Statistical model checking with `slimsim`

Various random thoughts

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Contents

Statistical model checking for COMPASS/HASDEL

Ever growing demand for probabilistic/dependability analysis.

HASDEL: Analysis of hybrid and probabilistic systems, used for space launchers and vehicles.

COMPASS:

- Analysis of timed/hybrid systems;
- Analysis of probabilistic systems;
- **No** analysis of timed and probabilistic systems.

Statistical model checking for COMPASS/HASDEL

Project goal: extend our capabilities to analyze timed and probabilistic systems, avoiding hard- and software failures and improve overall confidence.

Otherwise:

Insert any picture of rather expensive equipment unintentionally exploding here.

Statistical model checking for COMPASS/HASDEL

Problem: Currently no tools (or algorithms) that support probabilistic analysis of the systems that can be described in the toolset.

Our approach: Use (Monte Carlo) simulation to approximate the system behavior

Contents

SLIM

SLIM is a modeling language based on AADL

Example nominal:

```
device gpsDevice
    features
        measurement : out data port bool default false;
end gpsDevice;

device implementation gpsDevice.i
    flows
        measurement := true  in modes (active);
    modes
        acquisition : activation mode urgent in 20 sec;
        active      : mode;
    transitions
        acquisition -[ within 10 sec to 20 sec ]-> active;
end gpsDevice.i;
```

SLIM

SLIM is a modeling language based on AADL

Example error:

```
error model gpsError
    features
        nok : out error propagation;
end gpsError;

error model implementation gpsError.i
    events
        transient_fault   : error event occurrence poisson 0.001 per hour;
        hot_fault         : error event occurrence poisson 0.001 per day;
    states
        ok                : initial state;
        transient_failure_prop : error state;
        transient_failure : error state urgent in 400 msec;
        hot_failure_prop  : error state;
        hot_failure       : error state;
    transitions
        ok                -[ transient_fault                ]-> transient_failure_prop;
        transient_failure -[ nok within 200 msec to 400 msec ]-> ok;
        ok                -[ hot_fault                      ]-> hot_failure_prop;
        hot_failure       -[ @activation                    ]-> ok;
        transient_failure -[ @activation                    ]-> ok;
end gpsError.i;
```

SLIM

A fully formalized language derived from AADL, with support for:

- Timed behavior (clocks, guard and invariants)
- Data flows
- Synchronizing events
- Nonblocking events
- Probabilistic error events
- Component reconfiguration (@activation)

Monte Carlo simulation

Contents

Monte Carlo simulation

A brief intro

Generate samples for a process generating random events. When enough samples are generated, with a certain probability the likelyhood of the events can be determined.

In our case: Event = Property true/false. Generating event = Generating path

Randomized simulation

Contents

Randomized simulation

Non-determinism vs. simulation

Deterministic model: Given a state and input, the target state is *uniquely determined*.

Non-deterministic model: Target is a (possibly uncountably infinite) *set* of states.

Simulation approaches this by means of random number generation (RNG).

The point: The abstract model specifies a set of possible next states, but simulation always yields a single -- in practice deterministic -- result.

Randomized simulation

Random numbers

Approximate non-determinism by random numbers. A choice can be made on what probability distribution to use (or go fully deterministic).

In fact, our RNG implements a (probabilistic) scheduler determining the choice in a probabilistic manner (as opposed to the more conventional minimum and maximum).

Randomized simulation

A note on RNG implementations

It is possible to generate true random numbers (TRNG), but is relatively expensive/slow (2048 random bits in `/dev/random`). Fortunately, pseudo random numbers (PRNG) generally suffice for Monte Carlo simulation. But, please don't use `rand()`, at least for simulation.

Time to generate 2^{28} integers:

C rand()	7.28 s
C++ Mersenne twister	7.50 s
SFMT	6.10 s

Contents

Strategies

Contents

Path generation

Generate steps until property can be determined true or false. For CSL, simple to check for basic state formulae:

- $M \models a \Leftrightarrow a$ holds true in the current state
- $M \models \psi \wedge \phi \Leftrightarrow M \models \psi \wedge M \models \phi$
- etc...

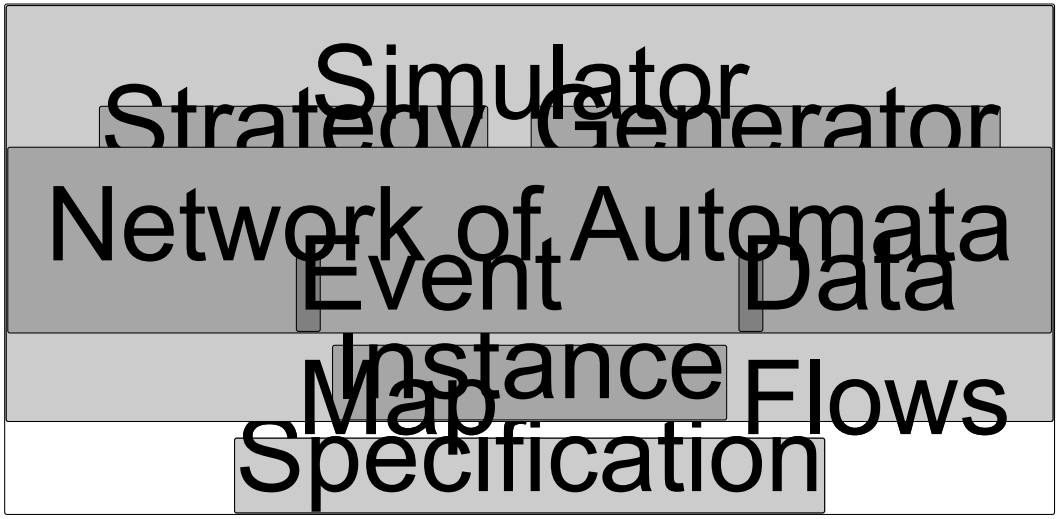
For path based formulae (i.e. $\psi \mathbf{U}^{[l,u]} \phi$), a tri-state approach is used (true, false, ``not yet known'').

For the operator $Pr_{\bowtie c}(\psi)$, (recursive) simulation is used. Note: `slimsim` currently does not support nesting.

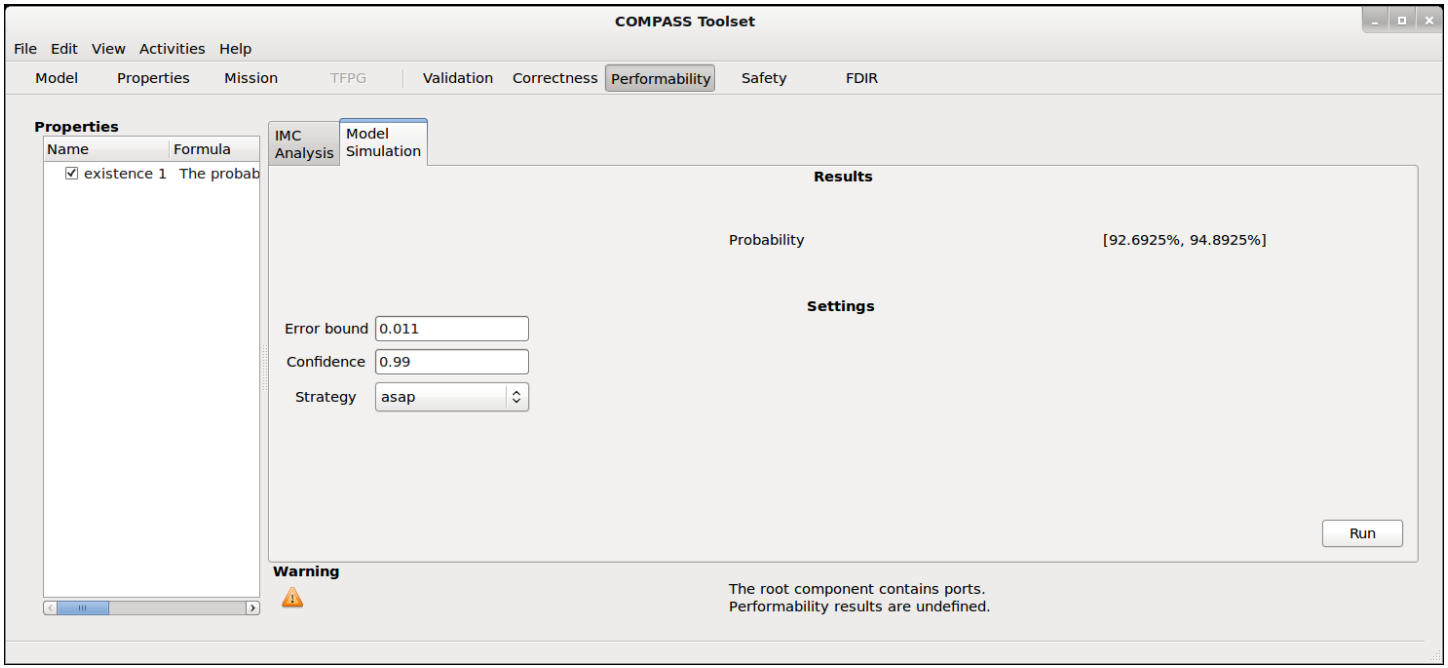
Special cases: Zeno behavior and deadlocks

Contents

Tool architecture



Toolset integration



Contents

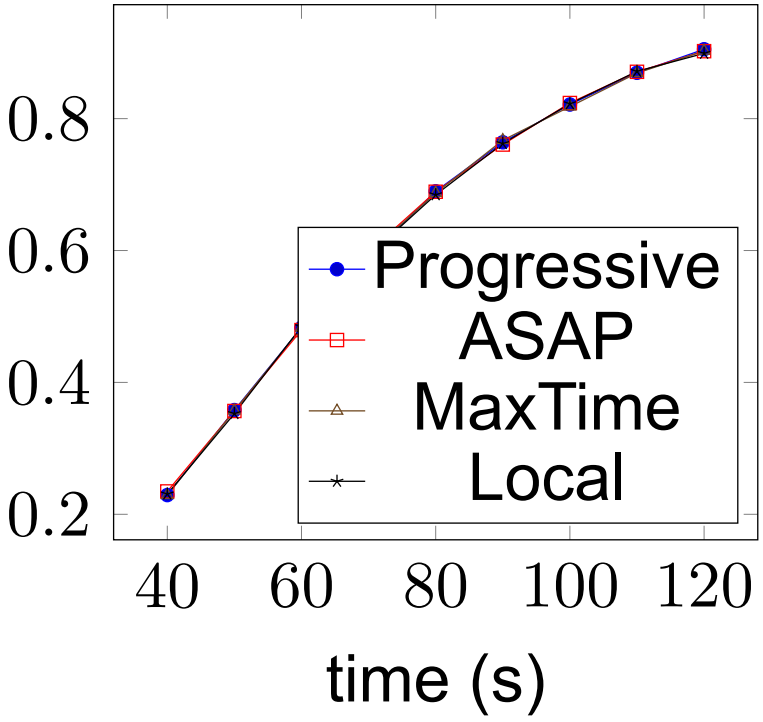
Avionics case study

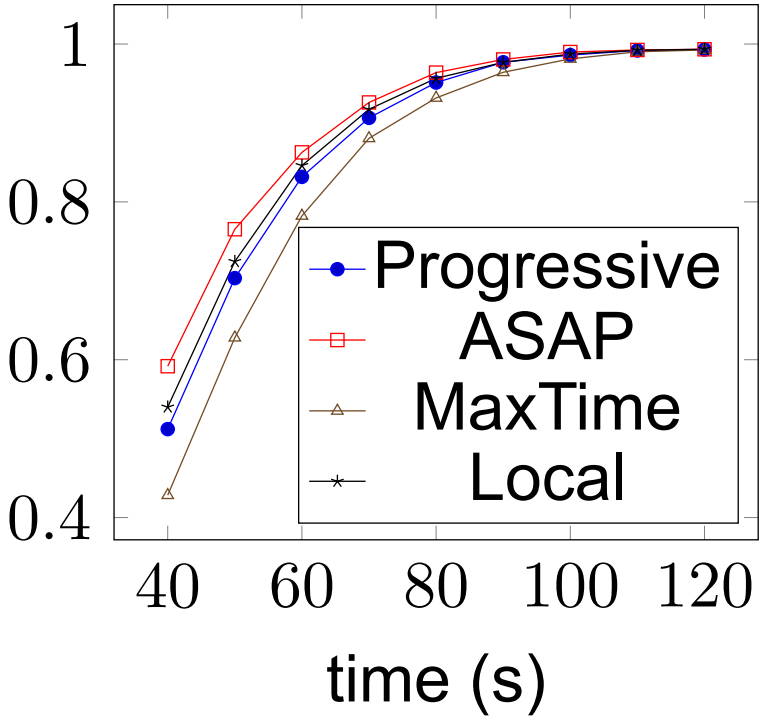
Experimental results

Contents

Experimental results

Probabilities of system failure containing DPUs without (left) and with (right) repair





Conclusion

Contents

Frame title

Thank you for your attention

Any questions?