at the position k by $\pi^k = (\pi_{k+n})_{n=0}^{\infty}$, the prefix that stops at k by $\pi^{-k} = \pi_0 \cdots \pi_k$, and all the executions of a given abstract strategy E continuing a given prefix by $E \upharpoonright ws = \{s\pi : ws\pi \in E\}$ for all $w \in S^*$ and $s \in S$.

1.	$E \vDash p$	iff $\forall\, \pi \in E \quad p \in \ell(\pi_0)$
2.	$E \vDash \neg\Phi$	iff $E \nvDash \Phi$
3.	$E \vDash \Phi_1 \wedge \Phi_2$	iff $E \vDash \Phi_1$ and $E \vDash \Phi_2$
4.	$E \vDash \mathbf{A}\,\phi$	iff $\forall\, \pi \in E \quad E \upharpoonright \pi_0, \pi \vDash \phi$
5.	$E \vDash \mathbf{E}\,\phi$	iff $\exists\, \pi \in E \quad E \upharpoonright \pi_0, \pi \vDash \phi$
6.	$E, \pi \vDash \Phi$	iff $E \vDash \Phi$
7.	$E, \pi \vDash \neg\varphi$	iff $E, \pi \nvDash \varphi$
8.	$E, \pi \vDash \varphi_1 \wedge \varphi_2$	iff $E, \pi \vDash \varphi_1$ and $E, \pi \vDash \varphi_2$
9.	$E, \pi \vDash \bigcirc\,\varphi$	iff $E \upharpoonright \pi_0\pi_1, \pi^1 \vDash \varphi$
10.	$E, \pi \vDash \diamond\,\varphi$	iff $\exists\, n \geq 0 \quad E \upharpoonright \pi^{-n}, \pi^n \vDash \varphi$
11.	$E, \pi \vDash \square\,\varphi$	iff $\forall\, n \geq 0 \quad E \upharpoonright \pi^{-n}, \pi^n \vDash \varphi$
12.	$E, \pi \vDash \varphi_1\,\mathbf{U}\,\varphi_2$	iff $\exists\, n \geq 0 \;\; E \upharpoonright \pi^{-n}, \pi^n \vDash \varphi_2 \;\wedge\; \forall\, 0 \leq k < n \;\; E \upharpoonright \pi^{-k}, \pi^k \vDash \varphi_1$

Observe that it only differs from the classical definition in the fact that the strategy is carried on. Path formulae φ are understood similarly, but here, a state property Φ does not only depend on the state but on the full state history. The extended and classical relations are linked by the following essential property:

Proposition 2. *Given a CTL* formula φ, $\mathcal{K}, s \vDash \varphi$ iff $\mathcal{K}, \Gamma_s^\omega \vDash \varphi$.*

Proof. Since the executions are unrestricted, $\Gamma_s^\omega \upharpoonright (s \cdots s') = \Gamma_{s'}^\omega$. Using this fact and other immediate arguments, the definition for strategies coincides almost syntactically with the standard one.

The following proposition justifies that model checking can be solved by the classical procedures applied on $\mathcal{M}$:

Proposition 3. $E(\alpha, t) \vDash \varphi \iff \mathcal{M}, \{t\,@\alpha\} \vDash \varphi$

Proof. We will show an inductive proof on the structure of CTL* formulae of the more general property $\mathcal{U}(E), w \vDash \varphi$ iff $\mathcal{K}, E \upharpoonright w \vDash \varphi$ for all $w \in S^+$. We need to handle path formulae simultaneously, so the inductive property also includes $\mathcal{U}(E), \pi \vDash \varphi$ iff $\mathcal{K}, E \upharpoonright \pi_0, \mathrm{flat}(\pi) \vDash \varphi$. Notice that in the left-hand side executions are successions of growing S^+ words while in the right-hand side they are successions of S states. To facilitate reading, we will omit the $\mathcal{U}(E)$ and $\mathcal{K}$ prefix when writing the satisfaction relations.

- (p, atomic propositions) By definition, $ws \vDash p$ iff $p \in \ell(s)$, and $E \upharpoonright ws \vDash p$ iff $p \in \ell(s')$ for all $s'w' \in E \upharpoonright ws = \{sw'' : wsw'' \in E\}$. Then, s' can only be s and both conditions coincide.
- ($\Phi_1 \wedge \Phi_2$) In the standard side, the conjunction is satisfied iff $w \vDash \Phi_i$ for both $i = 1, 2$. In the strategy side, this happens iff $E \upharpoonright w \vDash \Phi_i$. By induction hypothesis on Φ_i the equivalence holds.
- ($\neg\Phi$) The same inductive argument can be used for negation.

– ($\mathbf{A}\,\varphi$) This formula is satisfied iff $\pi \vDash \varphi$ for all $\pi \in \Gamma^\omega_{\mathcal{U}(E),w}$ in the $\mathcal{U}(E)$ side. In the strategy side, this is $E \upharpoonright w, \rho \vDash \varphi$ for all $\rho \in E \upharpoonright w$. Using Lemma 1, all these ρ are exactly those $\mathrm{flat}(\pi)$, and applying the induction hypothesis on φ, we get that both statements are equivalent.

Let π be $(ws_0)(ws_0s_1)\cdots$, we then target the path satisfaction cases:

– ($\bigcirc\,\varphi$) We should prove that $\pi \vDash \bigcirc\,\varphi$ is equivalent to $E \upharpoonright ws_0, s_0s_1 \cdots \vDash \bigcirc\,\varphi$. Their definitions translate these to $\pi^1 \vDash \varphi$ and $(E \upharpoonright ws_0) \upharpoonright s_0s_1, (s_0s_1\cdots)^1 \vDash \varphi$. But they are equivalent by induction hypothesis on φ, since $(E \upharpoonright ws_0) \upharpoonright s_0s_1 = E \upharpoonright ws_0s_1 = E \upharpoonright \pi_1 = E \upharpoonright (\pi^1)_0$ and $(s_0s_1\cdots)^1 = s_1s_2\cdots = \mathrm{flat}(\pi^1)$.
– ($\varphi_1\,\mathcal{U}\,\varphi_2$) The formula holds in the standard sense if there is an $n \in \mathbb{N}$ such that $\pi^n \vDash \varphi_2$ and for all k such that $0 \leq k < n$ then $\pi^k \vDash \varphi_1$. In the strategy side, the formula holds if again there is an $n \in \mathbb{N}$ such that $(E \upharpoonright ws_0) \upharpoonright s_1s_2\cdots s_n, s_ns_{n+1}\cdots \vDash \varphi_2$ and $(E \upharpoonright ws_0) \upharpoonright s_0s_1\cdots s_k, s_ks_{k+1}\cdots \vDash \varphi_1$ for all $0 \leq k < n$. Since $(E \upharpoonright ws_0) \upharpoonright s_0s_1\cdots s_k = E \upharpoonright ws_0s_1\cdots s_k = E \upharpoonright (\pi^k)_0$ and $s_ks_{k+1}\cdots = \mathrm{flat}(\pi^k)$ for all $k \in \mathbb{N}$, the induction hypothesis can be applied to φ_1 and φ_2 to conclude the property for $\varphi_1\,\mathcal{U}\,\varphi_2$.
– (Φ) $\pi \vDash \Phi$ is defined as $\pi_0 \vDash \Phi$ in the standard sense, and $E \upharpoonright ws_0, s_0s_1\cdots \vDash \Phi$ is $E \upharpoonright ws_0 \vDash \Phi$ in the strategy case. Since $\pi_0 = ws_0$, both statements are related as in the induction property. We can apply the hypothesis on Φ itself considering that state satisfaction is below path satisfaction (we never apply this argument in reverse), and then they are equivalent.

Only a complete subset of CTL* constructors has been handled in the proof, but simple propositional and first-order properties let us conclude that the following well-know semantic equivalences are also satisfied by the given extended CTL* definition for strategies:

– $\Phi_1 \vee \Phi_2 \equiv \neg(\neg\Phi_1 \wedge \neg\Phi_2)$ for any state or path formula Φ.
– $\mathbf{E}\,\varphi \equiv \neg(\mathbf{A}\,\neg\varphi)$ for any path formula φ.
– $\diamond\,\varphi \equiv \top\,\mathcal{U}\,\varphi$ for any path formula φ.
– $\Box\,\varphi \equiv \neg(\diamond\,\neg\varphi)$ for any path formula φ.

4.2 μ-calculus

We present a generalized definition of μ-calculus for strategies that mimics the original one. While in the original μ-calculus a valid formula φ is given meaning $[\![\varphi]\!]_\eta$ as the set of states in which it is satisfied, here, a formula will denote instead a set $\langle\!\langle\varphi\rangle\!\rangle_\eta$ of subtrees (in other words, strategies) in which φ is satisfied. Let η be an assignment from variables Z in the formula to subsets of $\mathcal{P}(\Gamma^\omega_{\mathcal{K}})$:

1. $\langle\!\langle p\rangle\!\rangle_\eta = \{T \subseteq \Gamma^\omega_{\mathcal{K}} : \forall sa\pi \in T \quad p \in \ell(s)\}$
2. $\langle\!\langle \neg\varphi\rangle\!\rangle_\eta = \mathcal{P}(\Gamma^\omega_{\mathcal{K}}) \backslash \langle\!\langle\varphi\rangle\!\rangle_\eta$
3. $\langle\!\langle \varphi_1 \wedge \varphi_2\rangle\!\rangle_\eta = \langle\!\langle\varphi_1\rangle\!\rangle_\eta \cap \langle\!\langle\varphi_2\rangle\!\rangle_\eta$
4. $\langle\!\langle Z\rangle\!\rangle_\eta = \eta(Z)$
5. $\langle\!\langle \langle a\rangle\varphi\rangle\!\rangle_\eta = \{T \subseteq \Gamma^\omega_{\mathcal{A}} : \exists\, sa\pi \in T \quad T \upharpoonright sa\pi_0 \in \langle\!\langle\varphi\rangle\!\rangle_\eta\}$
6. $\langle\!\langle \nu Z.\varphi\rangle\!\rangle_\eta = \bigcup \{F \subseteq \mathcal{P}(\Gamma^\omega_{\mathcal{A}}) : F \subseteq \langle\!\langle\varphi\rangle\!\rangle_{\eta[Z/F]}\}$

Other constructors like $[a]\varphi$ and $\mu Z.\varphi$ are defined by their usual equivalences to these. Provided that every variable is under an even number of negations, the definition is monotone and the fixpoints are well-defined. When the formula φ is ground, i.e. it does not have free variables, we omit the valuation subscript η. This generalization is connected with the original definition by the property:

Proposition 4. *Given a ground μ-calculus formula φ, $\langle\!\langle \varphi \rangle\!\rangle_{\mathcal{K},\eta} \ni s$ iff $\Gamma_{\mathcal{K},s} \in \langle\!\langle \varphi \rangle\!\rangle_{\mathcal{K},\eta}$ for any η and ξ.*

Proof. This property can be proven inductively, adding to the inductive property the premise that $\eta(Z) \ni s$ iff $\Gamma_s \in \xi(Z)$ for all variable Z. For the initial φ, this premise is trivially satisfied since we can take $\eta(Z) = \emptyset = \xi(Z)$ regardless of the given two, since the formula is closed. We will not detail some trivial cases:

- (p) By definition, $s \in \langle\!\langle p \rangle\!\rangle_\eta$ is $p \in \ell(s)$ and $\Gamma_s \in \langle\!\langle p \rangle\!\rangle_\xi$ is $\forall \pi \in \Gamma_s \; p \in \ell(\pi_0)$. Since Γ_s are the executions of $\mathcal{K}$ starting at s, $\pi_0 = s$ and both statements are equivalent.
- $(\langle a \rangle \varphi)$ $s \in \langle\!\langle \langle a \rangle \varphi \rangle\!\rangle_\eta$ if there is a $s' \in S$ such that $s \to^a s'$ and $s' \in \langle\!\langle \varphi \rangle\!\rangle_\eta$. On the other side, $\Gamma_s \in \langle\!\langle \langle a \rangle \varphi \rangle\!\rangle_\xi$ holds iff there is $saw \in \Gamma_s$ such that $\Gamma_s \upharpoonright saw_0 = \Gamma_{w_0} \in \langle\!\langle \varphi \rangle\!\rangle_\xi$. The induction hypothesis taking $s' = w_0$ let us conclude the property.
- $(\nu Z.\varphi)$ $s \in \langle\!\langle \nu Z.\varphi \rangle\!\rangle_\eta$ if there is a set V such that $s \in V$ and $V \subseteq \langle\!\langle \varphi \rangle\!\rangle_{\eta[Z/V]}$. In the strategy side, $\Gamma_s \in \langle\!\langle \nu Z.\varphi \rangle\!\rangle_\xi$ iff there is an F such that $\Gamma_s \in F$ and $F \subseteq \langle\!\langle \varphi \rangle\!\rangle_{\xi[Z/F]}$. Both implications of the equivalence can be proven like in the previous proposition, but taking $F = \{\Gamma_s : s \in V\}$ for a given V, and $V = \{s \in S : \Gamma_s \in F\}$ for a given F.

As for CTL*, the following proposition claims that a formula is satisfied for a strategy in the generalized sense iff it is satisfied in the merged labeled transition system generated by the nondeterministic semantics:

Proposition 5. *Given a ground μ-calculus formula φ, $[\![\varphi]\!]_{\mathcal{U}'(E),\xi} \ni t \;@\alpha$ iff $E \in \langle\!\langle \varphi \rangle\!\rangle_{\mathcal{K},\eta}$ for any η and ξ.*

Proof. We will prove the more general property that $\langle\!\langle \varphi \rangle\!\rangle_\eta \ni w$ iff $E \upharpoonright w \in \langle\!\langle \varphi \rangle\!\rangle_\xi$ provided that $\eta(Z) \ni w$ iff $E \upharpoonright w \in \xi(Z)$ for all variable Z.

- (p) By definition, $ws \in \langle\!\langle \varphi \rangle\!\rangle_\eta$ iff $p \in \ell(s)$. On the other side, $E \upharpoonright ws \in \langle\!\langle \varphi \rangle\!\rangle_\xi$ iff $p \in \ell(\pi_0)$ for all $\pi \in E \upharpoonright ws$. However, π_0 must be s since $E \upharpoonright ws = \{sw' : wsw' \in E\}$, so both sides are equivalent.
- (Z) The value of Z in both contexts is respectively $\eta(Z)$ and $\xi(Z)$, so the property directly follows from the assumption over these two functions.
- $(\varphi_1 \wedge \varphi_2)$ The standard definition says $\langle\!\langle \varphi_1 \wedge \varphi_2 \rangle\!\rangle_\eta = \langle\!\langle \varphi_1 \rangle\!\rangle_\eta \cap \langle\!\langle \varphi_2 \rangle\!\rangle_\eta$ and the strategy one is $\langle\!\langle \varphi_1 \wedge \varphi_2 \rangle\!\rangle_\xi = \langle\!\langle \varphi_1 \rangle\!\rangle_\xi \cap \langle\!\langle \varphi_2 \rangle\!\rangle_\xi$. Hence, the property holds by induction hypothesis on φ_1 and φ_2.
- $(\neg\varphi)$ By definition, $\langle\!\langle \neg\varphi \rangle\!\rangle_\eta = S^+ \backslash \langle\!\langle \varphi \rangle\!\rangle_\eta$ and $\langle\!\langle \neg\varphi \rangle\!\rangle_\xi = \mathcal{P}(\Gamma_\mathcal{K}) \backslash \langle\!\langle \varphi \rangle\!\rangle_\xi$, so the property holds by induction hypothesis on φ.
- $(\langle a \rangle \varphi)$ $ws \in \langle\!\langle \langle a \rangle \varphi \rangle\!\rangle_\eta$ iff there is an $(a, s') \in \lambda(ws)$ such that $wsas' \in \langle\!\langle \varphi \rangle\!\rangle_\eta$ according to the standard definition of μ-calculus and the transition relation on $\mathcal{U}'(E)$. On the other side, $E \upharpoonright ws \in \langle\!\langle \langle a \rangle \varphi \rangle\!\rangle_\xi$ if there is a $w' \in (S \cup A)^\infty$ such that $saw' \in E \upharpoonright ws$ and $(E \upharpoonright ws) \upharpoonright saw'_0 = E \upharpoonright wsaw'_0 \in \langle\!\langle \varphi \rangle\!\rangle_\xi$.
  By definition of λ and $E(\lambda)$, there is a $w' \in (S \cup A)^\infty$ such that $wsaw' \in E$ iff $(a, w'_0) \in \lambda(ws)$. Hence, by induction hypothesis on φ and taking $w'_0 = s'$, we conclude that the property holds.
- $(\nu Z.\varphi)$ According to the standard definition, $ws \in \langle\!\langle \nu Z.\varphi \rangle\!\rangle_\eta$ if there is a $V \subseteq S^+$ such that $V \subseteq \langle\!\langle \varphi \rangle\!\rangle_{\eta[Z/V]}$ and $ws \in V$. According to our definition for strategies, $E \upharpoonright ws \in \langle\!\langle \nu Z.\varphi \rangle\!\rangle_\xi$ iff there is an $F \subseteq \mathcal{P}(\Gamma_\mathcal{K})$ such that $F \subseteq \langle\!\langle \varphi \rangle\!\rangle_{\xi[Z/F]}$ and $E \upharpoonright ws \in F$.
  Assuming there exists a V with these properties ($\Rightarrow$), consider $F = \{E \upharpoonright w : w \in V\}$. The definition is precisely $w \in V$ iff $E \upharpoonright w \in F$, so $\eta[Z/V]$ and $\xi[Z/F]$ are properly related. Hence, by induction hypothesis on φ, $E \upharpoonright w \in \langle\!\langle \varphi \rangle\!\rangle_{\xi[Z/F]}$ iff $w \in \langle\!\langle \varphi \rangle\!\rangle_{\eta[Z/V]}$, so $F \subseteq \langle\!\langle \varphi \rangle\!\rangle_{\xi[Z/F]}$ as we wanted to prove. In the opposite direction ($\Leftarrow$), assuming the existence of an F with the mentioned properties, consider $V = \{w \in S^+ : E \upharpoonright w \in F\}$ and the proof is the same.

5 The Maude language module for LTSmin

According to the previous section, to check CTL* or μ-calculus properties on Maude specifications we should take the Kripke structure $\mathcal{B}$, already generated for the LTL model checker, merge its states as in $\mathcal{M}$ or $\mathcal{M}'$ and apply the standard algorithms on them. To avoid programming these algorithms for scratch, we have developed instead a language module for the language-independent model checker LTSmin [16]. Oversimplifying, this software allows defining *language frontends* that expose programs in a specification language like Maude as a labeled transition system to some builtin *algorithmic backends*, including model checkers for different logics. The Kripke-like C interface is called PINS (Partitioned Next State Interface) and promotes sharing additional information about the internal structure of the models to speed up algorithms. Frontends are included for various modeling formalisms like Promela, PNML, DIVINE, UPPAAL, etc., and custom language modules, like ours, can also be loaded by the LTSmin tools using the POSIX's `dlopen` API.

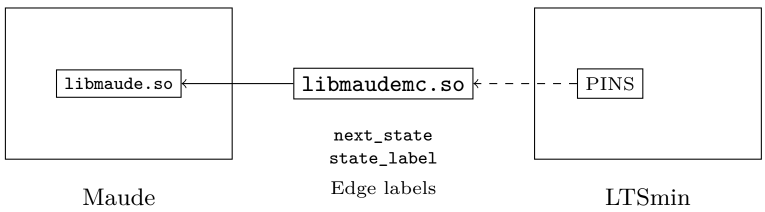

Fig. 1. Architecture of the Maude LTSmin plugin

The language module is the C library `libmaudemc.so` illustrated in Fig. 1. On the one hand, the module is linked with the C++ implementation of Maude 3.0 including the extended LTL model checker for strategy-controlled systems, which processes the Maude files and gives access to the transition system used for LTL model checking. On the other hand, the plugin implements the PINS interface by exporting some C functions that the LTSmin model-checking algorithms will call to introspect the model[3]: the `next_state` function provides the successors of a given state, including their edge labels, and `state_label` tests whether an atomic proposition holds in a state. The module itself takes care of merging states as required for CTL* and μ-calculus, and also removes states in which the strategy has failed, which were ignored automatically by the nested depth-first search of LTL model checking, but must be explicitly purged here. LTSmin lets frontend designers represent states as vectors of integers, which can be partitioned and whose dependencies can be declared as matrices that the algorithms may use to improve their efficiency and allow distributed implementations. However, in our case the state is a single integer that represents an internal state of the Maude model checker, since partitioning and inferring relations about arbitrary Maude specifications seems unpractical.

LTSmin includes different commands like `pins2lts-seq` for explicit state LTL model checking or `pins2lts-sym` for symbolic CTL/CTL*/μ-calculus that, once our module is loaded with `--loader=libmaudemc.so`, are ready to handle Maude specifications. The Maude source file, the initial term, and an optional strategy expression have to be provided as arguments to the command. A helper script `maude2lts` has been written to call the appropriate tool and configuration

[3] LTSmin loads `libmaudemc.so` using the `dlopen` API, which allows loading dynamic libraries at runtime, accessing their symbols, and calling their C functions.

for the appropriate formula among those supported by LTSmin: `invariant`, `ctl`, `ctl-star` and `mu`, which are documented in its webpage.

For example, after downloading the plugin from http://maude.ucm.es/strategies and LTSmin from https://ltsmin.utwente.nl, we can check the CTL property that every state of the river crossing puzzle can be continued to a solution $\mathbf{A}\,\Box\,\mathbf{E}\,\Diamond\, goal$. This formula is satisfied when the system is controlled by the `safe` strategy, but not when using the `eagerEating` strategy or when the system runs uncontrolled.

```
$ maude2lts river.maude initial --strat safe
    --aprops=goal --ctl 'A [] E <> goal'
maude-mc: selected module is RIVER-CROSSING-SCHECK
pins2lts-sym: Formula A [] E <> goal holds
              for the initial state
maude-mc: 16 system states explored, 110 rewrites

$ maude2lts river.maude initial --strat eagerEating
    --aprops goal --ctl 'A [] E <> goal'
pins2lts-sym: Formula ... does not hold ...
maude-mc: 43 system states explored, 314 rewrites

$ maude2lts river.maude initial
    --aprops goal --ctl 'A [] E <> goal'
pins2lts-sym: Formula ... does not hold ...
maude-mc: 36 system states explored, 322 rewrites
```

However, the property $\mathbf{A}\,\Box\,(bad \vee death \vee \mathbf{E}\,\Diamond\, goal)$ holds under the `eagerEating` strategy.

```
$ maude2lts river.maude initial --strat eagerEating
    --aprops goal:bad:death
    --ctl 'A [] (bad || death || E <> goal)'
pins2lts-sym: Formula ... holds ...
maude-mc: 43 system states explored, 658 rewrites
```

Since LTL properties can be checked both directly from Maude or using the LTSmin plugin. Against the model-checking examples available in our web page [13], LTSmin is 10,73% slower in average (or 11,21% using its builtin caching) and requires more memory. However, the communication costs and the partially redundant representation of the state can explain this difference. Moreover, since the PINS interface asks for all the successors of a state at once, the on-the-fly state space expansion is lazier in Maude and the order in which children are processed is reversed. The size of the property automata generated from the formulae by both tools coincide, except in one case when Maude's is one state smaller.

Other alternatives to bring CTL* and μ-calculus model checking to Maude have been considered like generating an equivalent model for a specific tool or exporting it to a somehow standard representation. For example, the Model Checking Contest uses the Petri Net Markup Language (PNML) to state the problems for all the competitor tools. In fact, we first wrote a metalevel prototype that outputs a model for the NuSMV model checker. Finally, we decided to use LTSmin because its interface is closer to our description of the transition system, and because of its live connection that allows generating the space state and checking propositions on the fly. Only LTL model checking, which was already covered, can benefit from the first advantage, but the second is always useful.

6 Related work

We have already commented in each section on related work for each topic, but we should also mention that other model checkers have been developed for Maude specifications, like a timed CTL model checker for Real-Time Maude [18], another for the more expressive Linear Temporal Logic of Rewriting [3] (LTLR), and an abstract logical model checker [2] using narrowing instead of rewriting.

The combination of strategies and model checking is not original. In the field of multiplayer games, various logics like ATL* [1] and *strategy logic* [21] have been proposed to reason about player strategies. Other logics like mCTL* [17] are extended to take past actions into account. However, our approach is different, since strategies are part of the specification of the model, keeping the property specification unaltered.

7 Conclusions

In this paper, the study of model checking for systems controlled by strategies is extended to branching-time properties, and a tool is presented to widen the range of properties that can be checked against standard Maude specifications and strategy-controlled ones. In a more general sense, this work aims to make strategies a more useful and convenient choice to specify and verify systems. While strategy-free models can be fully explored at the metalevel using the `metaXApply` function, there were no resources in the current metalevel to follow step by step the execution of a strategy, without implementing them from scratch. Our plugin exposes these Maude models to external tools for verification, visualization and other types of analysis.

References

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

[2] Kyungmin Bae, Santiago Escobar, and José Meseguer. Abstract logical model checking of infinite-state systems using narrowing. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands*, volume 21 of *LIPIcs*, pages 81–96. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. ISBN: 978-3-939897-53-8.

[3] Kyungmin Bae and José Meseguer. Model checking linear temporal logic of rewriting formulas under localized fairness. *Science Computer Programming*, 99:193–234, 2015.

[4] 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.

[5] 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.

[6] 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.

[7] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The Mit Press, 1999.

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

[9] Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott. *Maude Manual v3.0*. December 2019.

[10] Manuel Clavel and José Meseguer. Internal strategies in a reflective logic. In Bernhard Gramlich and Hélène Kirchner, editors, *Proceedings of the CADE-14 Workshop on Strategies in Automated Deduction*, pages 1–12, Townsville, Australia, 1997.

[11] Manuel Clavel and José Meseguer. Reflection and strategies in rewriting logic. In José Meseguer, editor, *Proceedings of the First International Workshop on Rewriting Logic and its Applications, WRLA'96, Asilomar, California, September 3-6, 1996*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 126–148. Elsevier, 1996.

[12] Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott. Programming and symbolic computation in Maude. *Journal of Logical and Algebraic Methods in Computer Programming*, 110, 2020.

[13] Steven Eker, Narciso Martí-Oliet, José Meseguer, Isabel Pita, Rubén Rubio, and Alberto Verdejo. Strategy language for Maude. URL: `http://maude.ucm.es/strategies`.

[14] 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.

[15] 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.

[16] Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: high-performance language-independent model checking. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 692–707. Springer, 2015. ISBN: 978-3-662-46680-3.

[17] 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.

[18] Daniela Lepri, Erika Ábrahám, and Peter Csaba Ölveczky. Sound and complete timed CTL model checking of timed Kripke structures and real-time rewrite theories. *Sci. Comput. Program.*, 99:128–192, 2015.

[19] 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.

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

[21] 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.

[22] Miguel Palomino, Narciso Martí-Oliet, and Alberto Verdejo. Playing with Maude. In Slim Abdennadher and Christophe Ringeissen, editors, *Proceedings of the 5th International Workshop on Rule-Based Programming, RULE 2004, Aachen, Germany, June 1, 2004*, volume 124 of number 1 in *Electronic Notes in Theoretical Computer Science*, pages 3–23. Elsevier, 2005.

[23] 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.

[24] Rubén Rubio, Narciso Martí-Oliet, Isabel Pita, and Alberto Verdejo. Parameterized strategies specification in Maude. In José Fiadeiro and Ionuț Țuțu, editors, *Recent Trends in Algebraic Development Techniques. 24th IFIP WG 1.3 International Workshop, WADT 2018, Egham, UK,*

July 2–5, 2018, Revised Selected Papers, volume 11563 of *Lecture Notes in Computer Science*, pages 27–44. Springer, 2019.

[25] Beatriz Santos-Buitrago, Adrián Riesco, Merrill Knapp, José Carlos R. Alcantud, Gustavo Santos-García, and Carolyn L. Talcott. Soft set theory for decision making in computational biology under incomplete information. *IEEE Access*, 7:18183–18193, 2019.

[26] Gustavo Santos-García and Miguel Palomino. Solving Sudoku puzzles with rewriting rules. In 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*, pages 79–93. Elsevier, 2007.

[27] Wolfgang Thomas. Computation tree logic and regular ω-languages. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 690–713. Springer, 1989. ISBN: 3-540-51080-X.

[28] 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.