UNIVERSITY OF TWENTE.

Symbolic Model Checking of Timed Automata using LTSmin

Sybe van Hijum

Overview

Introduction

Our Work

Results

Future Work

Transition System

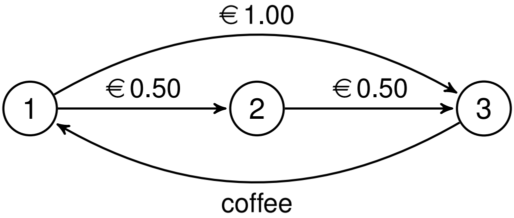
Definition (Labeled Transition System)

A labeled transition system is a 3-tuple $A = \langle S, Act, s_o \rangle$ *where*

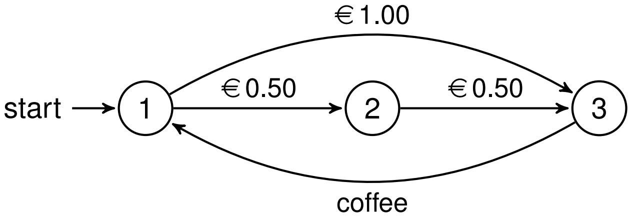
- S *is a finite set of states*
- Act *is a finite set of labelled actions*
- $s_o \in S$ *is a finite set of actions*

Transition System

Transition System



Transition System



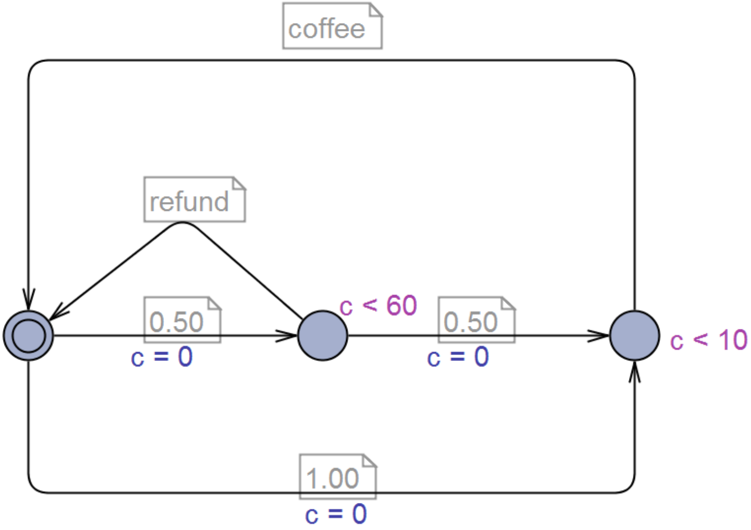
Timed Automata

Definition (Timed Automata)

An extended timed automaton is a 6-tuple $A = \langle L, C, Act, l_0, \rightarrow, I_c \rangle$ *where*

- L *is a finite set of locations, typically denoted by* l
- C *is a finite set of clocks, typically denoted by* c
- Act *is a finite set of actions*
- $l_0 \in L$ *is the initial location*
- $\rightarrow \subseteq L \times G(C) \times Act \times 2^C \times L$ *is the (non-deterministic) transition relation.*
- $I_C : L \rightarrow G(C)$ *is a function mapping locations to downwards closed clock invariants.*

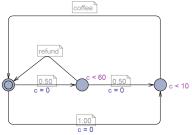
Timed Automata



Time Zones

Time not represented as a variable, but as a zone. Most used structure to represent zones: Different Bound Matrix (DBM)

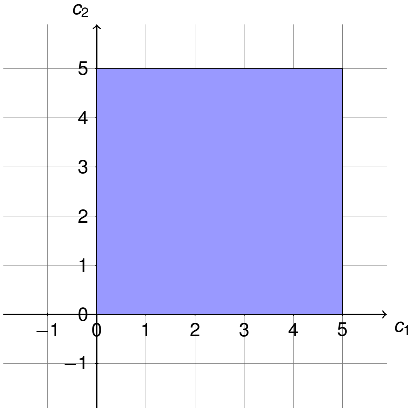
- Only convex zones
- Memory inefficient



$$0 \leq c < 60$$
$$\Downarrow$$
$$c - 0 < 60$$
$$0 - c \leq 0$$

$$\begin{array}{cc} & \begin{array}{cc} \mathbf{0} & c \end{array} \\ \begin{array}{c} \mathbf{0} \\ c \end{array} & \begin{pmatrix} (0, \leq) & (0, \leq) \\ (60, <) & (0, \leq) \end{pmatrix} \end{array}$$

$$\begin{array}{c|ccc} & \mathbf{0} & c_1 & c_2 \\ \hline \mathbf{0} & (0, \leq) & (0, \leq) & (0, \leq) \\ c_1 & (5, <) & (0, \leq) & (5, <) \\ c_2 & (5, <) & (5, <) & (0, \leq) \end{array}$$



Boolean Decision Diagram

- Expresses boolean expressions
- States can be seen as boolean expressions
- Memory efficient

Boolean Decision Diagram

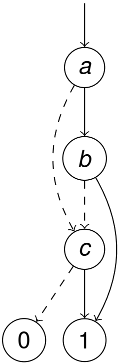
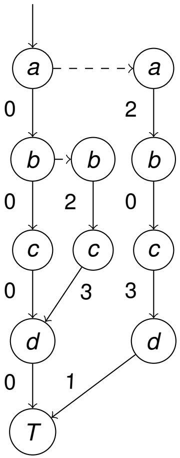


Figure: A BDD representing $(a \wedge b) \vee c$

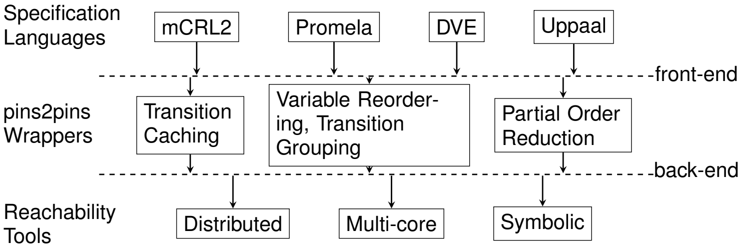
List Decision Diagram



LTSmin

- Language independent model checker
- Multiple algorithmic back ends
- Internal optimization wrappers

LTSmin


Specification Languages
mCRL2
Promela
DVE
Uppaal
front-end
pins2pins Wrappers
Transition Caching
Variable Reordering, Transition Grouping
Partial Order Reduction
back-end
Reachability Tools
Distributed
Multi-core
Symbolic

LTSmin

- States as integer vectors
- Partitioned next-state function
- Optimizations based on matrices
  - Read(r)
  - Must-write(w)
  - May-write(W)
  - Copy(-)

Matrices

$$
\begin{array}{lll}
1: & x = 1 \vee a[1] = 0 & \rightarrow a[1] := 1, x := 0, y := 5 \\
2: & a[0] = 1 \vee y = 5 & \rightarrow a[x] := 0, x := 1
\end{array}
$$

$$
\begin{array}{cc}
 & \begin{array}{cccc} x & y & a[0] & a[1] \end{array} \\
\begin{array}{c} 1 \\ 2 \end{array} & \begin{bmatrix} + & w & - & + \\ + & r & + & W \end{bmatrix}
\end{array}
$$

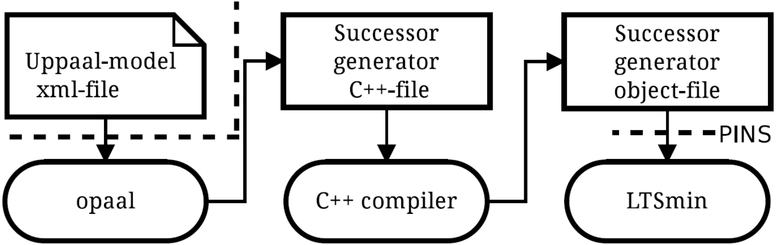
Problem: Model checkers are designed for discrete variables (integers), clocks have real values.

- Can we use the LTSmin symbolic model checker for timed automata?
- Can we optimize the symbolic back end for clocks?

Overview

Current LTSmin Uppaal setup

States as a vector of discrete locations and a pointer to a DBM.
Implemented in explicit-state multi-core tool.



First approach: values from DBM directly into an LDD

$$\begin{array}{c c} & \begin{array}{ccc} \mathbf{0} & c_1 & c_2 \end{array} \\ \begin{array}{c} \mathbf{0} \\ c_1 \\ c_2 \end{array} & \begin{pmatrix} (0, \leq) & (0, \leq) & (0, \leq) \\ (5, <) & (0, \leq) & (5, <) \\ (5, <) & (5, <) & (0, \leq) \end{pmatrix} \end{array}$$

Old situation: $\{l_0, ..., l_n, ptr\}$
New situation:
$\{l_0, ..., l_n, (0, \leq), (0, \leq), (5, <), (5, <), (5, <), (5, <)\}$

LDD solution

- Correct, working solution
- Variable reordering possible
- All variables seen as discrete values
- No optimizations based on time

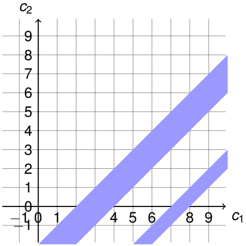
Difference Decision Diagram

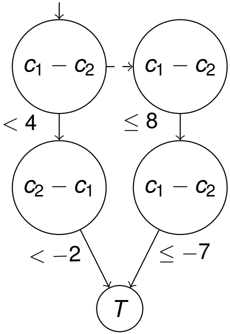
Definition (Difference Decision Diagram)

A difference decision diagram (DDD) is a directed acyclic graph (V, E). The vertex set V contains two terminals 0 and 1 with out-degree zero, and a set of non-terminal vertices with out-degree two and the following attributes.

Attribute	*Type*	*Description*
pos(v), neg(v)	***Var***	*Positive variable x_i, and negative variable x_j.*
op(v)	$\{<, \leq\}$	*Operator $<$ or $\leq$.*
const(v)	$\mathbb{D}$	*Constant c.*
high(v), low(v)	V	*High-branch h, and low-branch l.*

The set E contains the edges $(v, low(v))$ and $(v, high(v))$, where $v \in V$ is a non-terminal vertex.


c_2
9
8
7
6
5
4
3
2
1
0
−1
−1 0 1 4 5 6 8 9
c_1


$c_1 - c_2$
$c_1 - c_2$
< 4
≤ 8
$c_2 - c_1$
$c_1 - c_2$
< -2
≤ -7
T

Definition (Ordered DDD)

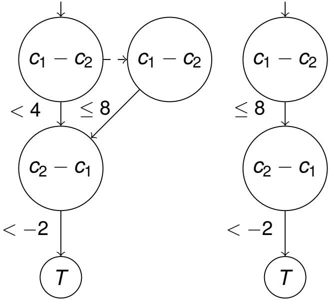
An ordered DDD (ODDD) is a DDD where each non-terminal vertex v satisfies:

1. $neg(v) \prec pos(v)$,
2. $var(v) \prec var(high(v))$,
3. $var(v) \prec var(low(v))$ *or*
   $var(v) = var(low(v))$ *and* $bound(v) \prec bound(low(v))$.

Definition (Locally Reduced DDD)

A locally reduced DDD ($R_L DDD$) is an ODDD satisfying, for all non-terminals u and v:

1. $\mathbb{D} = \mathbb{Z}$ *implies* $\forall v.op(v) =' \leq'$,
2. $(cstr(u), high(u), low(u)) = (cstr(v), high(v), low(v))$ *implies* $u = v$,
3. $low(v) \neq high(v)$,
4. $var(v) = var(low(v))$ *implies* $high(v) \neq high(low(v))$.



DDD Nodes

A node contains two 40 bit pointers, 32 bit value, type, operator and flag bit
Node is stored as a 128 bit struct, two 64 bit integers
Total information is 115 bit, 13 unused bits, all set to 0

low edge		value		high edge

Experiments

- LDD vs. DDD
- Different search strategies
- Reorderings for LDD
- Explicit state with flattened DBM
- Explicit state with pointer to DBM
- Uppaal

Overview

Results (Nodes)

Model	Discrete states	DDD	LDD
fischer6	16320	15156	85041
critRegion4	6629	55890	100006
Critical4	-	-	-
CSMACD8	10515	96098	321001
Viking12	241662	342	342
Lynch5	228579	49430	112397
bocdp	33	487	355
bocdpFIXED	33	488	427
bando	33	488	425
Milner8	128	11012	30887
hddi10	86	-	454246

Results (Time)

Model	DDD	LDD	mc-flattened	mc-original	Uppaal
fischer6	481.9	48.3	19.2	0.4	0.0
critRegion4	46.3	39.5	24.3	0.5	0.1
Critical4	TO	TO	1.1	0.5	0.6
CSMACD8	1.9	7.3	6.9	0.5	0.1
Viking12	17.6	18.7	10.4	0.7	1.0
Lynch5	34.2	120.0	50.0	0.3	0.0
bocdp	0.1	0.2	0.2	0.0	0.2
bocdpFIXED	0.2	0.2	0.1	0.0	0.3
bando	0.2	0.2	0.1	0.0	0.3
Milner8	0.4	1.2	1.4	0.1	0.0
hddi10	TO	93.3	43.1	0.0	0.0

Results

- DDD uses less nodes than LDD
- LDD reorderings not efficient
- No clearly faster symbolic solution
- All new options significantly slower than Uppaal
- Flattening DBM time expensive

Problems

- Too many function calls
- Dependency matrices densely filled
- Large state vectors

Overview

Future work

- DDD reordering
  - Needs mapping of positions and types
- Sparser matrix
  - Split timed and discrete transition
  - May-write matrix
  - Better insight in timing dependencies
- Multi threading
- Subsumption
- Skipping levels

Future work

- Multi threading
  - DDD is already thread save
  - Coupling to DBM not threadsafe
- Subsumption
- Skipping levels
  - All nodes with $(<, \infty)$ left out
  - Need explicit level of each node
  - Node reduction up to 90%