Comparing the Modeling Capabilities of UPPAAL and RealySt for Stochastic Hybrid Systems

Final Talk

Seraphim Zaytsev
Supervision: Prof. Erika Ábrahám (THS)

LuFG Theory of Hybrid Systems

SS 2025

RWTH AACHEN UNIVERSITY

Motivation

- **Hybrid systems:** continuous evolution of variables (flows) with discrete transitions (jumps)
- **Stochastic hybrid systems:** add probabilistic timing/events
- Need tools that handle time, uncertainty
- **Goal today:** Present the final modeling coverage and benchmark results for UPPAAL vs. RealySt.

Outline

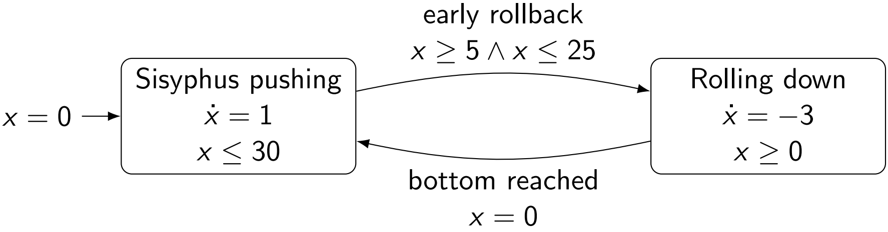
Outline

Hybrid Automata (HA): Intuition & Notation

$\mathcal{H} = (\mathrm{Loc}, \mathrm{Var}, \mathrm{Flow}, \mathrm{Inv}, \mathrm{Lab}, \mathrm{Edge}, \mathrm{Init})$

- **Flows:** in location ℓ, variables $x \in \mathbb{R}^d$ evolve with $\dot{x} = \mathrm{Flow}(\ell)(x)$ while $x \in \mathrm{Inv}(\ell)$
- **Jumps:** (ℓ, a, g, r, ℓ'): enabled if $x \in g$, apply reset r, target must satisfy $\mathrm{Inv}(\ell')$, $a \in Lab$
- **Runs:** alternate time steps and jumps

Example: Sisyphus as an HA



One variable $x \in [0, 30]$, linear rates, early rollback encodes the "curse"

Outline

Rectangular Automata (RA)

- **Guards & Invariants:** rectangles in valuation space: $x_1 \in [l_1, u_1]$, no diagonal constraints (e.g. $x_1 \leq x_2$)
- **Flows:** per-variable, rate bounds: $\dot{x}_i \in [a_i, b_i]$ (deterministic if $a_i = b_i$)
- **Intuition:** "boxy" geometry in both state and derivative spaces
- **Why RA here?** This structure is exactly what RealySt exploits for reachability

Outline

RA with Random Events (RAE)

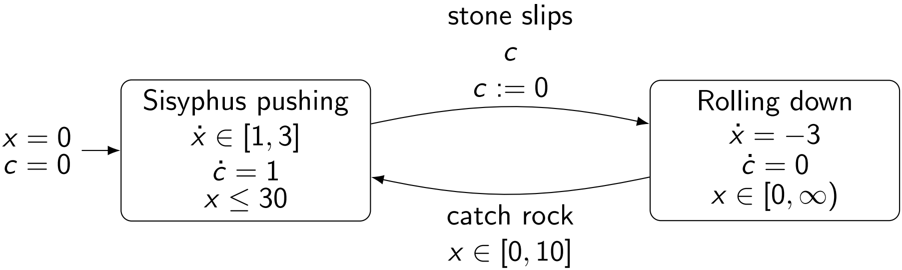
- Attach a random event to a jump: when the guard is enabled, sample a delay
- Semantics uses logical stopwatches μ_r: rate 1 while r-labelled jump is enabled, else 0
- Sample s_r initially and after each occurrence
- Fire when $\mu_r = s_r$ and the guard holds, then reset $\mu_r := 0$ and resample s_r
- Timers are handled *semantically* (no extra state variables needed)

Outline

RA with Random Clocks (RAC/RAR)

- Add a per-label random clock c (stopwatch) to store the *duration of enabledness*
- Sample an expiration $R_c \sim \mathrm{Distr}(c)$ initially and after each c-labelled firing
- In each location: $\dot{c} = 1$ iff a c-labelled jump is enabled, else $\dot{c} = 0$
- Fire when $c = R_c$ while enabled, upon firing set $c := 0$ and resample R_s from $\mathrm{Distr}(c)$
- Timing is handled *syntactically* (explicit clocks) $\Rightarrow$ enables geometric reachability and max-prob analysis

Example: RAC Sisyphus with an Explicit Random Clock



Random clock c measures *duration of enabledness*, slip delay e.g. $\sim \mathrm{Uniform}[0, 6]$

Outline

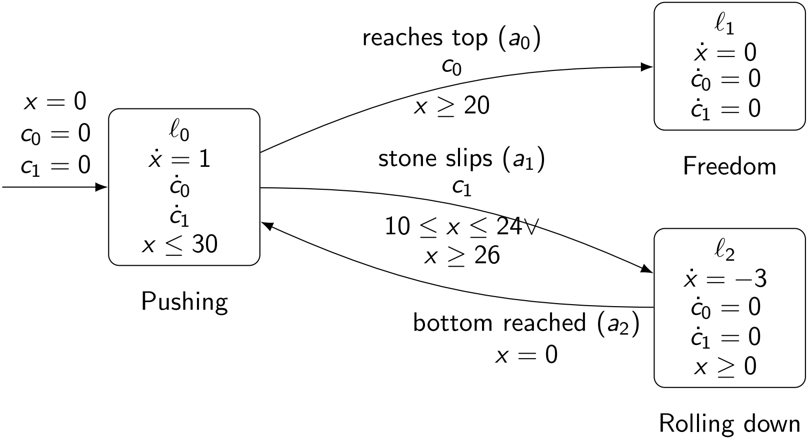
CAMELS: Scheduling & Realizability intuition

Axes:

- **Scheduling:** *Composed* (one global delay, then choose label) vs. *Decomposed* (per-label delays race)
- **Realizability:** *Lazy* (resample if invalid), *Eager Predictive* (only realizable delays), *Eager Non-predictive* (durations of enabledness)

	Lazy	Eager Predictive	Eager Non-predictive
Composed	✓	✓	✓
Decomposed	✓	✗	✓

Example: Decomposed Eager Non-predictive (two stopwatches)



In ℓ_0, both stopwatches (c_0 freedom, c_1 slip) accumulate enabledness, the first to hit its sampled duration wins. Includes resampling loops for each label when needed. We use $c_1 \sim \mathrm{Uniform}[0, 18]$ and $c_0 \sim \mathrm{Uniform}[0, 10]$.

Outline

Where do RA, RAE, and RAC sit within CAMELS?

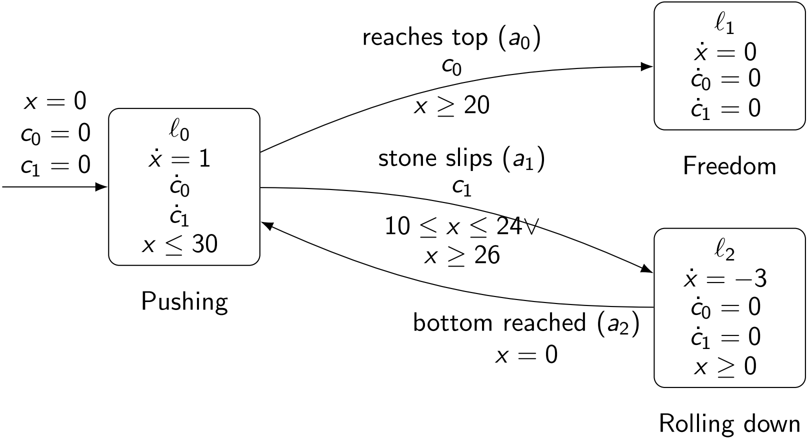
- **RA:** no stochastic delays $\Rightarrow$ *outside* CAMELS
- **RAE:** per-label stochastic delays interpreted as *enabledness durations* $\Rightarrow$ Decomposed, Eager Non-predictive
- **RAC/RAR:** makes those per-label timers explicit as continuous *random clocks*, syntactic subclass of RAE $\Rightarrow$ Decomposed, Eager Non-predictive

Outline

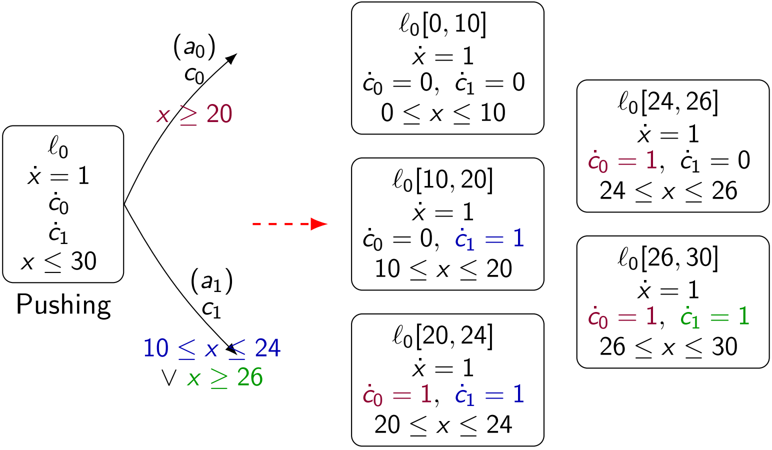
DENP $\Rightarrow$ RAC: General procedure

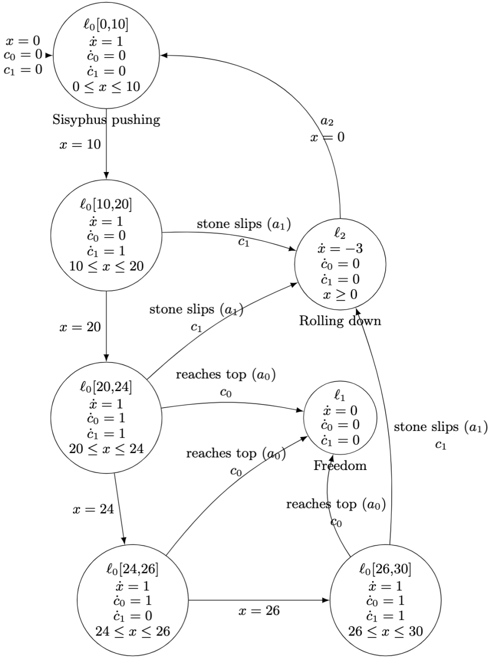
1. Take boundaries where enabledness changes.
2. Cut locations along these hyperplanes.
3. Set clock rates to $\dot{c}_a = 1$ iff a enabled in window, else $\dot{c}_a = 0$.
4. From locations where a's guard holds, add *unguarded* a-edge.
5. On firing, set $c_a := 0$ and resample its duration.
6. Add edges at each cut and keep source invariants.

Example: DENP (two stopwatches)



Example DENP to RAC conversion





$x = 0$
$c_0 = 0$
$c_1 = 0$
ℓ_0[0,10]
$\dot{x} = 1$
$\dot{c}_0 = 0$
$\dot{c}_1 = 0$
$0 \leq x \leq 10$
Sisyphus pushing
$x = 10$
a_2
$x = 0$
ℓ_0[10,20]
$\dot{x} = 1$
$\dot{c}_0 = 0$
$\dot{c}_1 = 1$
$10 \leq x \leq 20$
stone slips (a_1)
c_1
ℓ_2
$\dot{x} = -3$
$\dot{c}_0 = 0$
$\dot{c}_1 = 0$
$x \geq 0$
Rolling down
$x = 20$
stone slips (a_1)
c_1
ℓ_0[20,24]
$\dot{x} = 1$
$\dot{c}_0 = 1$
$\dot{c}_1 = 1$
$20 \leq x \leq 24$
reaches top (a_0)
c_0
ℓ_1
$\dot{x} = 0$
$\dot{c}_0 = 0$
$\dot{c}_1 = 0$
Freedom
stone slips (a_1)
c_1
reaches top (a_0)
c_0
$x = 24$
reaches top (a_0)
c_0
ℓ_0[24,26]
$\dot{x} = 1$
$\dot{c}_0 = 1$
$\dot{c}_1 = 0$
$24 \leq x \leq 26$
$x = 26$
ℓ_0[26,30]
$\dot{x} = 1$
$\dot{c}_0 = 1$
$\dot{c}_1 = 1$
$26 \leq x \leq 30$

Outline

UPPAAL

- **Networks:** timed automata with parallel composition and synchronization channels (`a!`/`a?`)
- **Locations:** with invariants, edges with guards, updates, and clock resets
- **Clocks:** progress uniformly, invariants restrict time elapse
- **Tooling:** Symbolic Simulator, Concrete Simulator (sampled runs), Verifier (reachability/safety)

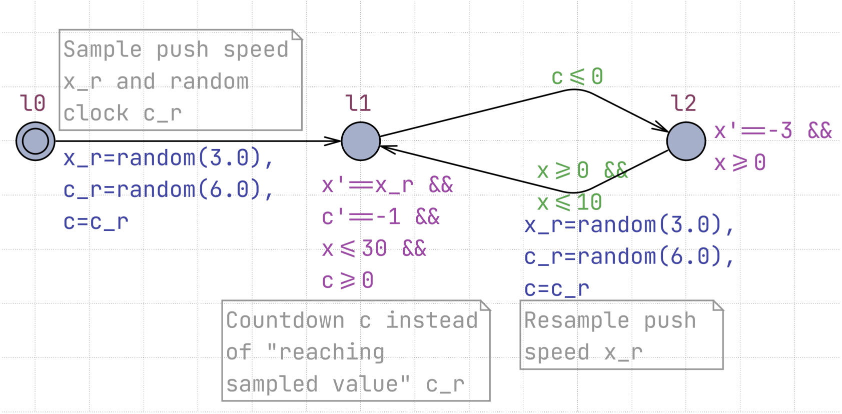
UPPAAL SMC: Stochastic Semantics & Queries

- **Race semantics:** concurrently enabled components draw delays (uniform/exponential), earliest enabled transition fires
- **Random sampling:** e.g. random(a,b) for variables/parameters
- **Continuous rates:** dynamics via `x'== expr` during simulation/model checking
- **Queries (statistical):** probability estimation, hypothesis testing, expectations with confidence

Examples:

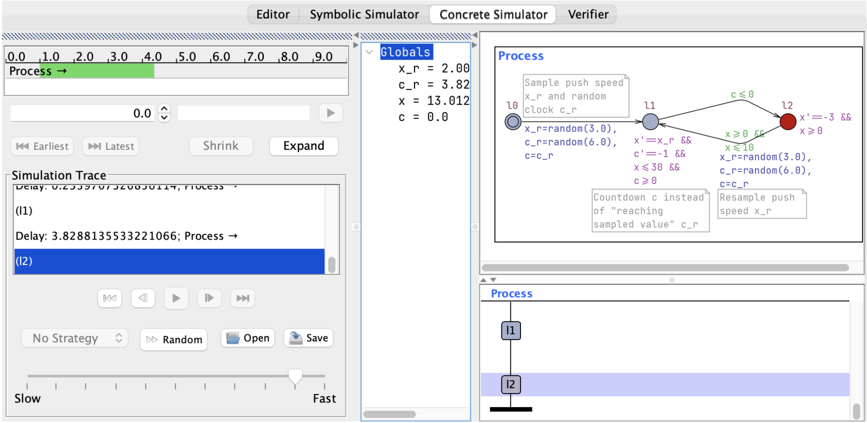
```
// probability that goal is reached within T
Pr[<=T](<> goal)

// hypothesis testing
Pr[<=T](<> goal) >= 0.9

// expected maximum of variable x up to T
E[<=T](max: x)
```

Example: Sisyphus as an UPPAAL model



Random uphill speed x_r sampled, early rollback with random clock c, roll at speed -3 back to $x \in [0, 10]$

UPPAAL Concrete Simulator



Concrete runs with realized delays, useful for validating race behavior before SMC queries.

UPPAAL Stratego: Strategy Optimization

- Synthesizes schedulers to optimize quantitative objectives (probability, time, cost)
- Simulation-driven policy iteration

Examples:

```
// maximize reachability probability by time T
strategy opt = maxPr[<=T](<> goal);

// minimize expected time to reach goal (bounded)
strategy fast = minE(time)[<=T]: goal;
```

UPPAAL: Core model

Component syntax $A = (\mathrm{Loc}, \mathrm{Clk}, \mathrm{Var}, \mathrm{Inv}, \mathrm{Flow}, E, \ell_0, v_0)$:

- **Locations**: $\mathrm{Loc} \in \{\mathit{normal}, \mathit{urgent}, \mathit{committed}\}$. Optional *exponential exit rate* $\lambda(\ell) \geq 0$ used only if the stay is unbounded.
  - Urgent/committed forbid time elapse, committed forces next discrete step to involve committed component.
- **Clocks**: evolve by rate equations $c' == e$ in locations.
- **Variables**: `int/bool/arrays/records/doubles` used in rates, guards, invariants, updates.
- **Invariants**: Inv bound time in locations.
- **Edges** E: Guards, optional weights, updates (incl. random draws), resets.

UPPAAL SMC: Execution semantics

Delays & sampling

- **Bounded stay**: sample $U[0, b(s)]$ and attempt to leave at expiry.
- **Unbounded stay** with $\lambda(\ell) > 0$: sample $\mathrm{Exp}(\lambda(\ell))$.
- Random assignments via SMC store values for later guards/flows.

Race in a network

- Nonblocked component sample delay, *minimum* wins the race.

Discrete choice at jump time

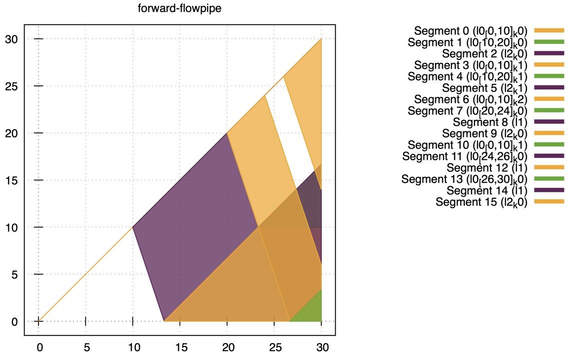
- Choose edge *proportionally to weights*, if none given choose uniformly.
- Set weight 0 to block an option.

Outline

RealySt: RAC/RAR with Explicit Random Clocks

- **Models:** rectangular/singular automata with random clocks
- **Pipeline:**
  1. Forward reachability via exact geometry
  2. Backward refinement to keep only states that can still reach the goal
  3. Project onto random-clock dimensions and integrate the probability densities (Monte Carlo / VEGAS) to obtain $P_{\max}(\Diamond \text{goal})$
- **Output:** maximum bounded-time reachability under *prophetic* schedulers.

RealySt: Flowpipe & Probability (bounded-time)



Cmd options (what they mean)

- `-t 30` - time bound
- `-d 100` - jump-depth
- `-b SISYPHUS` - benchmark
- `-m A` - model variant
- `--plotDimensions 0 1` - select variables as plot axes
- `-l trace` - logging level

```
CLI: ./realyst -t 30 -d 100 -b SISYPHUS -m A --plotDimensions 0 1 -l trace
```

Outline

ARCH-COMP'22 minimal examples: what we model

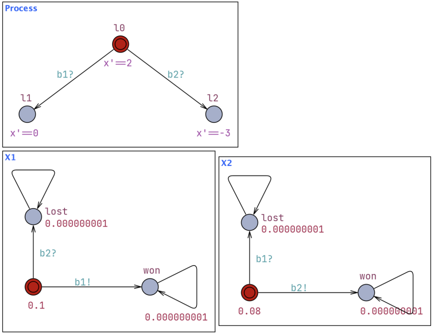
Cases

- **A:** Two independent event components race (`b1!`, `b2!`); system goes to hold or descend.
- **B:** Like A, but after `b2?` the system waits exactly 2 time units, then urgent non-determinist split.
- **C:** Like B, but flow in the waiting phase has Gaussian noise (we approximate by fixed ticks).
- **D:** Like A, but system has invariant $x \leq 6$ (potential timelock avoided by boundary jump).

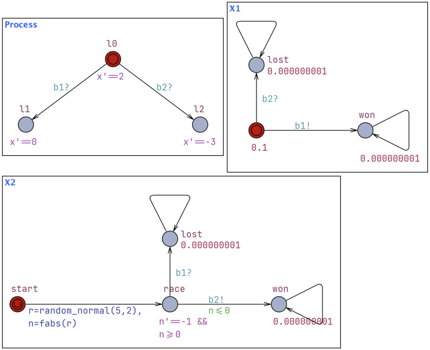
Each case has 2 subcases: *Exp/Exp* (both exponential) and *Exp/Normal* (X_1 exponential, X_2 single draw from folded normal $|N(5,2)|$).

ARCH'22 Case A: automata

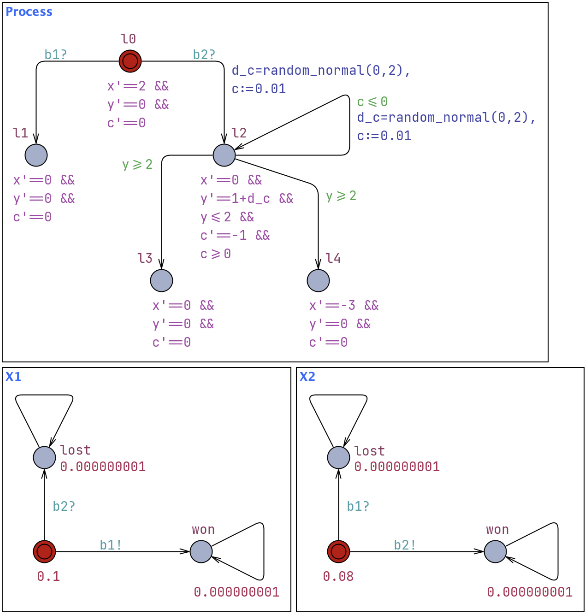
Exp/Exp



Exp/Normal



ARCH'22 Case C: automaton (Exp/Exp baseline)



ARCH'22 results: UPPAAL SMC vs. RealySt

Case	Variant / Property	UPPAAL (95% CI)	RealySt	Notes
A	Exp/Exp, ϕ (T=10)	0.288719 ± 0.000993	0.288236	match
A	Exp/Normal, ϕ (T=10)	0.448690 ± 0.000975	0.448211	match
B	Exp/Exp, ϕ (T=10)	0.125192 ± 0.000917	0.250016	max vs. prob. split
B	Exp/Exp, ϕ' (T=12)	0.144624 ± 0.000975	0.288236	max vs. prob. split
B	Exp/Normal, ϕ (T=10)	0.154144 ± 0.000846	0.308558	max vs. prob. split
B	Exp/Normal, ϕ' (T=12)	0.224051 ± 0.000977	0.448211	max vs. prob. split
C	*Exp/Exp*, ϕ (T=10)	0.125755 ± 0.000649	*N/A*	UPPAAL only
C	*Exp/Exp*, ϕ' (T=12)	0.143548 ± 0.000687	*N/A*	UPPAAL only
C	*Exp/Normal*, ϕ (T=10)	0.154234 ± 0.000708	*N/A*	UPPAAL only
C	*Exp/Normal*, ϕ' (T=12)	0.224111 ± 0.000817	*N/A*	UPPAAL only
D	Exp/Exp, ϕ (T=10)	0.185505 ± 0.000762	0.185280	match
D	Exp/Normal, ϕ (T=10)	0.130076 ± 0.000659	0.130280	match

Note: UPPAAL approximates Case C with fixed ticks Δt=0.01. For Case B UPPAAL matches other SMC tools.

Outline

CAMELS in REALYST

- **DENP** (RAC) supported natively.

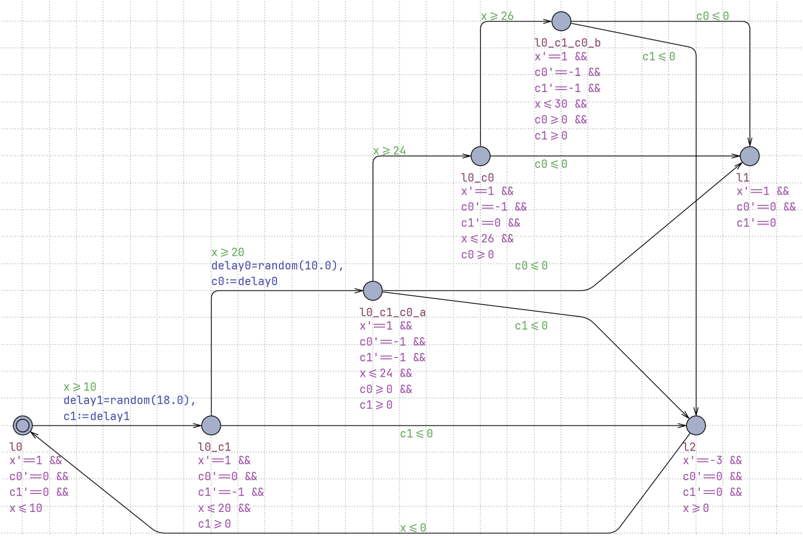
Not supported in general:

- **Composed** need one global delay + probabilistic label choice.
  - **CENP**: timing can match via single clock for all random jumps.
  - **CEP**: can match with precomputation of the set of possible samples for the global random clock.
- Current RealySt impl. random clocks fire only once.
- **DL not in RealySt:** needs absolute delays that keep ticking and resample (core semantics mismatch).

Workaround & stance

- Loop unrolling with fresh clocks per repetition is possible but in our runs probabilities did not match.
- Therefore only cross-tool comparison on **DENP**.

DENP in UPPAAL



DENP results (reachability of l1)

Tool	Bound	Estimate	Uncertainty
UPPAAL SMC	$t \leq 30$	0.222000	$\pm 8.14545 \times 10^{-4}$
	$t \leq 100$	0.617393	$\pm 9.52590 \times 10^{-4}$
RealySt	$t \leq 30$	0.2221555	$\pm 5.26794 \times 10^{-5}$

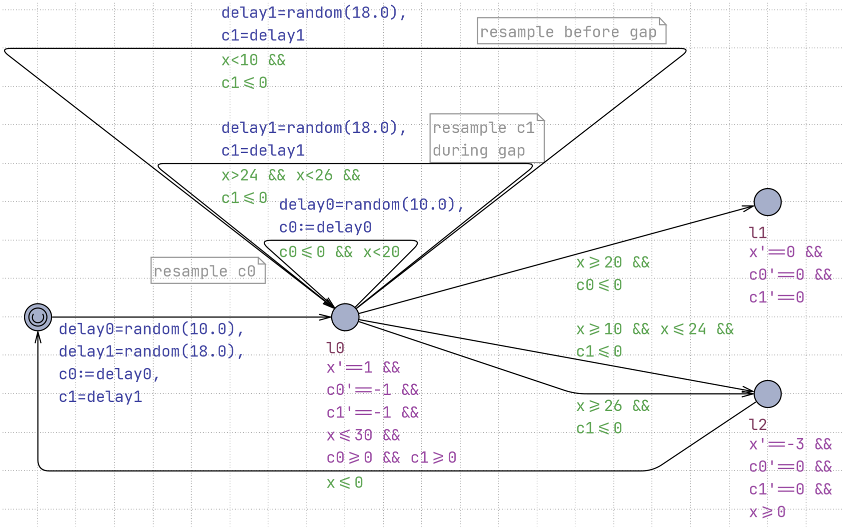
Analytic check (DENP): Pr(top before slip) $= \frac{2}{9} \approx 0.222$ (matches both).

UPPAAL: implementing DL

Idea

- Per label a_i: one countdown c_i that *never pauses*.
- Make expiry urgent: invariant $c_i \geq 0$ + guard all expiry edges with $c_i \leq 0$.
- At expiry: deterministic branch *fire vs. resample*.
  - Compile $\neg g_i$ into "gaps" and add one resampling loop per gap.

UPPAAL DL: automaton

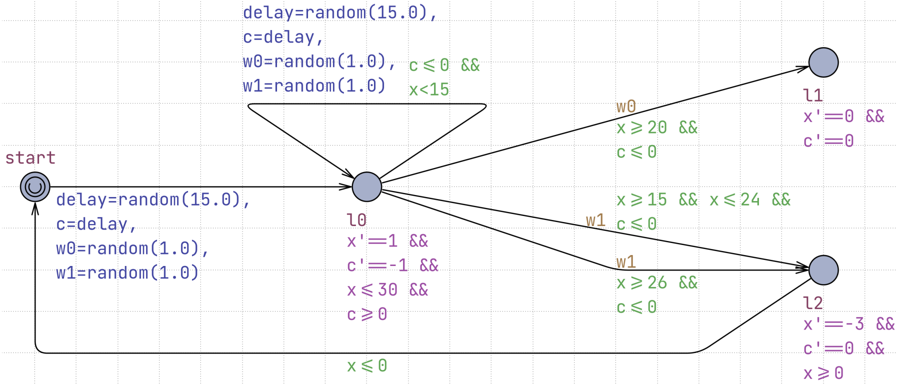


UPPAAL: implementing CL

Idea

- One global countdown c, never pauses.
- Weights w_i resolve overlaps among enabled random jumps.
- If expiry lands in a gap: resampling loop redraws c and w_i.

UPPAAL CL: automaton

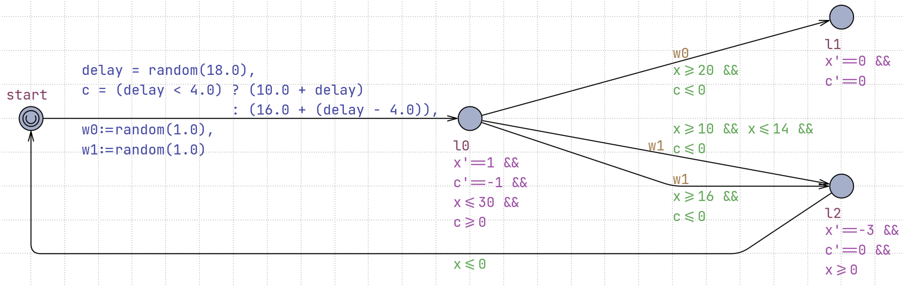


UPPAAL: implementing CEP

Idea

- One global countdown c, samples only from times where some label will be enabled.
  - Have to manually precompute the "enabled-samples".
- Overlaps still resolved by weights w_i.

UPPAAL CEP: automaton

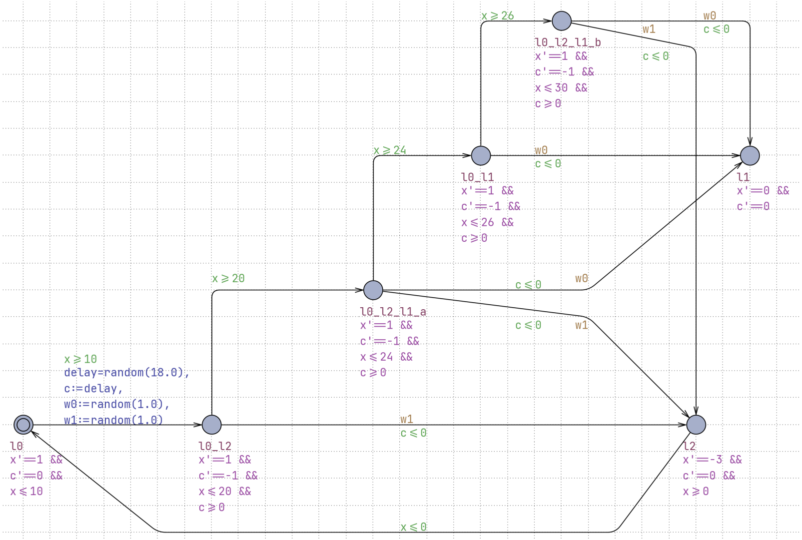


UPPAAL: implementing CENP

Idea

- One stopwatch c on the union $g = \bigvee_i g_i$, c keeps running through gaps of individual labels.
- Sample when g becomes true.
- Overlaps resolved by weights w_i.
- Same windows as DENP, c keeps running across single-label gaps.

UPPAAL CENP: automaton



Outline

Conclusion

- **Decision model difference:**
  - UPPAAL (SMC) resolves choices *probabilistically*
  - RealySt resolves overlaps *maximizing* (prophetic).
  - This is the main source of divergence in composed variants.
- **Expressivity (CAMELS):**
  - UPPAAL can model DENP, DL, CL, (CEP), CENP.
  - RealySt natively matches DENP/RAC.
- When semantics align, both tools agree.
- **Limits:**
  - RealySt currently fires a stochastic clock only once.
  - Composed variants only faithful under disjoint enablement (no tie at expiry).