Formal verification of real-time systems

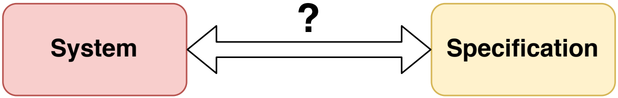
Frédéric Herbreteau (fh@labri.fr)

Bordeaux INP / LaBRI

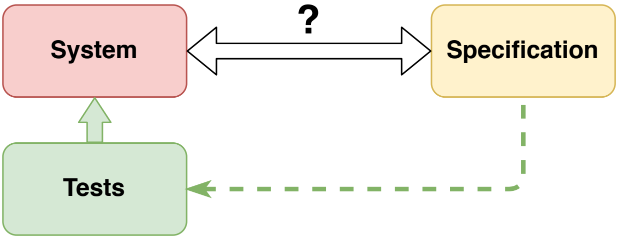
École Temps-Réel 2021 - Poitiers
September 21, 2021

Outline

System verification



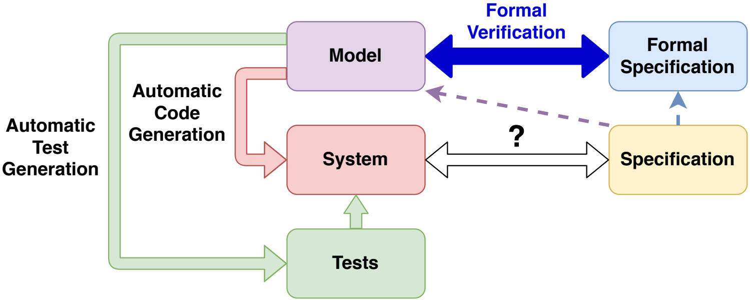
System verification



Standard solution: apply select **test cases** to the system

- **Non-exhaustive:** only a **few select** situations can be tested
- **Hard to reproduce:** in particular for **real-time systems**
- **Late bug discovery:** tests discover **bugs in the system**

Formal verification (model-checking)

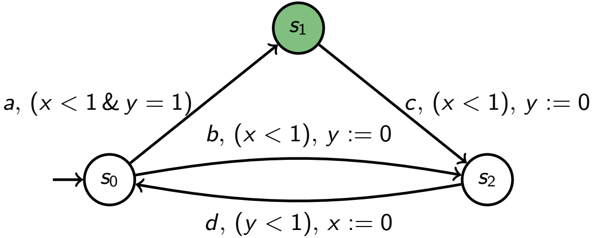


- **Formal models** are built **early** in development cycle
- **Model-checking:** ensures **automatically** and **exhaustively** that all behaviors **conform** to the specification
- **Recommended** for **critical systems** (e.g. ISO26262)

Outline

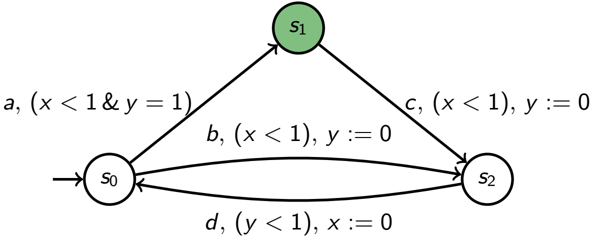
Timed automata [AD94]

Real-time system: correctness depends on **delays**



Timed automata [AD94]

Real-time system: correctness depends on **delays**

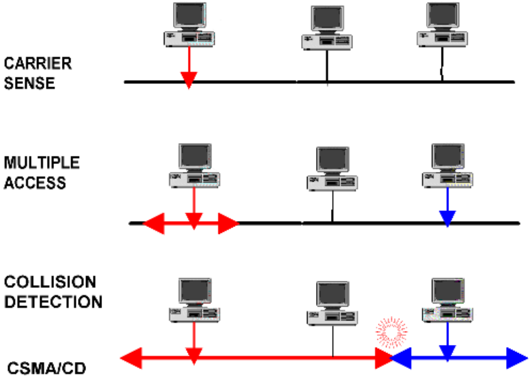


Run: finite **sequence of transitions**

	s_0	$\xrightarrow{0.4}$	s_0	$\xrightarrow{b}$	s_2	$\xrightarrow{0.3}$	s_2	$\xrightarrow{d}$	s_0	$\xrightarrow{0.7,\,a}$	s_1
x	0		0.4		0.4		0.7		0		0.7
y	0		0.4		0		0.3		0.3		1

$\langle q, v\rangle \xrightarrow{\delta,a} \langle q', v'\rangle$ if $\exists\, q \xrightarrow{a,g,R} q'$ s.t. $v + \delta \models g$ and $v' = [R](v + \delta)$.

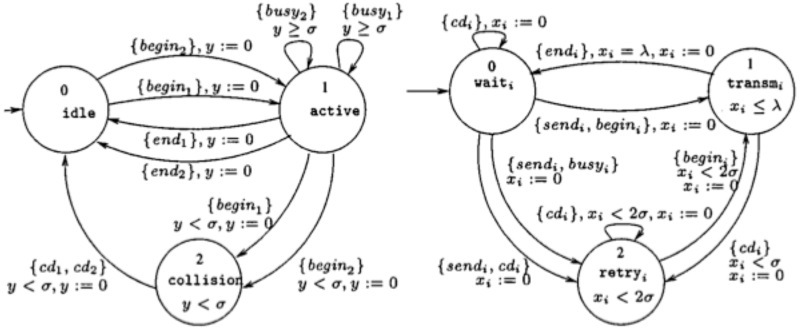
Example #1: the CSMA/CD protocol (1/2)



(source: https://dokteron.blogspot.com/2014/03/csmacd-csmaca.html)

Property to check: detection of **collisions** (based on delays)

Example #1: the CSMA/CD protocol (2/2)



Bus **Station**

(for $\lambda = 808$ and $\sigma = 26$)

Detection failure:

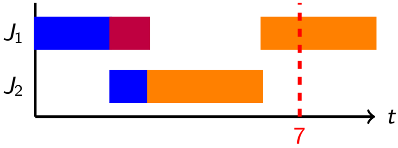
Reachability of a state with *collision* and $wait_1$ or $wait_2$?

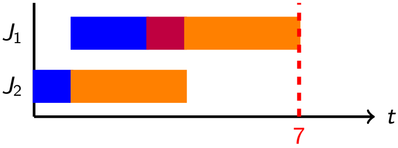
Example #2: scheduling jobs (1/2)

Jobs **compete** to execute tasks on machines

$J_1 : (m_1, 2)\,(m_2, 1)\,(m_3, 3)$ $\qquad$ $J_2 : (m_1, 1)\,(m_3, 3)$

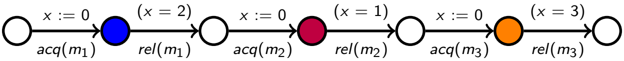
Property to check: can the jobs be **scheduled within** $7s$?

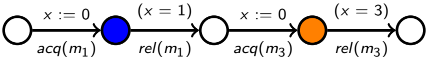




Example #2: scheduling jobs (2/2)

$J_1: \ (m_1, 2)\,(m_2, 1)\,(m_3, 3) \qquad J_2: \ (m_1, 1)\,(m_3, 3) \qquad$ within 7s.



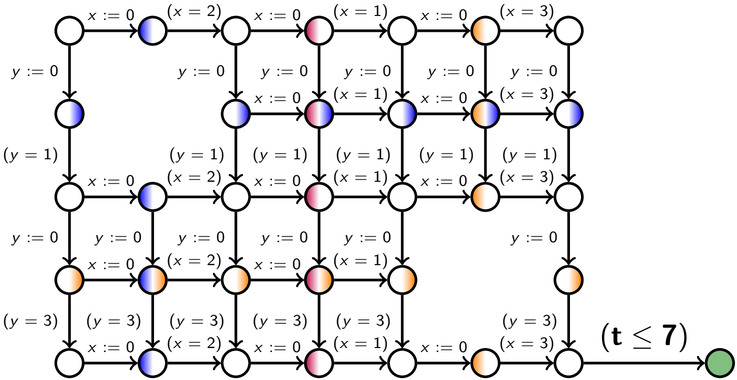


$acq(m)$: await m free, then set m busy

$rel(m)$: set m free

Example #2: scheduling jobs (2/2)

J_1 : $(m_1, 2)\,(m_2, 1)\,(m_3, 3)$ J_2 : $(m_1, 1)\,(m_3, 3)$ within 7s.



Schedulability:

Reachability of the green state?

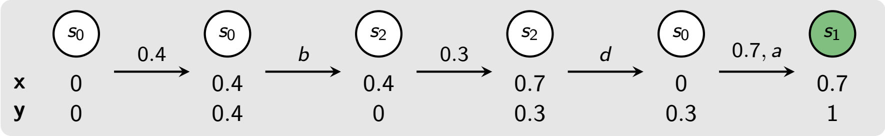
State reachability in timed automata

Specification: reachability of a state

Reachability problem:

INPUT: a timed automaton $\mathcal{A}$ and a state s

QUESTION: is there a run in $\mathcal{A}$ that ends in s?

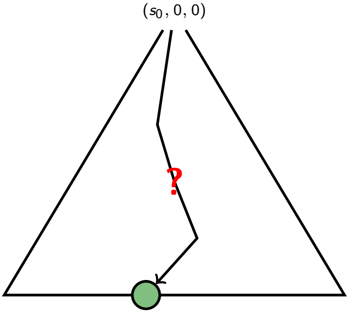


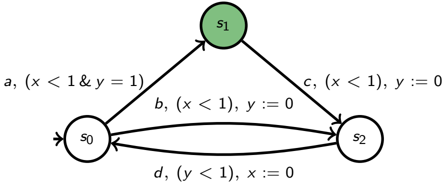
Theorem ([AD94, CY92])

The reachability problem is PSPACE-complete

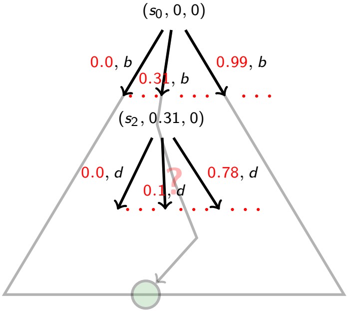
Outline

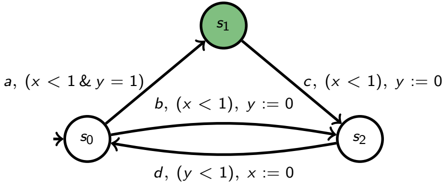
The uncountable state-space





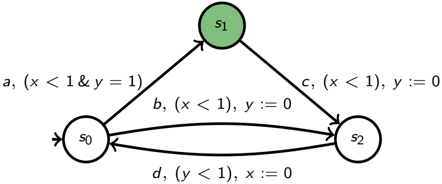
The uncountable state-space

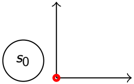




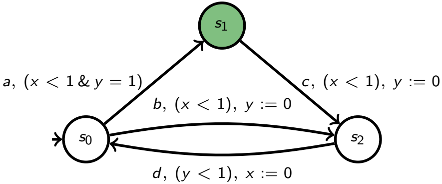
Uncountable state-space due to **density** of time

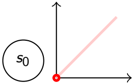
Symbolic semantics: zone graph (1/2)



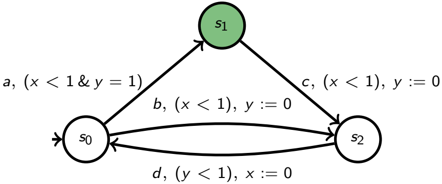


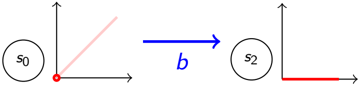
Symbolic semantics: zone graph (1/2)



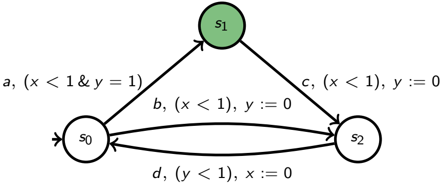


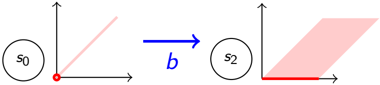
Symbolic semantics: zone graph (1/2)



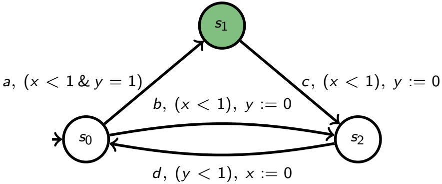


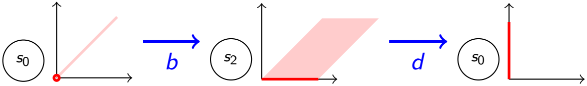
Symbolic semantics: zone graph (1/2)



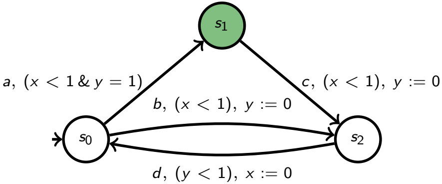


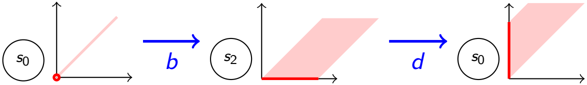
Symbolic semantics: zone graph (1/2)



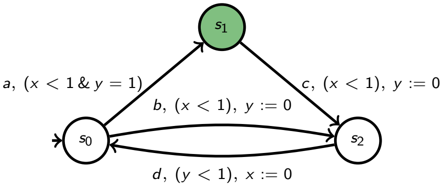


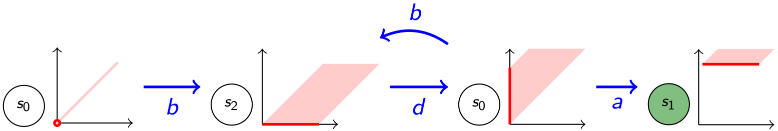
Symbolic semantics: zone graph (1/2)



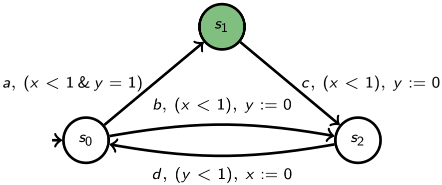


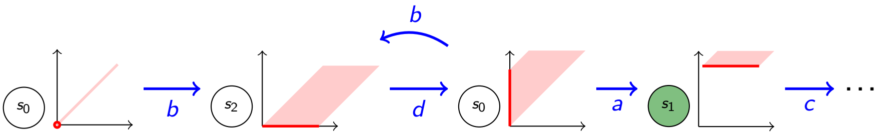
Symbolic semantics: zone graph (1/2)





Symbolic semantics: zone graph (1/2)





Symbolic semantics: zone graph (2/2)

Zone graph [DT98]:

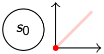
- **Zone:** set of valuations defined by simple constraints
  ($x - y \leq 1 \,\&\, y < 2$)

Symbolic semantics: zone graph (2/2)

Zone graph [DT98]:

- **Zone:** set of valuations defined by simple constraints
  $(x - y \leq 1 \,\&\, y < 2)$

- **Initial node:** $\langle q_0, Z_0 \rangle$ with $Z_0 = \{v_0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$



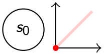
$\langle s_0, 0 \leq x = y \rangle$

Symbolic semantics: zone graph (2/2)

Zone graph [DT98]:

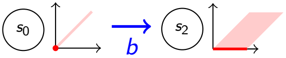
- **Zone:** set of valuations defined by simple constraints
  $(x - y \leq 1 \,\&\, y < 2)$

- **Initial node:** $\langle q_0, Z_0 \rangle$ with $Z_0 = \{v_0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$



$\langle s_0, 0 \leq x = y \rangle$

- **Edge:** $\langle q, Z \rangle \xrightarrow{a} \langle q', Z' \rangle$ if there is a transition $q \xrightarrow{a,g,R} q'$ s.t.
  $Z' = \{v' \mid \exists v \in Z. \exists \delta \in \mathbb{R}_{\geq 0}.\ v + \delta \models g$ and $v' = [R](v + \delta)\}$
  and $Z' \neq \emptyset$.



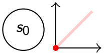
$\langle s_0, 0 \leq x = y \rangle \xrightarrow{b} \langle s_2, 0 \leq x - y < 1 \,\&\, 0 \leq y \rangle$

Symbolic semantics: zone graph (2/2)

Zone graph [DT98]:

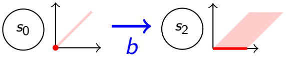
- **Zone:** set of valuations defined by simple constraints
  ($x - y \leq 1$ & $y < 2$)

- **Initial node:** $\langle q_0, Z_0 \rangle$ with $Z_0 = \{v_0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$



$\langle s_0, 0 \leq x = y \rangle$

- **Edge:** $\langle q, Z \rangle \xrightarrow{a} \langle q', Z' \rangle$ if there is a transition $q \xrightarrow{a,g,R} q'$ s.t.
  $Z' = \{v' \mid \exists v \in Z. \exists \delta \in \mathbb{R}_{\geq 0}.\ v + \delta \models g \text{ and } v' = [R](v + \delta)\}$
  and $Z' \neq \emptyset$.



$\langle s_0, 0 \leq x = y \rangle \xrightarrow{b} \langle s_2, 0 \leq x - y < 1 \ \&\ 0 \leq y \rangle$

Theorem ([DT98])

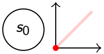
The zone graph is **sound** *and* **complete** *for reachability.*

Symbolic semantics: zone graph (2/2)

Zone graph [DT98]:

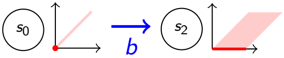
- **Zone:** set of valuations defined by simple constraints
  $(x - y \leq 1 \,\&\, y < 2)$

- **Initial node:** $\langle q_0, Z_0 \rangle$ with $Z_0 = \{v_0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$



$\langle s_0, 0 \leq x = y \rangle$

- **Edge:** $\langle q, Z \rangle \xrightarrow{a} \langle q', Z' \rangle$ if there is a transition $q \xrightarrow{a,g,R} q'$ s.t.
  $Z' = \{v' \mid \exists v \in Z. \exists \delta \in \mathbb{R}_{\geq 0}.\ v + \delta \models g \text{ and } v' = [R](v + \delta)\}$
  and $Z' \neq \emptyset$.



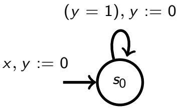
$\langle s_0, 0 \leq x = y \rangle \xrightarrow{b} \langle s_2, 0 \leq x - y < 1 \,\&\, 0 \leq y \rangle$

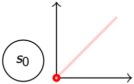
Theorem ([DT98])

The zone graph is **sound** *and* **complete** *for reachability.*

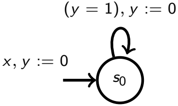
Efficient representation: Difference Bound Matrices [BM83, Dil89]

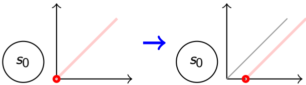
The zone graph may be infinite: abstraction! (1/2)



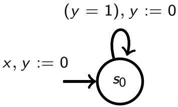


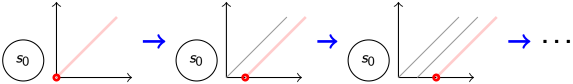
The zone graph may be infinite: abstraction! (1/2)



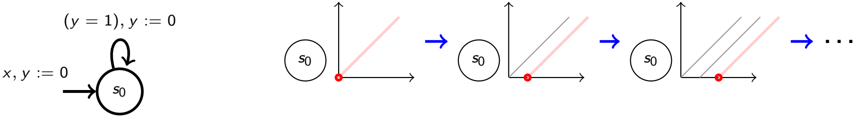


The zone graph may be infinite: abstraction! (1/2)



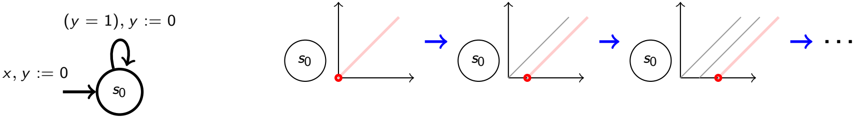


The zone graph may be infinite: abstraction! (1/2)

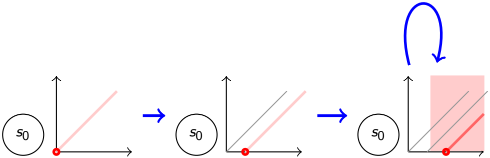


However, the **exact** value of x, y is **irrelevant** once bigger than 1

The zone graph may be infinite: abstraction! (1/2)



However, the **exact** value of x, y is **irrelevant** once bigger than 1



$(x - y = 2 \,\&\, y \geq 0)$ can **safely** be abstracted as $(x > 1 \,\&\, y \geq 0)$

The zone graph may be infinite: abstraction! (2/2)

Abstraction operator $\mathfrak{a}$ defined on the DBM representation of zones

Abstract zone graph:

- **Initial node:** $\langle q_0, \mathfrak{a}(\mathbf{Z_0})\rangle$ where Z_0 is the initial zone
- **Edge:** $\langle q, Z\rangle \xrightarrow{a}_{\mathfrak{a}} \langle q', \mathfrak{a}(\mathbf{Z'})\rangle$ if $Z = \mathfrak{a}(Z)$ and $\langle q, Z\rangle \xrightarrow{a} \langle q', Z'\rangle$ in the zone graph

Theorem ([DT98, BBLP06])

There exists abstractions $\mathfrak{a}$ s.t. the abstract zone graph is **finite**, **sound** and **complete** for finite reachability.

The zone graph may be infinite: abstraction! (2/2)

Abstraction operator $\mathfrak{a}$ defined on the DBM representation of zones

Abstract zone graph:

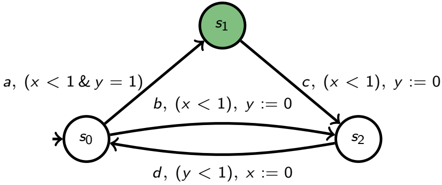
- **Initial node:** $\langle q_0, \mathfrak{a}(\mathbf{Z_0})\rangle$ where Z_0 is the initial zone
- **Edge:** $\langle q, Z\rangle \xrightarrow{a}_{\mathfrak{a}} \langle q', \mathfrak{a}(\mathbf{Z'})\rangle$ if $Z = \mathfrak{a}(Z)$ and $\langle q, Z\rangle \xrightarrow{a} \langle q', Z'\rangle$ in the zone graph

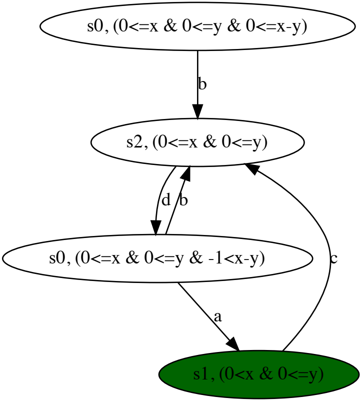
Theorem ([DT98, BBLP06])

There exists abstractions $\mathfrak{a}$ s.t. the abstract zone graph is **finite**, **sound** and **complete** for finite reachability.

The set of behaviors of a timed automaton can be represented as a **finite graph**

Example of finite abstract zone graph





Outline

Reachability algorithm

Search the finite abstract zone graph for an accepting state

```
INPUT: A timed automaton 𝒜
RETURN: true iff 𝒜 has a reachable accepting state

W := {⟨s₀, 𝔞(Z₀)⟩} ;  P := W
while (W ≠ ∅)
   pick and remove a node ⟨s, Z⟩ from W
   if (s is accepting)
      return true
   for each ⟨s, Z⟩ →𝔞 ⟨s′, Z′⟩ do
      if ⟨s′, Z′⟩ ∉ P
         add ⟨s′, Z′⟩ to P and W
   end
end
return false
```

Implementation with TChecker

```cpp
bool reach(tchecker::zg::zg_t & zg)
{
  std::stack<tchecker::zg::state_sptr_t> waiting;
  std::unordered_set<tchecker::zg::state_sptr_t, state_sptr_hash_t,
                     state_sptr_equal_t> passed;
  std::vector<tchecker::zg::zg_t::sst_t> v;

  zg.initial(v, tchecker::STATE_OK);
  for (auto && [status, s, t] : v) {
    waiting.push(s);
    passed.insert(s);
  }
  v.clear();

  while (! waiting.empty()) {
    tchecker::zg::const_state_sptr_t s{waiting.top()};
    waiting.pop();

    if (zg.satisfies(s, labels)) // accepting?
      return true;

    zg.next(s, v, tchecker::STATE_OK);
    for (auto && [status, next_s, t] : v) {
      if (passed.find(next_s) == passed.end()) {
        waiting.push(next_s);
        passed.insert(next_s);
      }
    }
    v.clear();
  }

  return false;
}
```

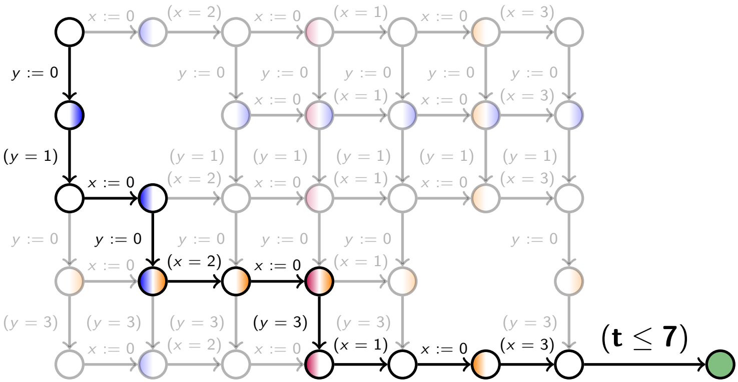
Some examples

CSMA/CD "Unreachability of a state with *collision* and $wait_1/wait_2$?" $\surd$

Some examples

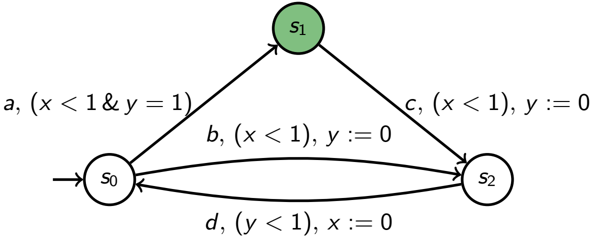
CSMA/CD "Unreachability of a state with *collision* and $wait_1 / wait_2$?" √

Scheduling "Unreachability of the green state?" ×

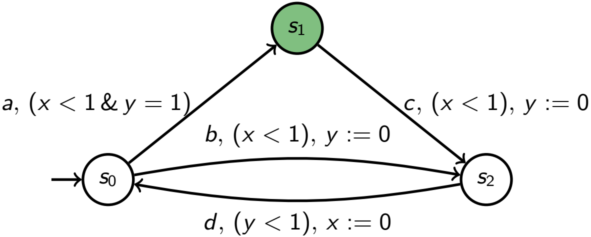


Outline

Liveness properties



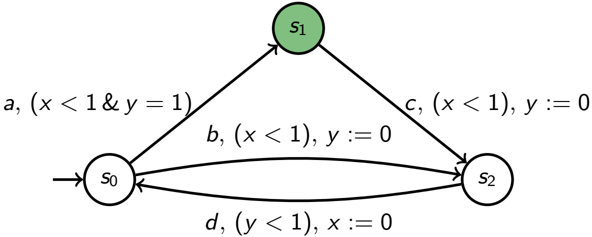
Liveness properties



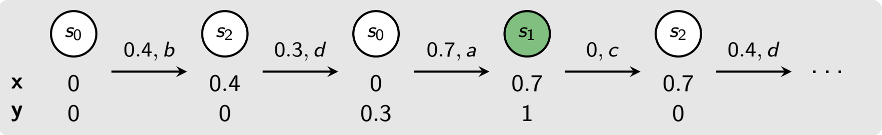
Infinite run: infinite sequence of transitions

	s_0		s_2		s_0		s_1		s_2		
		$\xrightarrow{0.4,\, b}$		$\xrightarrow{0.3,\, d}$		$\xrightarrow{0.7,\, a}$		$\xrightarrow{0,\, c}$		$\xrightarrow{0.4,\, d}$	$\cdots$
x	0		0.4		0		0.7		0.7		
y	0		0		0.3		1		0		

Liveness properties

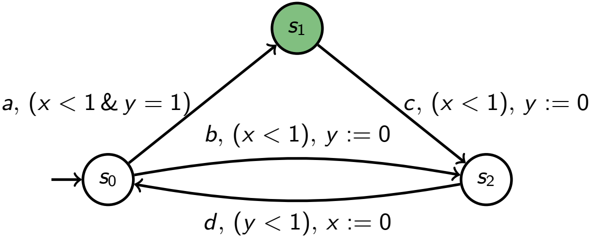


Infinite run: infinite sequence of transitions

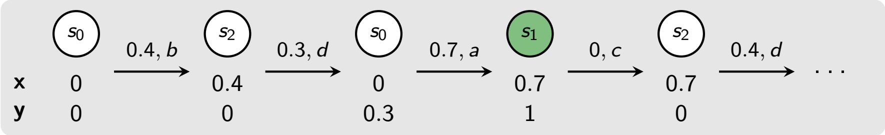


Liveness: visit an accepting state infinitely often

Liveness properties



Infinite run: infinite sequence of transitions

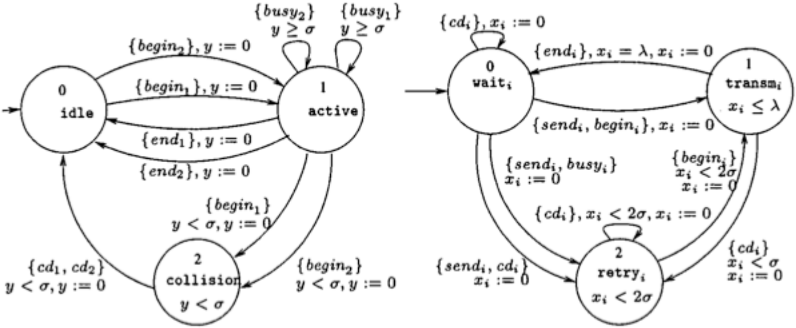


Liveness: visit an accepting state infinitely often

Theorem ([DT98, Li09])

The (abstract) zone graph is **sound** *and* **complete** *for liveness.*

Example #1: CSMA/CD (1/2)

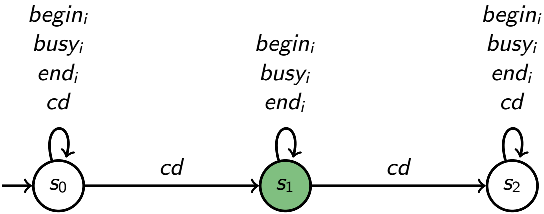


Bus **Station**

Few collisions don't prevent communication: is there a run with **finitely many collisions** and **infinitely many communications**?

Example #1: CSMA/CD (2/2)

Few collisions don't prevent communication: is there a run with **finitely many collisions** and **infinitely many communications**?



- **Product** of the CSMA/CD model and the property automaton
- The property above holds if the state s_1 **is visited infinitely often on a run** in the product

Liveness checking algorithm

Liveness problem:

INPUT: a timed automaton $\mathcal{A}$ and a state s

QUESTION: is there a run in $\mathcal{A}$ that visits s infinitely often?

Theorem ([AD94, CY92])

The liveness problem is PSPACE-complete

Liveness checking algorithm

Liveness problem:

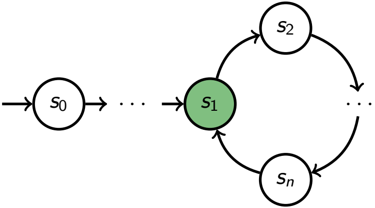
INPUT: a timed automaton $\mathcal{A}$ and a state s

QUESTION: is there a run in $\mathcal{A}$ that visits s infinitely often?

Theorem ([AD94, CY92])

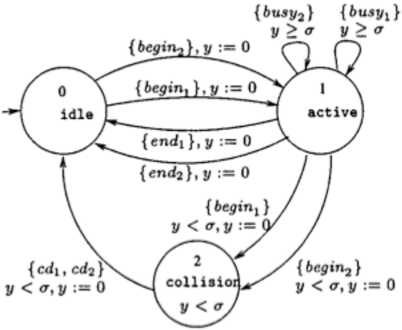
The liveness problem is PSPACE-complete

Algorithm: find an accepting cycle in the abstract zone graph

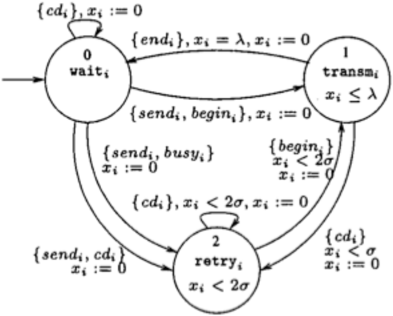


- nested depth-first search
- decomposition into strongly connected components

Example #1: fixing the CSMA/CD model



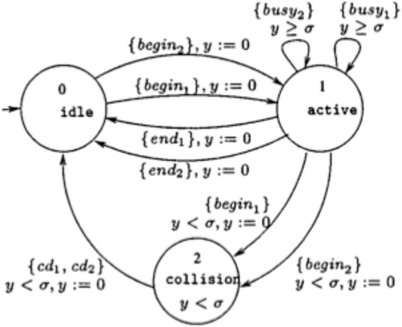
Bus



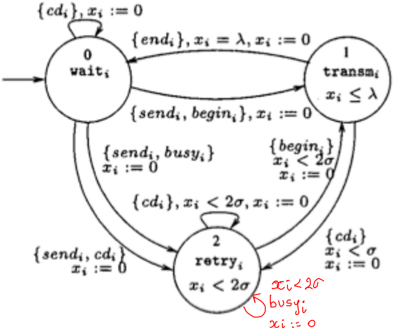
Station

Few collisions don't prevent communication: run with **finitely many collisions** and **infinitely many communications**? ×

Example #1: fixing the CSMA/CD model



Bus



Station

Few collisions don't prevent communication: run with **finitely many collisions** and **infinitely many communications**? √

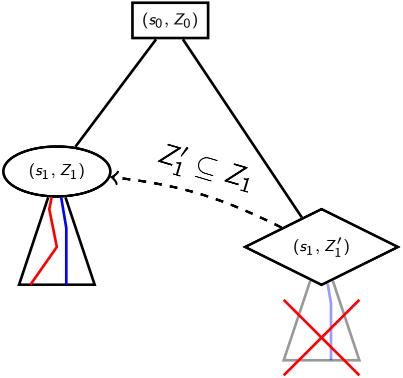
Summary on verification

- **Formal verification** has **sound mathematical** foundations
- **Specification =** Safety (unreachability) + Liveness
- **Abstract zone graph** is finite, sound and complete for verification (both safety and liveness)
- **Standard graph algorithms** can be used to verify timed automata

But many **optimisations** (coarse abstractions, etc) are required to apply model-checking to actual examples

Outline

Subsumption optimization for reachability checking

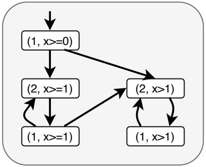


Don't explore (s_1, Z_1'): all its runs are possible from (s_1, Z_1)

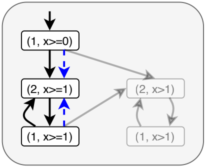
Recall: zones are sets of valuations

Subsumption graphs and reachability

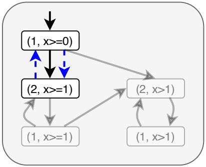
Zone graph $\mathcal{ZG}$



Subsumption graph 1



Subsumption graph 2



- **trace inclusion** when $\langle q, Z\rangle \subseteq \langle q, Z'\rangle$, i.e. $Z \subseteq Z'$
- Standard reachability algorithm: state-space traversal with:
  - Skip $\langle q, Z\rangle$ if **covered** by some visited node $\langle q, Z'\rangle$
  - Only keep **maximal nodes**
- The three graphs above **certify unreachability** of ○

Reachability algorithm with subsumption

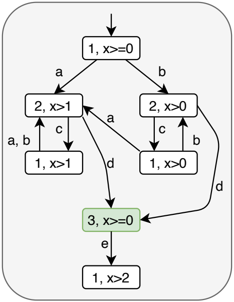
INPUT: A timed automaton $\mathcal{A}$
RETURN: true iff $\mathcal{A}$ has a reachable accepting state

$W := \{\langle s_0, \mathfrak{a}(Z_0)\rangle\}$; $P := W$
while $(W \neq \emptyset)$
  pick and remove a node $\langle s, Z\rangle$ from W
  if (s is accepting)
    return true
  for each $\langle s, Z\rangle \rightarrow_{\mathfrak{a}} \langle s', Z'\rangle$ **do**
    if $\forall \langle s', Z''\rangle \in P$ we have $Z' \not\subseteq Z''$
      remove all nodes $\langle s', Z''\rangle$ with $Z'' \subseteq Z'$ from P and W
      add $\langle s', Z'\rangle$ to P and W
    end
  end
end
return false

In practice: crucial optimisation to **scale** formal verification to models of significant size

Subsumption graphs and liveness

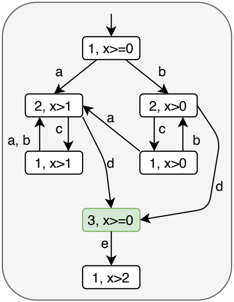
Zone graph $\mathcal{ZG}$ $\surd$



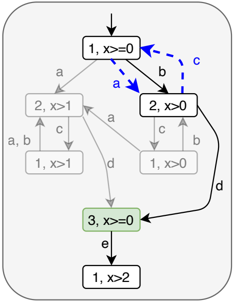
- A subsumption graph with **no accepting cycle** is a liveness **certificate**

Subsumption graphs and liveness

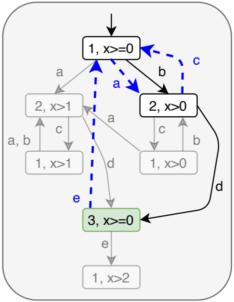
Zone graph $\mathcal{ZG}$ $\surd$



Subsumption graph 1 $\surd$

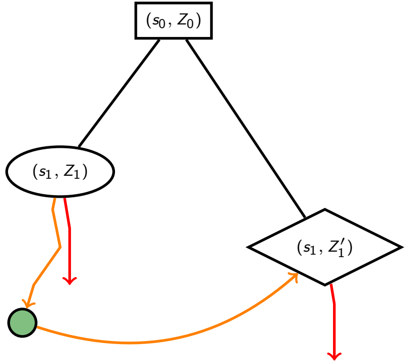


Subsumption graph 2 $\times$



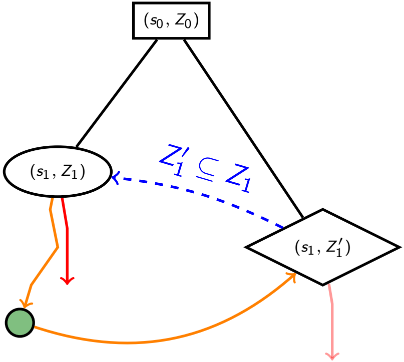
- A subsumption graph with **no accepting cycle** is a liveness **certificate**
- **Not all** subsumptions graphs are liveness certificates

Subsumption creates unsound accepting cycles



Without subsumption: no accepting cycle

Subsumption creates unsound accepting cycles

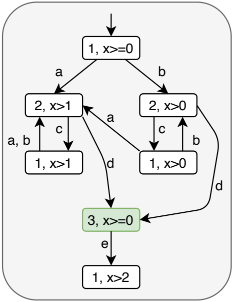


Without subsumption: no accepting cycle

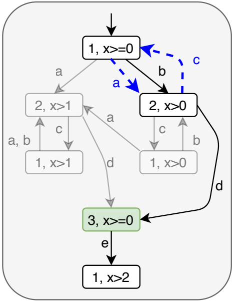
With subsumption: **spurious accepting cycle**, as we claim that $\langle s_1, Z_1' \rangle$ can do the orange path

Liveness compatible subsumption graphs

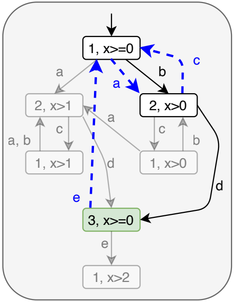
Zone graph $\mathcal{ZG}$ √



Subsumption graph 1 √



Subsumption graph 2 ×



- A subsumption graph is **liveness compatible** if it has no cycle with both ◯ and - - →
- Two main algorithms for computing liveness compatible subsumption graphs: **nested-DFS** [LOD+13] and **SCC-decomposition based refinement algorithm** [HSTW16, HSTW20].

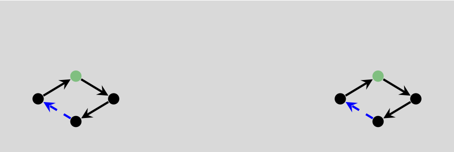
Iterative refinement algorithm [HSTW16, HSTW20]

Level 1

Subsumption graph

Iterative refinement algorithm [HSTW16, HSTW20]

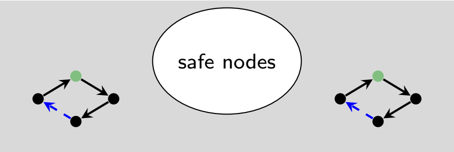
Level 1



Subsumption graph

Iterative refinement algorithm [HSTW16, HSTW20]

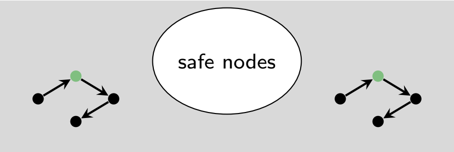
Level 1



Subsumption graph

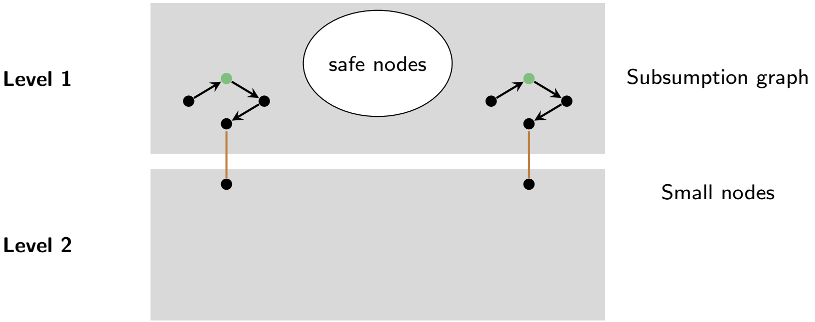
Iterative refinement algorithm [HSTW16, HSTW20]

Level 1

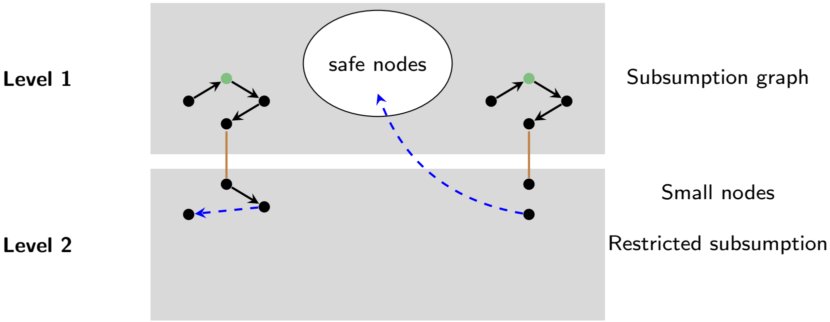


Subsumption graph

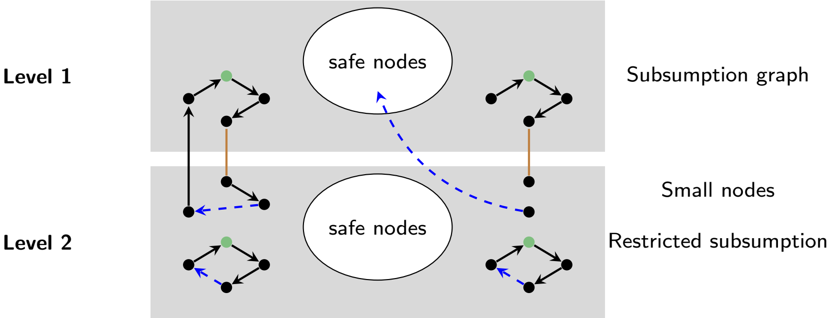
Iterative refinement algorithm [HSTW16, HSTW20]



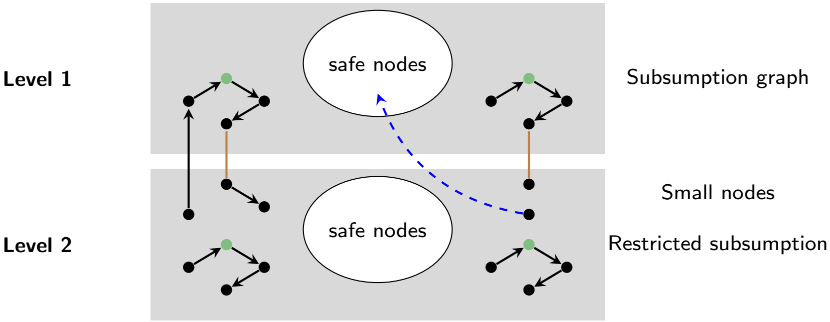
Iterative refinement algorithm [HSTW16, HSTW20]



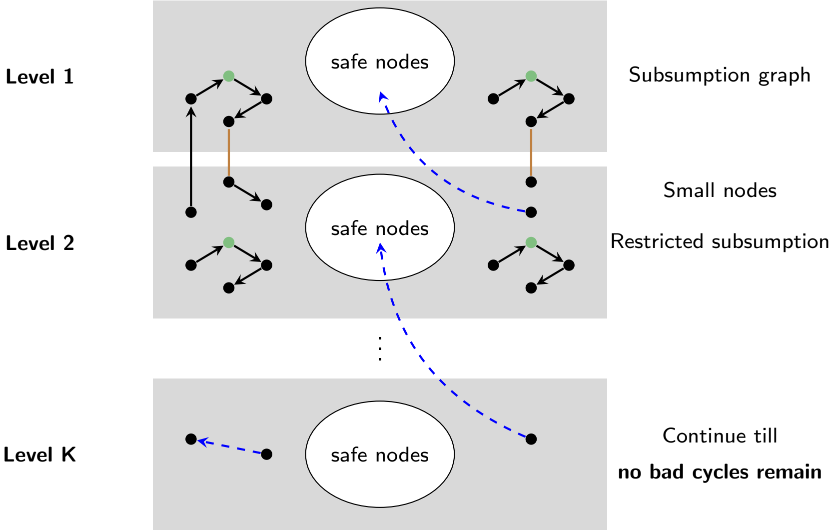
Iterative refinement algorithm [HSTW16, HSTW20]



Iterative refinement algorithm [HSTW16, HSTW20]



Iterative refinement algorithm [HSTW16, HSTW20]



Liveness with subsumption is hard

Inputs	Reachability	Liveness
$\mathcal{A}$	PSPACE-complete	PSPACE-complete
$\mathcal{A}, ZG(\mathcal{A})$	$\mathcal{O}(\lvert ZG(\mathcal{A})\rvert)$	$\mathcal{O}(\lvert ZG(\mathcal{A})\rvert)$
$\mathcal{A}, SubZG(\mathcal{A})$	$\mathcal{O}(\lvert SubZG(\mathcal{A})\rvert)$	PSPACE-complete

Liveness with subsumption is hard

Inputs	Reachability	Liveness
$\mathcal{A}$	PSPACE-complete	PSPACE-complete
$\mathcal{A}, ZG(\mathcal{A})$	$\mathcal{O}(\|ZG(\mathcal{A})\|)$	$\mathcal{O}(\|ZG(\mathcal{A})\|)$
$\mathcal{A}, SubZG(\mathcal{A})$	$\mathcal{O}(\|SubZG(\mathcal{A})\|)$	PSPACE-complete

The iterative refinement algorithm visits each node of $ZG(\mathcal{A})$ **at most 3 times**

Experiments on standard benchmarks: $SubZG(\mathcal{A})$ **is often enough to check liveness**

Outline

Beyond this talk (non exhaustive)

- **Timed automata model-checkers:** UPPAAL (https://uppaal.org/), PAT (https://pat.comp.nus.edu.sg/), ...
- **Effective:** case studies, e.g. Web service transaction protocol [RSV10], Aerial video tracking system [PRH+16]
- **Timed games & control:** see Ocan's talk (UPPAAL TiGa)
- **Quantitative analysis:** weighted timed automata (UPPAAL CORA), probabilistic timed automata (PRISM http://www.prismmodelchecker.org/), ...
- **Robustness & parametric analysis:** SYMROB (https://github.com/osankur/symrob), Imitator (https://www.imitator.fr/)
- **More expressive models:** stopwatches, hybrid systems PHAVer lite (https://www.cs.unipr.it/~zaffanella/PPLite/PHAVerLite), time Petri nets: Romeo (http://romeo.rts-software.org/), Tina (http://projects.laas.fr/tina/)

Timed automata verification in Bordeaux

- Complexity of timed automata verification and **efficient verification algorithms**
- Current challenge: verification of **concurrent real-time systems**
- Open-source implementation: the TChecker tool (`https://github.com/ticktac-project/tchecker`)
- Looking for **collaborations!**

References I

R. Alur and D.L. Dill.
A theory of timed automata.
Theoretical Computer Science, 126(2):183–235, 1994.

G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelanek.
Lower and upper bounds in zone-based abstractions of timed automata.
Int. Journal on Software Tools for Technology Transfer, 8(3):204–215, 2006.

Bernard Berthomieu and Miguel Menasche.
An enumerative approach for analyzing time petri nets.
In *IFIP Congress*, pages 41–46, 1983.

C. Courcoubetis and M. Yannakakis.
Minimum and maximum delay problems in real-time systems.
Form. Methods Syst. Des., 1(4):385–415, 1992.

D. Dill.
Timing assumptions and verification of finite-state concurrent systems.
In *AVMFSS*, volume 407 of *LNCS*, pages 197–212. Springer, 1989.

C. Daws and S. Tripakis.
Model checking of real-time reachability properties using abstractions.
In *TACAS'98*, volume 1384 of *LNCS*, pages 313–329. Springer, 1998.

Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz.
Why liveness for timed automata is hard, and what we can do about it.
In Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen, editors, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, volume 65 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

References II

Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz.
Why liveness for timed automata is hard, and what we can do about it.
ACM Trans. Comput. Log., 21(3):17:1–17:28, 2020.

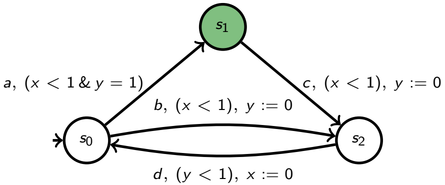
Guangyuan Li.
Checking timed büchi automata emptiness using lu-abstractions.
In Joël Ouaknine, editor, *Formal modeling and analysis of timed systems. 7th Int. Conf. (FORMATS)*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009.

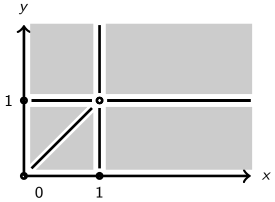
Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol.
Multi-core emptiness checking of timed büchi automata using inclusion abstraction.
In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 968–983. Springer, 2013.

Baptiste Parquier, Laurent Rioux, Rafik Henia, Romain Soulat, Olivier H. Roux, Didier Lime, and Étienne André.
Applying parametric model-checking techniques for reusing real-time critical systems.
In Cyrille Artho and Peter Csaba Ölveczky, editors, *Formal Techniques for Safety-Critical Systems - 5th International Workshop, FTSCS 2016, Tokyo, Japan, November 14, 2016, Revised Selected Papers*, volume 694 of *Communications in Computer and Information Science*, pages 129–144, 2016.

Anders P. Ravn, Jirí Srba, and Muhammad Saleem Vighio.
A formal analysis of the web services atomic transaction protocol with UPPAAL.
In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation - 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part I*, volume 6415 of *Lecture Notes in Computer Science*, pages 579–593. Springer, 2010.

Regions





The region abstraction above is a **bisimulation** relation for **all timed automata with constants at most 1**.