UNIVERSITY OF TWENTE.

Symbolic Model Checking of Timed Automata using LTSmin

Sybe van Hijum

Overview

Model Checking

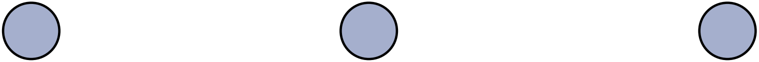
- Models a system, program, protocol, etc...
- Check if model meets specifications
- Problem: State space explosion
  - Grows exponentially with size of model
- Timed Automata adds time to these models

Research Problems

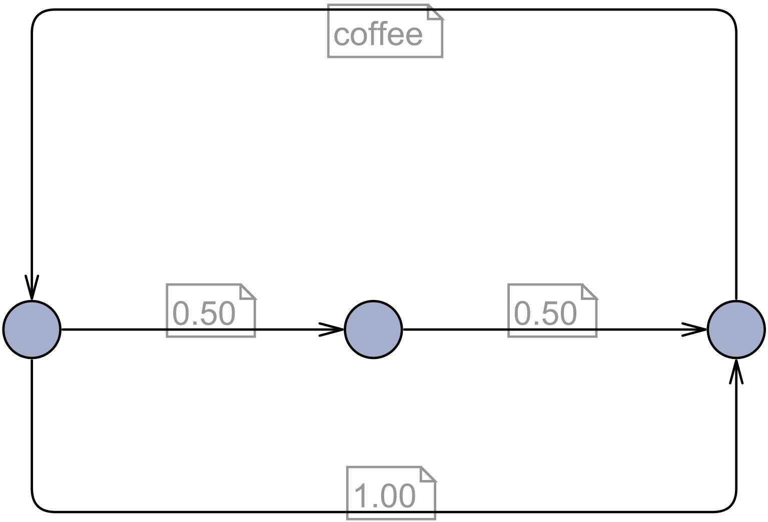
Problem: Model checkers are designed for discrete variables (integers), clocks have real values.

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

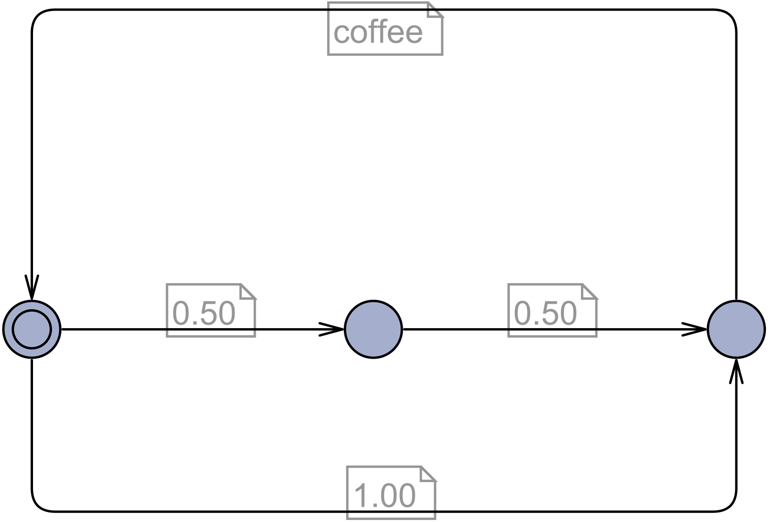
Transition System



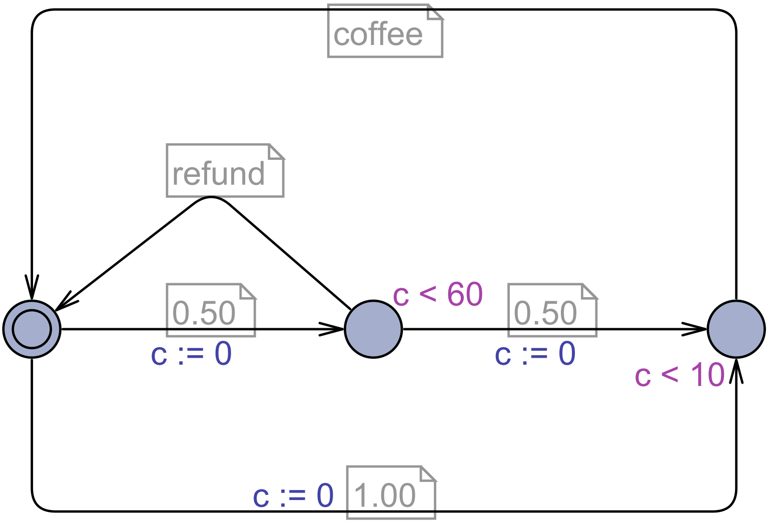
Transition System



Transition System



Timed Automata

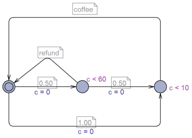


Overview

Time Zones

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

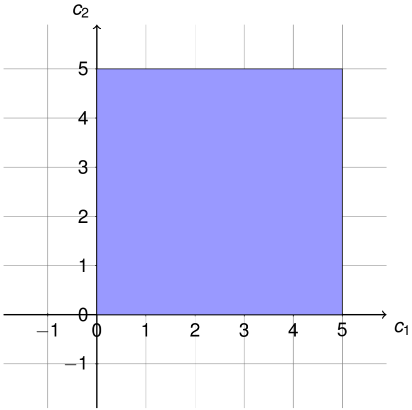
- Only convex zones
- Memory inefficient



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

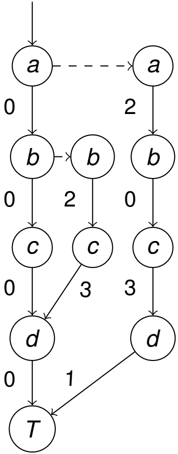
$$\begin{array}{cc} & \begin{array}{cc} \mathbf{0} & c \end{array} \\ \begin{array}{c} \mathbf{0} \\ c \end{array} & \begin{pmatrix} (0, \leq) & (0, \leq) \\ (60, <) & (0, \leq) \end{pmatrix} \end{array}$$

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



Overview

List Decision Diagram



- Diagram to represent set of valuations of integer variables
- Each node has high and low edge
- Each level represents a variable
- Order of variables important

Breadth First Search

procedure BFS(*initial*)
    $vis := cur := initial$
    while $cur \neq \emptyset$ **do**
        $cur := next(cur)$
        $cur := cur \setminus vis$
        $vis := vis \cup cur$
    end while
end procedure

DBM into state vector

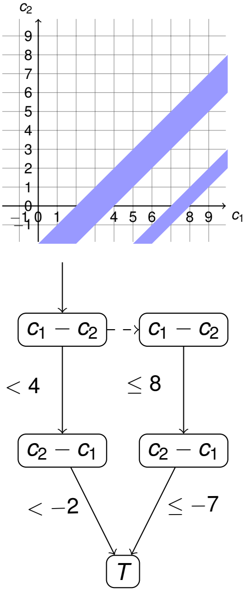
$$
\begin{array}{c}
 \\ \mathbf{0} \\ c_1 \\ c_2
\end{array}
\begin{array}{c}
\begin{array}{ccc} \quad\mathbf{0}\quad & \quad c_1\quad & \quad c_2\quad \end{array} \\
\begin{pmatrix}
(0, \leq) & (0, \leq) & (0, \leq) \\
(5, <) & (0, \leq) & (5, <) \\
(5, <) & (5, <) & (0, \leq)
\end{pmatrix}
\end{array}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

- 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

Overview



Difference Decision Diagram

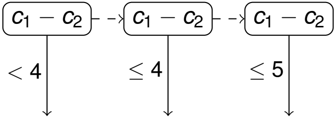
- Structure like LDD
- Added difference operator to each node
- Operator $<$ or $\leq$

Ordering

Definition (Ordered DDD)

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

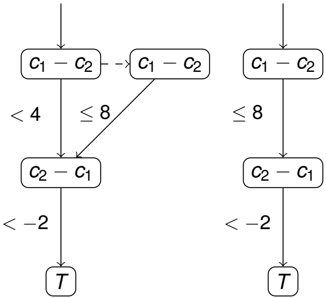
1. $neg(v) \prec pos(v)$,
2. $var(v) \prec var(high(v))$,
3. $var(v) \prec var(low(v))$ *or*
   $var(v) = var(low(v))$ *and* $bound(v) \prec bound(low(v))$.



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* $high(v) \neq high(low(v))$.

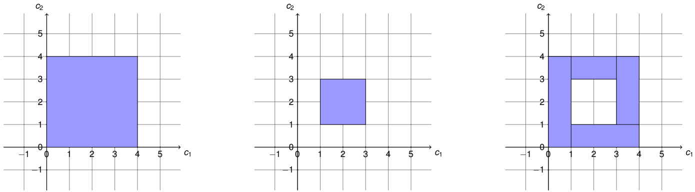


DDD Nodes

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

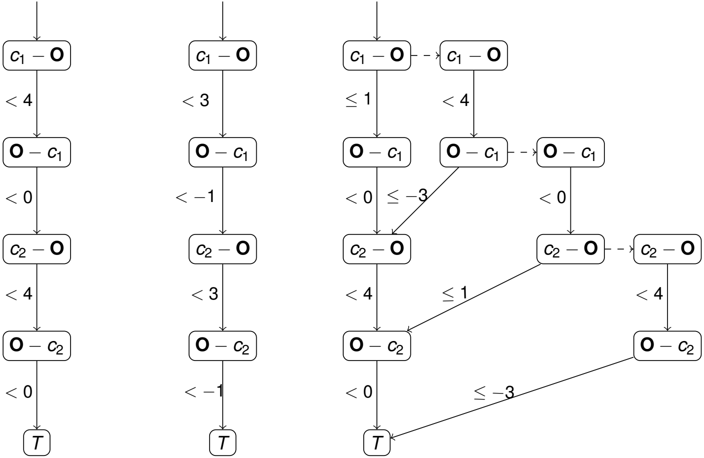
low edge		value		high edge

Minus



Difference of two convex zones not always convex

Minus



$c_1 - \mathbf{0}$
< 4
$\mathbf{0} - c_1$
< 0
$c_2 - \mathbf{0}$
< 4
$\mathbf{0} - c_2$
< 0
T
$c_1 - \mathbf{0}$
< 3
$\mathbf{0} - c_1$
< -1
$c_2 - \mathbf{0}$
< 3
$\mathbf{0} - c_2$
< -1
T
$c_1 - \mathbf{0}$
≤ 1
$\mathbf{0} - c_1$
< 0
≤ -3
$c_2 - \mathbf{0}$
< 4
≤ 1
$\mathbf{0} - c_2$
< 0
≤ -3
T
$c_1 - \mathbf{0}$
< 4
$\mathbf{0} - c_1$
$\mathbf{0} - c_1$
< 0
$c_2 - \mathbf{0}$
$c_2 - \mathbf{0}$
< 4
$\mathbf{0} - c_2$

Overview

Experiments

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

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
  - Must-write matrix
  - Better insight in timing dependencies

Future work

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

- Timed Automata
- Stored as discrete values in LDD
- Stored in specific diagram DDD
- Minus for DDD problematic
- Both solutions slower than original tools
- Many points for future work