The Maude strategy language

Narciso Martí-Oliet
(joint work with Isabel Pita, Rubén Rubio, Alberto Verdejo)

Facultad de Informática, Universidad Complutense de Madrid

IFIP WG 1.3, Massa Marittima, Italy, January 2020

Talk plan

Recent advances on the Maude strategy language
and model checking

1. Introduction to Maude
2. Maude strategy language
3. Model checking systems controlled by strategies

Maude

```
\||||||||||||||||||/
--- Welcome to Maude ---
/||||||||||||||||||\
```

http://maude.cs.illinois.edu

- Maude is a high-level language and high-performance system.
- It supports both equational and rewriting logic computation.
- It is a flexible and general semantic framework for giving semantics to a wide range of languages and models of concurrency.
- It is also a good logical framework, i.e., a metalogic in which many other logics can be naturally represented and implemented.
- Moreover, it is reflective allowing many advanced metaprogramming and metalanguage applications.

Maude specifications

Rewriting strategies	Strategy modules
Rewriting rules	System modules
Terms and equations	Functional modules

Maude specifications

Functional modules define membership equational logic theories.

- Order-sorted signature $\Omega = (K, \Sigma, S)$.
- Equations and membership axioms:

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

- Operator axioms, like commutativity, associativity, and identity.

Maude specifications

Functional modules define membership equational logic theories.

```
fmod RIVER is
  sort 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
```

Maude specifications

System modules are rewriting logic theories.

- $\mathcal{R} = (\Sigma, E \cup A, R)$ adds rewriting rules R on top of the equational theory.
- Rules do not have to be either confluent or terminating.

$$(\forall X) \quad t \Rightarrow t' \quad \text{if} \quad \bigwedge_i u_i = v_i \wedge \bigwedge_j u_j : s_j \wedge \bigwedge_k u_k \Rightarrow v_k$$

Maude specifications

System modules are rewriting logic theories.

```
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
```

Strategy language

- Maude provides commands **`rewrite`** and **`frewrite`** to obtain a single rule execution path, and **`search`** to get all of them.
- But the user may be interested in obtaining those paths satisfying a given constraint. Then, strategies are needed.
- Strategy α is seen as an operation transforming a term t into a set of terms, since the process is nondeterministic in general.
- Strategies can be executed with the command **`srewrite`** t **`using`** α.
- The most basic strategy is rule application

$$\texttt{top(label[}x_1 \leftarrow t_1\texttt{, ..., }x_n \leftarrow t_n\texttt{]\{}\alpha_1, \ldots, \alpha_k\texttt{\})}$$

The Maude strategy language

- Rule applications

$$\texttt{top}(\mathit{rlabel}[x_1 \to t_1, \ldots, x_n \to t_n]\{\alpha_1, \ldots, \alpha_m\})$$

- Tests

$$\texttt{match}\ P\ \texttt{s.t.}\ C$$

The Maude strategy language

- Rule applications

$$\texttt{top}(\mathit{rlabel}[x_1 \to t_1, \ldots, x_n \to t_n]\{\alpha_1, \ldots, \alpha_m\})$$

- Tests

$$\texttt{match}\ P\ \texttt{s.t.}\ C$$

- Strategy calls

$$\mathit{sl}(t_1, \ldots, t_n)$$

The Maude strategy language

- Concatenation $\alpha \; ; \; \beta$

The Maude strategy language

- Concatenation $\alpha \; ; \; \beta$
- Union $\alpha \mid \beta$

The Maude strategy language

- Concatenation $\alpha \;;\; \beta$
- Union $\alpha \mid \beta$
- Iteration $\alpha*$

The Maude strategy language

- Concatenation $\alpha\ ;\ \beta$
- Union $\alpha \mid \beta$
- Iteration $\alpha *$
- Constants `idle`, `fail`

The Maude strategy language

- Concatenation $\alpha \mathbin{;} \beta$
- Union $\alpha \mid \beta$
- Iteration $\alpha*$
- Constants `idle`, `fail`
- Conditionals $\alpha \mathbin{?} \beta : \gamma$

The Maude strategy language

- Rewriting of subterms

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

Strategy modules

$$\texttt{strat}\ \mathit{sname} : s_1 \ldots s_n \ @\ s \texttt{.}$$

$$\texttt{csd}\ \mathit{sname}(t_1, \ldots, t_n) := \alpha \quad \texttt{if} \quad \bigwedge_i u_i = v_i \wedge \bigwedge_j u_j : s_j \quad \texttt{.}$$

Strategy modules

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

  strats safe solution eagerEating @ River .

  *** Only safe moves such that no being dies
  sd safe := match left | G:Group ? idle :
               (oneCrossing ; not(eating) ; safe) .

  *** A known hardwired solution
  sd solution := goat ; alone ; cabbage ; goat ;
                wolf ; alone ; goat .

  *** Eating must happen (if possible) before moving
  sd eagerEating := match left | G:Group cabbage goat ? idle :
                     ((eating or-else oneCrossing) ; eagerEating) .

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

Executing strategies

```
Maude> srew initial using safe .

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

No more solutions
rewrites: 33
```

```
Maude> srew initial using eagerEating .

Solution 1
rewrites: 71
result River: left | right shepherd wolf goat cabbage

No more solutions
rewrites: 71
```

Parameterization

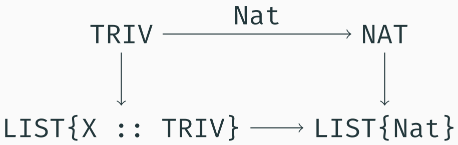
- Functional and strategic requirements are declared in a theory

```
fth T is ... endfth        sth T is ... endsth
```

- Parameterized modules receive arguments bound to a theory

```
fmod LIST{X :: TRIV} is ... endfm
```

- Views map sorts, operations, and strategies in a theory to their instances in a target module.
- Module instantiation is based on the pushout along a view.



Parameterization: backtracking example

Abstract problem definition

```
sth BT-PROBLEM is
  protecting BOOL .
  sort State .

  op isSolution : State → Bool .
  strat expand @ State .
endsth
```

Parameterization: backtracking example

Abstract problem definition

```
sth BT-PROBLEM is
  protecting BOOL .
  sort State .

  op isSolution : State → Bool .
  strat expand @ State .
endsth
```

Parameterized module

```
smod BT-STRAT{X :: BT-PROBLEM} is
  var S : X$State .

  strat solve @ X$State .
  sd solve := (match S s.t. isSolution(S)) ? idle
                  : (expand ; solve) .
endsm
```

Parameterization: backtracking example

Instance specification

```
mod QUEENS is
  protecting LIST{Nat} .
  protecting SET{Nat} .

  op isSolution : List{Nat} → Bool .
  op isValid : List{Nat} Nat → Bool .

  crl [next] : L ⇒ L N if N,S ≔ 1, 2, 3, 4, 5, 6, 7, 8 .

  eq isSolution(L) = size(L) == 8 .
  eq isValid(L, M) = isValid(L, M, 1) .

  *** [...]
endm
```

Parameterization: backtracking example

Instance specification

```
smod QUEENS-STRAT is
  protecting QUEENS .

  strat expand @ List{Nat} .
  var L : List{Nat} . var N : Nat .

  sd expand := top(next) ; match L N s.t. isValid(L, N) .
endsm
```

Parameterization: backtracking example

Instance specification

```
smod QUEENS-STRAT is
  protecting QUEENS .

  strat expand @ List{Nat} .
  var L : List{Nat} . var N : Nat .

  sd expand := top(next) ; match L N s.t. isValid(L, N) .
endsm
```

Strategy instantiation BT-STRAT{QueensBT}

```
view QueensBT from BT-PROBLEM to QUEENS-STRAT is
  sort State to List{Nat} .
  op isSolution to isSolution .
  strat expand to expand .
endv
```

Parameterization: backtracking example

```
Maude> srew [2] nil using solve .

Solution 1
rewrites: 285296
result NeList{Nat}: 1 5 8 6 3 7 2 4

Solution 2
rewrites: 285296
result NeList{Nat}: 1 6 8 3 7 4 2 5


Maude> continue .

[...]

Solution 92
rewrites: 556
result NeList{Nat}: 8 4 1 3 6 2 7 5

No more solutions.
rewrites: 688
```

Parameterization

Presented at WADT 2018 and illustrated by several examples available at `http://maude.ucm.es/strategies`

R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. Parameterized strategies specification in Maude. In J. Fiadeiro and I. 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.

Maude 3.0

- Symbolic reachability analysis of concurrent systems using narrowing
- The strategy language is incorporated to the official version
- Three new external objects (files, standard streams and meta-interpreters)
- Parameterized views
- Connection to SMT solvers CVC4 and Yices2
- Some bug fixes and other improvements

Checking models controlled by strategies

Motivation

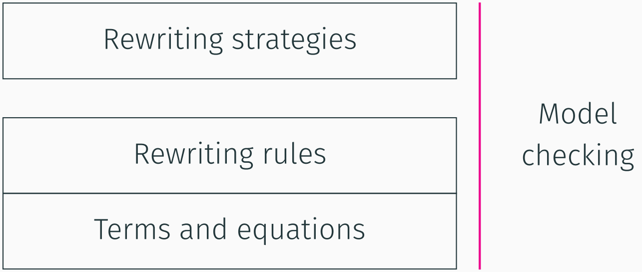
Rewriting strategies

Rewriting rules

Terms and equations

Model checking

Motivation


Rewriting strategies
Rewriting rules
Terms and equations
Model checking

Model checking

Kripke structure

$$\mathcal{K} = (S, \rightarrow, I, AP, \ell)$$

$$\ell : S \rightarrow \mathcal{P}(AP)$$

Temporal property

$$\varphi_{AP}$$

Model checking $\quad \mathcal{K} \models \varphi$

Model checking

Kripke structure

$$\mathcal{K} = (S, \to, I, AP, \ell)$$

$$\ell : S \to \mathcal{P}(AP)$$

Temporal property

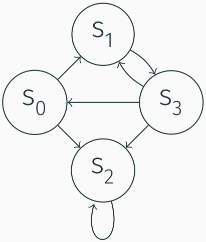
$$\varphi_{AP}$$

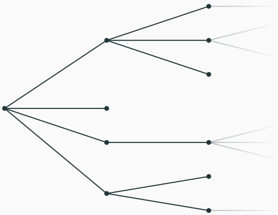
Model checking $\mathcal{K} \vDash \varphi$

$(T_{\Sigma/E}, \to_R^{1.}, I, AP, \ell)$ for a rewriting theory $\mathcal{R} = (\Sigma, E, R)$

Abstract strategies

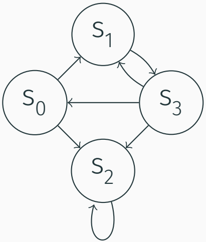
$\mathcal{A} = (S, \to)$ Γ_A

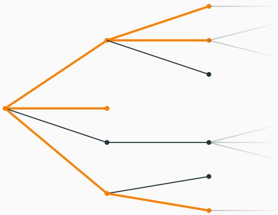




Abstract strategies

$\mathcal{A} = (S, \to)$ $\qquad E \subseteq \Gamma_A$





T. Bourdier, H. Cirstea, D. J. Dougherty, and H. Kirchner. Extensional and intensional strategies. In M. Fernández, editor, *WRS 2009, Brasilia, Brazil*, volume 15 of *EPTCS*, pages 1–19, 2009.

Model checking with strategies

Linear-time case

$\mathcal{K} \vDash \varphi$ iff $\ell(\pi) \vDash \varphi$ for every execution π in $\mathcal{K}$.

Definition: $(\mathcal{K}, E) \vDash \varphi$ if $\ell(\pi) \vDash \varphi$ for all $\pi \in E$.

Model checking with strategies

Linear-time case

$\mathcal{K} \vDash \varphi$ iff $\ell(\pi) \vDash \varphi$ for every execution π in $\mathcal{K}$.

Definition: $(\mathcal{K}, E) \vDash \varphi$ if $\ell(\pi) \vDash \varphi$ for all $\pi \in E$.

Model checking with strategies

Linear-time case

$\mathcal{K} \vDash \varphi$ iff $\ell(\pi) \vDash \varphi$ for every execution π in $\mathcal{K}$.

Definition: $(\mathcal{K}, E) \vDash \varphi$ if $\ell(\pi) \vDash \varphi$ for all $\pi \in E$.

To reuse standard algorithms, we find an ARS whose traces *coincide* with E.

Model checking with Maude strategies

Small-step non-deterministic operational semantics

- Execution states $\mathcal{XS}$: term + strategy progress.
- Control $\to_c$ and system $\to_s$ transitions (rule rewrites).

$$\twoheadrightarrow \; = \; \to_c^* \circ \to_s$$

$$E(\alpha, I) = \{\, \mathrm{term}(x) : t \,@\, \alpha \twoheadrightarrow x_2 \twoheadrightarrow \cdots \twoheadrightarrow x_k \twoheadrightarrow \cdots, t \in I \,\}$$

Model checking with Maude strategies

Small-step non-deterministic operational semantics

- Execution states $\mathcal{XS}$: term + strategy progress.
- Control $\to_c$ and system $\to_s$ transitions (rule rewrites).

$$\twoheadrightarrow \; = \; \to_c^* \circ \to_s$$

$$E(\alpha, I) = \{\, \mathrm{term}(x) : t \;@\; \alpha \twoheadrightarrow x_2 \twoheadrightarrow \cdots \twoheadrightarrow x_k \twoheadrightarrow \cdots, t \in I \,\}$$

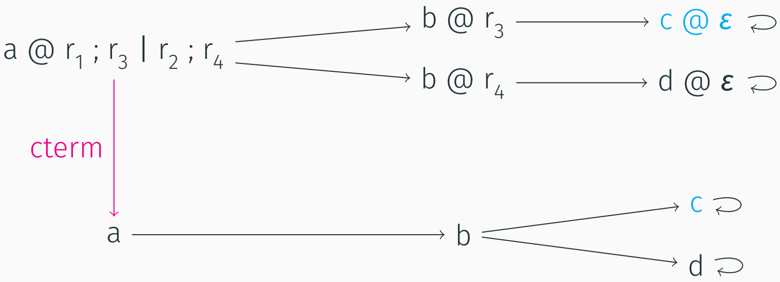
$$(\mathcal{K}, E(\alpha, I)) \vDash \varphi \iff (\mathcal{XS}, \twoheadrightarrow, \{t \;@\; \alpha\}_{t \in I}, \ell \circ \mathrm{term}) \vDash \varphi$$

Branching-time case

Initial idea: use the *equivalent* ARS of the linear-time case.

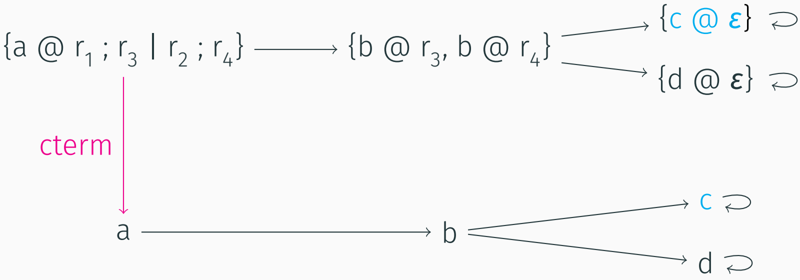
Model checking with strategies (branching-time case)

The branching structure of the executions is not preserved.



Model checking with strategies (branching-time case)

However, this can be solved by merging states.



Model checking CTL* properties

$$
\begin{array}{lcll}
E \models p & \iff & \forall \pi \in E & p \in \ell(\pi_0) \\
E \models \mathbf{A}\,\phi & \iff & \forall \pi \in E & E_{\pi_0}, \pi \models \phi \\
E \models \mathbf{E}\,\phi & \iff & \exists \pi \in E & E_{\pi_0}, \pi \models \phi
\end{array}
$$

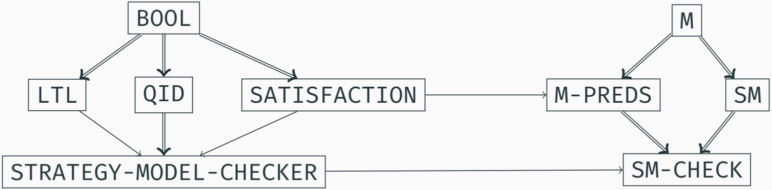
$$
\begin{array}{lcl}
E, \pi \models \Phi & \iff & E \models \Phi \\
E, \pi \models \circ\,\phi & \iff & E_{\pi_0 \pi_1}, \pi[1..] \models \phi \\
E, \pi \models \Diamond\,\phi & \iff & \exists\, n \geq 0 \quad E_{\pi[..n]}, \pi[n..] \models \phi \\
E, \pi \models \Box\,\phi & \iff & \forall\, n \geq 0 \quad E_{\pi[..n]}, \pi[n..] \models \phi \\
E, \pi \models \phi_1 \,\mathcal{U}\, \phi_2 & \iff & \exists\, n \geq 0 \quad \begin{array}{l} E_{\pi[..n]}, \pi[n..] \models \phi_2 \,\wedge \\ \forall\, 0 \leq k < n \quad E_{\pi[..k]}, \pi[k..] \models \phi_1 \end{array}
\end{array}
$$

$$
E_{\pi_1 \cdots \pi_n} = \{\pi_n \pi_+ \,:\, \pi_1 \cdots \pi_n \pi_+ \in E\}
$$

W. Thomas. Computation tree logic and regular ω-languages. In *REX 1988*, volume 354 of *LNCS*, pages 690–713. Springer, 1988.

The Maude model checker

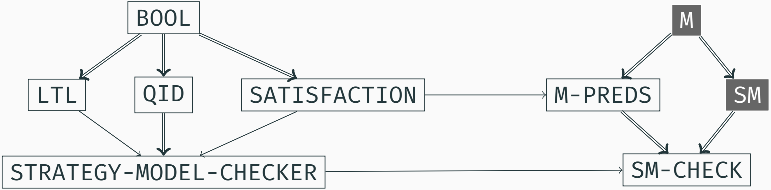
The Maude LTL model checker for strategy-controlled systems



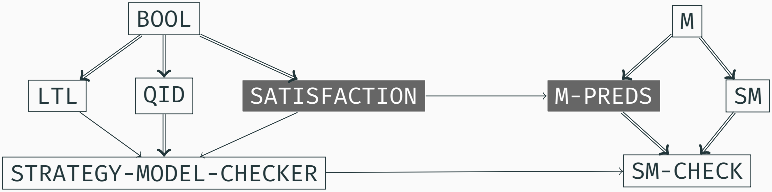
R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. Model checking strategy-controlled rewriting systems. In H. Geuvers, editor, *FSCD 2019*, volume 131 of *LIPIcs*, 34:1–34:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In F. Gadducci and U. Montanari, editors, *WRLA 2002, Pisa, Italy*, volume 71 of *ENTCS*, pages 162–187. Elsevier, 2004.

The Maude LTL model checker for strategy-controlled systems



The Maude LTL model checker for strategy-controlled systems




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

The Maude LTL model checker for strategy-controlled systems

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

  subsort River < State .
  ops goal death : → Prop [ctor] .

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

  eq left | right shepherd wolf goat cabbage |= goal = true .
  eq R |= goal = false [owise] .

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

The Maude LTL model checker for strategy-controlled systems


```
modelCheck :  State Formula Qid QidList Bool
              -> ModelCheckResult
```

The Maude LTL model checker for strategy-controlled systems


```
modelCheck :  State Formula Qid QidList Bool
              -> ModelCheckResult
```

```
modelCheck(initial, φ, 'safe)
```

The Maude LTL model checker for strategy-controlled systems

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

```
Maude> red modelCheck(initial, [] ~ death ∧ <> goal, 'safe) .
rewrites: 69
result ModelCheckResult: counterexample(
   {left shepherd wolf goat cabbage | right,'goat}
   {left wolf cabbage | right shepherd goat,'alone}
   {left shepherd wolf cabbage | right goat,'wolf}
   {left cabbage | right shepherd wolf goat,'goat}
   {left shepherd goat cabbage | right wolf,'cabbage},
   {left goat | right shepherd wolf cabbage,'alone}
   {left shepherd goat | right wolf cabbage,'alone})
```

The Maude LTL model checker for strategy-controlled systems

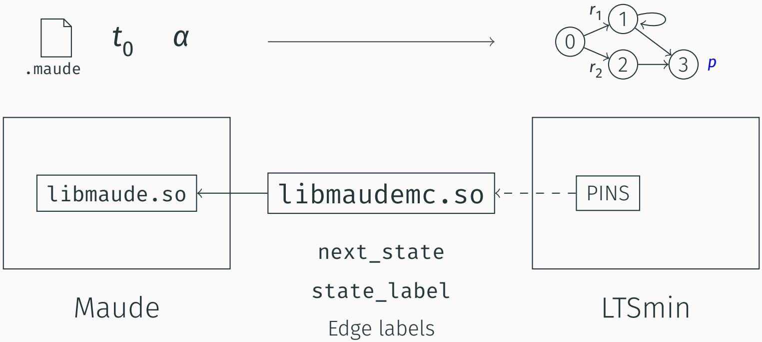
```
Maude> red modelCheck(initial, [] ~ death, 'eagerEating) .
rewrites: 14
result ModelCheckResult: counterexample(
   {left shepherd wolf goat cabbage | right,'alone}
   {left wolf goat cabbage | right shepherd,'wolf-eats}
   {left wolf cabbage | right shepherd,'alone},
   {left shepherd wolf cabbage | right,'alone}
   {left wolf cabbage | right shepherd,'alone})
```

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

Model checking CTL* properties

G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom, and T. van Dijk. LTSmin: high-performance language-independent model checking. In *TACAS 2015, London, UK*, volume 9035 of *LNCS*, pages 692–707. Springer, 2015.

Model checking CTL* properties



G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom, and T. van Dijk. LTSmin: high-performance language-independent model checking. In *TACAS 2015, London, UK*, volume 9035 of *LNCS*, pages 692–707. Springer, 2015.

Model checking CTL* properties

```
$ maude2lts river.maude initial --strat safe
    --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, 55 rewrites
```

```
$ maude2lts river.maude initial --strat eagerEating
    --ctl 'A [] E <> goal'
pins2lts-sym: Formula A [] E <> goal does not hold
    for the initial state
maude-mc: 43 system states explored, 157 rewrites
```

```
$ maude2lts river.maude initial --strat eagerEating
    --ctl 'A [] (death || bad || E <> goal)'
pins2lts-sym: Formula A [] (death || bad || E <> goal) holds
    for the initial state
maude-mc: 43 system states explored, 329 rewrites
```

Conclusions

- The full strategy language is now available at Maude 3.0.
- Rewriting can be controlled at the object level and modular strategies can be written using strategy modules.
- Rewriting systems control can be specified compositionally, allowing the execution and analisys of alternative semantics or behaviors easily.
- Model-checking Maude specifications with strategies is also possible.

Strategy language for narrowing

Combinator	*Classical*	*Symbolic*
$rlabel$	rewriting	narrowing
match P	matching	unification
matchrew P	matching	unification

Future work

- Specify and check properties of more complex and interesting examples of systems using strategies.
- Compare the verbosity and performance of specifications whether using strategies or not.

`http://maude.ucm.es/strategies/`

Bibliography

- R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. Parameterized strategies specification in Maude. In J. Fiadeiro and I. 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.
- R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. Model checking strategy-controlled rewriting systems. In H. 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.
- F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. Programming and symbolic computation in Maude. *Journal of Logical and Algebraic Methods in Computer Programming*, 110, 2020.
- M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. *Maude Manual v3.0*. December 2019.

Thank you

maude.cs.illinois.edu · maude.ucm.es/strategies