

Multi-Core Reachability for Timed Automata

Andreas Dalsgaard² Alfons Laarman¹ Kim G. Larsen²
Mads Chr. Olesen² Jaco van de Pol¹

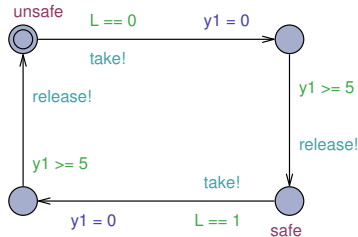
¹Formal Methods and Tools, University of Twente {a.w.laarman,vdpol}@cs.utwente.nl

²Department of Computer Science, Aalborg University {andrase,kgl,mchro}@cs.aau.dk

FORMATS'2012, London, 19th September 2012

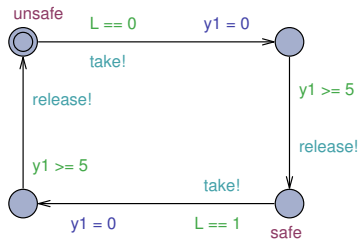
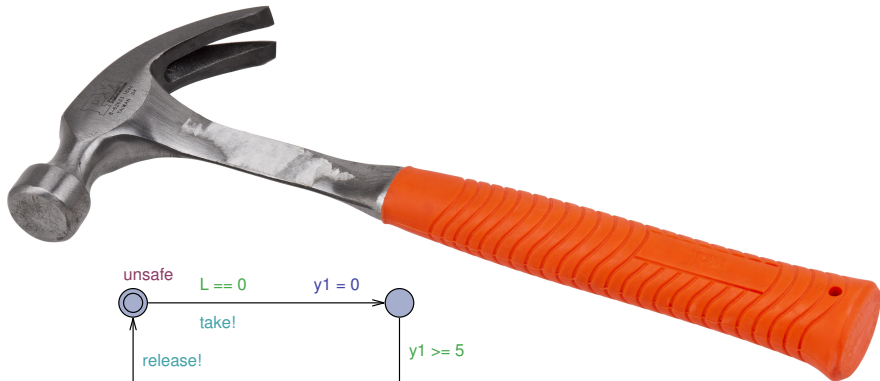
Hammers

1 core

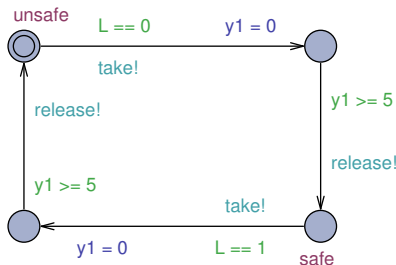


Hammers

Many cores



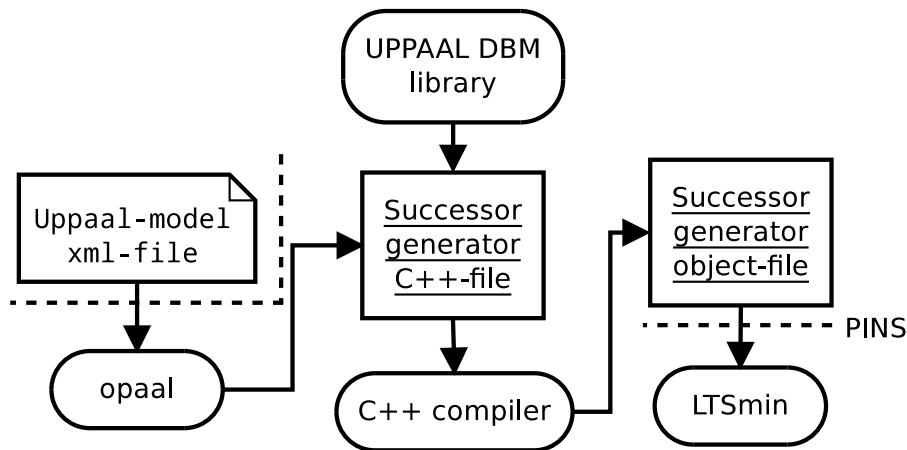
Timed Automata



- Clocks, etc.
- Model checking symbolic
- Partial order on behaviour (clock zones, represented as Difference Bounded Matrix (DBM))

- Viewed as Well-Structured Transition System
- State (s, σ) has discrete part, and symbolic part with comparison operator $\sqsubseteq$

Toolchain



LTSmin

PINS Interface with subsumption

- $\text{INITIAL-STATE}() = (s_0, \sigma_0)$
- $\text{NEXT-STATE}((s, \sigma)) = \{(s_1, \sigma_1), \dots, (s_n, \sigma_n)\}$ returning all successors of (s, σ) , $(s, \sigma) \rightarrow (s_i, \sigma_i)$
- $\text{COVERS}(\sigma', \sigma) = \sigma \sqsubseteq \sigma'$ returning whether the symbolic part σ' subsumes σ

Implementation-wise, a state is a vector of 32-bit integers, and a pointer to a DBM:

$$\langle i_1, \dots, i_m, \text{DBM} \rangle$$

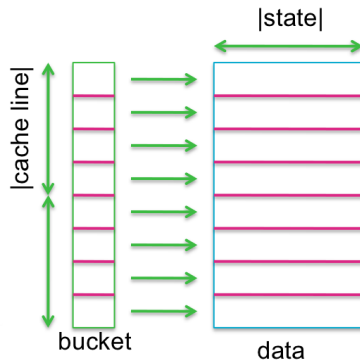
Reachability with subsumption

Algorithm 1 Reachability with subsumption

```
1  proc reachability( $s_g$ )
2     $W := \{ \text{INITIAL-STATE}() \}; P := \emptyset$ 
3    while  $W \neq \emptyset$ 
4       $W := W \setminus (s, \sigma)$  for some  $(s, \sigma) \in W$ 
5       $P := P \cup \{(s, \sigma)\}$ 
6      for  $(t, \tau) \in \text{NEXT-STATE}((s, \sigma))$  do
7        if  $t = s_g$  then report & exit
8        if  $\nexists \rho: (t, \rho) \in W \cup P \wedge \text{COVERS}(\rho, \tau)$ 
9           $W := W \setminus \{(t, \rho) \mid \text{COVERS}(\tau, \rho)\} \cup (t, \tau)$ 
```

Multicore LTSmin Passed-Waiting List

- Hash table optimized for multicore
- Open addressing
- Separate data
- Lockless (CAS + write bit)
- Pre-allocation of entire structure (no allocation there-after)



DBM Storage Data Structure

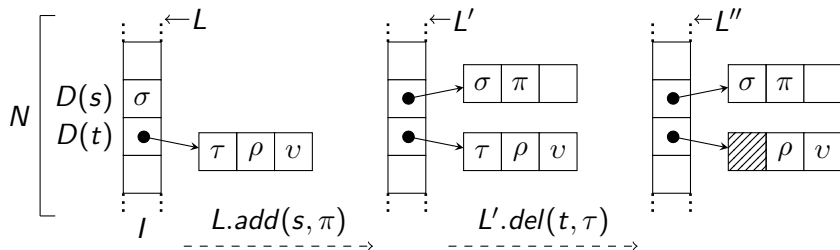


Figure: Data structure for L , and operations

- Updates can be either blocking or non-blocking
 - Blocking maintains invariant: waiting states in L do not subsume each other
- Another hash table used for duplicate detection of DBMs

Parallel Reachability

- More searchers
- Share Passed hash table
- Local Waiting list
- Work-stealing

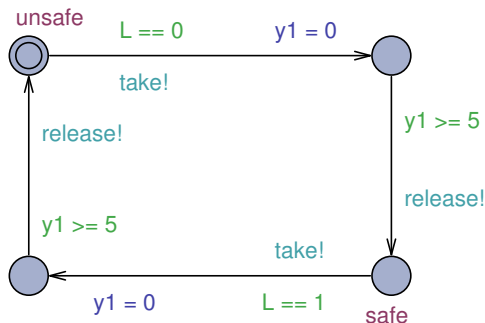
global $L : \mathcal{S} \rightarrow (\Sigma \times \{\text{waiting}, \text{passed}\})^*$

```
proc search( $((s_0, \sigma_0), s_g, p)$ )  
   $Q_p :=$  if  $p = 1$  then  $\{(s_0, \sigma_0)\}$  else  $\emptyset$   
  while  $Q_p \neq \emptyset \vee \text{balance}(Q_p)$   
     $Q_p := Q_p \setminus (s, \sigma)$  for some  $(s, \sigma) \in Q_p$   
    if  $\neg \text{grab}(s, \sigma)$  then continue  
    for  $(t, \tau) \in \text{NEXT-STATE}((s, \sigma))$  do  
      if  $t = s_g$  then report & exit  
      if  $\neg \text{update}(t, \tau)$   
         $Q_p := Q_p \cup (t, \tau)$ 
```

opaal

Successor Generator Generator

- Generate C++ code specific for the model
- Max clock constant analysis (location-based k -extrapolation)
- Python

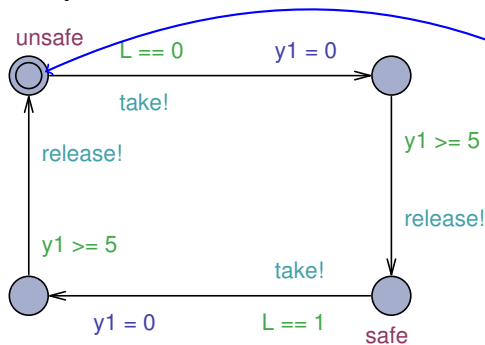


State vector:

- Process1
- int L
- DBM: y1

Successor Generator Generator

- Generate C++ code specific for the model
- Max clock constant analysis (location-based k -extrapolation)
- Python

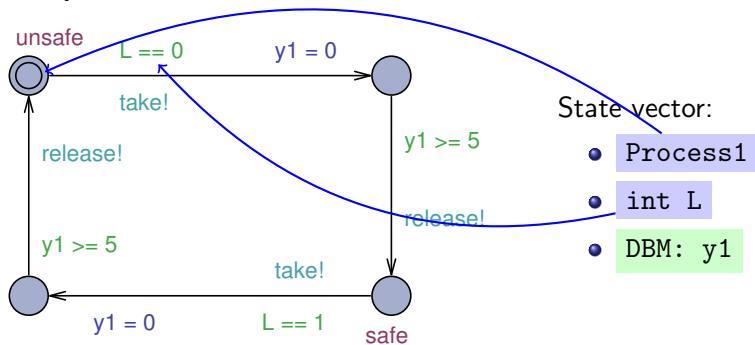


State vector:

- Process1
- int L
- DBM: y1

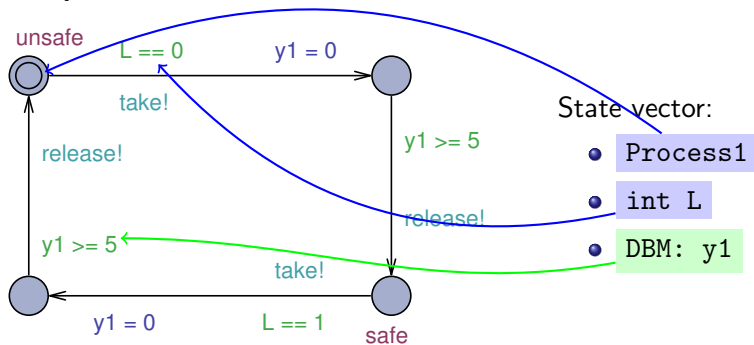
Successor Generator Generator

- Generate C++ code specific for the model
- Max clock constant analysis (location-based k -extrapolation)
- Python



Successor Generator Generator

- Generate C++ code specific for the model
- Max clock constant analysis (location-based k -extrapolation)
- Python



Structure of generated code

```
1  switch (Process1 location) {  
2      case 1:  
3          if (guard1) //normal transition  
4              callback with successor  
5          if (guard2) { //synchronisation on chan!  
6              switch (Process2 location) {  
7                  case X:  
8                      if (guardX) callback with successor  
9              }  
10             ...  
11         }  
12         ...  
13     }  
14 //Transitions for Process2  
15 ...
```

Supported Language Features

- Templates
- constants
- bounded integer variables
- arrays
- selects
- guards
- updates
- invariants on both variables and clocks
- committed and urgent locations
- binary synchronisation
- broadcast channels
- urgent synchronisation
- much of the C-like language that UPPAAL uses to express guards and variable updates

Experiments

Expectations

- Search order matters!
- Finding larger zones faster allow faster termination
- Multi-core means search order depends on interleaving
- Hope for linear speedup

Experiments

Table: Runtimes, strict BFS

	UPPAAL T	opaal+LTSMIN (cores)					
		T_1	T_2	T_8	T_{16}	T_{32}	T_{48}
train-gate-N10	837.4	573.3	297.8	76.7	39.4	21.1	14.4
vikings17	207.8	331.5	172.5	44.2	22.7	11.9	8.6
train-gate-N9	76.8	52.4	28.5	7.7	4.1	2.4	2.0
vikings15	38.0	67.0	34.8	9.7	5.1	3.0	2.3
train-crossing	48.3	24.5	37.2	5.8	2.7	2.0	2.1
fischer6	0.1	219.2	129.2	46.4	36.1	32.9	31.8

Scalability Strict Search Order

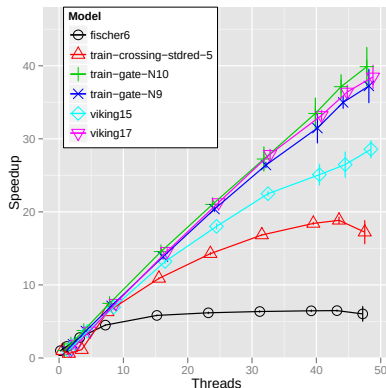


Figure: Speedup strict BFS

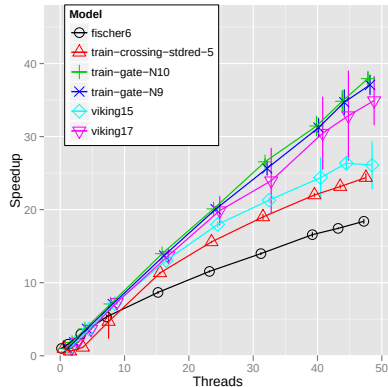


Figure: Speedup non-blocking strict BFS

Scalability Non-Strict Search Order

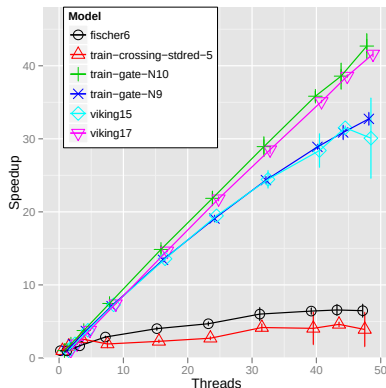


Figure: Speedup Pseudo BFS

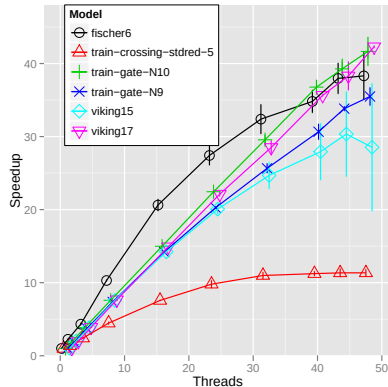


Figure: Speedup Randomized Pseudo DFS

Conclusion

- A scalable hammer
- Promising experiments
- Lots of efficiency improvements (LU-clock extrapolation, symmetry, partial order reduction, memory consumption, ...)
- Multi-core liveness?

Websites

<http://opaal-modelchecker.com/>

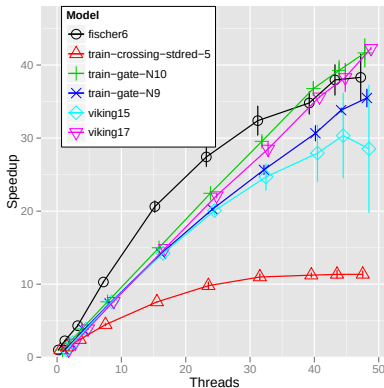
<http://fmt.cs.utwente.nl/tools/ltsmin/>

Conclusion

Websites

<http://opaal-modelchecker.com/>

<http://fmt.cs.utwente.nl/tools/ltsmin/>



Questions?

1 Introduction

- Hammers
- Timed Automata
- Toolchain

2 LTSmin

- PINS Interface with subsumption
- Reachability with subsumption
- Multicore LTSmin Passed-Waiting List
- DBM Storage Data Structure
- Parallel Reachability

3 opaal

- Successor Generator Generator
- Structure of generated code
- Supported Language Features

4 Experiments

- Expectations
- Experiments
- Scalability Strict Search Order
- Scalability Non-Strict Search Order