Strategies, qualitative and quantitative model checking in Maude

Narciso Martí-Oliet, Rubén Rubio

Universidad Complutense de Madrid

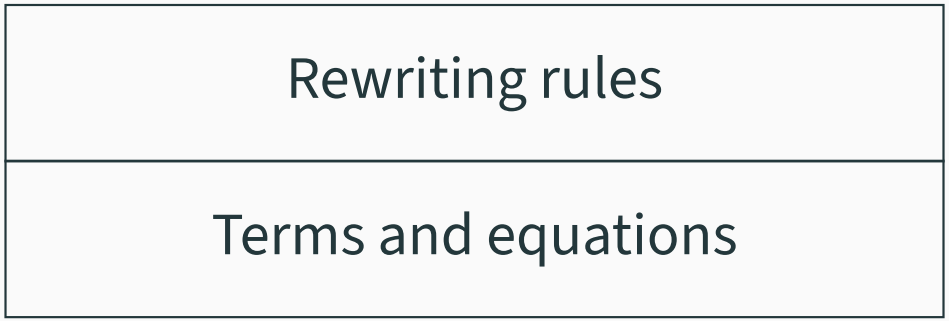
LAC 2025 — October 2, 2025, Fukuoka, Japan

Maude

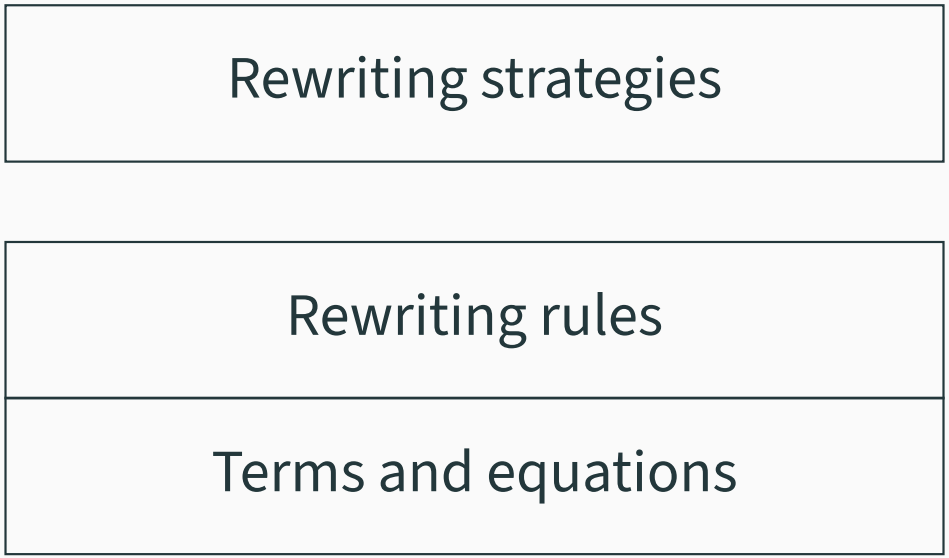
https://maude.cs.illinois.edu

- 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.

Anatomy of a Maude specification



Anatomy of a Maude specification



Anatomy of a Maude specification

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

$$(\forall X) \quad \frac{t = t'}{t : s} \quad \textbf{if} \quad \bigwedge_i u_i = v_i \wedge \bigwedge_j u_j : s_j$$

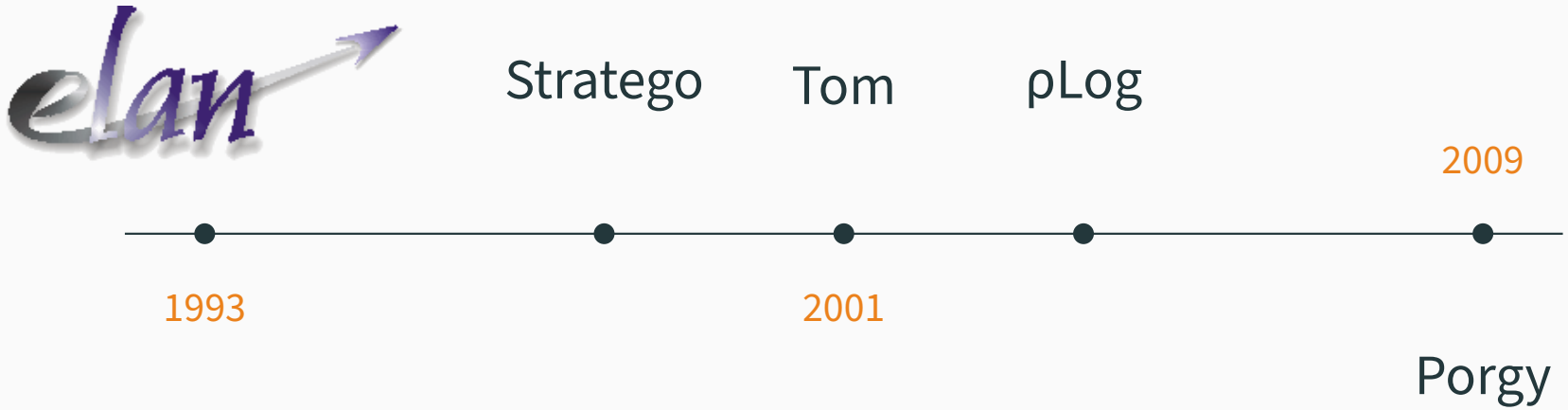
- Operator axioms, like commutativity, associativity, and identity.

Anatomy of a Maude specification

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

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

Earlier strategy languages and reflection 1993-



Earlier strategy languages and reflection 1993-

Stratego Tom ρLog

2009

1993 2001

Porgy

Reflection and Strategies in Rewriting Logic [1]

Manuel Clavel and José Meseguer

Computer Science Laboratory
SRI International

© *1996 Elsevier Science B. V.*

Design and first prototype 2004-2006

Towards a Strategy Language for Maude[⋆]

Narciso Martí-Oliet[a], José Meseguer[b], and Alberto Verdejo[a]

[a] *Facultad de Informática, UCM, Madrid, Spain*

[b] *Department of Computer Science, UIUC, Urbana-Champaign, USA*

Deduction, Strategies, and Rewriting

Steven Eker[a], Narciso Martí-Oliet[b], José Meseguer[c], and Alberto Verdejo[b]

[a] *Computer Science Laboratory, SRI International, Menlo Park, CA, USA*

[b] *Facultad de Informática, Universidad Complutense de Madrid, Spain*

[c] *Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA*

MaudE3

maude.cs.illinois.edu

Journal of Logical and Algebraic Methods in Programming 110 (2020) 100497

Programming and symbolic computation in Maude

Francisco Durán [a], Steven Eker [b], Santiago Escobar [c], Narciso Martí-Oliet [d,*], José Meseguer [e], Rubén Rubio [d], Carolyn Talcott [b]

[a] *Universidad de Málaga, Spain*
[b] *SRI International, CA, USA*
[c] *Universitat Politècnica de València, Spain*
[d] *Universidad Complutense de Madrid, Spain*
[e] *University of Illinois at Urbana-Champaign, IL, USA*

Examples with the Maude strategy language

`maude.ucm.es/strategies`

Language semantics

- λ-calculus
- Prolog (negation and cut)
- Eden
- CCS

Algorithms & deduction

- Equational completion
- SAT solving
- Simplex algorithm
- Sudoku solver

Concurrency problems

- Lamport's bakery
- Dining philosophers

Computational models

- Membrane systems
- Population protocols

Plan of the talk

1. The language
2. Examples
3. Model checking
4. Probabilistic extension
5. Probabilistic and statistical model checking

The Maude strategy language

Introductory example — Crossing the river

Example — Crossing the river

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

  ops left right : -> Side [ctor] .
  ops shepherd wolf goat cabbage : -> Group [ctor] .

  ops __  : Group Group -> Group [ctor assoc comm prec 40] .
  op  _|_ : Group Group -> River [ctor comm] .

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

Example — Crossing the river

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

  ops left right : -> Side [ctor] .
  ops shepherd wolf goat cabbage : -> Group [ctor] .

  ops __  : Group Group -> Group [ctor assoc comm prec 40] .
  op  _|_ : Group Group -> River [ctor comm] .

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

Example — Crossing the river

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

  ops left right : -> Side [ctor] .
  ops shepherd wolf goat cabbage : -> Group [ctor] .

  ops __  : Group Group -> Group [ctor assoc comm prec 40] .
  op  _|_ : Group Group -> River [ctor comm] .

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

Example — Crossing the river

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

  ops left right : -> Side [ctor] .
  ops shepherd wolf goat cabbage : -> Group [ctor] .

  ops __  : Group Group -> Group [ctor assoc comm prec 40] .
  op  _|_ : Group Group -> River [ctor comm] .

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

Example — Crossing the river

```
mod RIVER-CROSSING is
  protecting RIVER .

  vars G G' : Group .

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

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

Example — Crossing the river

```
mod RIVER-CROSSING is
  protecting RIVER .

  vars G G' : Group .

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

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

Example — Crossing the river

```
mod RIVER-CROSSING is
  protecting RIVER .

  vars G G' : Group .

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

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

Rule application

label

Rule application

label

```
Maude> srewrite shepherd left | shepherd right using alone .

Solution 1
result River: shepherd shepherd left | right

Solution 2
result River: left | shepherd shepherd right

No more solutions.
```

Rule application

label

```
Maude> srew shepherd left | shepherd right using alone .
```

- `shepherd shepherd left | right`
- `left | shepherd shepherd right`

Rule application

all

```
Maude> srew shepherd wolf left | right using all .
```

- wolf left | shepherd right
- left | shepherd wolf right

Rule application

all

```
Maude> srew shepherd wolf left | right using all .
```

- `wolf left | shepherd right` **rl** [alone]
- `left | shepherd wolf right` **rl** [wolf]

Rule application

$$label\ [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$$

```
Maude> srew shepherd left | shepherd right
       using alone[G ← left] .
```

- `left | shepherd shepherd right` `rl shepherd left | ...`

Rule application

$$label\ [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$$

```
Maude> srew shepherd left | shepherd right
       using alone[G ← right] .
```

- `shepherd shepherd left | right` `rl shepherd right | ...`

Rule application

$$label\ [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$$

$$\textbf{crl}\ l(\overline{x}) \Rightarrow r(\overline{x}, \overline{y})\ \textbf{if}\ C(\overline{x}, \overline{y})\ [\text{nonexec}]\ .$$

Rule application

$$label\ [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$$

$$\textbf{crl}\ l(\overline{x}) \Rightarrow r(\overline{x}, \overline{y})\ \textbf{if}\ C(\overline{x}, \overline{y})\ [\text{nonexec}]\ .$$

```
rl [replace] : G => G' [nonexec] .
```

Rule application

$$label\ [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$$

$$\texttt{crl}\ l(\overline{x}) \Rightarrow r(\overline{x}, \overline{y})\ \texttt{if}\ C(\overline{x}, \overline{y})\ \texttt{[nonexec] .}$$

```
rl [replace] : G => G' [nonexec] .
```

```
Maude> srew goat wolf using replace[G' ← cabbage] .
```

- cabbage
- goat cabbage
- cabbage wolf

Rule application

top(β)

```
Maude> srew goat wolf using top(replace[G' ← cabbage]) .
```

- cabbage

Rule application

$$label\ [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]\{\alpha_1, \ldots, \alpha_m\}$$

$$\texttt{crl}\ l \Rightarrow r\ \texttt{if}\ C \wedge l_1 \Rightarrow r_1 \wedge C'\ .$$

Tests

`match` P `s.t.` C

Tests

match P s.t. C

```
Maude> srew left | right using match G | G' s.t. G =/= G' .
```

- `left | right`

Tests

match P s.t. C

```
Maude> srew goat | goat using match G | G' s.t. G ≠ G' .

No solution.
```

Tests

xmatch P s.t. C

```
Maude> srew wolf goat cabbage using xmatch wolf cabbage .
```

- wolf goat cabbage

Tests

amatch P s.t. C

```
Maude> srew left wolf | right using amatch wolf .
```

- left wolf | right

Concatenation

$$\alpha \ ; \ \beta$$

Concatenation

$$\alpha \ ; \ \beta$$

```
Maude> srew shepherd left | wolf right using alone .
```

- `left | shepherd wolf right`

Concatenation

$$\alpha \ ; \ \beta$$

```
Maude> srew shepherd left | wolf right using alone ; wolf .
```

- shepherd wolf left | right

One solution

one(α)

```
Maude> srew shepherd left | shepherd right using one(alone)
.
```

- `left | shepherd shepherd right`

Disjunction

$$\alpha \mid \beta$$

```
Maude> srew shepherd wolf goat left | right
       using wolf | goat .
```

- goat left | shepherd wolf right
- wolf left | shepherd goat right

Regular expressions

α * idle fail

```
Maude> srew wolf goat goat | shepherd using wolf-eats * .
```

- wolf goat goat | shepherd
- wolf goat | shepherd
- wolf | shepherd

Regular expressions

α * idle fail

```
Maude> srew wolf goat goat | shepherd using wolf-eats + .
```

- ~~wolf goat goat | shepherd~~
- wolf goat | shepherd
- wolf | shepherd

Regular expressions

α * idle fail

```
Maude> srew wolf goat goat | shepherd using wolf-eats ! .
```

- ~~wolf goat goat | shepherd~~
- ~~wolf goat | shepherd~~
- wolf | shepherd

Regular expressions

α * `idle` `fail`

```
Maude> srew shepherd using idle .
```

- shepherd

Regular expressions

α * idle fail

```
Maude> srew shepherd using fail .

No solution.
```

Conditionals

$$\alpha \ ? \ \beta \ : \ \gamma$$

```
Maude> srew wolf goat left | shepherd right
       using wolf-eats ? idle : alone .
```

- `wolf left | shepherd right`

Conditionals

$$\alpha \ ? \ \beta \ : \ \gamma$$

```
Maude> srew wolf goat shepherd left | right
       using wolf-eats ? idle : alone .
```

- `wolf goat left | shepherd right`

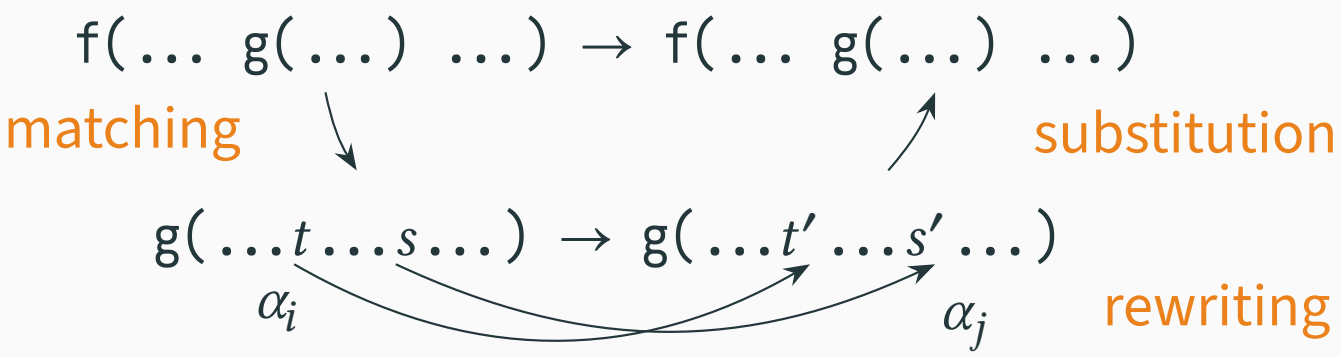
Conditionals

$$\alpha \texttt{ ? } \beta \texttt{ : } \gamma$$

$$\begin{aligned}
\alpha \texttt{ or-else } \beta &\equiv \alpha \texttt{ ? idle : } \beta \\
\texttt{try}(\alpha) &\equiv \alpha \texttt{ ? idle : idle} \\
\texttt{not}(\alpha) &\equiv \alpha \texttt{ ? fail : idle} \\
\texttt{test}(\alpha) &\equiv \texttt{not}(\texttt{not}(\alpha))
\end{aligned}$$

Rewriting of subterms

`matchrew` P `s.t.` C `by` x_1 `using` α_1 `,` $\ldots$ `,` α_n `using` x_n



Rewriting of subterms

matchrew P s.t. C by x_1 using α_1 , ... , α_n using x_n

```
Maude> srew wolf left | goat using matchrew G left | G' by G
       using replace[G' ← cabbage] .
```

- `cabbage left | goat`

Rewriting of subterms

matchrew P s.t. C by x_1 using α_1 , ... , α_n using x_n

```
Maude> srew wolf left | goat using matchrew G left | G' by G
       using replace[G' ← G'] .
```

- goat left | goat

Overview of the strategy language

label$[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]\{\alpha_1, \ldots, \alpha_m\}$

match P **s.t.** C **idle** **fail**

α ; β α | β α* α ? β : γ

matchrew P **s.t.** C **by** x_1 **using** α_1, ..., x_n **using** α_n

Strategy calls

$$slabel\,(t_1,\ \ldots,\ t_n)$$

```
Maude> srew wolf goat goat goat goat | shepherd
       using repeat(2, 3) .
```

- wolf goat goat
- wolf goat

Strategy modules

Rewrite theory

System module

Equational theory

Functional module

Strategy modules

Rewrite strategy

Strategy module

Rewrite theory

System module

Equational theory

Functional module

Strategy modules

Declarations

strat *slabel* : $s_1 \cdots s_n$ @ s .

Definitions

sd *slabel* $(t_1, \ldots, t_n)$:= α .

csd *slabel* $(t_1, \ldots, t_n)$:= α **if** C .

A recursive strategy

```
strat repeat : Nat Nat @ Word .

vars N M : Nat .

sd repeat(0, N) := idle .
sd repeat(0, s N) := wolf-eats ; repeat(0, N) .
sd repeat(s M, s N) := wolf-eats ; repeat(M, N) .
```

A recursive strategy

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

  strat repeat : Nat Nat @ Word .

  vars N M : Nat .

  sd repeat(s M, s N) := wolf-eats ; repeat(M, N) .
  sd repeat(0, s N) := idle | wolf-eats ; repeat(0, N) .
endsm
```

Reflection of the strategy language

```
op _[_]{_} : Qid Substitution StrategyList -> RuleApplication .
op match_s.t._ : Term EqCondition -> Strategy .

op sd_:=_[_]. : CallStrategy Strategy AttrSet -> StratDefinition .
op smod_is_sorts_._______endsm : ... -> StratModule .

op metaSrewrite : Module Term Strategy ... ~> ResultPair? .
```

R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. **Metalevel transformation of strategies.** *J. Log. Algebr. Methods Program.*, 124, 2022.

Search controlled by a strategy

Crossing the river

```
Maude> search initial =>* left | right shepherd wolf goat cabbage .

Solution 1 (state 32)
empty substitution

No more solutions.
```

Crossing the river

```
Maude> search initial =>* left | right shepherd wolf goat cabbage .

Maude> show path 32 .

state 0, River: left shepherd wolf goat cabbage | right
===[ wolf ]==>
state 2, River: left goat cabbage | right shepherd wolf
===[ alone ]==>
state 8, River: left shepherd goat cabbage | right wolf
===[ goat ]==>
state 16, River: left cabbage | right shepherd wolf goat
===[ alone ]==>
state 24, River: left shepherd cabbage | right wolf goat
===[ cabbage ]==>
state 32, River: left | right shepherd wolf goat cabbage
```

Crossing the river

```
Maude> search initial =>* left | right shepherd wolf goat cabbage .

Maude> show path 32 .

state 0, River: left shepherd wolf goat cabbage | right
===[ wolf ]==>
state 2, River: left goat cabbage | right shepherd wolf
===[ alone ]==>
state 8, River: left shepherd goat cabbage | right wolf
===[ goat ]==>
state 16, River: left cabbage | right shepherd wolf goat
===[ alone ]==>
state 24, River: left shepherd cabbage | right wolf goat
===[ cabbage ]==>
state 32, River: left | right shepherd wolf goat cabbage
```

Crossing the river

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

  strats eagerEating oneCrossing eating @ River .

  var G : Group .

  sd eagerEating := match left | G cabbage goat ? idle
                  : ((eating or-else oneCrossing) ; eagerEating) .

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

Crossing the river

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

  strats eagerEating oneCrossing eating @ River .

  var G : Group .

  sd eagerEating := match left | G cabbage goat ? idle
                  : ((eating or-else oneCrossing) ; eagerEating) .

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

Crossing the river

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

  strats eagerEating oneCrossing eating @ River .

  var G : Group .

  sd eagerEating := match left | G cabbage goat ? idle
                  : ((eating or-else oneCrossing) ; eagerEating) .

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

Crossing the river

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

  strats eagerEating oneCrossing eating @ River .

  var G : Group .

  sd eagerEating := match left | G cabbage goat ? idle
                  : ((eating or-else oneCrossing) ; eagerEating) .

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

Crossing the river

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

  strats eagerEating oneCrossing eating @ River .

  var G : Group .

  sd eagerEating := match left | G cabbage goat ? idle
                  : ((eating or-else oneCrossing ) ; eagerEating) .

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

Crossing the river with eagerEating

```
Maude> search initial =>* left | right shepherd wolf goat cabbage
                using eagerEating .

Solution 1 (state 30)
empty substitution

No more solutions.
```

Crossing the river with eagerEating

```
Maude> show path 30 .

state 0, River: left shepherd wolf goat cabbage | right
===[ goat ]==>
state 23, River: left wolf cabbage | right shepherd goat
===[ alone ]==>
state 24, River: left shepherd wolf cabbage | right goat
===[ wolf ]==>
state 25, River: left cabbage | right shepherd wolf goat
===[ goat ]==>
state 27, River: left shepherd goat cabbage | right wolf
===[ cabbage ]==>
state 28, River: left goat | right shepherd wolf cabbage
===[ alone ]==>
state 29, River: left shepherd goat | right wolf cabbage
===[ goat ]==>
state 30, River: left | right shepherd wolf goat cabbage
```

Strategic failure

```
sd safe := (match left | G)
              ? idle
              : (oneCrossing ; not(eating) ; safe) .
```

Strategic failure

```
sd safe := (match left | G)
              ? idle
              : (oneCrossing ; (eating ? fail : idle) ; safe) .
```

Strategic failure

```
sd safe := (match left | G)
              ? idle
              : (oneCrossing ; not(eating) ; safe) .
```

```
Maude> search initial =>* R s.t. risky(R) using safe .

No solution.
```

Strategic failure

```
sd safe := (match left | G)
              ? idle
              : (oneCrossing ; not(eating) ; safe) .
```

Strategic failure

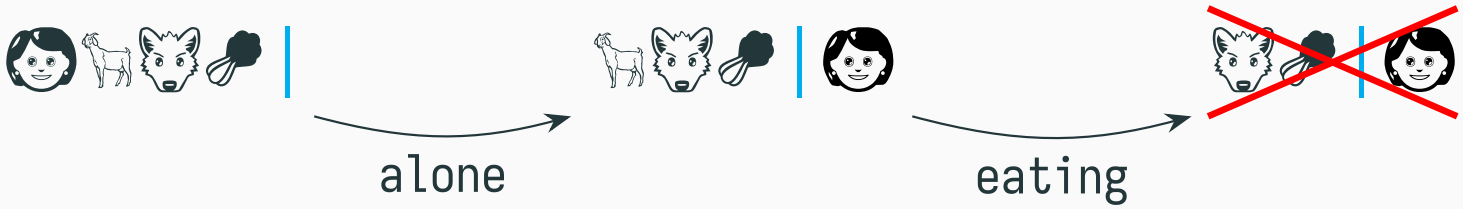
```
sd safe := (match left | G)
             ? idle
             : (oneCrossing ; not(eating) ; safe) .
```

Strategic failure

```
sd safe := (match left | G)
             ? idle
             : (oneCrossing ; not(eating) ; safe) .
```

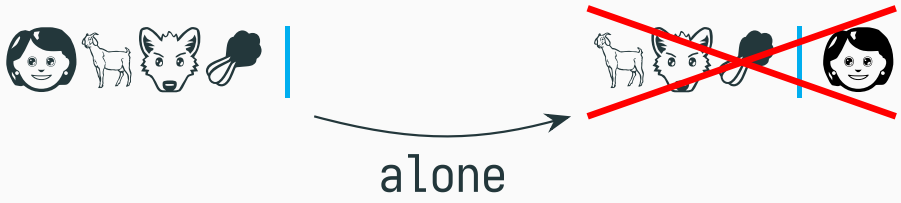
Strategic failure

```
sd safe := (match left | G)
              ? idle
              : (oneCrossing ; not(eating) ; safe) .
```



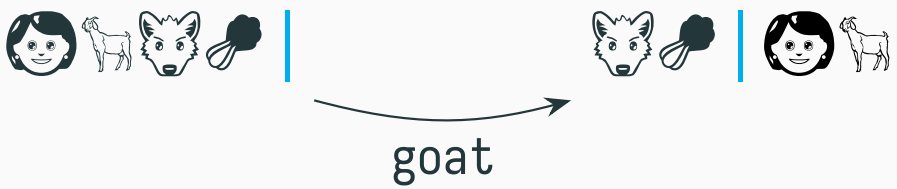
Strategic failure

```
sd safe := (match left | G)
              ? idle
              : (oneCrossing ; not(eating) ; safe) .
```



Strategic failure

```
sd safe := (match left | G)
             ? idle
             : (oneCrossing ; not(eating) ; safe) .
```



Model checking with strategies

Model checking

Model (Kripke structure)

$$\mathscr{K} = (S, \to, I, AP, \ell)$$
$$\ell : S \to \mathscr{P}(AP)$$

Temporal property

$$\varphi_{AP}$$

Model checking $\mathscr{K} \models \varphi$

Model checking

Model (Kripke structure)

$$\mathscr{K} = (S, \to, I, AP, \ell)$$
$$\ell : S \to \mathscr{P}(AP)$$

Temporal property

$$\varphi_{AP}$$

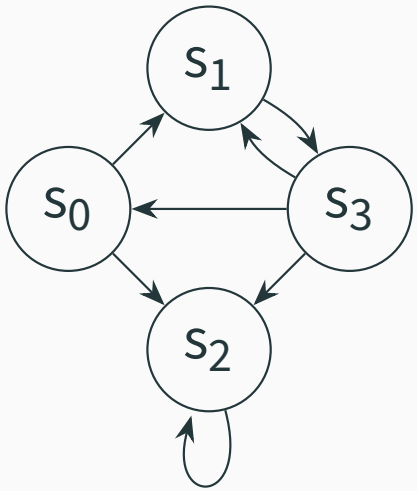
Model checking $\mathscr{K} \models \varphi$

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

Abstract strategies

$\mathscr{A} = (S, \to)$

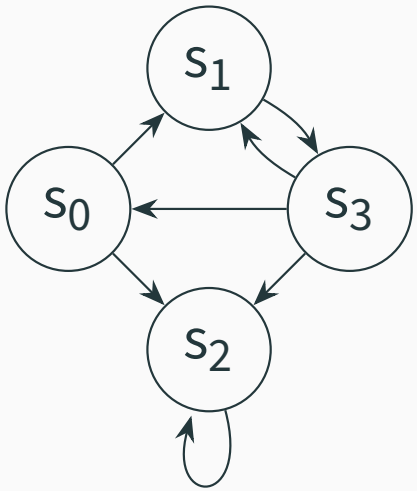
Γ_A

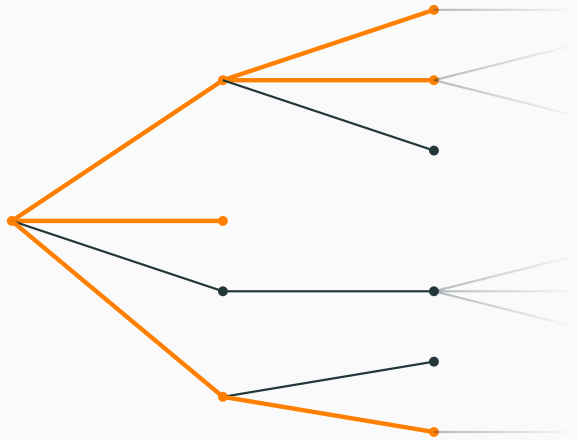


Abstract strategies

$\mathscr{A} = (S, \to)$

$E \subseteq \Gamma_A$





Model checking with strategies — linear-time case

Standard $\mathscr{K} \models \varphi$ iff $\ell(\pi) \models \varphi$ for every execution π in $\mathscr{K}$

Model checking with strategies — linear-time case

Standard $\quad \mathscr{K} \quad \models \varphi$ iff $\ell(\pi) \models \varphi$ for every execution π in $\mathscr{K}$

Strategy $\quad (\mathscr{K}, E) \models \varphi$ iff $\ell(\pi) \models \varphi$ for every execution $\pi \in E$

Model checking with strategies — linear-time case

Standard $\mathscr{K} \models \varphi$ iff $\ell(\pi) \models \varphi$ for every execution π in $\mathscr{K}$

Strategy $(\mathscr{K}, E) \models \varphi$ iff $\ell(\pi) \models \varphi$ for every execution $\pi \in E$

In practice: to reuse standard algorithms, find an ARS whose traces *coincide* with E.

Model checking with Maude strategies — linear-time case

Small-step non-deterministic operational semantics

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

$$\twoheadrightarrow \;=\; \rightarrow_c^* \circ \rightarrow_s$$

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

Model checking with Maude strategies — linear-time case

Small-step non-deterministic operational semantics

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

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

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

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

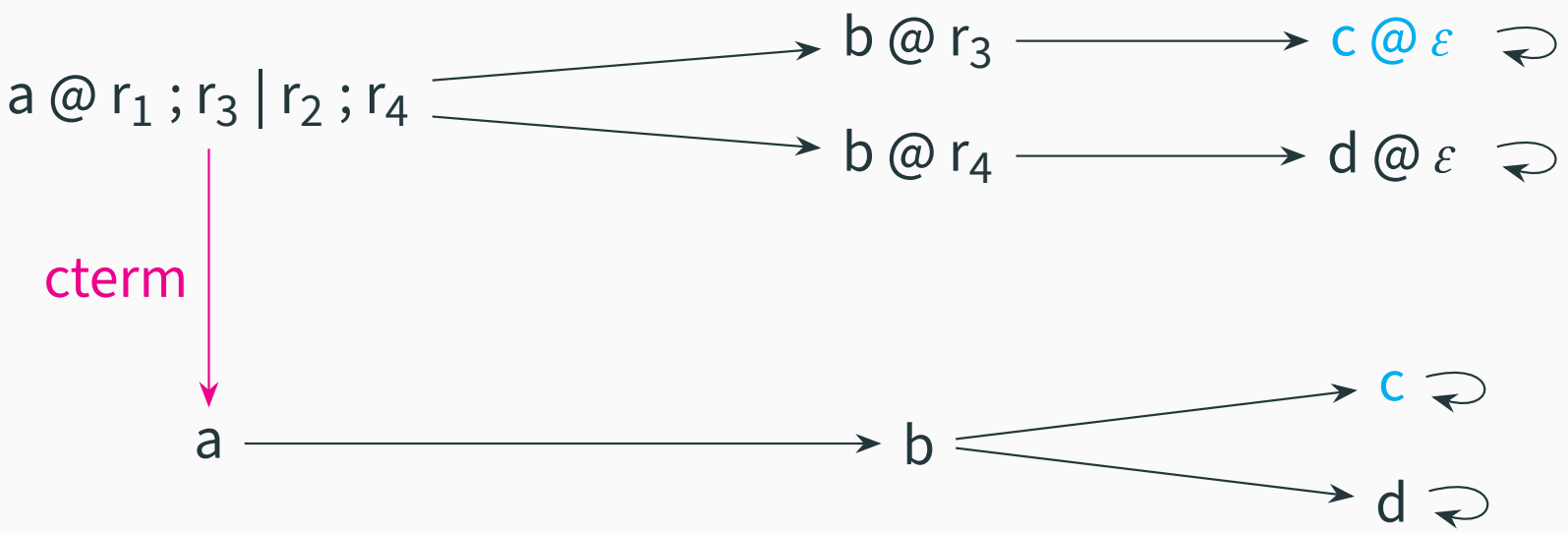
Model checking with strategies — branching-time case

And for branching time?

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

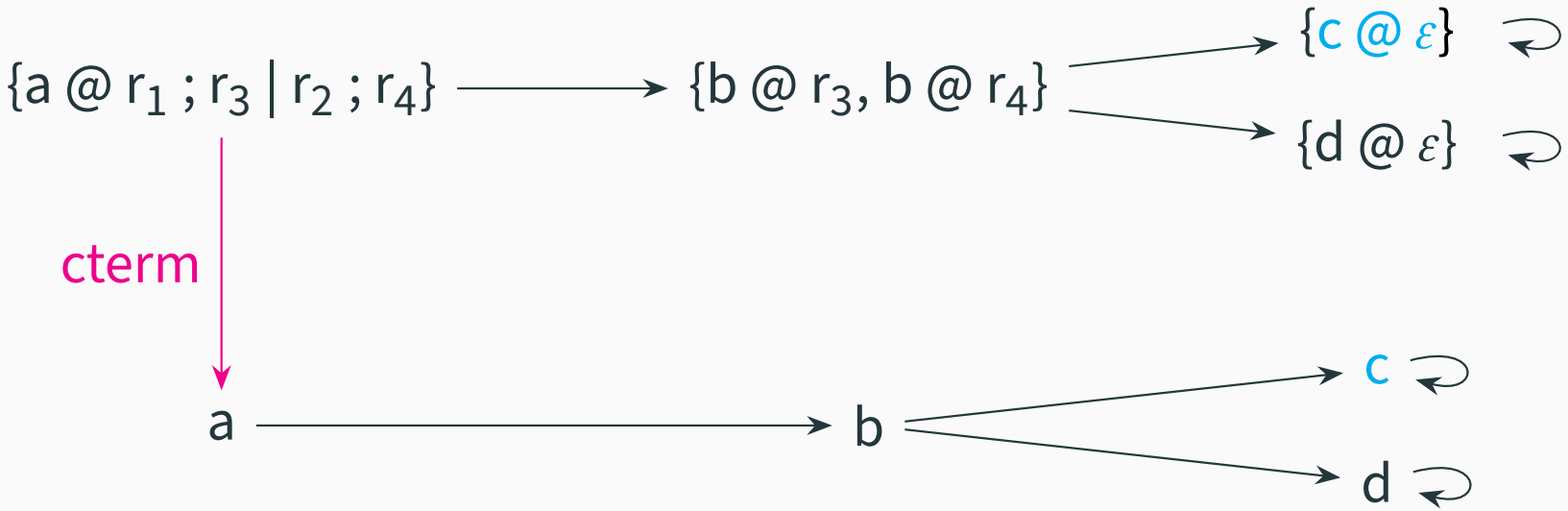
Model checking with strategies (branching-time case)

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

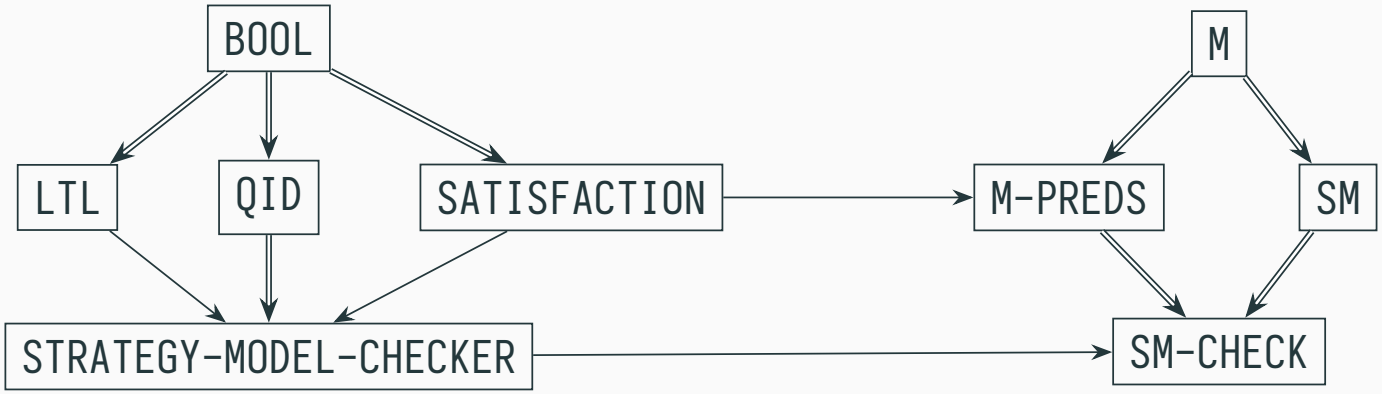
$$
\begin{aligned}
E \vDash p &\iff \forall\, \pi \in E \quad p \in \ell(\pi_0) \\
E \vDash \mathbf{A}\,\phi &\iff \forall\, \pi \in E \quad E_{\pi_0}, \pi \vDash \phi \\
E \vDash \mathbf{E}\,\phi &\iff \exists\, \pi \in E \quad E_{\pi_0}, \pi \vDash \phi
\end{aligned}
$$

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

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

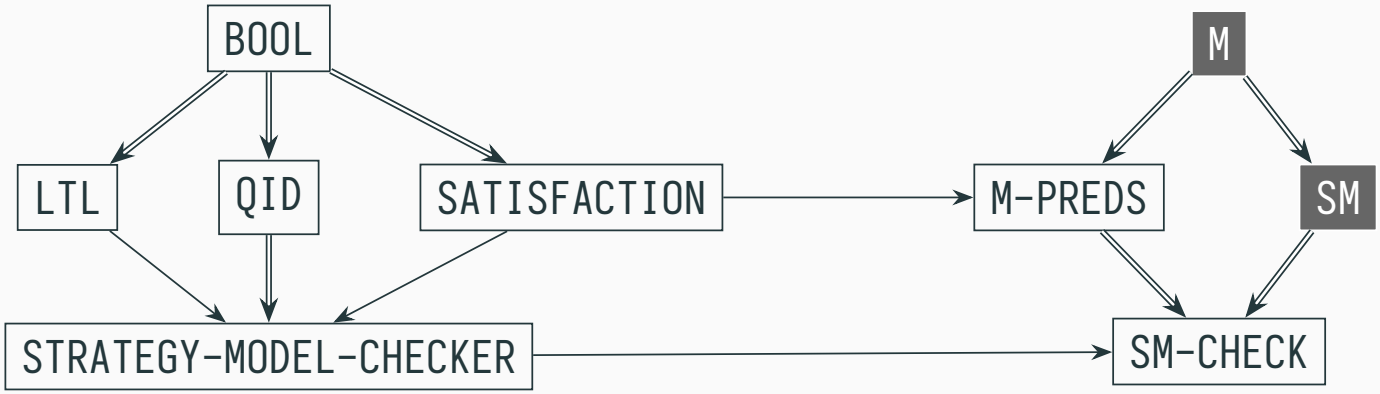
Model checking in Maude

The Maude model checker for strategy-controlled systems



R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. **Model checking strategy-controlled systems in rewriting logic.** *Automat. Softw. Eng.*, 29(1), 2022.

The Maude model checker for strategy-controlled systems



The Maude model checker for strategy-controlled systems


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

The Maude model checker for strategy-controlled systems


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

The Maude model checker for strategy-controlled systems


```
op modelCheck : State Formula Qid QidList
                -> ModelCheckResult .
```

Model checking — Crossing the river

```
Maude> reduce modelCheck(initial,
                         [] (risky -> <> death),
                         'eagerEating) .
rewrites: 129
result Bool: true
```

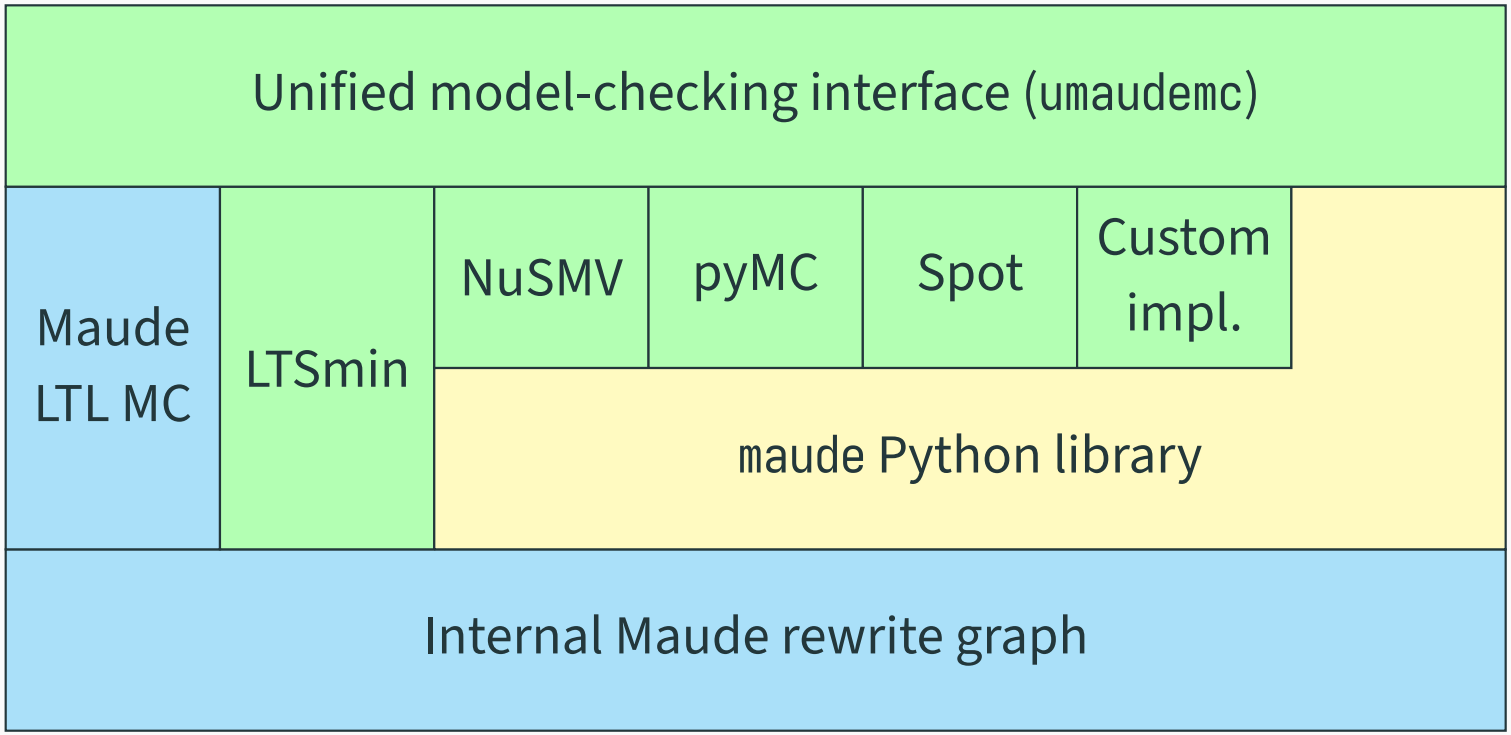
Model checking — Crossing the river

```
Maude> reduce modelCheck(initial,
                         [] (risky -> <> death),
                         'eagerEating) .
rewrites: 44
result ModelCheckerResult: counterexample(..., ...)
```

Model checking — Crossing the river

```
Maude> reduce modelCheck(initial,
                         <> goal,
                         'eagerEating) .
rewrites: 24
result ModelCheckerResult: counterexample(..., ...)
```

External model checkers for branching-time logics



`github.com/fadoss/umaudemc` · `pip install umaudemc`

Supported logics

	LTL	CTL	CTL*	μ-calculus
Extended Maude	✓			
LTSmin	✓	✓	✓	✓
pyModelChecking	✓	✓	✓	
NuSMV	✓	✓		
Spot	✓			
Builtin		✓		✓

R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. **Strategies, model checking and branching-time properties in Maude.** *J. Log. Algebr. Methods Program.*, 123, 2021.

Model checking — Crossing the river

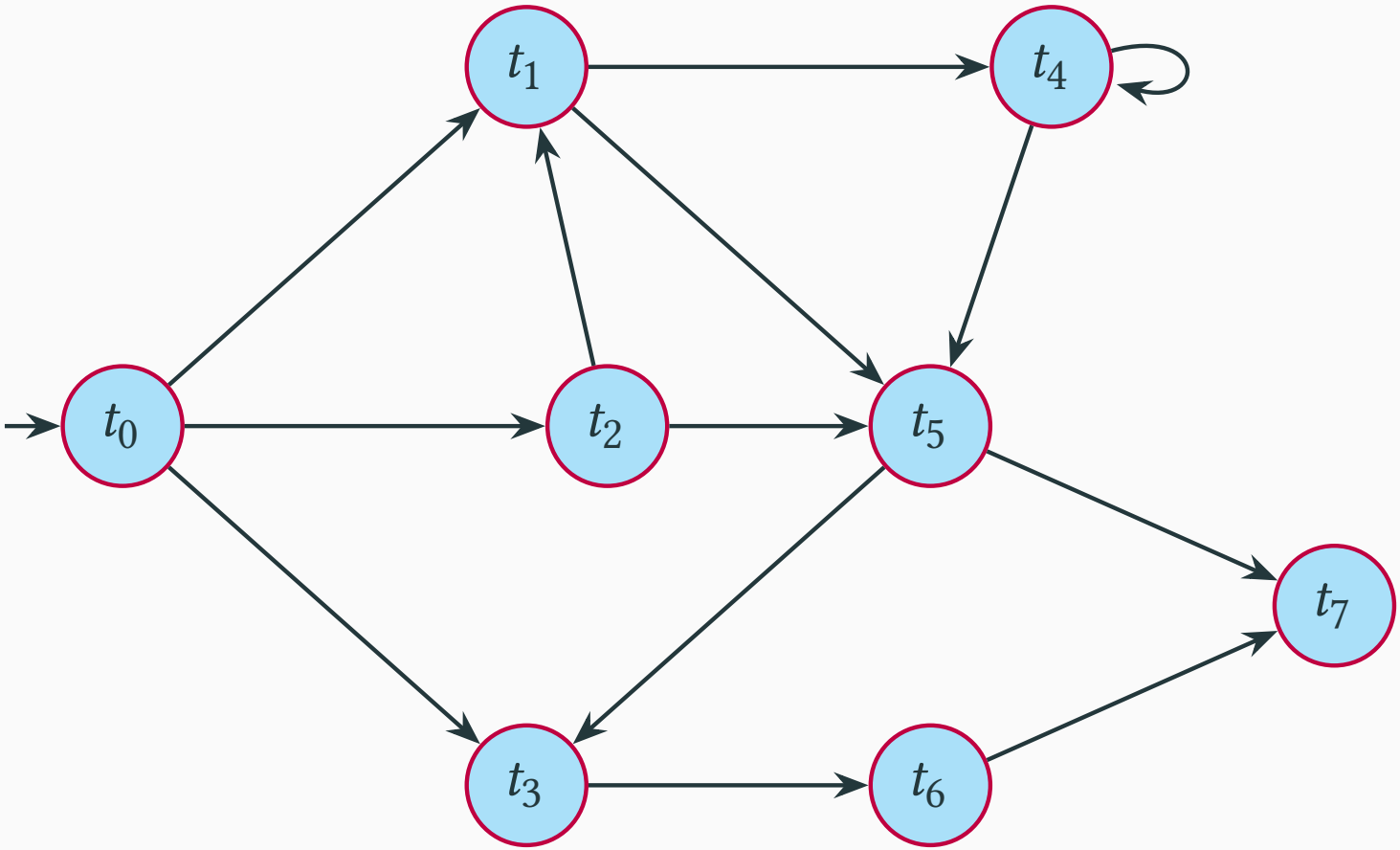
```
$  umaudemc check river.maude initial 'A [] E <> goal' safe
The property is satisfied in the initial state
(11 system states, 44 rewrites, holds in 11/11 states)
```

Model checking — Crossing the river

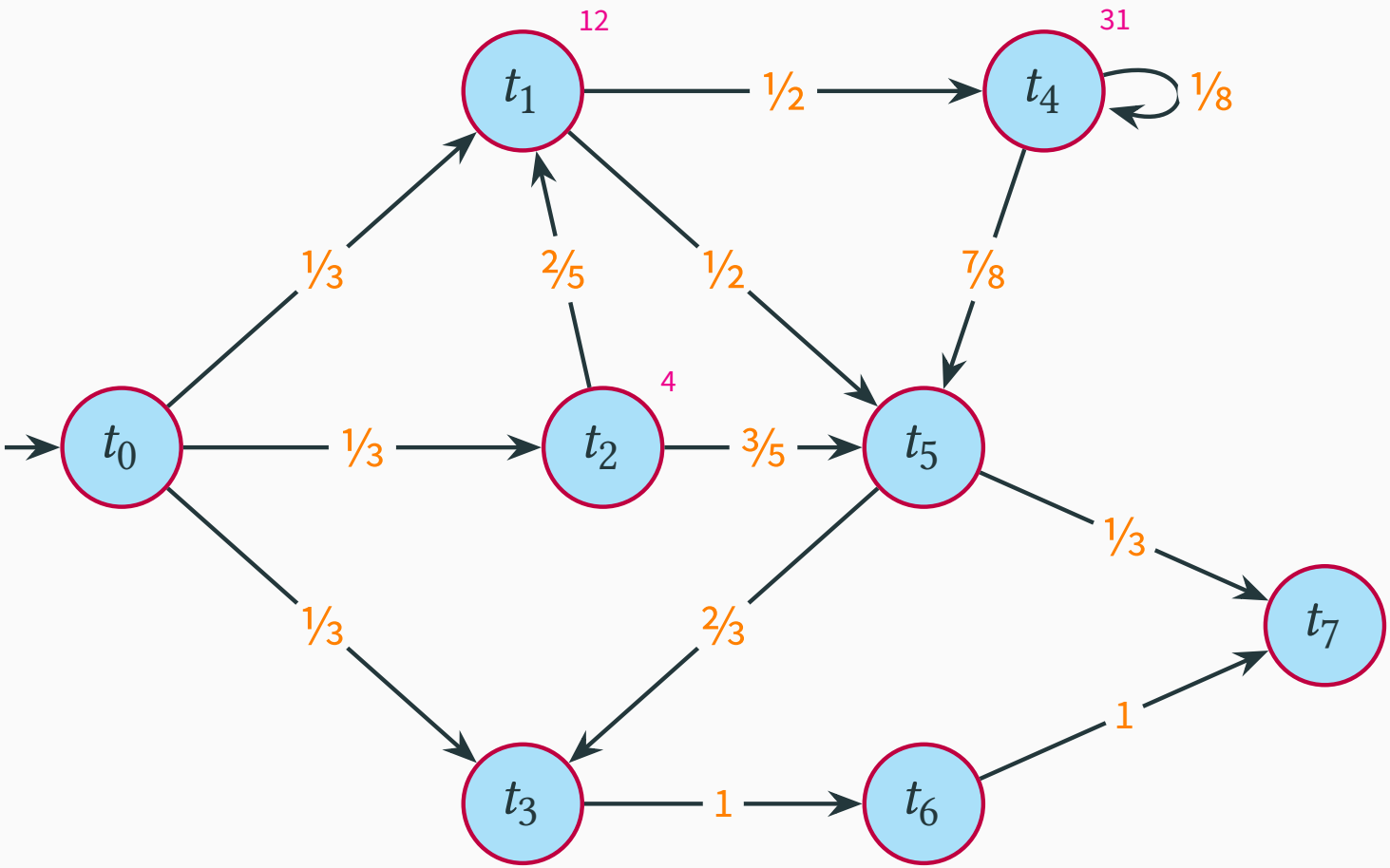
```
$  umaudemc check river.maude initial 'A [] E <> goal' eagerEating
The property is not satisfied in the initial state
(35 system states, 106 rewrites)
| left shepherd wolf goat cabbage | right
v  rl G' | shepherd G => G | shepherd G' [label alone] .
0 left wolf goat cabbage | right shepherd
```

Quantitative specification and verification

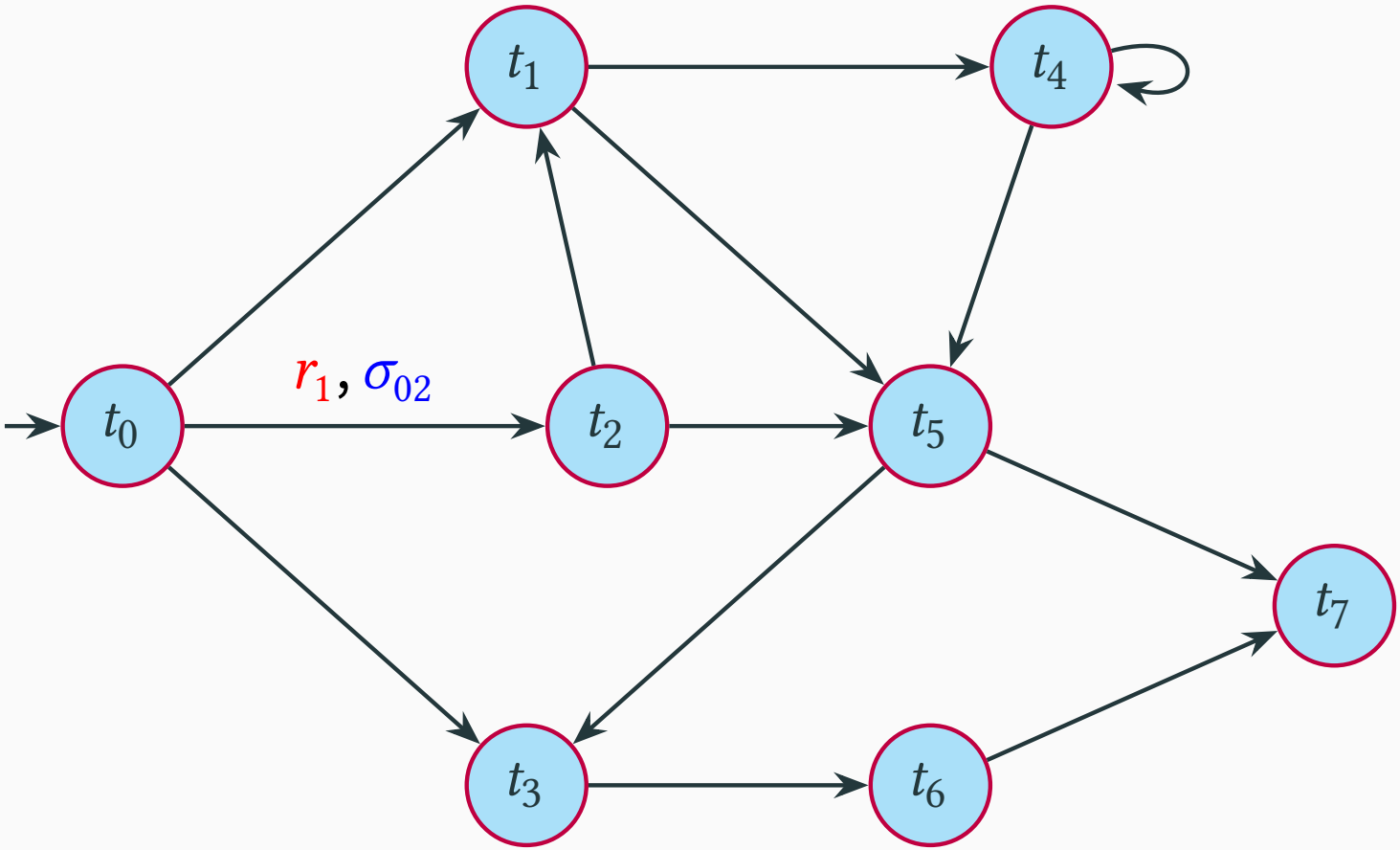
Probability assignment



Probability assignment



Probability assignment



Probability assignment methods

- `uniform`
- `uaction(`r_1`=`w_1`, ..., `r_m`=`w_m`)`
- `term(`e`)`
- `metadata`
- `pmaude` (scheck only)
- `strategy`

Probabilistic extension of the strategy language

$$\alpha_1 \mid \cdots \mid \alpha_n$$

`matchrew` P `s.t.` C `by` $\cdots$

Probabilistic extension of the strategy language

$$\texttt{choice}(w_1 : \alpha_1, \ \ldots \ , w_n : \alpha_n)$$

$$\texttt{matchrew } P \texttt{ s.t. } C \texttt{ with weight } w \texttt{ by } \cdots$$

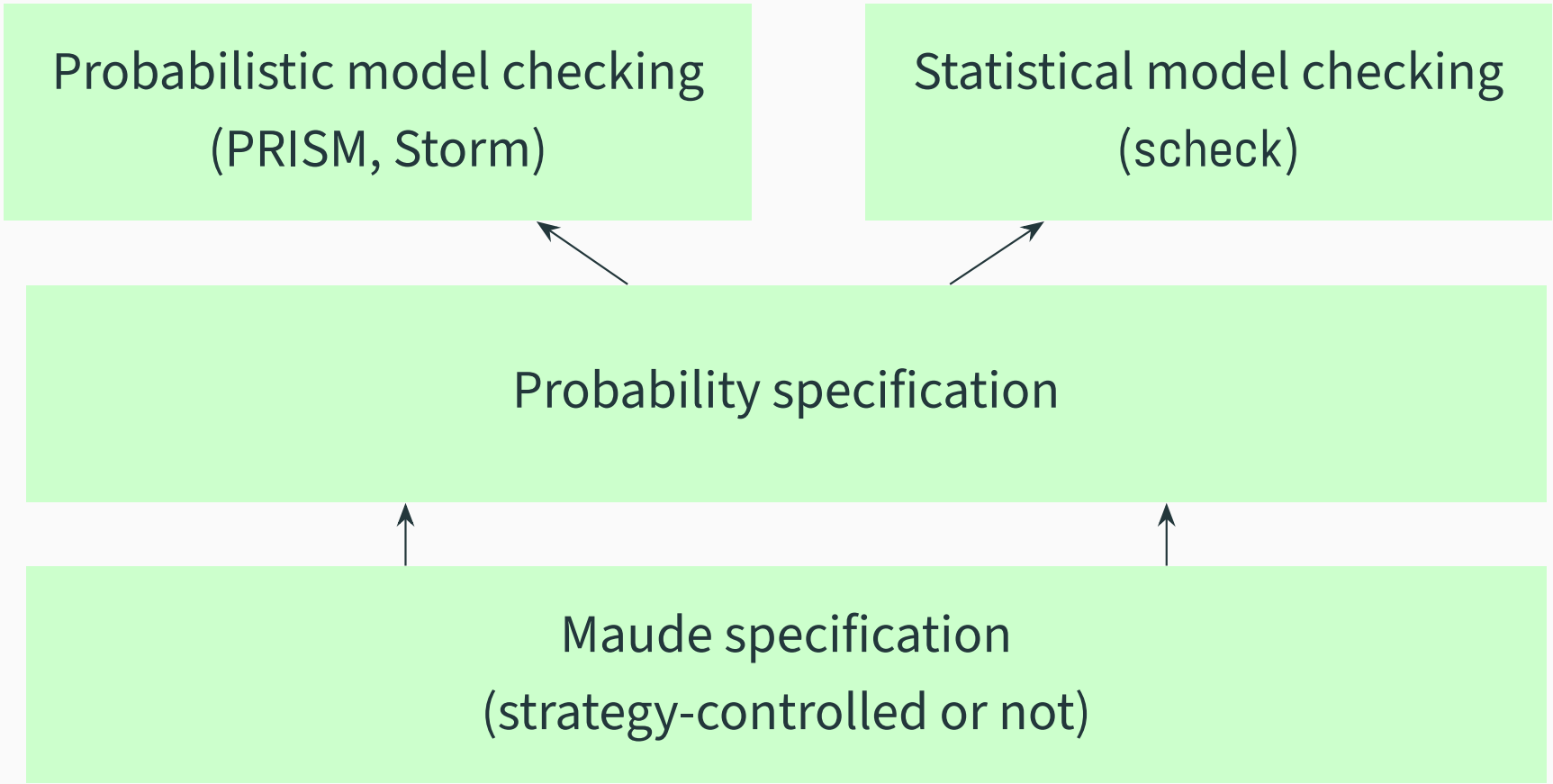
Probabilistic extension of the strategy language

$$\texttt{choice}(w_1 : \alpha_1, \ \ldots \ , w_n : \alpha_n)$$

$$\texttt{matchrew } P \texttt{ s.t. } C \texttt{ with weight } w \texttt{ by } \cdots$$

$$\texttt{sample X := } \pi(t_1, \ \ldots, \ t_n) \texttt{ in } \alpha$$

Quantitative specification and verification



R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. **QMaude: quantitative specification and verification in rewriting logic.** In *FM 2023*, volume 14000 of *LNCS*. Springer, 2023.

Quantitative verification

- Probabilistic model checking
  - Probability of LTL or PCTL formulas
  - Steady-state analysis
  - Transient-state analysis
  - Expected values of rewards
  - With any assignment method except continuous distributions
  - For models without unquantified nondeterminism (DTMC) or even some well-ordered nondeterminism (MDP)

- Statistical model checking
  - Monte Carlo estimation of QuaTEx formulas
  - Parameterized queries
  - Multithreaded or distributed

Quantitative model checking — Example

```
$ umaudemc pcheck river initial '<> goal' --assign uniform
                  eagerEating

Result: 0.006211 (relative error 6.366e-06)
```

Quantitative model checking — Example

```
$ umaudemc pcheck river initial '<> goal' --assign uniform
                  safe

Result: 1.0
```

Quantitative model checking — Example

```
$ umaudemc pcheck river initial '<> goal' --assign uniform
                  safe --steps

Result: 54.9975 (relative error 9.778e-06)
```

```
$ umaudemc scheck river initial eaten.quatex --assign step
        'choice(1 : alone, 2 : wolf, 1 : goat, 1 : cabbage)'

Number of simulations = 7470
  μ = 2.6088353413    σ = 4.40767926235    r = 0.0999696468
```

Statistical model checking — QuaTEx formulas

```
Eaten(n) = if s.rval("S:State |= bad") then
               n
           else
               # Eaten(n + 1)      // next operator
           fi;

eval E[Eaten(0)];
```

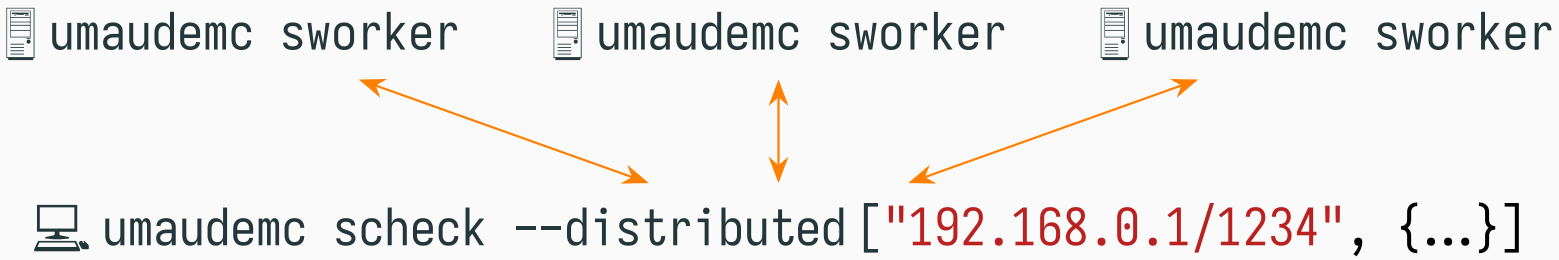
Statistical model checking — Parallelism

Monte Carlo simulations can be easily parallelized

Process-level parallelism

- scheck supports it with `--jobs n` or `-j n`

Machine-level parallelism



References

`maude.ucm.es/strategies`

- M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. ***Maude Manual v3.5.1.*** July 2025. URL: `maude.cs.illinois.edu`.
- F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott. **Programming and symbolic computation in Maude.** *J. Log. Algebraic Methods Program.*, 110, 2020.
- R. Rubio. ***Model checking of strategy-controlled systems in rewriting logic.*** PhD thesis, Universidad Complutense de Madrid, 2022.
- R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo. **QMaude: quantitative specification and verification in rewriting logic.** In *FM 2023*, volume 14000 of *LNCS*. Springer, 2023.

Thank you

Strategies, qualitative and quantitative model checking in Maude

Narciso Martí-Oliet, Rubén Rubio
Universidad Complutense de Madrid

LAC 2025