QMaude and statistical model checking

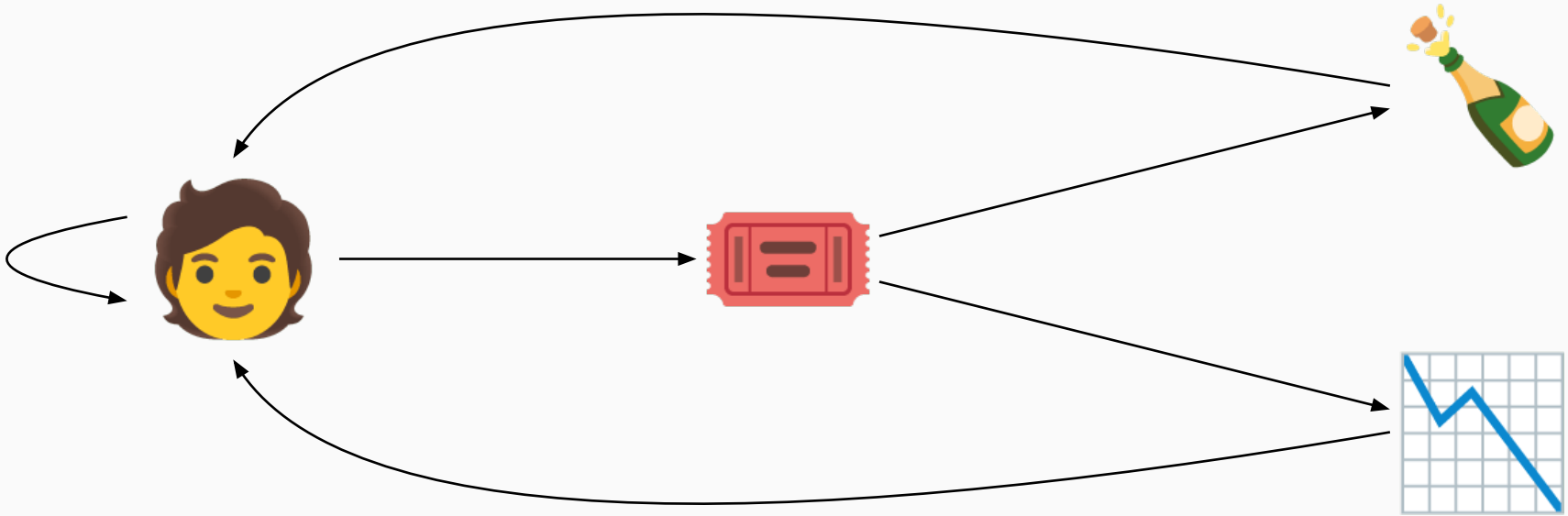
Rubén Rubio[1] with help from
Minyoung Kim[2] · Narciso Martí-Oliet[1] · José Meseguer[2]
Peter Csaba Ölveczky[2] · Carolyn Talcott[2]

Invited tutorial · WRLA 2026 · April 11, Torino, Italia

[1] Supported by the AEI project ProCode 10 (PID2023-149943OB-I00)

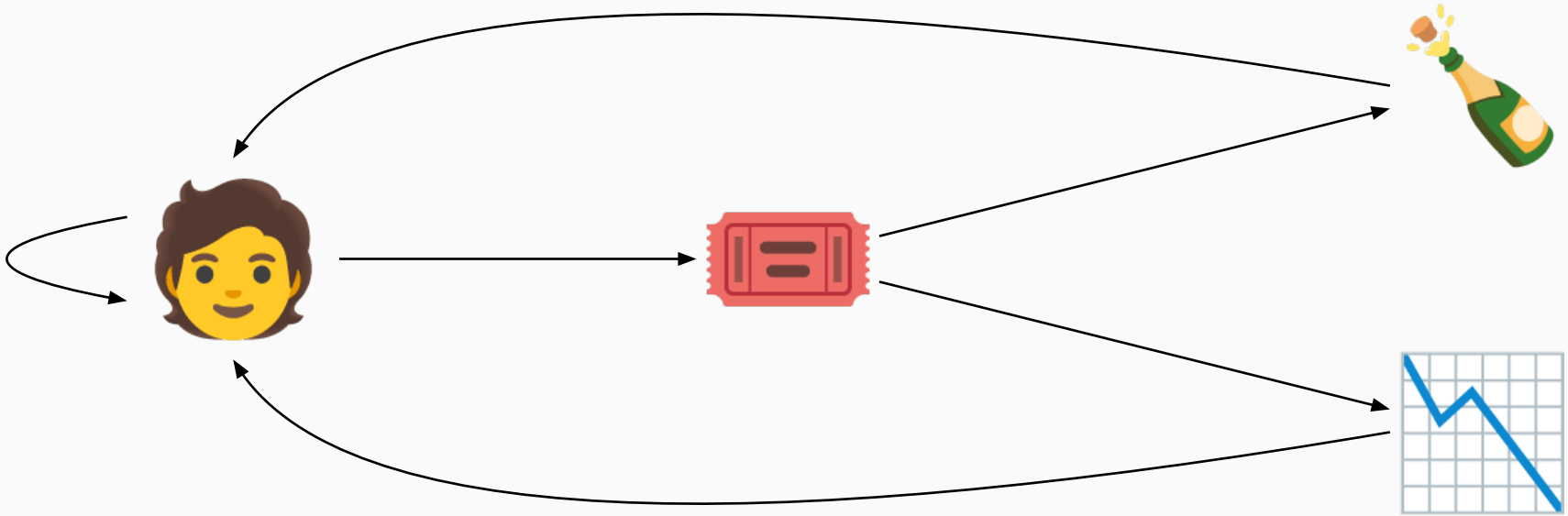
[2] Supported by the Maude-HCS project of the DARPA Provably Weird Network Deployment and Detection (PWND[2]) program

Qualitative and quantitative properties



Qualitative and quantitative properties

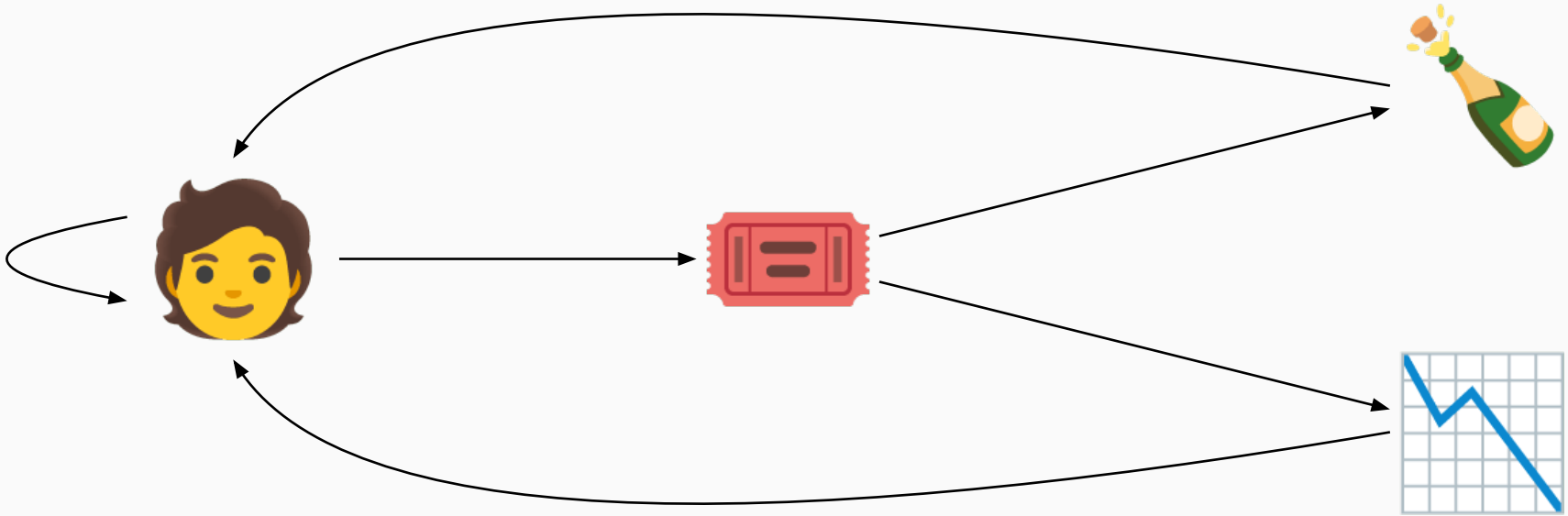
You cannot win the lottery without buying a ticket $(\neg$ 🍾 $)\ W$ 🎟



Standard model checking

Qualitative and quantitative properties

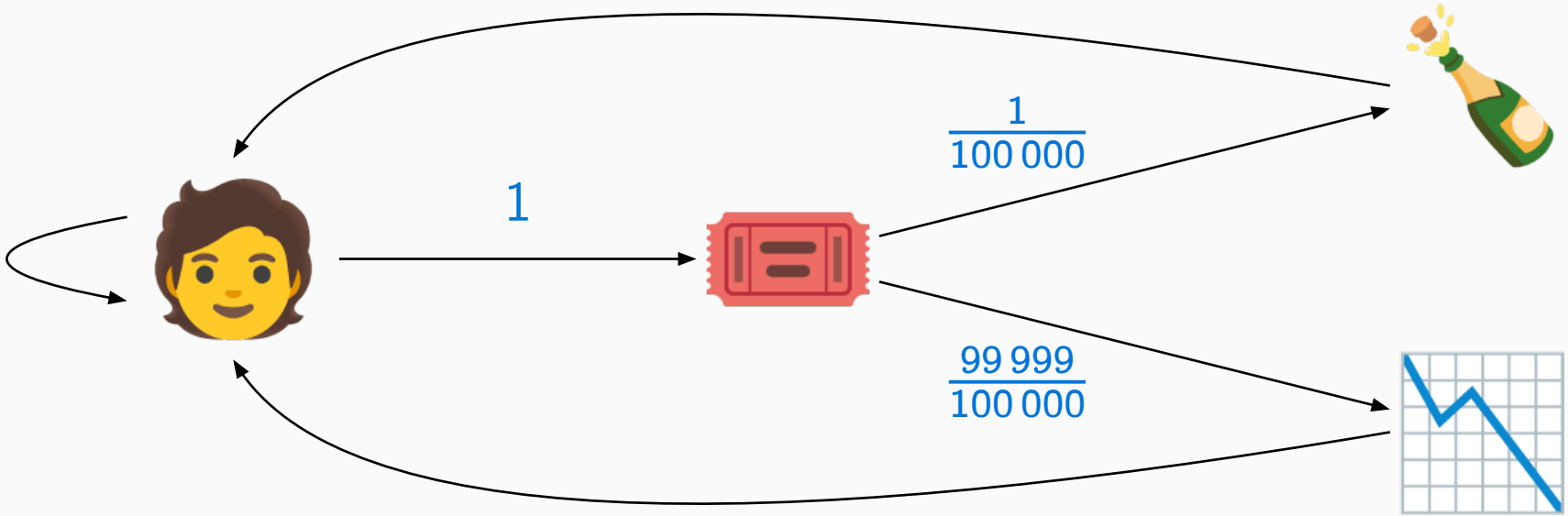
~~*You will eventually win the lottery*~~ ◇🍾



Standard model checking

Qualitative and quantitative properties

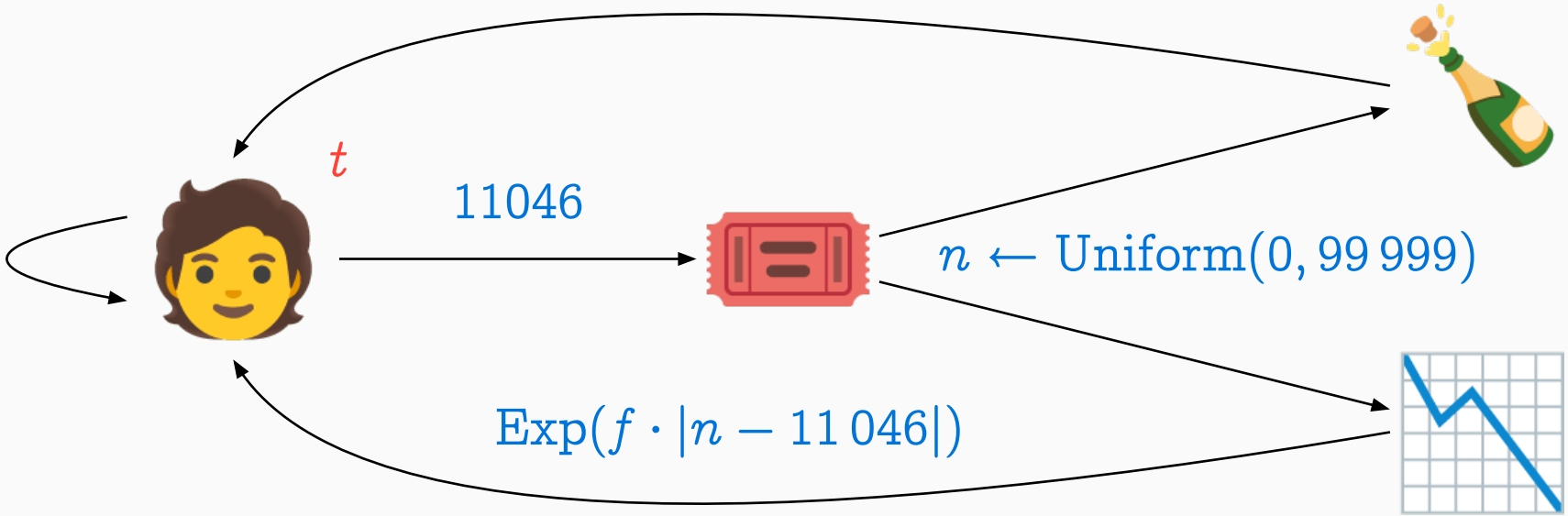
Expected number of attempts before winning the lottery



Probabilistic model checking

Qualitative and quantitative properties

Expected time for winning the lottery



Statistical model checking

Statistical model checking

Estimation of quantitative values by Monte Carlo simulations

1. Simulate a probabilistic model
2. Obtain numerical values from each execution/simulation
3. Statistically analyze the results

SMC in Maude

Electronic Notes in Theoretical Computer Science 153 (2006) 213–239

PMaude: Rewrite-based Specification Language for Probabilistic Object Systems

Gul Agha[1] José Meseguer[2] Koushik Sen[3]

Department of Computer Science,
University of Illinois at Urbana Champaign, USA.

FM 2023, LNCS 14000, pp. 240–259, 2023.

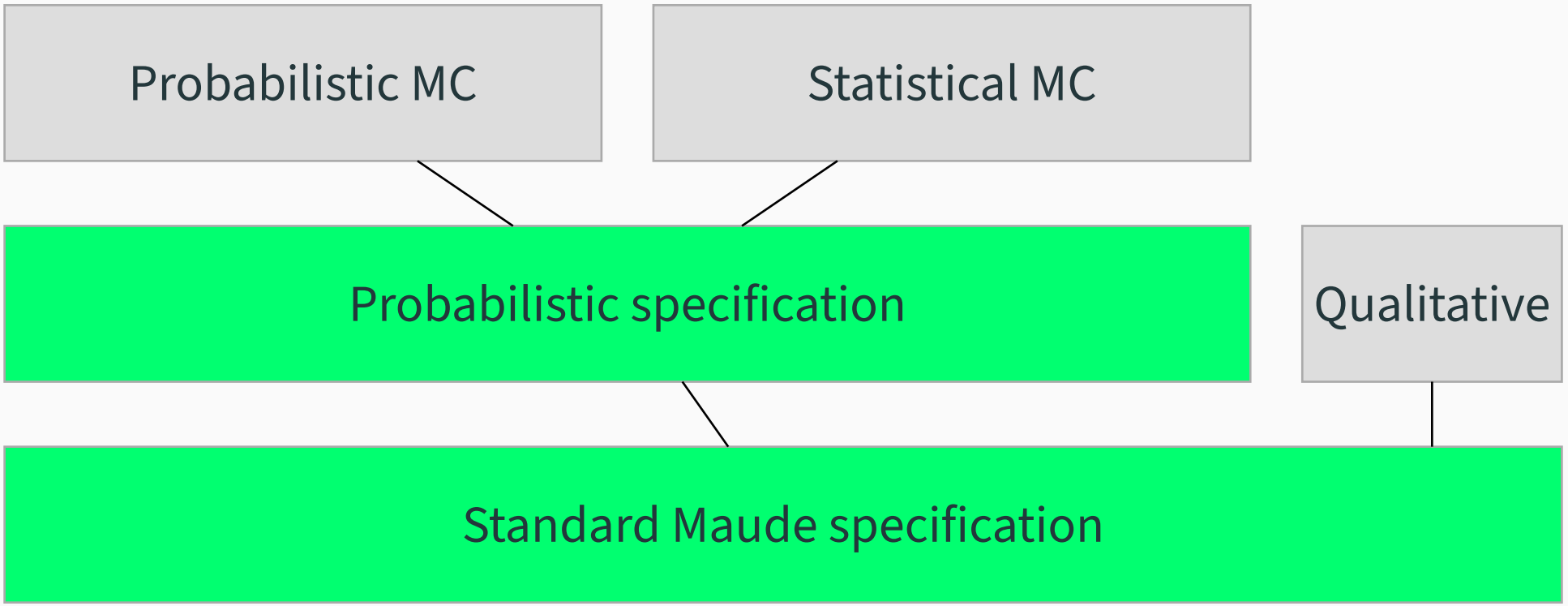
QMaude: Quantitative Specification and Verification in Rewriting Logic

Rubén Rubio(✉), Narciso Martí-Oliet, Isabel Pita, and Alberto Verdejo

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

Methods for assigning probabilities + `umaudemc {pcheck, scheck}`

QMaude overview



Unified Maude model-checking tool

`check`	Qualitative model checking with LTL, CTL, CTL*, and μ-calculus using internal and external tools
`graph`	Visual representation of the rewrite graph (standard, strategy-controlled, probabilistic, etc.)
`pcheck`	Probabilistic model checking with LTL, PCTL, expected rewards, etc. through external tools
`scheck`	Statistical model checking for QuaTEx expressions

Tutorial: unified Maude model-checking tool

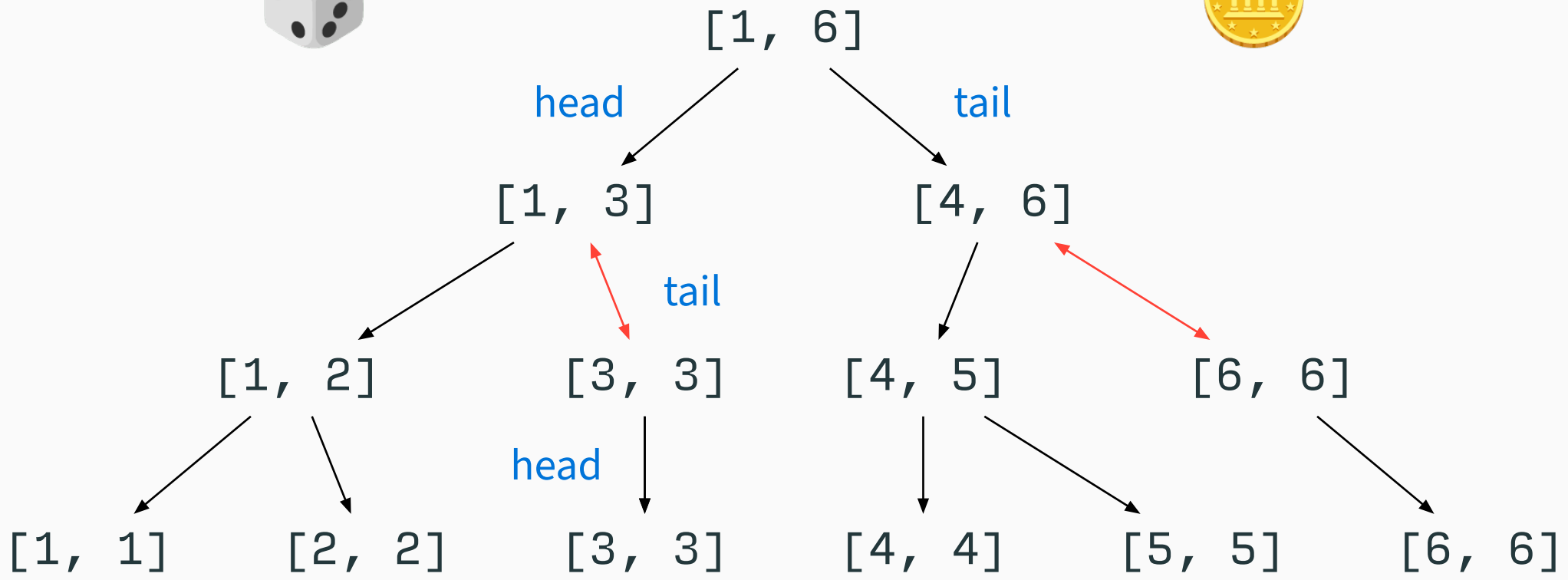
The tool can be installed with

```
$ pip install umaudemc
```

and probably `scipy` too.

Documentation	`fadoss.github.io/umaudemc`
Source code	`github.com/fadoss/umaudemc`
Material	`maude.ucm.es/qmaude/wrla26`

Knuth-Yao die (example)



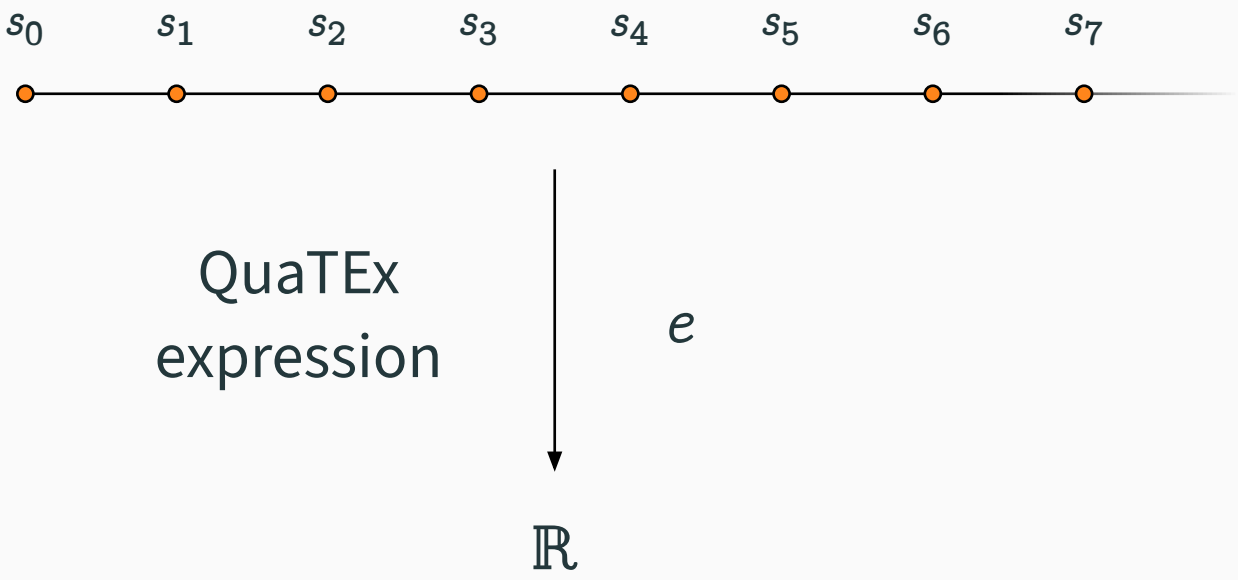
scheck command

```
usage: umaudemc scheck [-h] [-m NAME] [-M TERM] ...
                       file initial query [strategy]

positional arguments:
  file                  Maude source file
  initial               initial term
  query                 QuaTEx query
  strategy              strategy expression

options:
  -h, --help            show this help message and exit
  -m, --module NAME     specify the module for model checking
  -M, --metamodule TERM
```

Quantitative **T**emporal **Ex**pressions



Quantitative **T**emporal **Ex**pressions

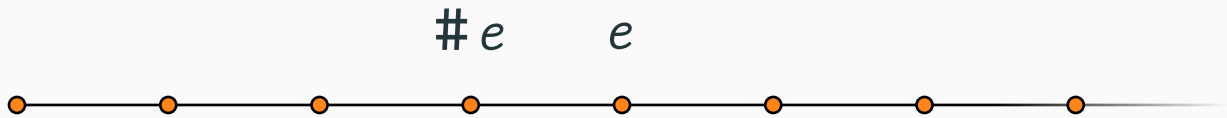
Literals	`1.0` `1` `"1"`
Variables	x
Operators	`+` `-` `*` `/` `%` `==` `!=` `>` `>=` `&&` `||`
Conditionals	`if` C `then` e_1 `else` e_2 `fi`

Quantitative **T**emporal **Ex**pressions

Literals	`1.0 1 "1"`
Variables	x
Operators	`+ - * / % == != > >= && ||`
Conditionals	`if` C `then` e_1 `else` e_2 `fi`
Observations	`s.rval("`t`")`

Quantitative **T**emporal **Ex**pressions

Literals	`1.0` `1` `"1"`
Variables	x
Operators	`+` `-` `*` `/` `%` `==` `!=` `>` `>=` `&&` `||`
Conditionals	`if` C `then` e_1 `else` e_2 `fi`
Observations	`s.rval("`t`")`
Next operator (○)	`#` e



Quantitative **T**emporal **Ex**pressions

Literals	`1.0 1 "1"`
Variables	x
Operators	`+ - * / % == != > >= && ||`
Conditionals	`if` C `then` e_1 `else` e_2 `fi`
Observations	`s.rval("`t`")`
Next operator ($\bigcirc$)	`#` e
Function definition	$f(\bar{x})$ `=` φ `;`
Query	`eval E[`φ`] ;`

Observations

`"steps"`	Number of steps from the beginning of the execution
`"time"`	Coincides with `"steps"` except for `pmaude` and `ctmc-`
Other	Numeric or Boolean Maude term with any number of ocurrences of a single variable that will be substituted by the current term

Statistics of a sample

Given a sample $\{x_1, ..., x_n\} \subseteq \mathbb{R}$

$$\mu = \frac{1}{n} \sum_{k=1}^{n} x_k$$

Mean

$$\sigma = \sqrt{\frac{\sum_{k=1}^{n} (x_k - \mu)^2}{n - 1}}$$

Standard deviation

$$[\mu - r, \mu + r]$$

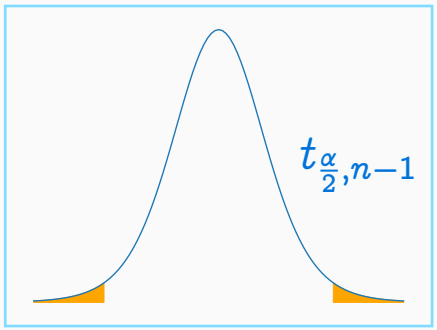
Confidence interval

Confidence interval

Given a sample $\{x_1, ..., x_n\}$ from a distribution with mean θ.

$$\frac{\mu - \theta}{\sigma/\sqrt{n}} \sim \text{Student}(n) \overset{\text{CLT}}{\to} \text{Normal}(0, 1)$$

$$I = (\mu - r, \mu + r) \quad \text{with} \quad r = \frac{t_{\frac{\alpha}{2}, n-1} \cdot \sigma}{\sqrt{n}}$$



Informally, "with probability $1 - \alpha$ we find an I that contains θ."

Simulation parameters

`--alpha` α — significance level ($1 - \alpha$ is the confidence level)

`--delta` δ — maximum radius of the confidence interval

`--block` b — block size between convergence checks

`--nsims` n — fixed number of simulations or range n–N

MultiVeSta's parametric queries

$$\texttt{eval parametric(E[}e\texttt{], }x\texttt{, }start\texttt{, }step\texttt{, }end\texttt{) ;}$$

will evaluate E[e] for all values

$$x := start + k \cdot step \leq end$$

MultiVeSta's parametric queries

`eval parametric(E[`e`], `x`, `$start$`, `$step$`, `end`) ;`

will evaluate `E[`e`]` for all values

$$x := start + k \cdot step \leq end$$

- Format strings `f"{x} * S"`
- Plots can be obtained (requires Matplotlib) `--plot`

Probability assignment methods

local weight-based

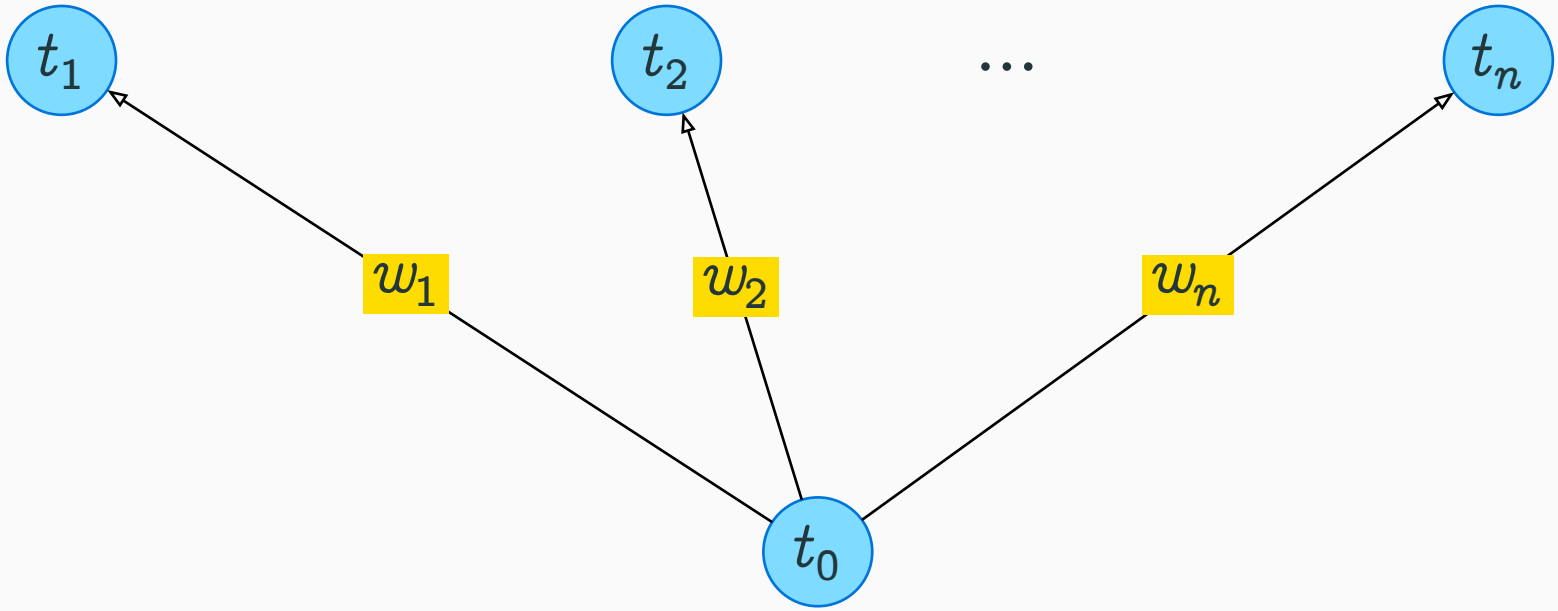
- `uniform`
- `uaction(`r_1`=`w_1`, …, `r_n`=`w_n`)`
- `term`
- `metadata`

- `strategy`

statistical only

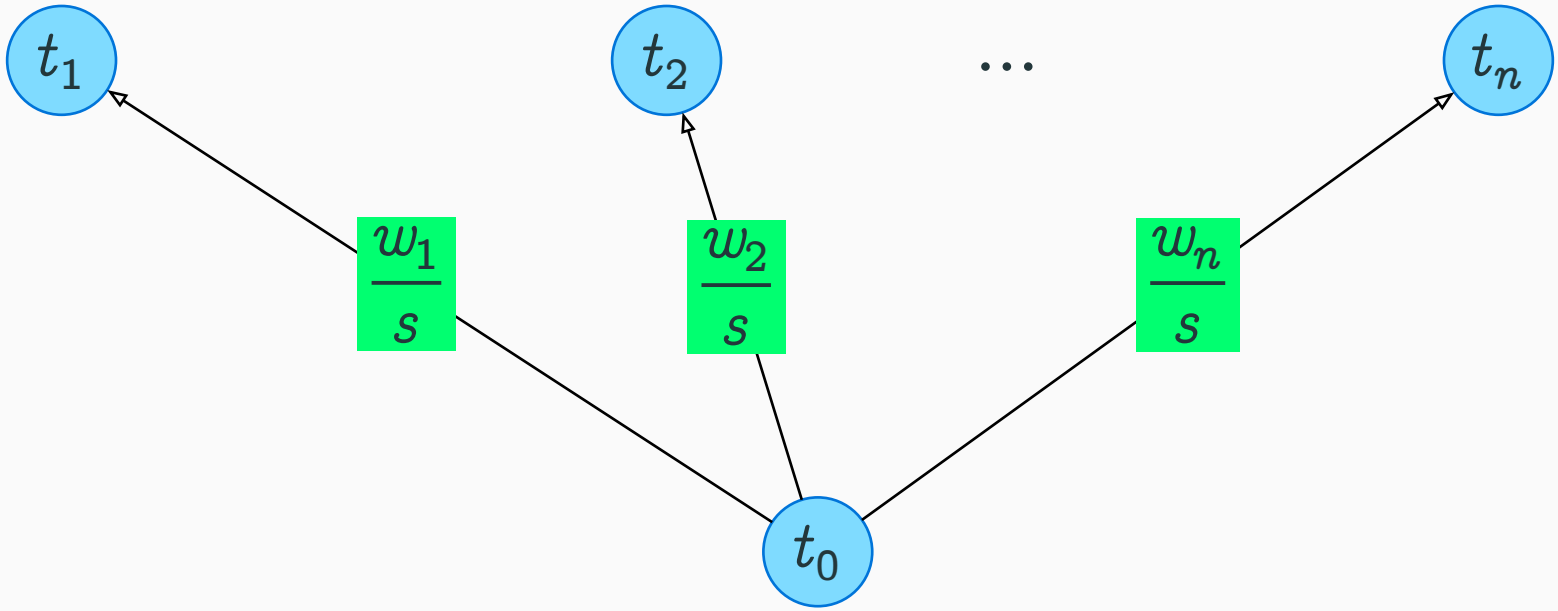
- `step`
- `pmaude`

Weight-based assignment methods



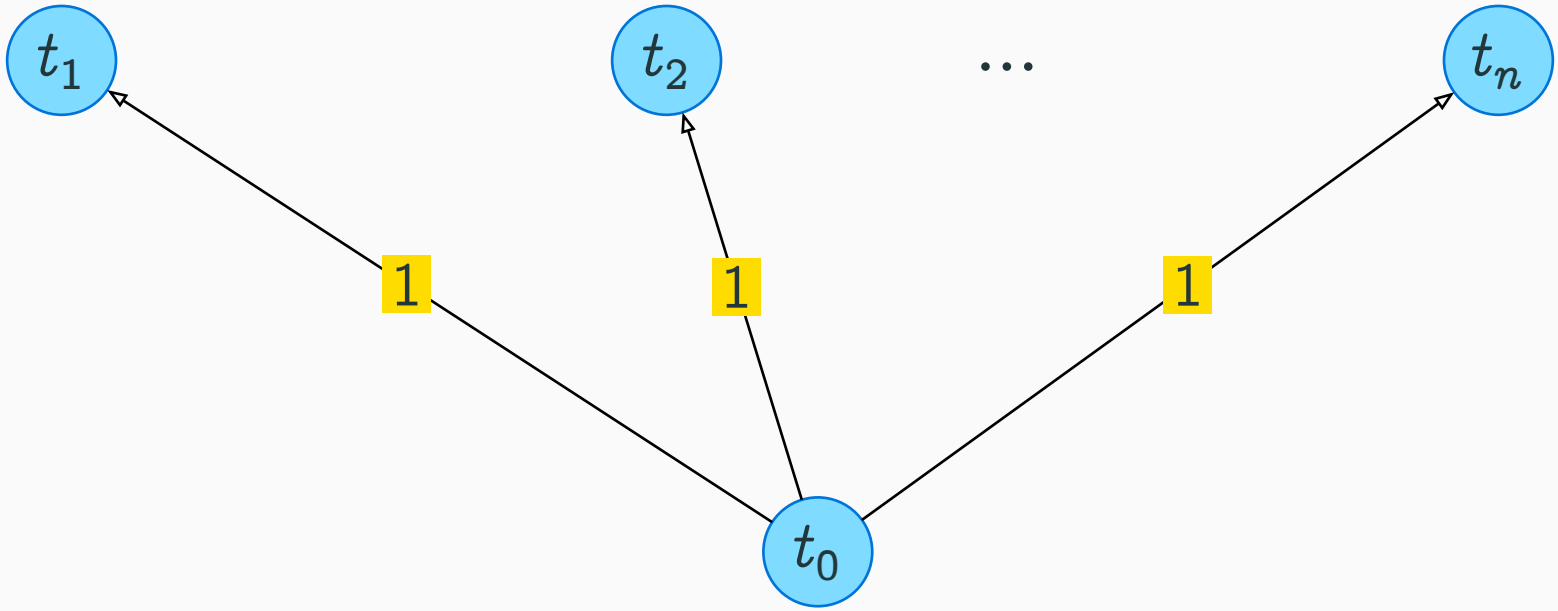
$$s = w_1 + \cdots + w_n$$

Weight-based assignment methods



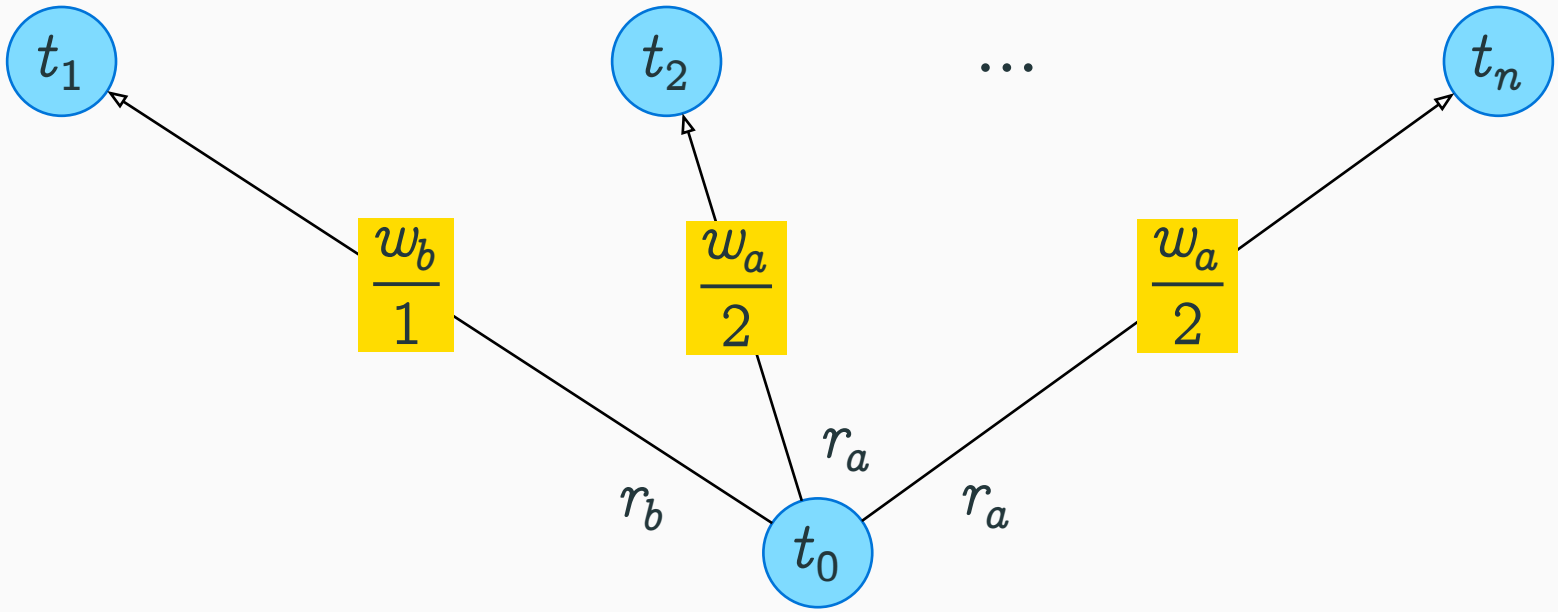
$$s = w_1 + \cdots + w_n$$

Weight-based assignment methods



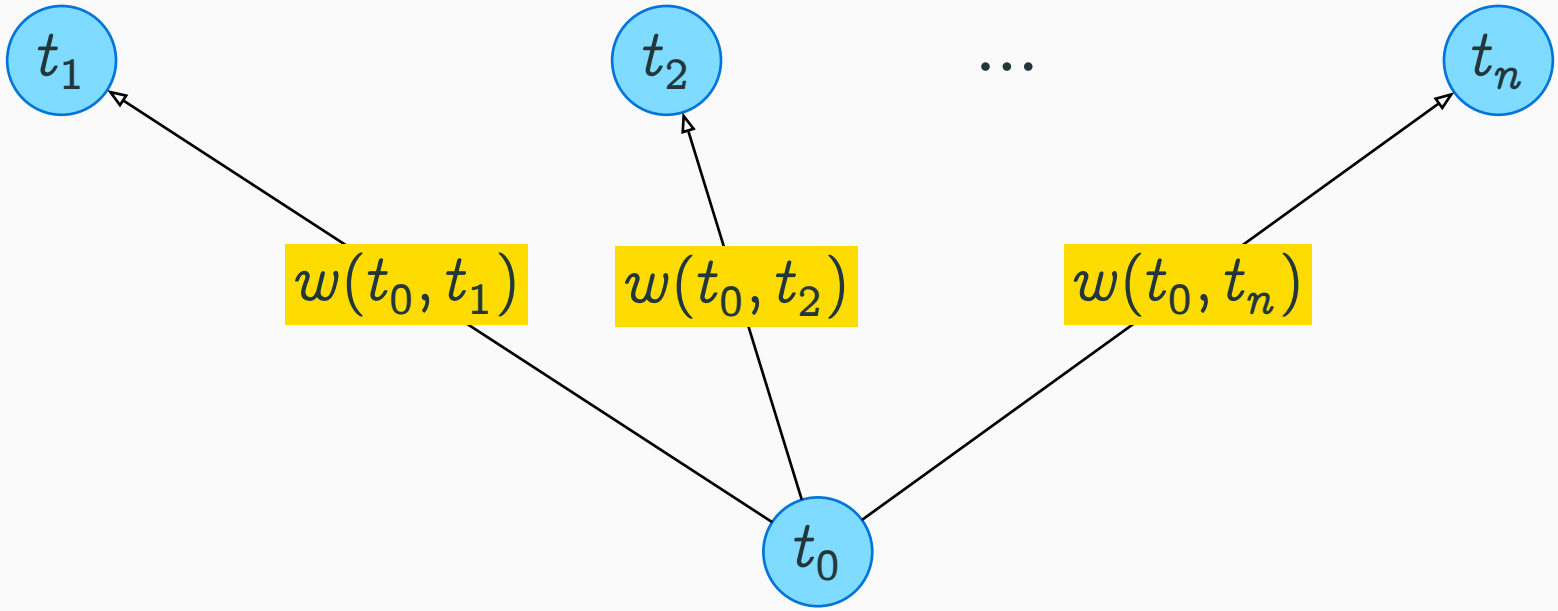
`uniform`

Weight-based assignment methods



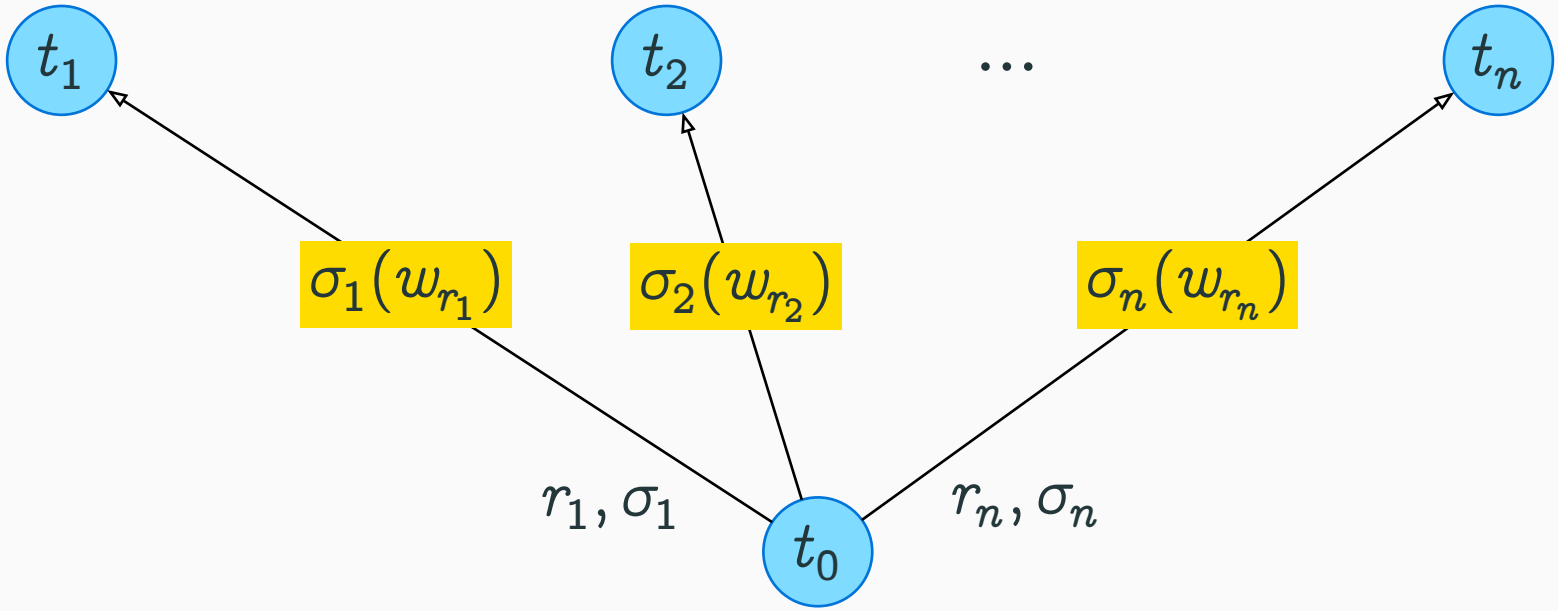
`uaction(`r_a`=`w_a`, …, `r_z`=`w_z`)`

Weight-based assignment methods



`term(`w`(`**L**`,`**R**`))`

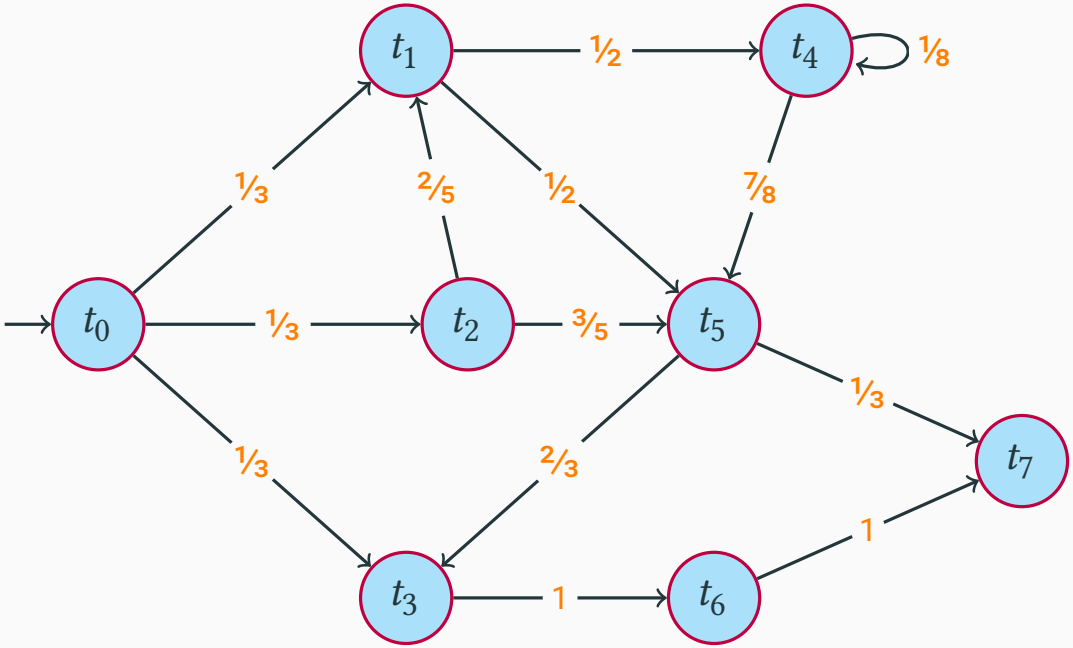
Weight-based assignment methods




```
metadata          rl l => r if C [metadata "w"]
```

Weight-based assignment methods

The quantified model becomes a discrete-time Markov chain (DTMC) (S, P, s_0) with $P : S \times S \to [0, 1]$ and $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$



Weight-based assignment methods

Each method m comes with a `mdp-`m and `ctmc-`m variant.

- `mdp-`m describes a Markov Decision Process (MDP), not for SMC

Weight-based assignment methods

Each method m comes with a `mdp-`m and `ctmc-`m variant.

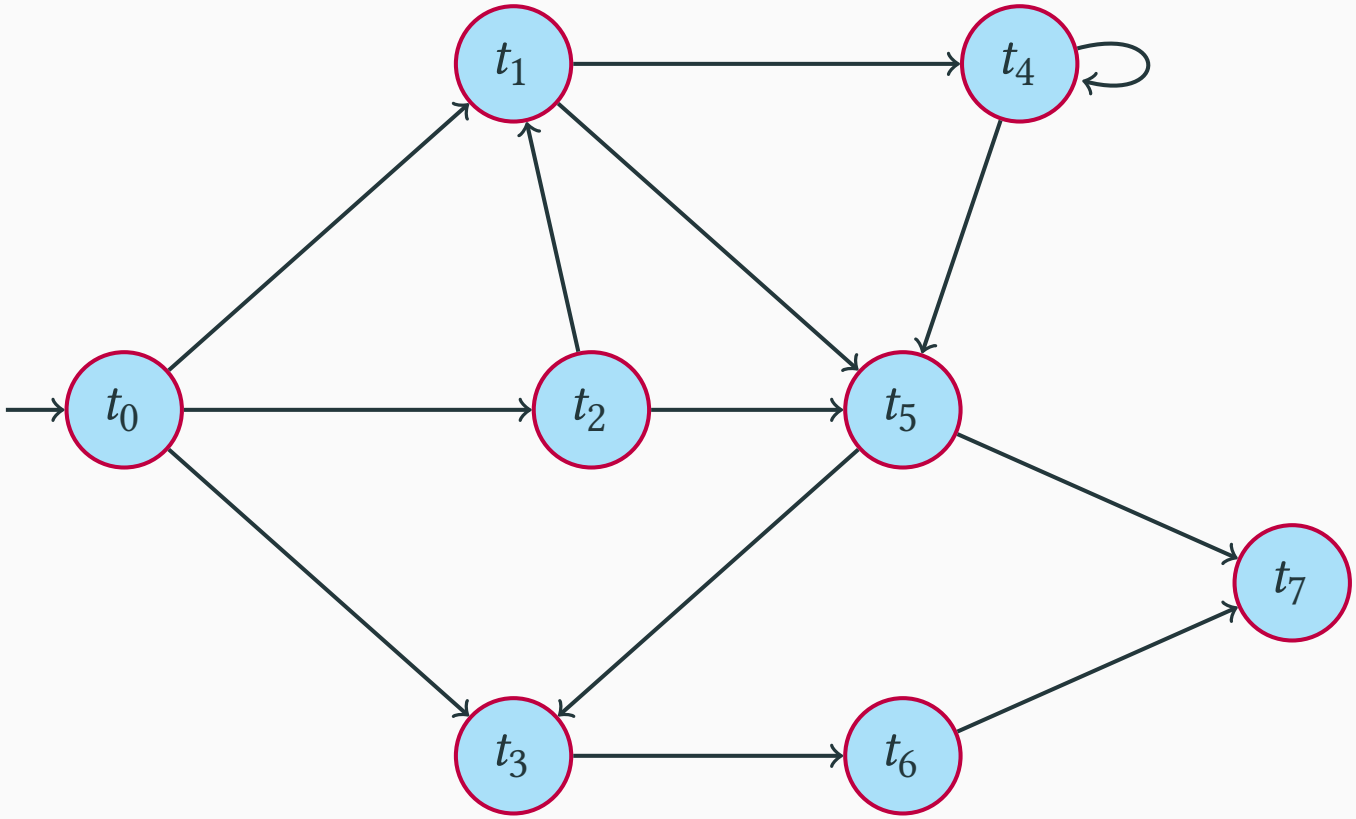
- `ctmc-`m describes a Continuous-Time Markov Chain (CTMC)
- The weights of each transition are interpreted as firing rates of an exponential distribution governing the transition.

Weight-based assignment methods

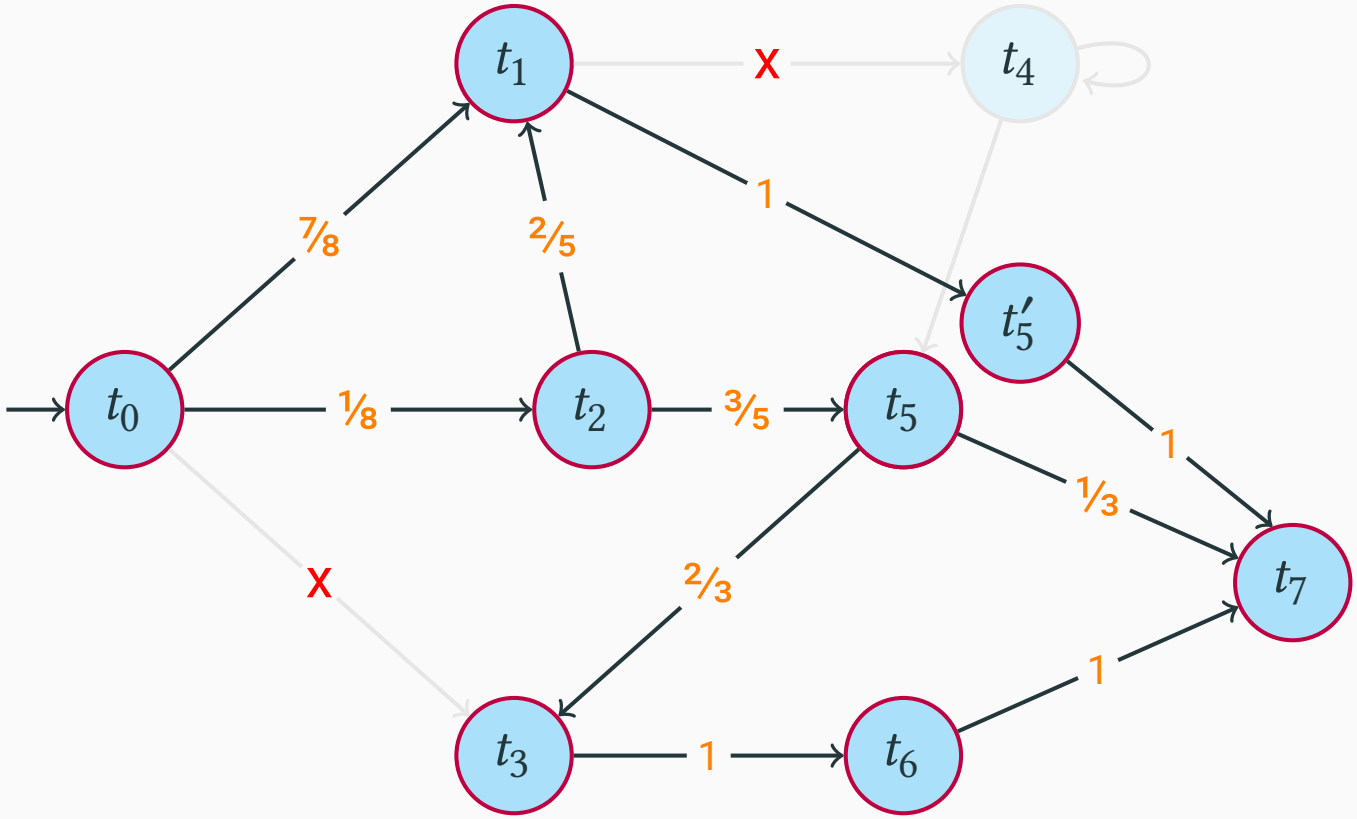
Each method m comes with a `mdp-`m and `ctmc-`m variant.

- `ctmc-`m describes a Continuous-Time Markov Chain (CTMC)
- The weights of each transition are interpreted as firing rates of an exponential distribution governing the transition.
- The probability of taking each transition is the same…
- …but it gives a notion of time passed in each state.

`strategy` assignment method



`strategy` assignment method



Maude strategy language

Controlled application of rules $\alpha \mapsto E \subseteq \mathrm{Paths}(\mathcal{R})$

📕 Eker, Martí-Oliet, Meseguer, Rubio, Verdejo. *The Maude strategy language*. JLAMP 134

Maude strategy language

Controlled application of rules $\alpha \mapsto E \subseteq \mathrm{Paths}(\mathcal{R})$

Application of a rule	$label[\rho]$
Test	`match` P `s.t.` C
Concatenation	α `;` β
Non-deterministic choice	α_1 `|` $\cdots$ `|` α_n
Iteration	α `*`

📕 Eker, Martí-Oliet, Meseguer, Rubio, Verdejo. *The Maude strategy language*. JLAMP 134

Maude strategy language

Controlled application of rules $\alpha \mapsto E \subseteq \mathrm{Paths}(\mathcal{R})$

Application of a rule	$label[\rho]$
Test	`match` P `s.t.` C
Conditional	α `?` β `:` γ
Rewriting of subterms	`matchrew`
Recursive strategies	

📕 Eker, Martí-Oliet, Meseguer, Rubio, Verdejo. *The Maude strategy language*. JLAMP 134

Probabilistic strategy language

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

Quantified matching $\texttt{matchrew}\ P\ \texttt{with weight}\ w$

Probabilistic strategy language

Quantified choice	$\texttt{choice}(w_1 : \alpha_1, \ldots, w_n : \alpha_n)$
Quantified matching	$\texttt{matchrew}\ P\ \texttt{with weight}\ w$
Sampling	$\texttt{sample}\ x := \pi(\bar{t})\ \texttt{in}\ \alpha$

$\pi \in \{\texttt{bernoulli}, \texttt{uniform}, \texttt{exp}, \texttt{norm}, \texttt{gamma}\}$

Probabilistic strategy language

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

Quantified matching $\texttt{matchrew}\ P\ \texttt{with weight}\ w$

Sampling $\texttt{sample}\ x := \pi(\bar{t})\ \texttt{in}\ \alpha$

$\pi \in \{\texttt{bernoulli}, \texttt{uniform}, \texttt{exp}, \texttt{norm}, \texttt{gamma}\}$

🔮 Future work: add other distributions: `lognormal`, `chi2`, `studentt`, `geometric`, etc.

`strategy` assignment method

- The most general of the assignment methods
- Memoryful strategies (Turing complete)
- Detects unquantified non-determinism and complains
- Full state space expansion due to failure semantics: `strategy-fast`

`strategy` assignment method

- The most general of the assignment methods
- Memoryful strategies (Turing complete)
- Detects unquantified non-determinism and complains
- Full state space expansion due to failure semantics: `strategy-fast`

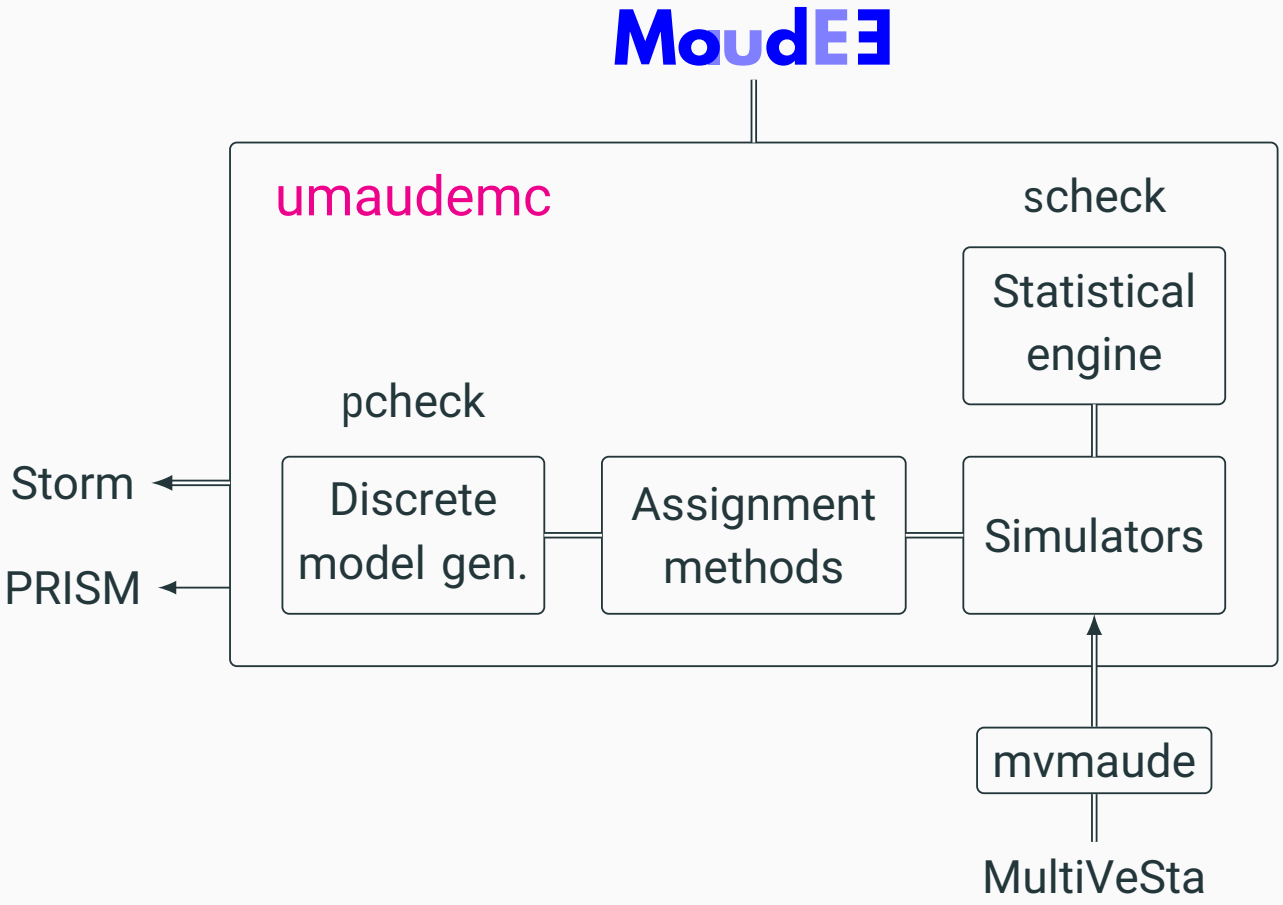
```
var F : Float .
sd bias(F) = choice(F : head, 1.0 - F : tail)
  ? bias(if F > 0.25 then 0.9 * F else F fi)
  : idle .
```

`step` assignment method

- Only for statistical model checking
- No explicit check for unquantified non-determinism
- The default method for `scheck` when a strategy is given

Strategy	✅	❌
Step	Full execution of the (probabilistic) strategy	Single rule rewrite

Extensible architecture



Recapitulation

- How to use `umaudemc scheck` for statistical model checking
- QuaTEx to express quantitative properties
- Assignment methods to add probabilities to Maude models

- Knuth-Yao example: number of throws, probability of faces

Conditional expected values

`discard` discards the current execution for that query

$\longrightarrow$ conditional expected values

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \qquad E[f|B] = \frac{1}{P(B)} \int_B f \, dP$$

where B would be “not being discarded”.

The probabilistic language Prob (example)

Syntax

```
op _return_ : ProbStmt FTuple -> ProbProgram [ctor] .
op _=_ : FVar FExpr -> ProbStmt [ctor] .
op _~_ : FVar FDistribution -> ProbStmt [ctor] .
op _;_ : ProbStmt ProbStmt -> ProbStmt [ctor assoc] .
op while_do_done : BoolExpr ProbStmt -> ProbStmt [ctor] .
```

The probabilistic language Prob (example)

Syntax

```
op _return_ : ProbStmt FTuple -> ProbProgram [ctor] .
op _=_ : FVar FExpr -> ProbStmt [ctor] .
op _~_ : FVar FDistribution -> ProbStmt [ctor] .
op _;_ : ProbStmt ProbStmt -> ProbStmt [ctor assoc] .
op while_do_done : BoolExpr ProbStmt -> ProbStmt [ctor] .
```

Operational semantics

```
rl [step] : < FV = F ; S return R | VM > =>
            < S return R | VM[FV / instantiate(F, VM)] > .
```

The probabilistic language Prob (example)

Syntax

```
op _return_ : ProbStmt FTuple -> ProbProgram [ctor] .
op _=_ : FVar FExpr -> ProbStmt [ctor] .
op _~_ : FVar FDistribution -> ProbStmt [ctor] .
op _;_ : ProbStmt ProbStmt -> ProbStmt [ctor assoc] .
op while_do_done : BoolExpr ProbStmt -> ProbStmt [ctor] .
```

Operational semantics

```
rl [sample] : < FV ~ FD ; S return R | VM > =>
              < S return R | VM[FV / F] > [nonexec] .
```

The probabilistic language Prob (example)

Execution step

```
sd sstep := step
  | matchrew X s.t. < FV ~ D ; S return R | VM > := X
     by X using sample(D, VM) .
```

The probabilistic language Prob (example)

Execution step

```
sd sstep := step
  | matchrew X s.t. < FV ~ D ; S return R | VM > := X
     by X using sample(D, VM) .
```

```
sd sample(bernoulli(F), VM) = … .
sd sample(uniform(F), VM) = … .
sd sample(exponential(F), VM) =
      sample FC := exp(instantiate(F, VM))
      in sample[F <- FC] .
```

The probabilistic language Prob (example)

```
eq ex1a = c1 ~ exponential(1.0) ;
          c2 ~ exponential(1.0)
          return (c1 + c2) .
```

The probabilistic language Prob (example)

```
eq ex1a = c1 ~ exponential(1.0) ;
          c2 ~ exponential(1.0)
          return (c1 + c2) .


eq ex2b = c1 ~ exponential(1.0) ;
          c2 ~ exponential(1.0) ;
          observe(c1 >= c2)              *** discard
          return (c1 + c2) .
```

SMC in Maude

Electronic Notes in Theoretical Computer Science 153 (2006) 213–239

PMaude: Rewrite-based Specification Language for Probabilistic Object Systems

Gul Agha[1] José Meseguer[2] Koushik Sen[3]

Department of Computer Science,
University of Illinois at Urbana Champaign, USA.

Probabilistic rewrite theories

$$\texttt{rl}\ l(\bar{x})\ \texttt{=>}\ r(\bar{x},\bar{y})\ \texttt{s.t.}\ C(\bar{x})\ \texttt{with probability}\ \bar{y} \coloneqq \pi(\bar{x})\ \texttt{.}$$

- Probabilistic rewrite theories $\mathcal{R} = (\Sigma, E \cup A, R, \pi)$
- Syntax for defining probabilistic rules (not implemented in practice)

Probabilistic rewrite theories

$$\texttt{rl}\ l(\bar{x})\ \texttt{=>}\ r(\bar{x},\bar{y})\ \texttt{s.t.}\ C(\bar{x})\ \texttt{with probability}\ \bar{y} \coloneqq \pi(\bar{x})\ \texttt{.}$$

- Probabilistic rewrite theories $\mathcal{R} = (\Sigma, E \cup A, R, \pi)$
- Syntax for defining probabilistic rules (not implemented in practice)
- The choice of rule, position and substitution for $\bar{x}$ is still non-deterministic, so Actor PMaude

PMaude operational behavior

As implemented in Vesta, PVesta, MultiVeSta, and QMaude's `pmaude`:

🧑 Users need to provide

1. a `Config` sort
2. `op getTime : Config -> Float`
3. `op tick : Config -> Config`
4. `op val : Nat Config -> Float` for observations (optional)

⚙️ The engine

1. rewrites the initial configuration exhaustively to obtain t_0
2. rewrites `tick`(t_k) exhaustively to obtain t_{k+1}
3. repeats (2) on demand of the QuaTEx formula

PMaude operational behavior

The user *programs* the random behavior with

- the `random : Nat -> Nat` operator of the Maude prelude
- the special `counter` constant of sort `Nat` of the Maude prelude, which rewrites to increasing natural numbers each time
- a library of probability distributions programmed in Maude

PMaude operational behavior

The user *programs* the random behavior with

- the `random : Nat -> Nat` operator of the Maude prelude
- the special `counter` constant of sort `Nat` of the Maude prelude, which rewrites to increasing natural numbers each time
- a library of probability distributions programmed in Maude

😀 Lots of flexibility

PMaude operational behavior

The user *programs* the random behavior with

- the `random : Nat -> Nat` operator of the Maude prelude
- the special `counter` constant of sort `Nat` of the Maude prelude, which rewrites to increasing natural numbers each time
- a library of probability distributions programmed in Maude

😀 Lots of flexibility

😞 No separation of concerns, no declarative, and error prone (unquantified non-determinism) ⟶ APMaude + P transform

Exponential clock (example)

```
***          time  charge
op clock  : Float Float -> Clock [ctor] .
op broken : Float Float -> Clock [ctor] .

vars T C D : Float .  var B : Bool .

rl [advance] : clock(T     , C                ) =>
    if B then  clock(T + D, C - C / 1000.0)
         else broken(T     , C - C / 1000.0)
    fi
  with probability B := bernoulli(C / 1000.0),
                   D := exponential(1.0) .
```

Exponential clock as written (example)

```
protecting PMAUDE-DISTRIBUTIONS + COUNTER .

rl [advance] : clock(T    , C              ) => frozen(
    if B then  clock(T + D, C - C / 1000.0)
         else broken(T    , C - C / 1000.0)
    fi)
if B := sampleBernoulli(random(counter), C / 1000.0),
/\ D := sampleExpWithRate(random(counter), 1.0) .

subsort Clock < Config .
eq getTime(clock(T, C)) = T . *** same for others
eq tick(frozen(S:State)) =  S:State .
```

Exponential clock as written (example)

```
protecting PMAUDE-DISTRIBUTIONS + COUNTER .

rl [advance] : clock(T    , C              ) => frozen(
    if B then  clock(T + D, C - C / 1000.0)
         else broken(T    , C - C / 1000.0)
    fi)
if B := sampleBernoulli(random(counter), C / 1000.0),
/\ D := sampleExpWithRate(random(counter), 1.0) .
```

```
subsort Clock < Config .
eq getTime(clock(T, C)) = T . *** same for others
eq tick(frozen(S:State)) =  S:State .
```

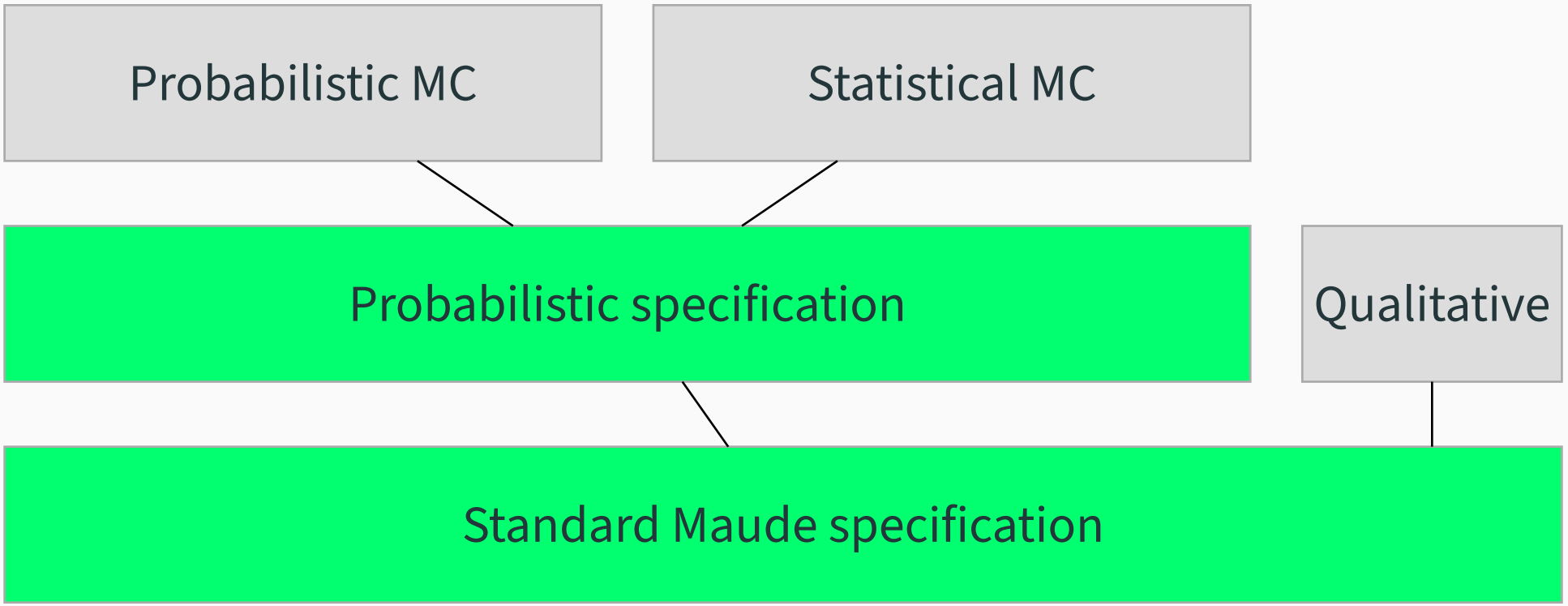
Exponential clock (example)

QuaTEx formula for the expected charge when the clock gets broken:

```
BreakCharge() = if (s.rval("isBroken(Clk:Clock)"))
  then
    s.rval("charge(Clk:Clock)")  // or "time"
  else
    # BreakCharge()
  fi ;

eval E[ BreakCharge() ];
```

QMaude overview



Object-oriented modules in Maude

Module	`omod` M `is` `...` `endom`
Class definitions	`class` C `|` a_1 `:` s_1`,` …`,` a_n `:` s_n
Subclass relations	`subclass` C_1 `<` C_2
Messages	`msg` m `:` s_1 ⋯ s_n `->` s
Objects	`<` o `:` C `|` a_1 `:` t_1`,` …`,` a_n `:` t_n `>`

- Objects and messages live in a soup (configuration, multiset)
- Objects are assumed to have a fixed set of attributes
- Unused or unmodified attributes can be ommited in equations and rules

Actor PMaude and generalized actor rewrite system

- Rules

Message-triggered	$\texttt{<}\ o\ \texttt{:}\ C\ \texttt{|}\ attrs\ \texttt{>}\ \texttt{(to}\ o\ \texttt{[from}\ o'\texttt{]}\ \texttt{:}\ mp\texttt{)}$ $\texttt{=>}\ \texttt{<}\ o\ \texttt{:}\ C\ \texttt{|}\ attrs'\ \texttt{>}\ msgs\ newobjs\ \texttt{[if}\ C\texttt{]}$
Object-triggered	$\texttt{<}\ o\ \texttt{:}\ C\ \texttt{|}\ attrs\ \texttt{>}$ $\texttt{=>}\ \texttt{<}\ o\ \texttt{:}\ C\ \texttt{|}\ attrs'\ \texttt{>}\ msgs\ newobjs\ \texttt{[if}\ C\texttt{]}$

- Probability of message delays (per rule and initial, basic and modulation)

$$\Pi = \{(\pi_r, \delta_r)\}_{r \in R} \quad \cup \quad \{(\pi_{\text{init}}, \delta_{\text{init}})\}$$

P and related transformations

From $\mathcal{R} = (\Sigma, E, L, R)$ a generalized actor rewrite theory, and Π defining the message delay distributions:

- $P : (\mathcal{R}, initconf, \Pi) \mapsto (\mathcal{R}_\Pi, initconf_\Pi)$
- $Sim : \mathcal{R}_\Pi \mapsto Sim(\mathcal{R}_\Pi)$ for simulation (what Vesta/`scheck` will use)
- M (*monitor*) adds logging on $Sim(\mathcal{R}_\Pi)$ to facilitate expressing properties

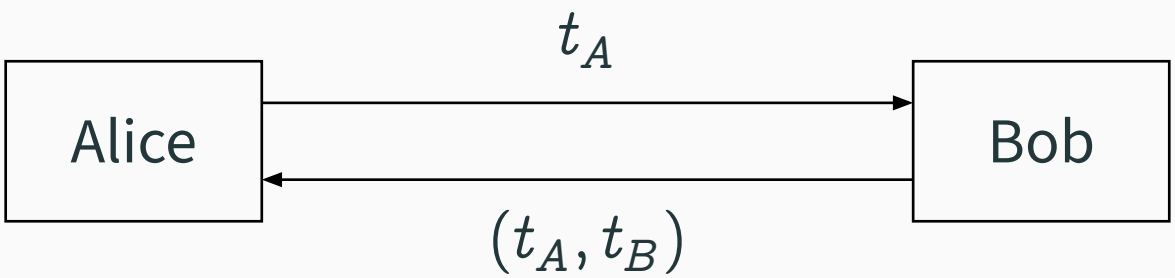
📕 Liu, Meseguer, Ölveczky, Zhang, Basin. *Bridging the semantic gap between qualitative and quantitative models of distributed systems*. OOPSLA 2022

Conditions for the P transformation

(1) In any concrete configuration of objects and messages, any object *enabled* by a message-triggered rule to receive a given message addressed to it, is enabled to receive that message by the application of a *unique* message-triggered rule with a *unique* substitution.
(2) In any concrete configuration, any object *enabled* to perform a transition by an object-triggered rule, is so enabled by a *unique* object-triggered rule with a *unique* substitution.
(3) *At most one* object in *initconf* is enabled to be rewritten by an object-triggered rule.
(4) In *initconf*, if an object has a message addressed to it, then it is enabled to receive it, and it is *not* enabled to be rewritten by an object-triggered rule.
(5) In any configuration reachable from *initconf*, if an object is in a state *not* enabled by any object-triggered rule and the configuration contains a message addressed to it, then it is enabled to receive such a message.
(6) In any configuration reachable from *initconf*, if a (†) or (‡) rule applies to an object with the ground substitution θ, then all addressees in the (normal form of the) ground set of messages $msgs\theta$ in the instance of the applied rewrite rule are objects that either: (i) belong to the current configuration, or (ii) are among the new objects in the set $newobjs\theta$ introduced in the instance of the applied rewrite rule.
(7) Any rewrite sequence where in each step some object-triggered rule is applied to the same object must be finite.

Round trip time (example)

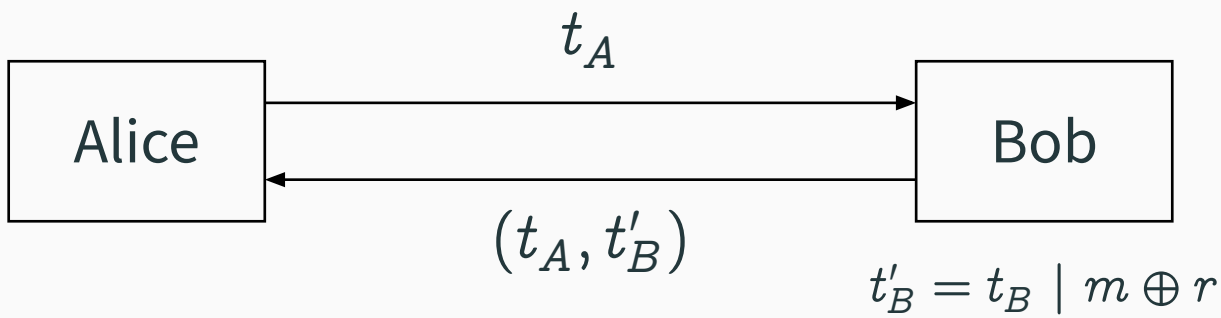
Hidden information transmission in a protocol to calculate the round trip time



from the Maude-HCS project by Minyoung Kim and Carolyn Talcott

Round trip time (example)

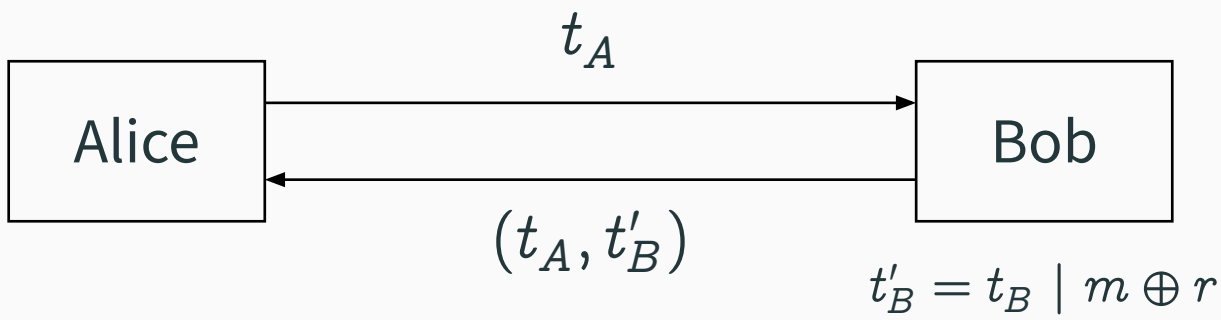
Hidden information transmission in a protocol to calculate the round trip time



from the Maude-HCS project by Minyoung Kim and Carolyn Talcott

Round trip time (example)

Hidden information transmission in a protocol to calculate the round trip time



$t_B - t_A$ 🕵 entropy

from the Maude-HCS project by Minyoung Kim and Carolyn Talcott

Round trip time (example)

Alice sends t_A to Bob

```
crl [sndRttReq] :
  < A : Snd | timers: [tt ; per ; data], tsq: tsl,
              clock: T, stopT: Tstop, rcv: B >
=>
  < A : Snd | timers: [tt0 ; per ; data], tsq: (tsl ; ts) >
  [ (to B from A : rttReq(ts)) ]

if tt == zero
/\ tt0 := if T ge Tstop then infty else nat2t(per) fi
/\ ts  := t2ts(T) .
```

Round trip time (example)

Bob receives t_A from Alice and replies

```
rl [rcvRttReq]:
   < B : Rcv | clock: T, snd: A >
   (to B from A : rttReq(ts))
=>
   < B : Rcv | >
   [ (to A from B : rttResp(ts, t2ts(T))) ]
.
```

Round trip time (example)

Alice receives (t_A, t_B) from Bob

```
rl [rcvRttResp] :
  < A : Snd | clock: T, rcv: B, timers: [tt0 ; per ; data],
              tsq: (tsl0 ; ts ; tsl1), rttq: rttl >
  (to A from B : rttResp(ts, ts0))
=>
  < A : Snd | timers: [tt0 ; per ; data],
              tsq: (tsl0 ; tsl1),
              rttq: (rttl ; computeRtt(T, ts)) > .
```

Round trip time (example)

Bob inserts a hidden byte in t_B

```
crl [rcvWRttReq] :
  < B : bWRcv | clock: T, snd: A, bctr: i, ctr: j,
                byteList: bytel >
  (to B from A : rttReq(ts0))
=>
  < B : bWRcv | bctr: s i, ctr: s j,
                byteList: bytel ; byte >
  [ (to A from B : rttResp(ts0, ts1)) ]
if ts   := t2ts(T)
/\ byte := b(sampleUniWithIntX(i, 256))
/\ ts1  := embed(ts, j, byte) .
```

Round trip time (example)

Alice recovers the byte from Bob

```
crl [rcvWRttResp] :
  < A : bWSnd | clock: T, rcv: B, rttq: rttl, ctr: j,
                tsq: (tsl ; ts0 ; tsl1), byteList: bytel >
  (to A from B : rttResp(ts0, ts1))
=>
  < A : bWSnd | rttq: (rttl ; computeRtt(T,ts0)), ctr: s j,
                tsq: (tsl ; ts0 ; tsl1),
                byteList: (bytel ; byte)>

if byte := extract(ts1, j) .
```

Round trip time (example)

QuaTEx expression for RTT and entropy as measured by the observer:

```
wrtt() = s.rval("getRttAv(getObserver(S))");
entropy() = s.rval("getEnAv(getObserver(S))");

eval E[ wrtt()    ];                    // Query 1
eval E[ entropy() ];                    // Query 2
```

Round trip time (example)

QuaTEx expression for RTT and entropy as measured by the observer:

```
wrtt() = s.rval("getRttAv(getObserver(S))");
entropy() = s.rval("getEnAv(getObserver(S))");

eval E[ wrtt()    ];                    // Query 1
eval E[ entropy() ];                    // Query 2
```

❓ Would the inclusion of the hidden message be noticed?

Round trip time (example)

RTT and entropy with no message

```
$ umaudemc scheck rtt-smc iSrtt wrtt.quatex --assign pmaude -d 0.5
Number of simulations = 180
Query 1 (wrtt.quatex:3:1)
  μ = 100.79013227513225     σ = 3.2633107589237724     r = 0.47997265122538463
Query 2 (wrtt.quatex:4:1) (30 simulations)
  μ = 0.6135072604012983     σ = 0.0790158588647808     r = 0.02950500660132106
```

RTT and entropy with hidden message

```
$ umaudemc scheck rtt-smc iSwrtt wrtt.quatex --assign pmaude -d 0.5
Number of simulations = 180
Query 1 (wrtt.quatex:3:1)
  μ = 100.60623015873011     σ = 3.1662056469951327     r = 0.46569028541254426
Query 2 (wrtt.quatex:4:1) (30 simulations)
  μ = 0.6124656163684576     σ = 0.0722734517126548     r = 0.02698735039419551
```

Overriding delta per query

```
wrtt() = s.rval("getRttAv(getObserver(S))");
entropy() = s.rval("getEnAv(getObserver(S))");
eval E[wrtt()];                                  // Query 1 default delta = 0.5
eval E[entropy()] with delta = 0.01;             // Query 2
```

```
$ umaudemc scheck rtt-smc iSrtt wrtt.quatex  --assign pmaude
Number of simulations = 240
Query 1 (wrtt.quatex:3:1) (180 simulations)
  μ = 101.50120370370372     σ = 3.161024938498901      r = 0.46492829902023936
Query 2 (wrtt.quatex:4:1)
  μ = 0.6044756398224311     σ = 0.07507941114512935    r = 0.009547025985316458
```

The inner workings

Statistical model-checking algorithm

Chub-Robbins method

run initial block of simulations

check radius attained

while radius or minimum of executions not attained

> run another block of simulations
>
> check radius attained

return results

Statistical model-checking algorithm

Chub-Robbins method

run initial block of simulations

check radius attained

while radius or minimum of executions not attained

> run another block of simulations
>
> check radius attained

return results

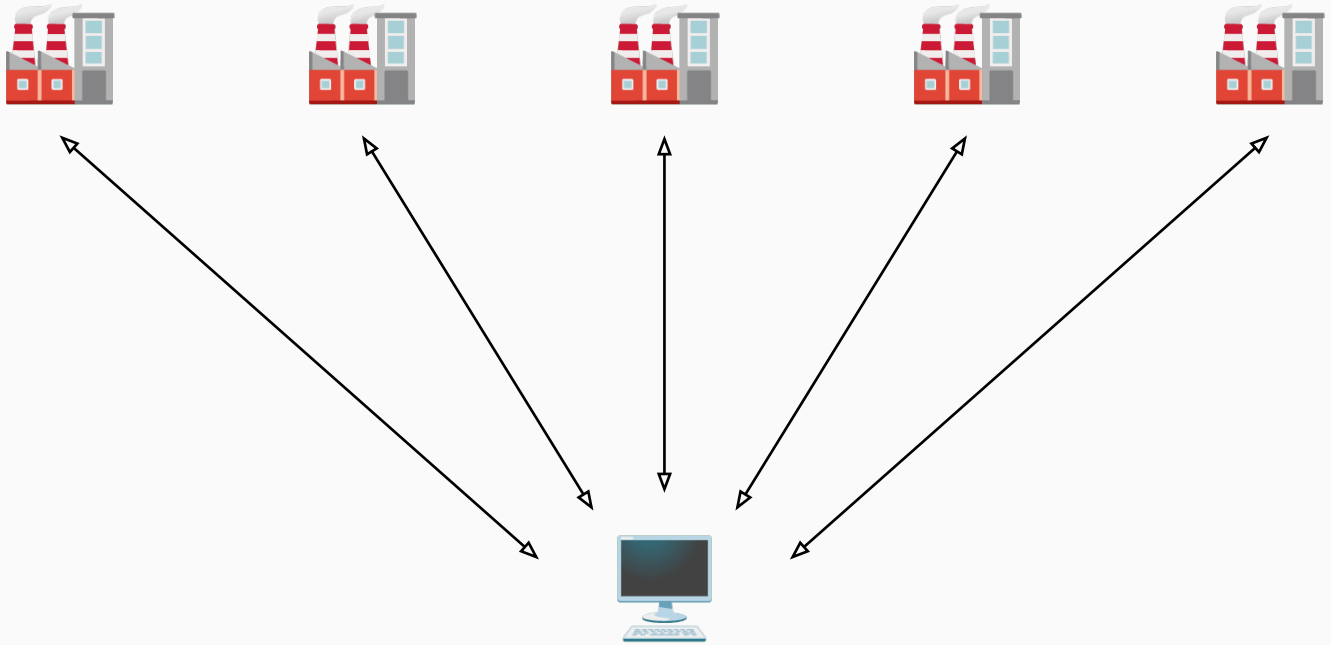
🔮 Future work: implement alternative convergence criteria

Parallelization

How the blocks of executions are run?

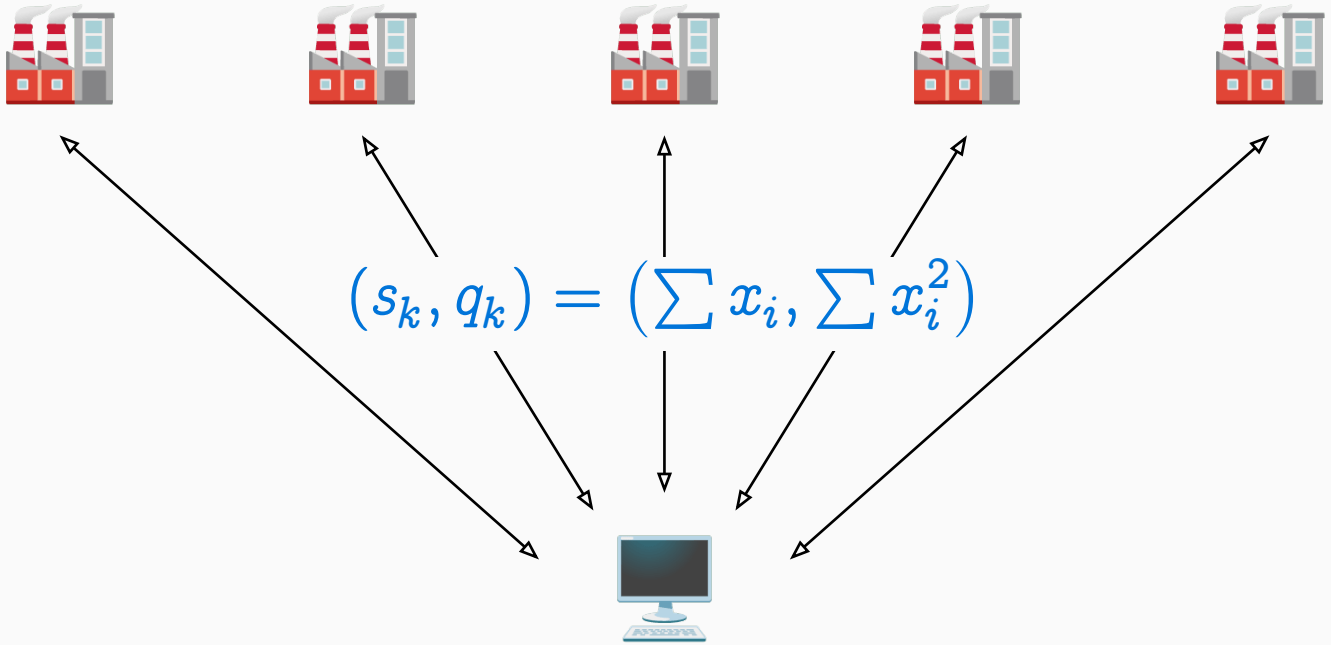
- Sequential
- Parallel (in the same machine)
- Distributed (multiple machines)

Parallel simulation



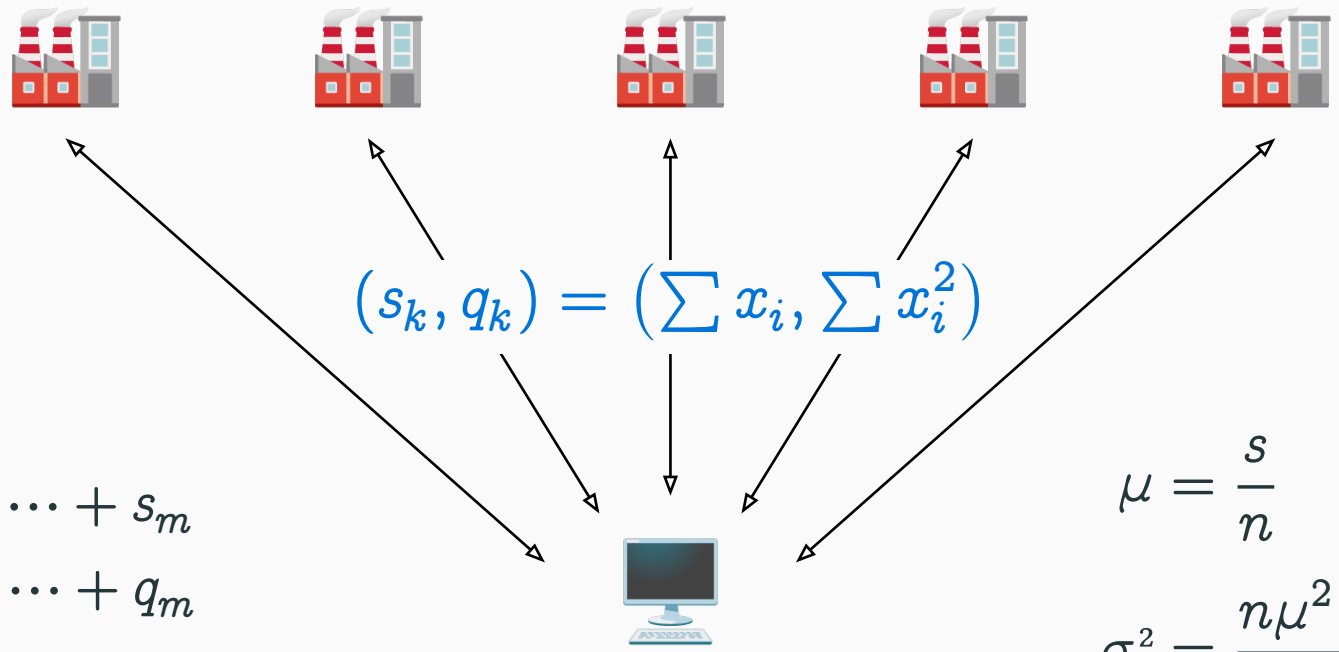
```
scheck -j <jobs> ...
```

Parallel simulation




```
scheck -j <jobs> ...
```

Parallel simulation



$$s = s_1 + \cdots + s_m$$

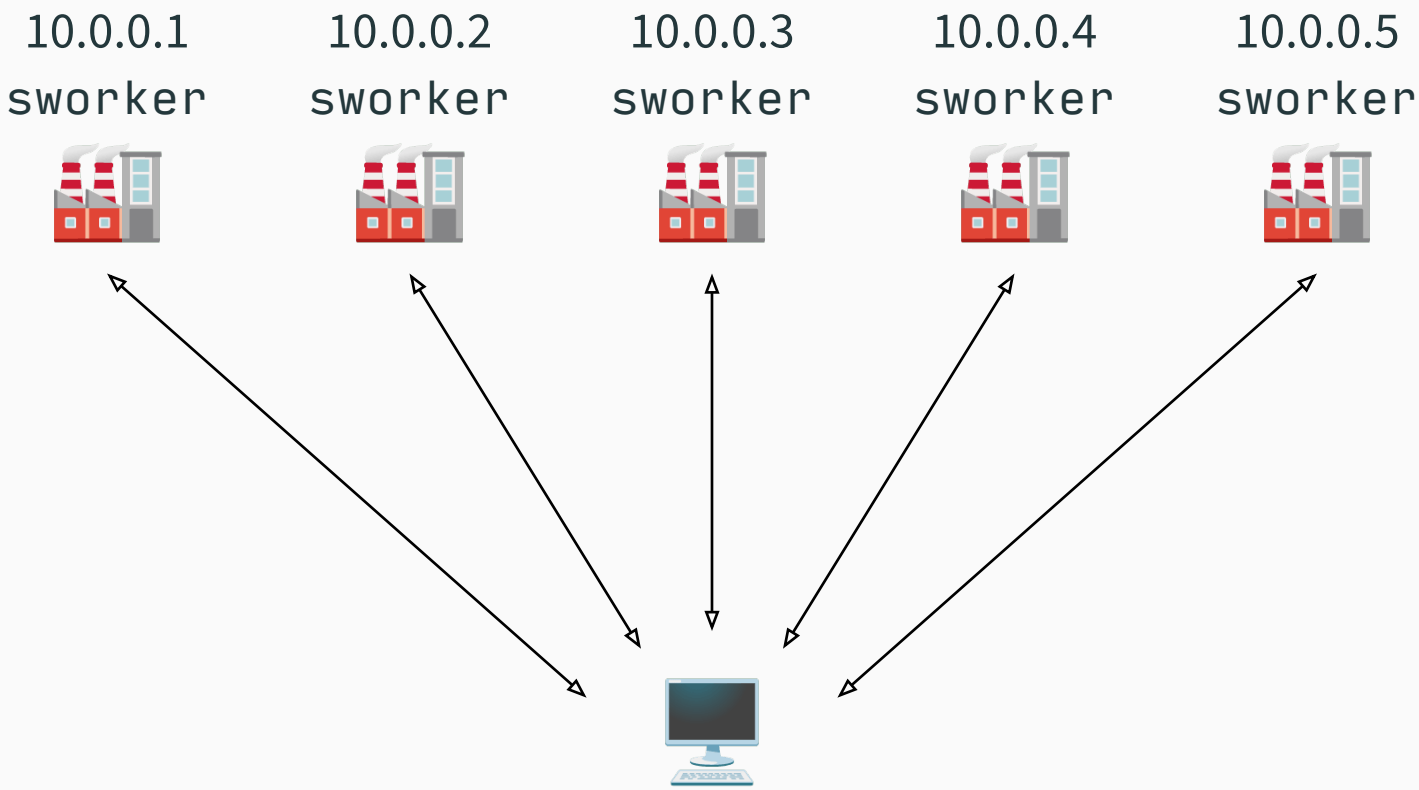
$$q = q_1 + \cdots + q_m$$

$$\mu = \frac{s}{n}$$

$$\sigma^2 = \frac{n\mu^2 + q - 2\mu s}{n - 1}$$

```
scheck -j <jobs> ...
```

Distributed simulation




```
scheck --distribute <workers> ...
```

Distributed simulation

In each machine running simulations:

```
$ umaudemc sworker -p <port>
```

Distributed simulation

In each machine running simulations:

```
$ umaudemc sworker -p <port>
```

In the mothership:

```
$ umaudemc scheck ... --distribute workers.toml
```

with the TOML (or JSON or YAML) file:

```
workers = ["10.0.0.1:1234", "10.0.0.2:1234", ...]
```

New features of QMaude

- Distributed model checking
- `--dump` to obtain all simulation values
- Early termination of parallel queries
- `step` assignment method without strategies
- QuaTEx
  - Overwriting δ per query (`with delta`)
  - `discard`
  - Importation of QuaTEx files
  - Constants: defined with `-D`c`=`v and used as `$`c

Summing up

- How to do SMC with `umaudemc scheck` and QuaTEx
- Alternative probability assignment methods
- Advanced features: conditional expected values, etc.
- PMaude and Actor PMaude specifications

🔮 Future work: implement $P \circ Sim$ as another assignment method, `actor`.

Some applications of SMC with Maude

📕 Liu, Rahman, Skeirik, Gupta, Meseguer. *Formal modelling and analysis of Cassandra in Maude*. ICFEM 2014

📕 Alturki, Kanovich, Kirigin, Nigam, Scedrov, Talcott. *Statistical model checking of distance fraud attacks on the Hancke-Kuhn family of protocols*. CPS-SPC 2018

📕 AlTurki, Rosu. *Statistical model checking of distance fraud attacks on the Hancke-Kuhn family of protocol*. FM 2019

📕 Durán, Ramírez, Rocha. *A rewriting logic semantics for the analysis of P programs*. JLAMP 144 (2025)

📕 Olarte, Ramírez, Rocha, Valencia. *Unified opinion formation analysis in rewriting logic*. JLAMP 148 (2026)

Acknowledgements

- Thanks to the DARPA Provably Weird Network Deployment and Detection (PWND^2) program for using QMaude and providing analysis challenges that lead to several advanced features of QMaude, thus enhancing the expressiveness and usability.

- Thanks to the Spanish *Agencia Estatal de Investigación* (MCIU/AEI/10.13039/501100011033/FEDER,EU) for supporting the development of QMaude and the associated research through projects ProCode and ProCode 10.

Thank you

maude.ucm.es/qmaude