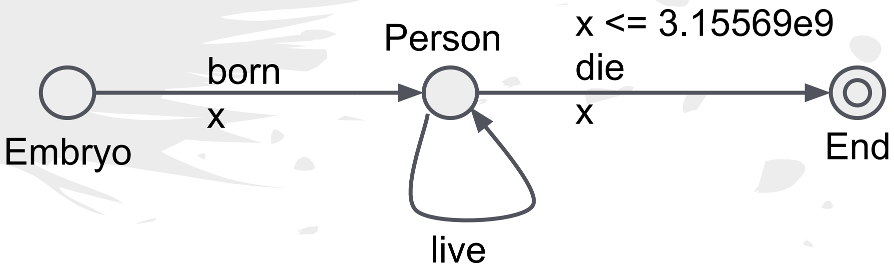
Are Timed Automata Bad for a Specification Language?

Language Inclusion Checking for Timed Automata

Contributors

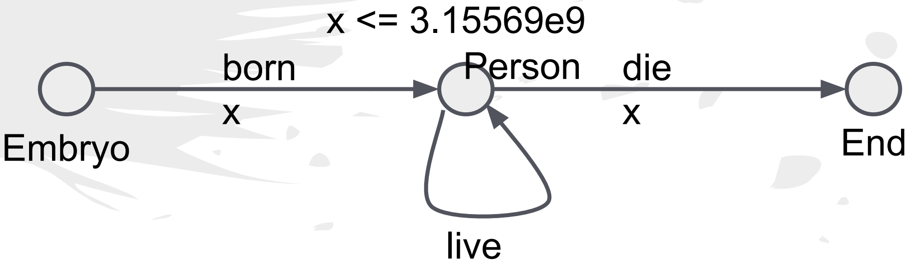
Ting Wang, Zhejiang University
Jun Sun, SUTD
Yang Liu, NTU
Xinyu Wang, Zhejiang University
Shanping Li, Zhejiang University

Timed Buchi Automata



* “A Theory of Timed Automata”, 1994, Dill and Alur

Timed Safety Automata



* “Symbolic Model Checking for Real-Time Systems”, 1992, Hezinger et al.

** Timed Automata means Timed Safety Automata hereafter

Languages

A rooted run of the timed automaton:

<Embryo, 50, Embryo, born, Person, live, Person, 1000, Person, live, Person, die, End>

A word of the timed automaton:

<(50,born),(0,live),(1000,live),(0,die)>

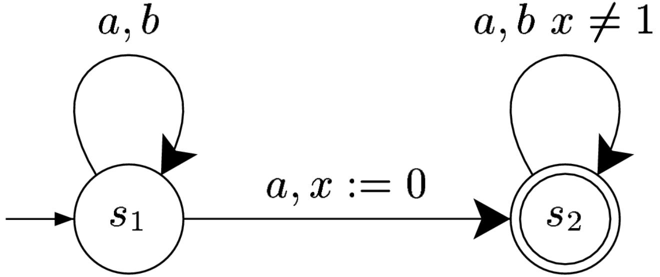
The Problem

Let *Impl* be a timed automaton modeling an implementation; *Spec* be a timed automaton modeling a specification of the system.

Can we check *Impl* refines *Spec*, i.e., any word in *Impl* is in *Spec*?

The Problem is Undecidable

Timed automata are un-determinable*.



* "Decision Problems for Timed Automata: a Survey", 1994, Dill and Madhusudan

The Conclusion?

"... this result is an obstacle in using timed automata as a specification language ..."*

Shall we look at event-clock timed automata, one lock timed automata, instead?

* "A Theory of Timed Automata", 1994, Dill and Alur

This Work

We propose a semi-algorithm for checking whether an arbitrary timed automaton refines another.

We would argue that timed automata are not a bad specification language.

The Result

Are timed automata good for specify commonly used timed properties?

Our semi-algorithm always terminates on commonly used timed properties.

The Result

Does the Semi-Algorithm terminate often?

Highly likely (the answer is related to the transition density of the Spec).

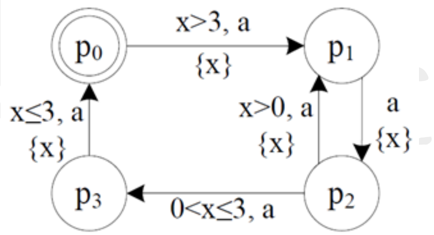
The Result

Is the Semi-Algorithm Scalable in Practice?

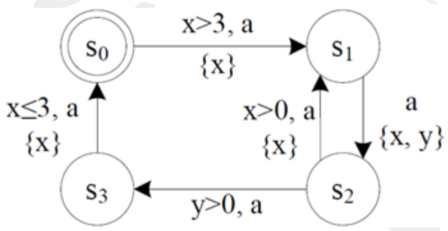
With the reduction techniques in place, it is perhaps as scalable as Uppaal is.

The Approach

Here it goes ...

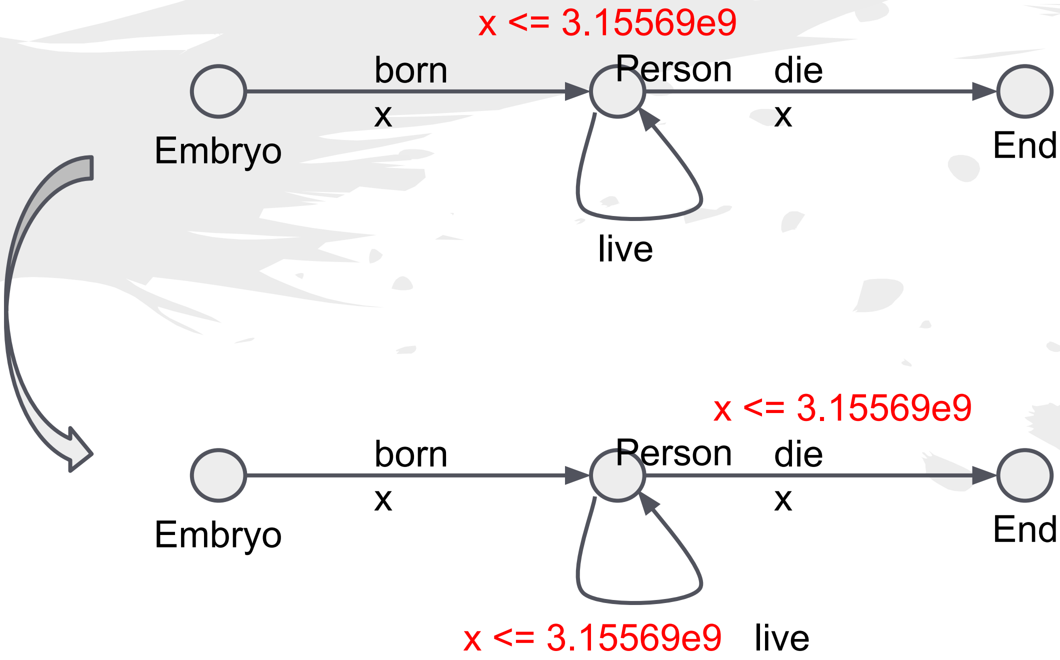


Impl

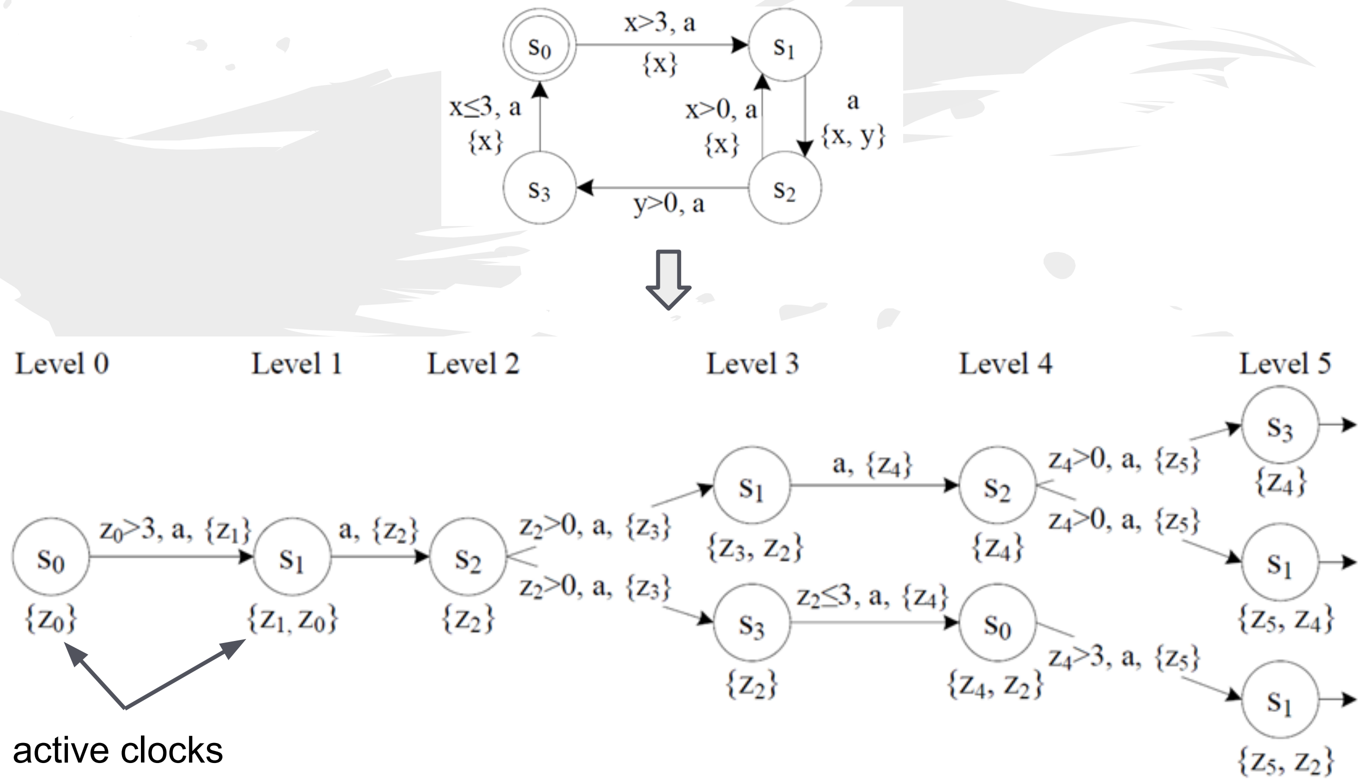


Spec

Step 0: Remove Invariants



Step 1: Unfold Spec

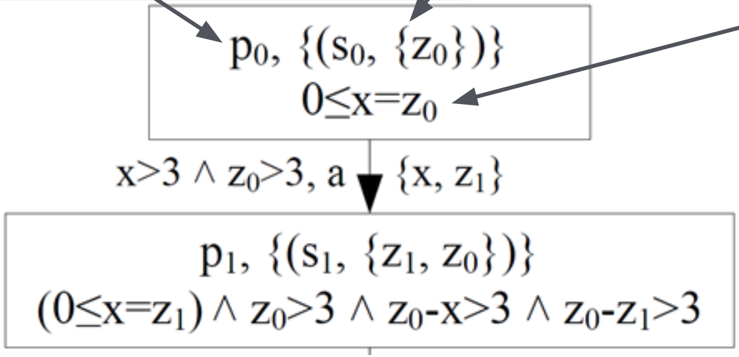


Step 2: Compute the Product

current state in Impl

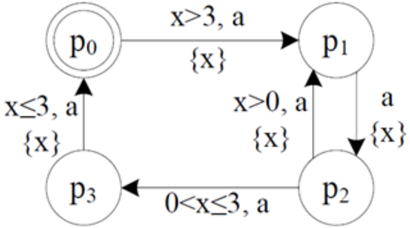
current states in Spec
with active clocks

a zone on all clocks

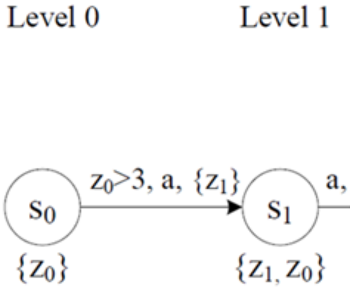


Prod

Impl

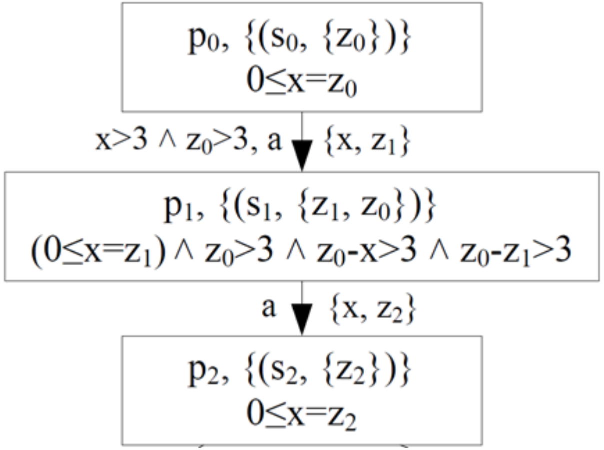


Spec

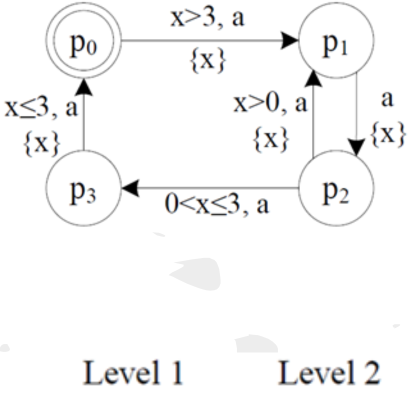


Step 2: Compute the Product

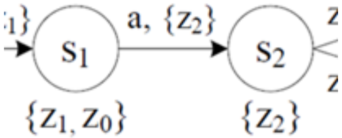
Prod



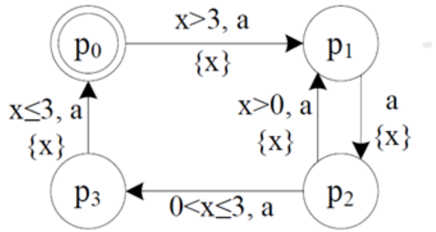
Impl



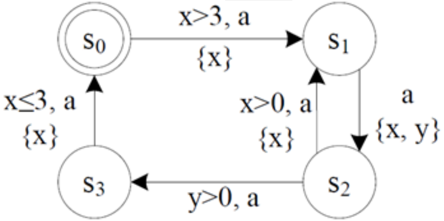
Spec



Step 2: Compute the Product

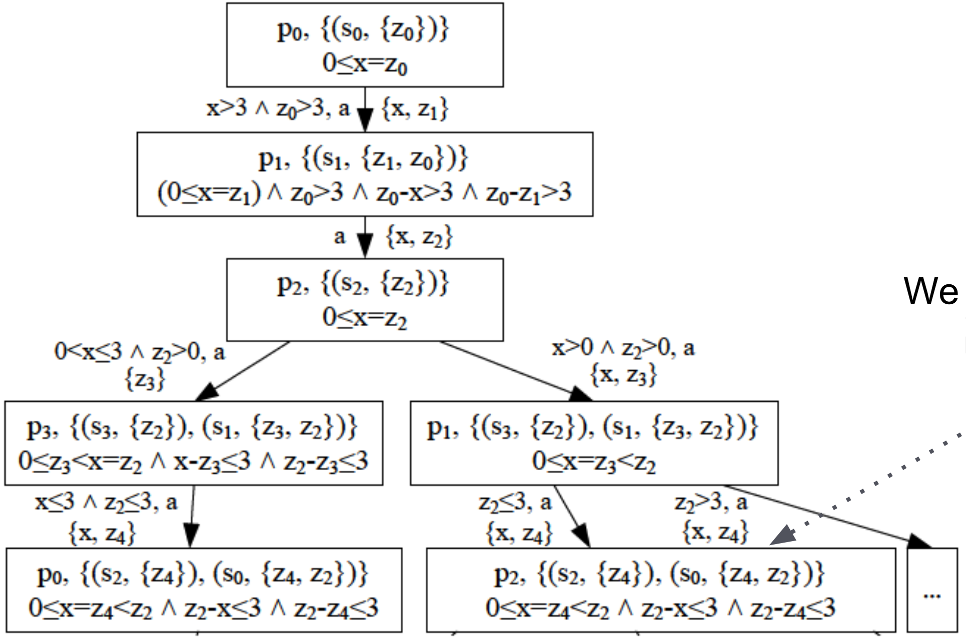


Impl



Spec

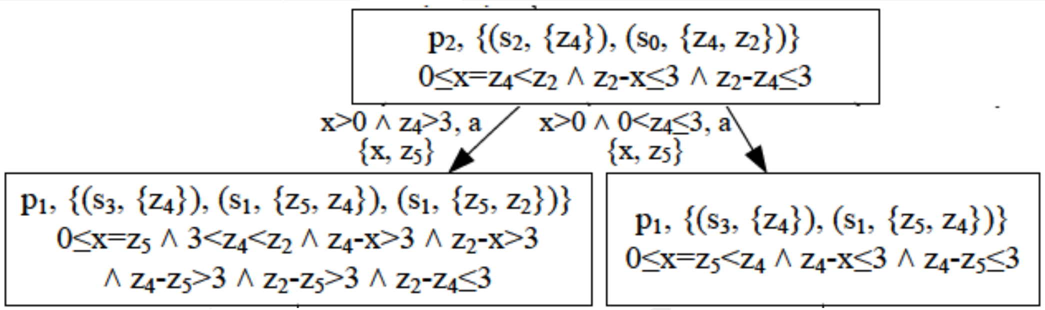
Prod



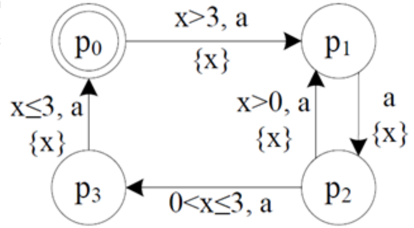
We will look at this one.

Step 2: Compute the Product

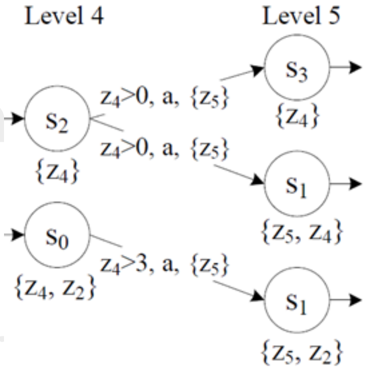
Prod



Impl



Spec

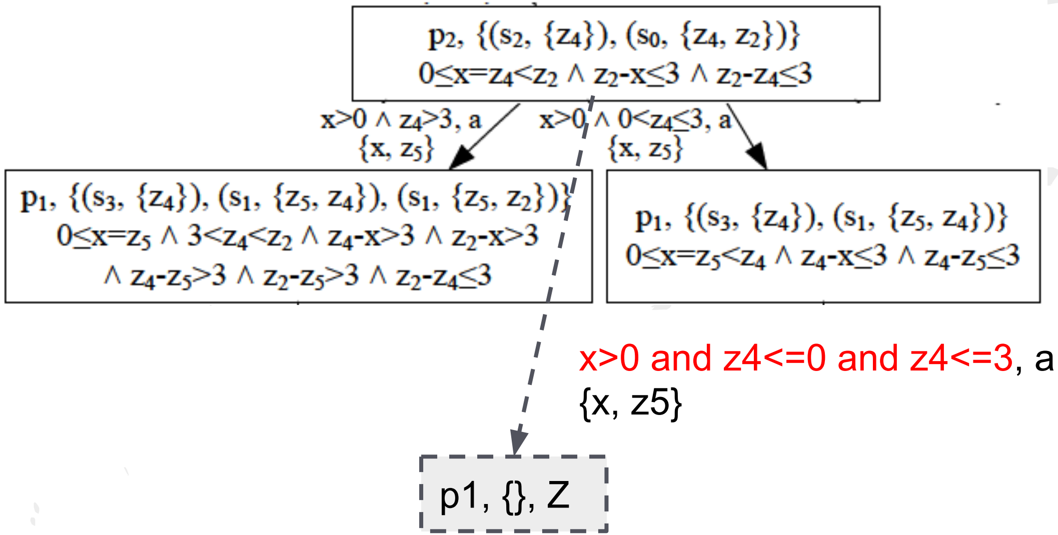


Four combinations:

- x>0 and z4>0 and z4>3
- x>0 and z4>0 and z4<=3
- x>0 and z4<=0 and z4>3
- x>0 and z4<=0 and z4<=3

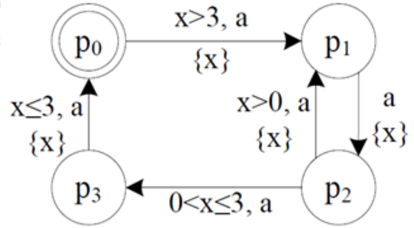
Step 2: Compute the Product

Prod

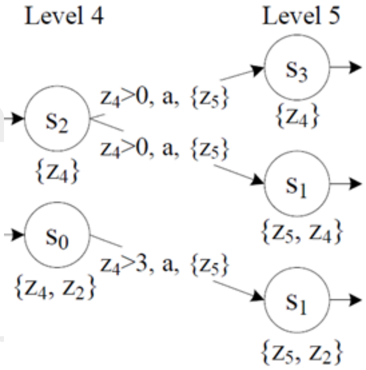


What if Z is not empty?

Impl



Spec



Theorem

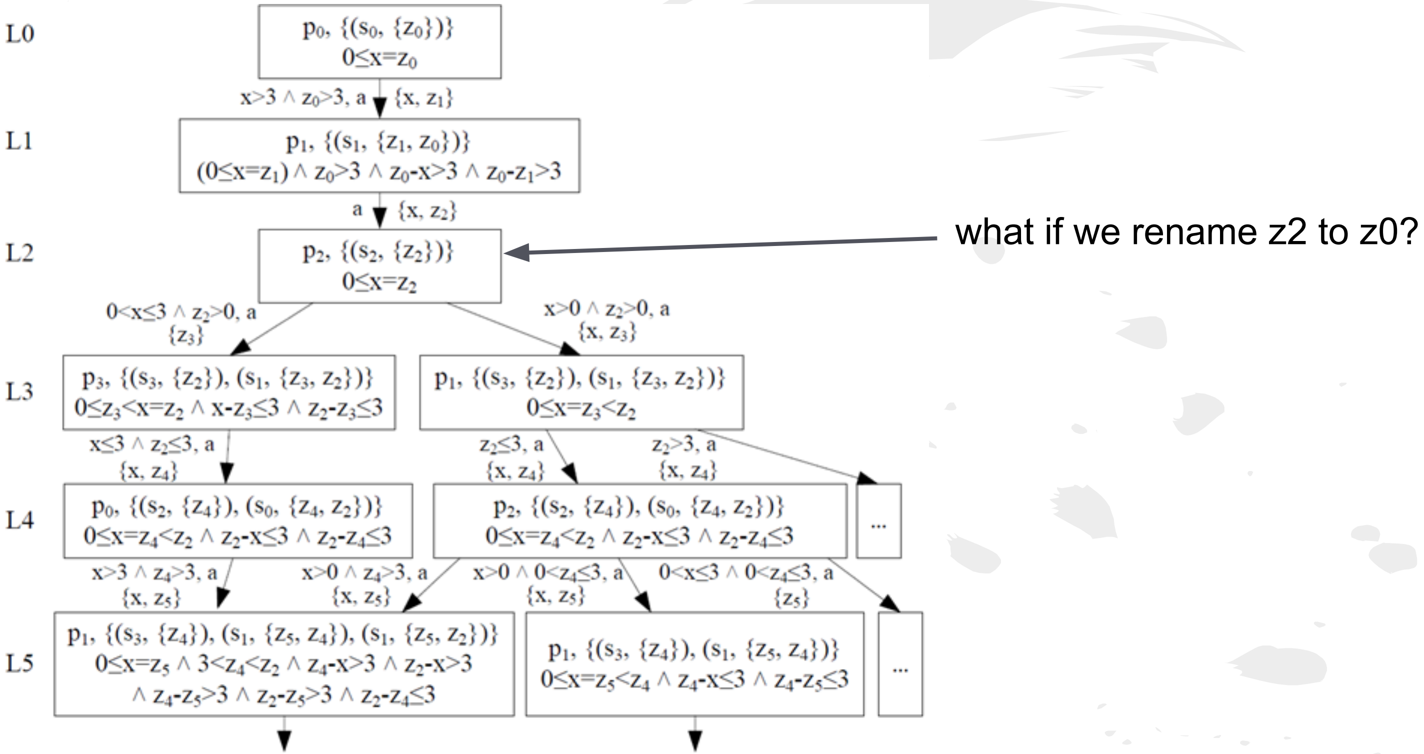
Impl refines *Spec* iff there is no reachable state (p, {}, Z) in *Prod*.

One minor problem: the product has infinitely many states.

Reducing Prod

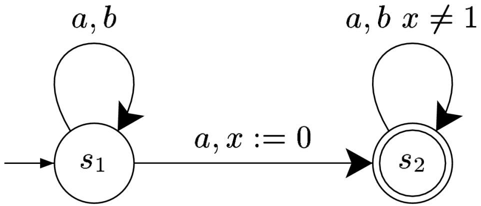
as much as we could …

Clock Renaming



Infinite Clocks

There might be infinitely many active clocks at a state in Prod.



If #clocks are bounded, Prod is finite after clock renaming (with zone normalization).

Simulation Reduction

If s simulates s' (w.r.t a set of accepting states), then if s' can be skipped if s has been explored.

Identifying the simulation relationship is expensive in general.

LU-Simulation

Let (p1, X1, Z1) and (p2, X2, Z2) be two states in Prod. (p2, X2, Z2) simulates (p1, X1, Z1) iff

- $p2=p1$ and $X2 = X1$ and
- for all clock valuation v1 in Z1, there exists v2 in Z2 such that $v1(x) = v2(x)$, or $L(x)<v2(x)<v1(x)$, or $U(x)<v1(x) <v2(x)$ for all x.

where L(x) is the maximal constant from a clock constraint of the form $x>k$ or $x>=k$; U(x) is the maximal constant from a clock constraint of the form $x<k$ or $x<=k$.

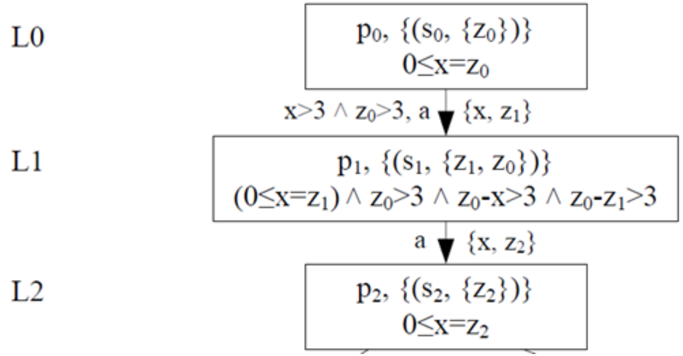
Zone Extrapolation

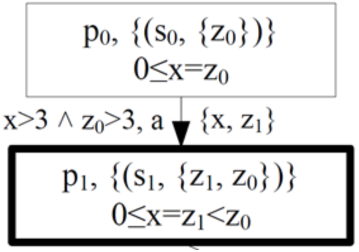
Given a state (p, X, Z), enlarge Z s.t. it contains all clock valuation v1 s.t. there exists v2 in Z such that v1(x) = v2(x), or L(x)<v2(x)<v1(x), or U(x)<v1(x)<v2(x) for all x*.

All clock valuations added to Z are simulated by an existing one.

LU-Simulation: Example

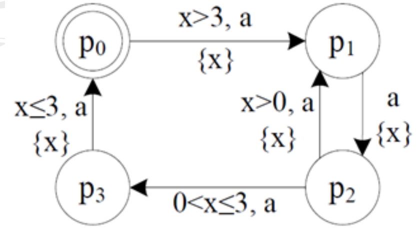
Prod



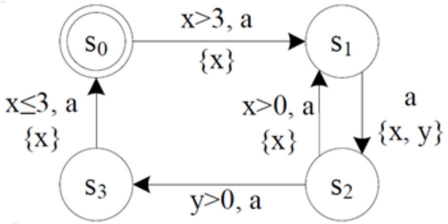


L(x) = 3; U(x)=3
L(z0)=U(z0)=0

Impl



Spec

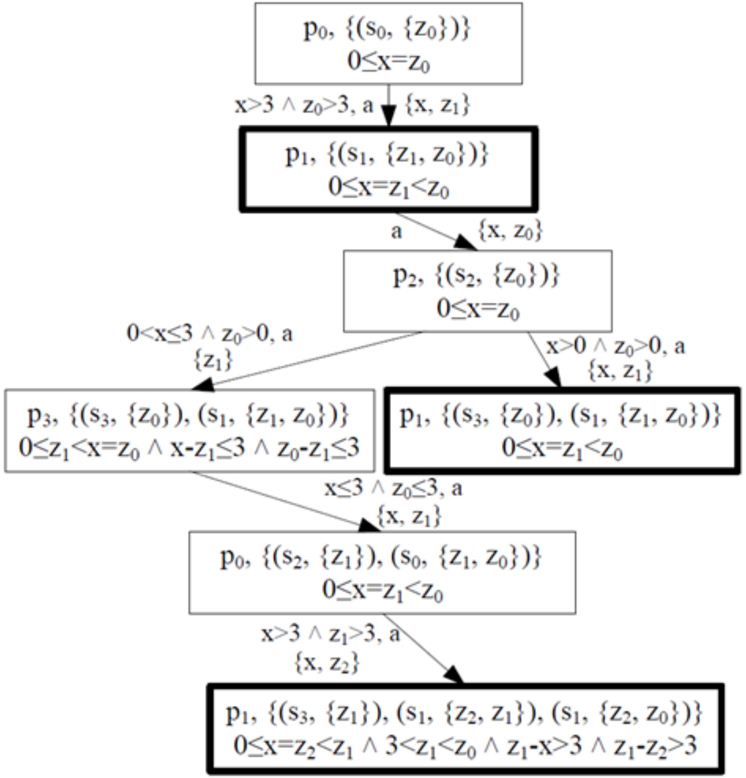


LU Simulation Reduction

During exploration, a state (p, X, Z) can be skipped if a state (p, X, extra(Z')) where Z is a subset of extra(Z') has been explored.

* extra(Z) is the enlarged zone based on Z.

Anti-Chain

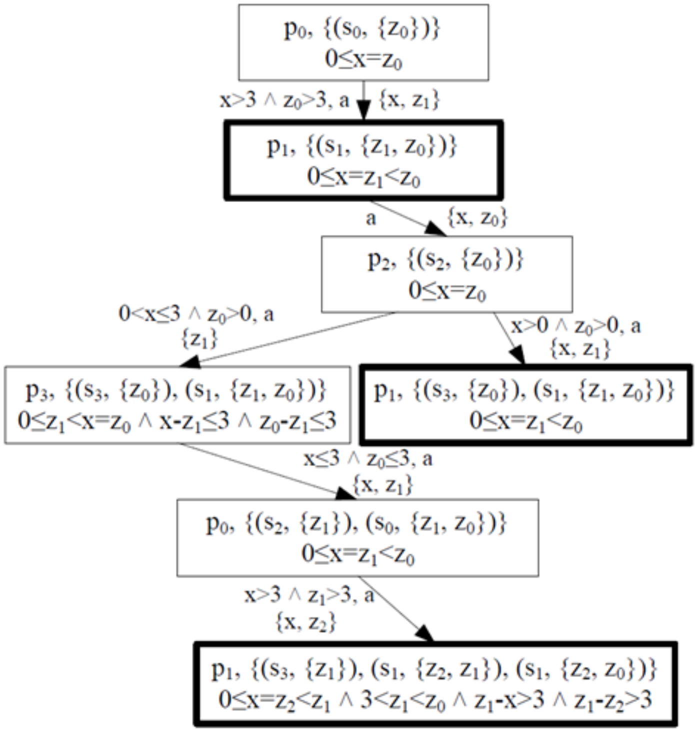


Can we skip this state?

Anti-Chain

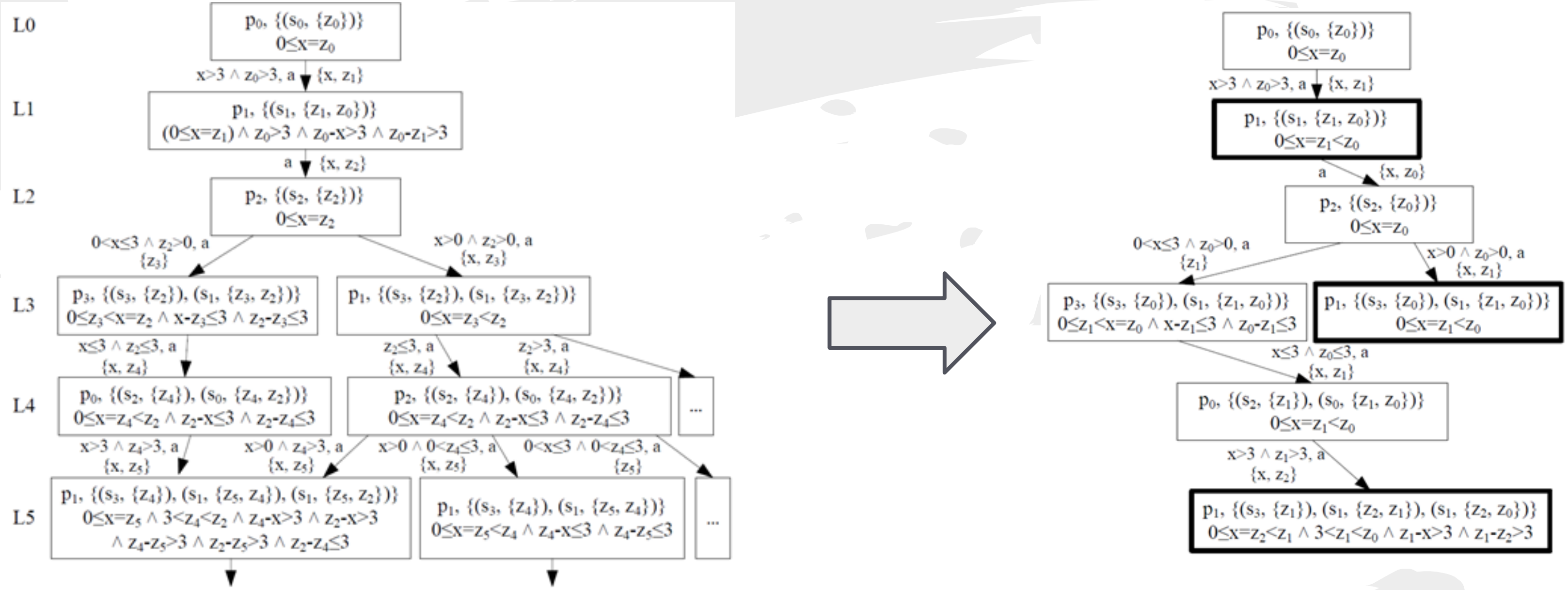
(p1, X1, Z1) simultes (p2, X2, Z2) iff

- p1 = p2 and
- X1 is a subset of X2 and
- Z2 is a subset of Z1*



*with clock renaming

The Reduction



The Algorithm

Algorithm 1 Language inclusion checking

let $working := Init$;
let $done := \emptyset$;
while $working \neq \emptyset$ **do**
  remove $ps = (s_p, X_s, \delta)$ from $working$;
  add ps into $done$ and remove all $ps' \in done$ s.t. $ps' \sqsubseteq ps$;
  for all $(s'_p, X'_s, \delta') \in post(ps, \mathcal{Z}_r^{LU})$ **do**
    if $X'_s = \emptyset$ **then**
      return false;
    end if
    if $\nexists\, ps' \in done$ such that $(s'_p, X'_s, \delta') \sqsubseteq ps'$ **then**
      put (s'_p, X'_s, δ') into $working$;
    end if
  end for
end while
return true;

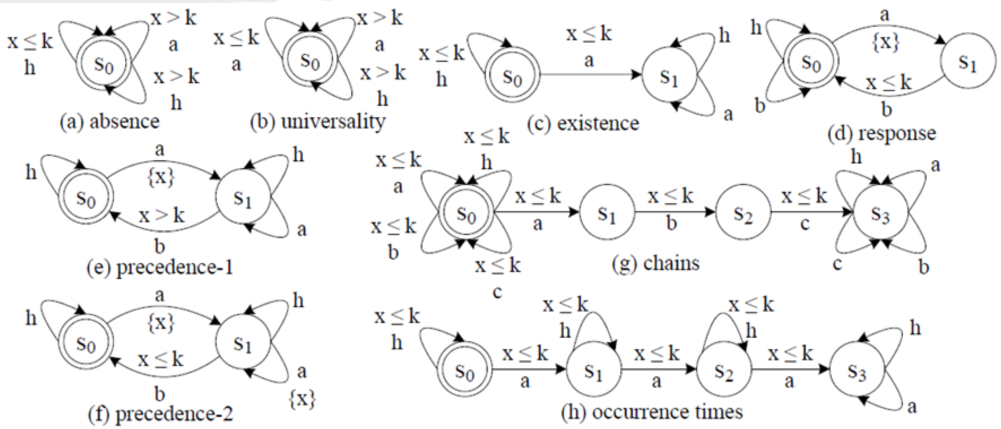
Termination

Always terminates if active clocks are bounded (which includes SNZ, Event-clock timed automata, timed automata with integer resets).

Always terminates for one-clock timed automata.

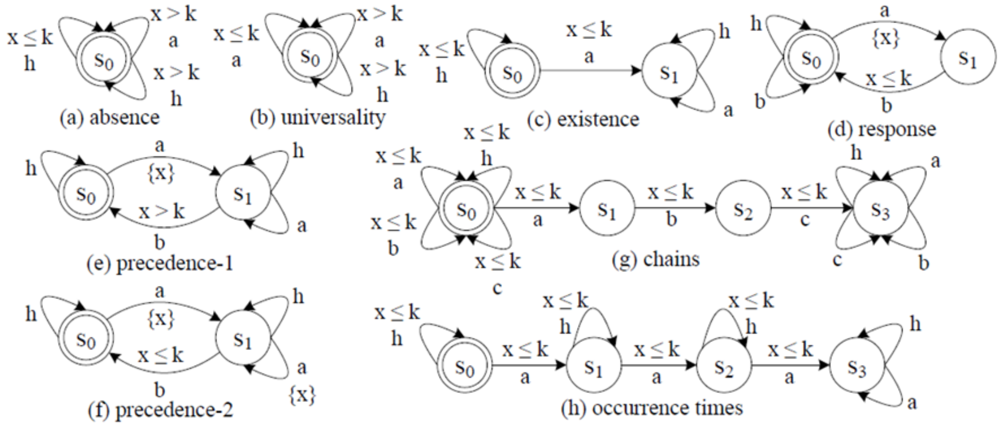
Evaluation 0

Is the algo always terminates given a common timed property? Yes.



Evaluation 0

Is the algo always terminates given a common timed property? Yes.



Evaluation 1

Is the algo scalable?

Table 1. Experiments on Language Inclusion Checking for Timed Systems

System	$\lvert C_s \rvert$	Det	$\sqsubseteq$ +*LU*			*LU*			$\sqsubseteq$		
			stored	total	time	stored	total	time	stored	total	time
Fischer*8	1	Yes	91563	224208	28.3	138657	300384	516.7	-	-	-
Fischer*6	6	No	38603	78332	537.0	-	-	-	-	-	-
Fischer*6	2	No	27393	58531	6.8	36218	70348	30.3	-	-	-
Fischer*7	2	No	121782	271895	42.9	159631	326772	661.7	-	-	-
Railway*8	1	Yes	796154	1124950	142.1	-	-	-	-	-	-
Railway*6	6	No	23265	33427	7.2	27903	39638	20.4	-	-	-
Railway*7	7	No	180034	260199	66.7	222806	318698	1352.8	-	-	-
Lynch*5	1	Yes	3852	11725	0.6	16193	48165	6.0	45488	421582	377.2
Lynch*7	1	Yes	79531	400105	34.9	-	-	-	-	-	-
Lynch*5	2	No	8091	29686	2.4	63623	208607	151.3	56135	324899	290.1
Lynch*6	2	No	35407	162923	16.7	477930	1828668	5751.1	-	-	-
FDDI*7	7	Yes	1198	1590	7.4	8064	9592	36.4	8452	11836	125.5
CSMA*7	1	Yes	9840	36255	4.5	-	-	-	-	-	-

Evaluation 2

Does it terminate?

Table 2. Experiments on Random Timed Automata

$\|S\|$	$\|C\|$	$Dt = 0.6$	$Dt = 0.8$	$Dt = 1.0$	$Dt = 1.1$	$Dt = 1.3$
4	1	1.00\0.99\0.98	0.99\0.93\0.74	0.99\0.82\0.59	0.99\0.63\0.39	0.89\0.18\0.09
4	2	0.99\0.98\0.94	0.98\0.87\0.68	0.94\0.72\0.51	0.85\0.49\0.33	0.45\0.12\0.06
4	3	0.99\0.98\0.93	0.95\0.82\0.65	0.89\0.67\0.52	0.75\0.42\0.28	0.31\0.10\0.06
6	1	1.00\0.99\0.98	0.99\0.97\0.90	0.99\0.61\0.41	0.97\0.43\0.29	0.83\0.13\0.08
6	2	0.99\0.99\0.98	0.99\0.96\0.88	0.88\0.49\0.32	0.79\0.34\0.22	0.44\0.09\0.05
6	3	0.99\0.99\0.98	0.99\0.94\0.85	0.78\0.44\0.29	0.69\0.31\0.21	0.34\0.11\0.07
8	1	1.00\0.99\0.99	0.99\0.92\0.83	0.96\0.53\0.40	0.94\0.37\0.31	0.55\0.08\0.07
8	2	0.99\0.99\0.99	0.99\0.91\0.84	0.84\0.48\0.37	0.73\0.32\0.25	0.25\0.10\0.09
8	3	0.99\0.99\0.99	0.98\0.91\0.83	0.78\0.47\0.38	0.70\0.40\0.32	0.20\0.08\0.07

Dt = #transitions/#states; a\b\c: percentage of termination (a: with reduction; b: without reduction; c: due to Spec being determinizable)

Related Work

Zone abstraction

LU simulation reduction

Anti-chain simulation reduction

Ongoing Work

How to extend the algorithm to deal with non-Zenoness?

What is the best way to verify timed automata with the assumption of non-Zenoness?

Q?