Can Zones be used for reachability in Pushdown Timed Automata?

S. Akshay

Dept of CSE, Indian Institute of Technology Bombay, India

Joint work with Paul Gastin, Karthik R. Prakash
Based on work that appeared in CAV’21.

* Work supported by ReLaX CNRS IRL 2000, DST/CEFIPRA/INRIA project EQuaVE
  & SERB Matrices grant MTR/2018/00074.

TickTac meeting March 2022
The timed automaton model

- Introduced by Alur & Dill in 1990 [AD90]
- Clocks as variables, guards on transitions and resets.
- Reachability is PSPACE-complete – Region Abstraction
  - Exploration of regions: always finite but often large.
- Well studied model with many extensions.
Big leap forward: Making Timed Automata Practical

Zone based abstractions of Timed automata

- Zones: union of regions, "better" abstractions of constraints
  - Exploration of zone graph: Can be infinite but often small.
  - Simulation/subsumption or extrapolation guarantees finiteness.
- UPPAAL [BLL+95, LPY97, PL00, BDL+06], TChecker [HP19], many tools use this!
- Widely used as feasible in practice for many benchmarks...
Big leap forward: Making Timed Automata Practical

Zone based abstractions of Timed automata

- Zones: union of regions, "better" abstractions of constraints
  - Exploration of zone graph: Can be infinite but often small.
  - Simulation/subsumption or extrapolation guarantees finiteness.
- UPPAAL [BLL+95, LPY97, PL00, BDL+06], TChecker [HP19], many tools use this!
- Widely used as feasible in practice for many benchmarks...

Does the “Zone approach” work for extensions of TA?
Pushdown timed automata (PDTA)

A natural extension combining Time and Recursion

- Introduced in [BER94], just after Timed automata!
- PDTA = Timed automata + (pushdown) stack!
Pushdown timed automata (PDTA)

A natural extension combining Time and Recursion
- Introduced in [BER94], just after Timed automata!
- PDTA = Timed automata + (pushdown) stack!

Many theoretical results and extensions
- For instance, [AAS12, CL15, AGK18, CLLM17, AGJK19, CL21]
Pushdown timed automata (PDTA)

A natural extension combining Time and Recursion

- Introduced in [BER94], just after Timed automata!
- PDTA = Timed automata + (pushdown) stack!

Many theoretical results and extensions

- For instance, [AAS12, CL15, AGK18, CLLM17, AGJK19, CL21]
- But very few implementations: [AGKS17, AGKR20].
Pushdown timed automata (PDTA)

A natural extension combining Time and Recursion
- Introduced in [BER94], just after Timed automata!
- PDTA = Timed automata + (pushdown) stack!

Many theoretical results and extensions
- For instance, [AAS12, CL15, AGK18, CLLM17, AGJK19, CL21]
- But very few implementations: [AGKS17, AGKR20].

No known zone based approach... Why?!
Our problem statement

The well-nested control-state reachability problem for PDTA

- Is there a run in PDTA, from initial state to target state s.t.,
  - at initial and target states, the stack is empty.
  - in between stack can grow arbitrarily.

\[ y \leq 2, \text{push}_a \quad x \geq 1, \{x\} \]
Our problem statement

The well-nested control-state reachability problem for PDTA

- Is there a run in PDTA, from initial state to target state s.t.,
  - at initial and target states, the stack is empty.
  - in between stack can grow arbitrarily.
- Our goal: Develop a Zone-based reachability algorithm to compute set of all reachable states (with empty stack).
Our problem statement

The well-nested control-state reachability problem for PDTA

- Is there a run in PDTA, from initial state to target state s.t.,
  - at initial and target states, the stack is empty.
  - in between stack can grow arbitrarily.

  **Our goal:** Develop a Zone-based reachability algorithm to compute set of all reachable states (with empty stack).

Main Challenge

- Each recursive call starts a new exploration of zone graph.
- Can we still use simulations to prune and obtain finiteness?
 Outline of the talk

 1. Re-look at zone algorithms for TA, using re-write rules.
   - Strategies to prune: Simulations and equivalences
Outline of the talk

1. Re-look at zone algorithms for TA, using re-write rules.
   - Strategies to prune: Simulations and equivalences

2. Pinpointing the difficulty in lifting simulations to PDTA
Outline of the talk

1. Re-look at zone algorithms for TA, using re-write rules.
   - Strategies to prune: Simulations and equivalences

2. Pinpointing the difficulty in lifting simulations to PDTA
   - Why using simulations naively in PDTA instead of TA is not sound.
Outline of the talk

1. **Re-look at zone algorithms for TA, using re-write rules.**
   - Strategies to prune: Simulations and equivalences

2. **Pinpointing the difficulty in lifting simulations to PDTA**
   - Why using simulations naively in PDTA instead of TA is not sound.

3. **Refining the rules - New Zone algorithms for PDTA-reach!**
Outline of the talk

1. Re-look at zone algorithms for TA, using re-write rules.
   - Strategies to prune: Simulations and equivalences

2. Pinpointing the difficulty in lifting simulations to PDTA
   - Why using simulations naively in PDTA instead of TA is not sound.

3. Refining the rules - New Zone algorithms for PDTA-reach!
   - A saturation algorithm for well-nested control state reachability in PDTA.
Outline of the talk

1. **Re-look at zone algorithms for TA, using re-write rules.**
   - Strategies to prune: Simulations and equivalences

2. **Pinpointing the difficulty in lifting simulations to PDTA**
   - Why using simulations naively in PDTA instead of TA is not sound.

3. **Refining the rules - New Zone algorithms for PDTA-reach!**
   - A saturation algorithm for well-nested control state reachability in PDTA.

4. **Prototype implementation built on Open source tool, TChecker.**
Outline of the talk

1. **Re-look at zone algorithms for TA, using re-write rules.**
   - Strategies to prune: Simulations and equivalences

2. **Pinpointing the difficulty in lifting simulations to PDTA**
   - Why using simulations naively in PDTA instead of TA is not sound.

3. **Refining the rules - New Zone algorithms for PDTA-reach!**
   - A saturation algorithm for well-nested control state reachability in PDTA.

4. **Prototype implementation built on Open source tool, TChecker.**
   - Challenges in implementing the above algorithm.
Outline of the talk

1. **Re-look at zone algorithms for TA, using re-write rules.**
   - Strategies to prune: Simulations and equivalences

2. **Pinpointing the difficulty in lifting simulations to PDTA**
   - Why using simulations naively in PDTA instead of TA is not sound.

3. **Refining the rules - New Zone algorithms for PDTA-reach!**
   - A saturation algorithm for well-nested control state reachability in PDTA.

4. **Prototype implementation built on Open source tool, TChecker.**
   - Challenges in implementing the above algorithm.

5. **Experimental results and comparisons.**
Initial clock valuation: \((x = y = 0)\).

Allowing time elapse: \((y - x = 0, x \geq 0)\)

\((x = y = 0) = (y - x = 0 \land x \geq 0)\) is the initial zone, \(Z_0\)
Recall: Zones in Timed automata

- Initial clock valuation: \((x = y = 0)\).
- Allowing time elapse: \((y - x = 0, x \geq 0)\)
  - \((x = y = 0) = (y - x = 0 \land x \geq 0)\) is the initial zone, \(Z_0\)
- From zone \(Z\), when we fire transition \(t = (g, R)\), we get...
Recall: Zones in Timed automata

- Initial clock valuation: \( (x = y = 0) \).
- Allowing time elapse: \( (y - x = 0, x \geq 0) \)
  - \( (x = y = 0) = (y - x = 0 \land x \geq 0) \) is the initial zone, \( Z_0 \)
- From zone \( Z \), when we fire transition \( t = (g, R) \), we get
  \[ Z \land g \]
Recall: Zones in Timed automata

- Initial clock valuation: \((x = y = 0)\).
- Allowing time elapse: \((y - x = 0, x \geq 0)\)
  - \((x = y = 0) = (y - x = 0 \land x \geq 0)\) is the initial zone, \(Z_0\)
- From zone \(Z\), when we fire transition \(t = (g, R)\), we get
  \([R](Z \land g)\)
Recall: Zones in Timed automata

- Initial clock valuation: \((x = y = 0)\).
- Allowing time elapse: \((y - x = 0, x \geq 0)\)
  - \((x = y = 0) = (y - x = 0 \land x \geq 0)\) is the initial zone, \(Z_0\)
- From zone \(Z\), when we fire transition \(t = (g, R)\), we get
  \[ Z' = [R](Z \land g) \]
Recall: Zone based Reachability in Timed Automata

Zone graph is defined on nodes, i.e., (state, Zone) pairs

\[(q, Z) \xrightarrow{t} (q', Z') \text{ if } t = (q, g, R, q'), Z' = [R](Z \land g)\]
Recall: Zone based Reachability in Timed Automata

- Zone graph is defined on nodes, i.e., (state, Zone) pairs

\[(q, Z) \xrightarrow{t} (q', Z')\] if \(t = (q, g, R, q')\), \(Z' = [R](Z \land g)\)

**First re-look:** We view this as a fix pt computation

\[
\begin{align*}
S &:= \{(q_0, Z_0)\} \\
(q, Z) &\in S \\
q &\xrightarrow{g, R} q' \\
Z' &\equiv R(g \land Z) \neq \emptyset
\end{align*}
\]

\[
S := S \cup \{(q', Z')\}
\]
Recall: Zone based Reachability in Timed Automata

- Zone graph is defined on nodes, i.e., (state, Zone) pairs

\[(q, Z) \xrightarrow{t} (q', Z') \text{ if } t = (q, g, R, q'), Z' = [R](Z \land g)\]

- Reachability using Zone graph construction is sound, and complete, but non-terminating.
Recall: Getting a finite Zone graph using simulations

\[
(q_0, Z_0) \rightarrow (q_1, Z_1) \rightarrow (q_0, Z_1) \rightarrow (q_2, Z_2) \rightarrow (q_0, Z_2) \rightarrow (q_0, Z_3) \rightarrow (q_1, Z_3) \\
\cdots \cdots \cdots
\]
Recall: Getting a finite Zone graph using simulations

\[
(q_0, Z_0) \rightarrow (q_1, Z_1) \rightarrow (q_0, Z_1) \rightarrow (q_2, Z_2) \rightarrow (q_0, Z_3) \rightarrow (q_1, Z_3)
\]

\[\vdots\]

**Simulation**

\[(q_0, Z_2) \preceq_{q_0} (q_0, Z_1) \quad \text{(Behaviour of } Z_2 \text{ captured by } Z_1 \text{ at } q_0)\]
Recall: Getting a finite Zone graph using simulations

Simulation

- $(q_0, Z_2) \preceq_{q_0} (q_0, Z_1)$ (Behaviour of $Z_2$ captured by $Z_1$ at $q_0$).

$$(q_0, Z_2) \preceq_{q_0} (q_0, Z_1)$$

$$(q_0, Z_2) \preceq_{q_0} (q_0, Z_1)$$

$$(q_n, Z_n) \preceq_{q_n} (q_n, Z'_n)$$
Recall: Getting a finite Zone graph using simulations

Finite Simulation

- \((q_0, Z_2) \preceq_{q_0} (q_0, Z_1)\) (Behaviour of \(Z_2\) captured by \(Z_1\) at \(q_0\)).

- For any infinite sequence of nodes \((q_0, Z_0), (q_1, Z_1), \cdots\), there must exist \(j < i\), s.t., \(q_i = q_j\) and \((q_i, Z_i) \preceq_{q_i} (q_j, Z_j)\)
Recall: Getting a finite Zone graph using simulations

Finite Simulation

- \((q_0, Z_2) \preceq_{q_0} (q_0, Z_1)\) (Behaviour of \(Z_2\) captured by \(Z_1\) at \(q_0\)).
- For any infinite sequence of nodes \((q_0, Z_0), (q_1, Z_1), \ldots\), there must exist \(j < i\), s.t., \(q_i = q_j\) and \((q_i, Z_i) \preceq_{q_i} (q_j, Z_j)\)

Finite simulations guarantee finite zone graph preserving soundness, completeness!
Recall: Getting a finite Zone graph using simulations

Finite Simulation

- $(q_0, Z_2) \preceq_{q_0} (q_0, Z_1)$ (Behaviour of $Z_2$ captured by $Z_1$ at $q_0$).
- For any infinite sequence of nodes $(q_0, Z_0), (q_1, Z_1), \cdots$, there must exist $j < i$, s.t., $q_i = q_j$ and $(q_i, Z_i) \preceq_{q_i} (q_j, Z_j)$

Finite simulations guarantee finite zone graph preserving soundness, completeness!
- There are many known finite simulations, e.g., $LU$-abstraction [BBLP06].
Recall: Getting a finite Zone graph using simulations

Finite Simulation

- \((q_0, Z_2) \preceq_{q_0} (q_0, Z_1)\) (Behaviour of \(Z_2\) captured by \(Z_1\) at \(q_0\)).
- For any infinite sequence of nodes \((q_0, Z_0), (q_1, Z_1), \ldots\), there must exist \(j < i\), s.t., \(q_i = q_j\) and \((q_i, Z_i) \preceq_{q_i} (q_j, Z_j)\)

Finite simulations guarantee finite zone graph preserving soundness, completeness!

- There are many known finite simulations, e.g., \(LU\)-abstraction [BBLP06].
Recall: Getting a finite Zone graph using simulations

\[(x = 1, \{x\})\]

\[q_0 \rightarrow \{x, y\} \rightarrow q_1\]

\[
\begin{align*}
(q_0, Z_0) &\rightarrow (\{x, y\}) \\
(q_1, Z_0) &\rightarrow (x = 1, \{x\}) \\
(q_1, (y - x = 1)) &\rightarrow (x = 1, \{x\}) \\
&\cdots
\end{align*}
\]
Recall: Getting a finite Zone graph using simulations

\[(q_0, Z_0) \xrightarrow{(x, y)} (q_1, Z_0) \xrightarrow{(x = 1, \{x\})} (q_1, (y - x = 1)) \xrightarrow{(x = 1, \{x\})} (q_0, Z_0) \]

\[\vdash_{q_1} \]

We don’t need to know any more details. We only care that such finite simulations exist!
Recall: Getting a finite Zone graph using simulations

A modified re-write rule based saturation algorithm

\[
S := \{(q_0, Z_0)\} \quad \text{start}
\]

\[
(q, Z) \in S \quad q \xrightarrow{g, R} q' \quad Z' = R(g \land Z) \neq \emptyset
\]

\[
S := S \cup \{(q', Z')\}, \text{ unless } \exists (q', Z'') \in S, Z' \preceq_{q} Z''
\]

Trans
Recall: Getting a finite Zone graph using simulations

A modified re-write rule based saturation algorithm

\[
S := \{(q_0, Z_0)\} \quad \text{start}
\]

\[
(q, Z) \in S \quad q \xrightarrow{g,R} q' \quad Z' = R(g \land Z) \neq \emptyset
\]

\[
S := S \cup \{(q', Z')\}, \text{ unless } \exists (q', Z'') \in S, Z' \preceq_{q'} Z''
\]

This algorithm is sound, complete and terminating for computing set of reachable nodes in TA.
The well-nested control-state reachability problem for PDTA

- Given PDTA A, an initial state $q_0$ and a target state $q_f$, is there a run of A from $q_0$ to $q_f$ s.t.,
  - at initial and target states stack is empty.
  - in between stack can grow arbitrarily.
The well-nested control-state reachability problem for PDTA

- Given PDTA $A$, an initial state $q_0$ and a target state $q_f$, is there a run of $A$ from $q_0$ to $q_f$ s.t.,
  - at initial and target states stack is empty.
  - in between stack can grow arbitrarily.
- As in TA, we will instead compute set of all reachable nodes (with empty stack).
The well-nested control-state reachability problem for PDTA

- Given PDTA $A$, an initial state $q_0$ and a target state $q_f$, is there a run of $A$ from $q_0$ to $q_f$ s.t.,
  - at initial and target states stack is empty.
  - in between stack can grow arbitrarily.
- As in TA, we will instead compute set of all reachable nodes (with empty stack).

Let us try the same approach as above!
We start with the initial node $(q_0, Z_0)$.
We start with the initial node and explore as before as long as we see internal transitions (no push-pop).
When we see a Push, we start a new tree/context!
When we see a Push, we start a new tree/context!
Continue as long as we only see internal transitions.
Continue as long as we only see internal transitions.

When we see a "matching" Pop transition,
When we see a "matching" Pop transition, we return to original context and continue from corresponding Push.
Reachability rules for PDTA

- We construct set of nodes explored, as in TA, but parametrized by the root $S_{(q_0, Z_0)}$.

$$S_{(q_0, Z_0)} := \{(q_0, Z_0)\}$$

$$(q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset$$

$$S_{(q, Z)} := S_{(q, Z)} \cup \{(q'', Z'')\},$$

Internal

Start
Reachability rules for PDTA

- We construct set of nodes explored, as in TA, but parametrized by the root $S_{(q_0, Z_0)}$.
- In addition, we maintain the set of roots $\mathcal{G}$:

$$\mathcal{G} := \{(q_0, Z_0)\}, \quad S_{(q_0, Z_0)} := \{(q_0, Z_0)\}$$

$$(q, Z) \in \mathcal{G} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset$$

$$S_{(q, Z)} := S_{(q, Z)} \cup \{(q'', Z'')\},$$

Internal
Reachability rules for PDTA

\[ \mathcal{G} := \{(q_0, Z_0)\}, \quad S_{(q_0, Z_0)} := \{(q_0, Z_0)\} \]

\[ (q, Z) \in \mathcal{G} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{\mathcal{G}, \text{nop}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset \]

\[ S_{(q, Z)} := S_{(q, Z)} \cup \{(q'', Z'')\}, \quad \text{Internal} \]

- When we see a push we add it to set of roots, and start exploration from here.

\[ (q, Z) \in \mathcal{G} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{push}_a, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset \]

\[ \mathcal{G} := \mathcal{G} \cup \{(q'', Z'')\}, \quad S_{(q'', Z'')} = \{(q'', Z'')\}, \quad \text{Push} \]
Reachability rules for PDTA

\[ S := \{(q_0, Z_0)\}, \quad S(q_0, Z_0) := \{(q_0, Z_0)\} \]

Start

\[
\begin{align*}
(q, Z) &\in S \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset \\
S(q, Z) &:= S(q, Z) \cup \{(q'', Z'')\}.
\end{align*}
\]

Internal

\[
\begin{align*}
(q, Z) &\in S \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{push}_a, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset \\
S(q, Z) &:= S(q, Z) \cup \{(q'', Z'')\}.
\end{align*}
\]

Push

Finally, when we see pop, we continue exploring tree where corresponding push happened.

\[
\begin{align*}
(q, Z) &\in S \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{push}_a, R} q'' \quad Z'' = R(g \land Z') \\
(q'', Z'') &\in S \quad (q'_1, Z'_1) \in S(q'', Z'') \quad q'_1 \xrightarrow{g_1, \text{pop}_a, R_1} q_2 \quad Z_2 = R_1(g_1 \land Z'_1) \neq \emptyset \\
S(q, Z) &:= S(q, Z) \cup \{(q_2, Z_2)\}
\end{align*}
\]

Pop
Reachability rules for PDTA

- **Start**

\[
\mathcal{G} := \{ (q_0, Z_0) \}, \quad S(q_0, Z_0) := \{ (q_0, Z_0) \}
\]

\[
S(q, Z) := S(q, Z) \cup \{ (q’, Z’) \}, \quad (q’, Z’) \in S(q, Z)
\]

\[
S(q, Z) := S(q, Z) \cup \{ (q”, Z”) \}, \quad q’ \xrightarrow{g, \text{nop}} q’’ \\
Z” = R(g \wedge Z’) \neq \emptyset
\]

\*

**Internal**

\[
S(q, Z) := S(q, Z) \cup \{ (q’, Z’) \}, \quad q’ \xrightarrow{g, \text{nop}} q’’ \\
Z” = R(g \wedge Z’) \neq \emptyset
\]

**Push**

\[
S(q, Z) := S(q, Z) \cup \{ (q’, Z’) \}, \quad q’ \xrightarrow{g, \text{push}} q’’ \\
Z’’ = R(g \wedge Z’) \neq \emptyset
\]

\[
S(q, Z) := S(q, Z) \cup \{ (q”, Z”) \}, \quad (q”, Z”) \in \mathcal{G}
\]

\[
S(q, Z) := S(q, Z) \cup \{ (q’, Z’) \}, \quad q’ \xrightarrow{g, \text{push}} q’’ \\
Z’’ = R(g \wedge Z’) \neq \emptyset
\]

**Pop**

\[
S(q, Z) := S(q, Z) \cup \{ (q_2, Z_2) \}, \quad (q_2, Z_2) \in \mathcal{G}
\]

\[
S(q, Z) := S(q, Z) \cup \{ (q’, Z’) \}, \quad q’ \xrightarrow{g, \text{pop}} q’’ \\
Z’’ = R(g \wedge Z’) \neq \emptyset
\]

\[
S(q, Z) := S(q, Z) \cup \{ (q_2, Z_2) \}
\]

This set of rules is sound and complete for well-nested control-state reachability in PDTA.

**Issue:** But it is not terminating!
How to handle Push-Pop in the Zone graph

$(q_0, Z_0) \xrightarrow{} (q_1, Z_1) \xrightarrow{} (q_2, Z_2)$

- Two sources of infinity!
How to handle Push-Pop in the Zone graph

\[ \rightarrow (q_0, Z_0) \rightarrow (q_1, Z_1) \rightarrow (q_2, Z_2) \]

- Two sources of infinity!
  - Number of nodes in a tree
How to handle Push-Pop in the Zone graph

- Two sources of infinity!
  - Number of nodes in a tree
  - Number of root nodes, since each push starts tree at new root!
How to handle Push-Pop in the Zone graph

- Two sources of infinity!
  - Number of nodes in a tree
  - Number of root nodes, since each push starts tree at new root!
- Simulation inside a tree (i.e., within each tree) handles the first.
How to handle Push-Pop in the Zone graph

- Two sources of infinity!
  - Number of nodes in a tree
  - Number of root nodes, since each push starts tree at new root!

- Simulation inside a tree (i.e., within each tree) handles the first.
- But not the second! We lose soundness...
The problem with simulation & soundness

\[ (q_0, Z_0) \rightarrow (q, Z) \rightarrow (q_1, Z_1) \rightarrow (q_1, Z_2) \]

\[ (q', Z') \]

\[ \text{push}_a \]

\[ \text{push}_b \]

Can Zones be used for Reachability in PDTA?

TicTacToe 2022
The problem with simulation & soundness

\[
(q_0, Z_0) \rightarrow (q, Z) \rightarrow (q_1, Z_1) \rightarrow (q_1, Z_2)
\]

Can Zones be used for Reachability in PDTA? TicTacToe 2022
The problem with simulation & soundness

\[ (q_0, Z_0) \rightarrow (q_1, Z_1) \]

\[ (q, Z) \quad \text{push}_a \quad (q'_1, Z'_1) \]

\[ (q'_2, Z'_2) \quad \text{pop}_b \quad (q_1', Z'_1) \]

\[ (q_1, Z_2) \]

\[ (q', Z') \quad \text{push}_b \]
The problem with simulation & soundness

$(q_0, Z_0) \xrightarrow{} (q_1, Z_1)$

$(q, Z) \xrightarrow{\text{push}_a} (q', Z')$

$(q_2, Z_2) \xrightarrow{\text{pop}_a} (q_1, Z_1)$

$(q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2)$

$(q_f, Z_f) \xrightarrow{\text{pop}_b} (q_1', Z_1')$
The problem with simulation & soundness

\[(q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \xrightarrow{\text{push}_b} (q_1, Z_2) \xrightarrow{\text{pop}_b} (q_1', Z_1') \xrightarrow{\text{pop}_a} (q_f, Z_f)\]

\[(q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_2} (q_1, Z_1) \xrightarrow{\text{push}_b} (q_1, Z_2) \xrightarrow{\text{pop}_b} (q_1', Z_1') \xrightarrow{\text{pop}_a} (q_f, Z_f)\]
The problem with simulation & soundness

\[(q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \rightarrow (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \not\rightarrow \exists (q', Z')\]

\[(q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \xrightarrow{\text{push}_b} (q_1, Z_2) \xrightarrow{\text{pop}_a} (q', Z') \xrightarrow{\text{pop}_b} (q', Z''_1) \rightarrow (q_0, Z_0)\]
The problem with simulation & soundness

\[ (q_0, Z_0) \rightarrow (q, Z) \rightarrow (q_1, Z_1) \rightarrow (q', Z') \rightarrow (q_1, Z_2) \rightarrow \cdots \]

\[ (q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \rightarrow (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \not\rightarrow \]

Not Sound!

So how do we fix it?
The problem with simulation & soundness

\[ (q_0, Z_0) \xrightarrow{\text{push}_a} (q_1, Z_1) \xrightarrow{\text{push}_b} (q_1, Z_2) \xrightarrow{\text{pop}_b} (q_1', Z_1') \xrightarrow{\text{pop}_a} (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \xrightarrow{\text{pop}_b} (q_1', Z_1') \xrightarrow{\text{pop}_a} (q', Z') \xrightarrow{\text{push}_a} (q_1, Z_1) \xrightarrow{\text{push}_b} (q_1, Z_2) \]

Use equivalence!
The problem with simulation & soundness

\[ (q_0, Z_0) \rightarrow (q_1, Z_1) \]

\[ (q, Z) \]

\[ (q_1, Z_1) \]

\[ (q_2, Z_2) \]

\[ (q', Z') \]

\[ (q_1, Z_2) \]

\[ (q', Z') \]

\[ (q_1', Z_1') \]

\[ (q, Z) \rightarrow (q_1, Z_1) \rightarrow (q', Z') \rightarrow (q_1', Z_1') \rightarrow (q', Z') \rightarrow (q_1', Z_1') \rightarrow (q_2, Z_2') \rightarrow (q_f, Z_f) \]
The problem with simulation & soundness

$(q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \rightarrow (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \rightarrow (q_1, Z_2) \xrightarrow{\text{pop}_b} (q_1, Z_1) \xrightarrow{\text{pop}_a} (q_0, Z_0)$
The solution: equivalence leads to soundness

- Thus one solution is to require equivalence between roots.
  - $(q, Z) \sim_q (q', Z')$ if $(q, Z) \preceq_q (q', Z') \land (q', Z') \succeq_q (q, Z)$
The solution: equivalence leads to soundess

- Thus one solution is to require equivalence between roots.
  - \((q, Z) \sim_q (q, Z')\) if \((q, Z) \preceq_q (q, Z') \land (q, Z') \succeq_q (q, Z)\)
- But does this give finiteness of exploration?

Finiteness of simulation is not enough.

We need that in any infinite sequence of nodes there must be an equivalent pair—This is called Strongly finiteness!

Luckily, all standard simulations, e.g., LU-abstraction, are strongly finite!

An aside: Does there exist a relation that is finite but not strongly finite?

Upshot
Equivalence among root nodes, and simulation among nodes within tree, gives a sound, complete and terminating procedure.
The solution: equivalence leads to soundness

- Thus one solution is to require equivalence between roots.
  - \((q, Z) \sim_q (q, Z')\) if \((q, Z) \preceq_q (q, Z') \land (q, Z') \succeq_q (q, Z)\)
  - But does this give finiteness of exploration? Finiteness of simulation is not enough.
The solution: equivalence leads to soundness

- Thus one solution is to require equivalence between roots.
  - \((q, Z) \sim_q (q', Z')\) if \((q, Z) \preceq_q (q', Z') \land (q', Z') \succeq_q (q, Z)\)
  - But does this give finiteness of exploration? Finiteness of simulation is not enough.
  - We need that in any infinite sequence of nodes there must be an equivalent pair
The solution: equivalence leads to soundness

- Thus one solution is to require equivalence between roots.
  - \((q, Z) \sim (q, Z')\) if \((q, Z) \leq (q, Z') \land (q, Z') \geq (q, Z)\)
  - But does this give finiteness of exploration? Finiteness of simulation is not enough.
  - We need that in any infinite sequence of nodes there must be an equivalent pair
    - This is called 

S. Akshay, IIT Bombay Can Zones be used for Reachability in PDTA? TicTacToe 2022
The solution: equivalence leads to soundness

- Thus one solution is to require *equivalence* between roots.
  - \((q, Z) \sim (q, Z')\) if \((q, Z) \preceq (q, Z') \land (q, Z') \succeq (q, Z)\)
  - But does this give finiteness of exploration? Finiteness of simulation is not enough.
  - We need that in any infinite sequence of nodes there must be an equivalent pair
    - This is called **Strongly finiteness!**

- Luckily, all standard simulations, e.g., LU-abstraction, are strongly finite!
Thus one solution is to require equivalence between roots.

- \((q, Z) \sim_q (q', Z')\) if \((q, Z) \preceq_q (q, Z') \land (q, Z') \succeq_q (q, Z)\)
- But does this give finiteness of exploration? Finiteness of simulation is not enough.
- We need that in any infinite sequence of nodes there must be an equivalent pair
  - This is called Strongly finiteness!

- Luckily, all standard simulations, e.g., LU-abstraction, are strongly finite!

- An aside: Does there exist a relation that is finite but not strongly finite?
The solution: equivalence leads to soundness

- Thus one solution is to require equivalence between roots.
  - \((q, Z) \sim_q (q, Z')\) if \((q, Z) \preceq_q (q, Z') \land (q, Z') \succeq_q (q, Z)\)
- But does this give finiteness of exploration? Finiteness of simulation is not enough.
- We need that in any infinite sequence of nodes there must be an equivalent pair
  - This is called **Strongly finiteness!**
- Luckily, all standard simulations, e.g., LU-abstraction, are strongly finite!

- An aside: Does there exist a relation that is finite but not strongly finite?

**Upshot**

Equivalence among root nodes, and simulation among nodes within tree, gives a sound, complete and terminating procedure.
Rules for PDTA to regain finiteness

\[ \mathcal{S} := \{(q_0, Z_0)\}, \quad S_{(q_0, Z_0)} := \{(q_0, Z_0)\} \]
Rules for PDTA to regain finiteness

\[
\mathcal{G} := \{(q_0, Z_0)\}, \quad S(q_0, Z_0) := \{(q_0, Z_0)\}
\]

Start

\[
(q, Z) \in \mathcal{G} \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset
\]

Internal

\[
S(q, Z) := S(q, Z) \cup \{(q'', Z'')\}, \text{ unless } \exists (q'', Z''') \in S(q, Z), Z'' \preceq q'' Z'''
\]

\[
(q, Z) \in \mathcal{G} \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{push}, R} q'' \quad Z'' = R(g \land Z') \sim q'' Z_1
\]

\[
(q'', Z_1) \in \mathcal{G} \quad (q_1, Z_1') \in S(q'', Z_1) \quad q' \xrightarrow{g_1, \text{pop}, R_1} q_2 \quad Z_2 = R_1(g_1 \land Z_1') \neq \emptyset
\]

Pop

\[
S(q, Z) := S(q, Z) \cup \{(q_2, Z_2)\}, \text{ unless } \exists (q_2, Z'_2) \in S(q, Z), Z_2 \preceq q_2 Z'_2
\]

Main Theorem

This set of rules is sound, complete & terminating for well-nested control-state reachability in PDTA.
Rules for PDTA to regain finiteness

\[ \mathcal{S} := \{(q_0, Z_0)\}, \quad S_{(q_0, Z_0)} := \{(q_0, Z_0)\} \]

\[
\text{Start}
\]

\[
(q, Z) \in \mathcal{S} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset
\]

\[
S_{(q, Z)} := S_{(q, Z)} \cup \{(q'', Z'')\}, \text{ unless } \exists (q'', Z''') \in S_{(q, Z)}, Z'' \preceq q'' Z'''
\]

\[
(q, Z) \in \mathcal{S} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{push}_{a}, R} q'' \quad Z'' = R(g \land Z') \sim q'' Z_1
\]

\[
(q'', Z_1) \in \mathcal{S} \quad (q'_1, Z'_1) \in S_{(q'', Z_1)} \quad q'_1 \xrightarrow{g_1, \text{pop}_{a}, R_1} q_2 \quad Z_2 = R_1(g_1 \land Z'_1) \neq \emptyset
\]

\[
S_{(q, Z)} := S_{(q, Z)} \cup \{(q_2, Z_2)\}, \text{ unless } \exists (q_2, Z'_2) \in S_{(q, Z)}, Z_2 \preceq q_2 Z'_2
\]

\[
(q, Z) \in \mathcal{S} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{push}_{a}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset
\]

\[
\mathcal{S} := \mathcal{S} \cup \{(q'', Z'')\}, \quad S_{(q'', Z'')} := \{(q'', Z'')\}, \text{ unless } \exists (q'', Z''') \in \mathcal{S}, Z'' \sim q'' Z'''
\]

---

Main Theorem

This set of rules is sound, complete & terminating for well-nested control-state reachability in PDTA.
Rules for PDTA to regain finiteness

\[ \mathcal{S} := \{(q_0, Z_0)\}, \quad S_{(q_0, Z_0)} := \{(q_0, Z_0)\} \]

\[ (q, Z) \in \mathcal{S} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = \overrightarrow{R(g \land Z')} \neq \emptyset \]

\[ S_{(q, Z)} := S_{(q, Z)} \cup \{(q'', Z'')\}, \text{ unless } \exists (q'', Z''') \in S_{(q, Z)}, \quad Z'' \preceq q'' Z''' \]

\[ (q, Z) \in \mathcal{S} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{push}, R} q'' \quad Z'' = \overrightarrow{R(g \land Z')} \sim_{q''} Z_1 \]

\[ (q'', Z_1) \in \mathcal{S} \quad (q'_1, Z'_1) \in S_{(q'', Z_1)} \quad q'_1 \xrightarrow{g_1, \text{pop}, R_1} q_2 \quad Z_2 = \overrightarrow{R_1(g_1 \land Z'_1)} \neq \emptyset \]

\[ S_{(q, Z)} := S_{(q, Z)} \cup \{(q_2, Z_2)\}, \text{ unless } \exists (q_2, Z''') \in S_{(q, Z)}, \quad Z_2 \preceq_{q_2} Z' \]

\[ (q, Z) \in \mathcal{S} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{push}, R} q'' \quad Z'' = \overrightarrow{R(g \land Z')} \neq \emptyset \]

\[ \mathcal{S} := \mathcal{S} \cup \{(q'', Z'')\}, \quad S_{(q'', Z'')} = \{(q'', Z'')\}, \text{ unless } \exists (q'', Z''') \in \mathcal{S}, \quad Z'' \sim_{q''} Z''' \]

Main Theorem

This set of rules is sound, complete & terminating for well-nested control-state reachability in PDTA.
Towards an efficient implementation

The rules give a fix pt saturation algorithm.

To implement it efficiently, we need to

1. Come up with a good data structure.
2. Decide on order of exploration
3. Avoid/reduce revisiting explored nodes. (see paper)
Towards an efficient implementation

The rules give a fix pt saturation algorithm.

To implement it efficiently, we need to

1. Come up with a good data structure.
2. Decide on order of exploration
3. Avoid/reduce revisiting explored nodes. (see paper)

For the data structure, we use two level hash tables

1. First level for roots
2. Second level for the set of nodes explored from each root
Towards an efficient implementation

Storing Roots

Storing graph of each root
Experiments and comparison

- Implemented tool\(^1\) on top of the Open Source tool TChecker.

\(^1\)https://github.com/karthik-314/PDTA_Reachability.git
Experiments and comparison

- Implemented tool\(^1\) on top of the Open Source tool TChecker.
- Tried two ways of pruning
  - Simulation within trees and equivalence across roots.
  - Equivalence everywhere
- Also compared region based approach from [AGKS17]

\(^1\)https://github.com/karthik-314/PDTA_Reachability.git
Experiments and comparison

- Implemented tool\(^1\) on top of the Open Source tool TChecker.
- Tried two ways of pruning
  - Simulation within trees and equivalence across roots.
  - Equivalence everywhere
- Also compared region based approach from [AGKS17]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>(\leq \text{LU})</th>
<th>(\leq \text{LU})</th>
<th>(\sim \text{LU})</th>
<th>(\sim \text{LU})</th>
<th>Region</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td># nodes</td>
<td>Time</td>
<td># nodes</td>
<td>Time</td>
<td># nodes</td>
</tr>
<tr>
<td>(B_1)</td>
<td>0.2</td>
<td>17</td>
<td>0.2</td>
<td>17</td>
<td>235.6</td>
<td>4100</td>
</tr>
<tr>
<td>(B_2)</td>
<td>20.0</td>
<td>5252</td>
<td>20.7</td>
<td>5252</td>
<td>T.O.</td>
<td>(\geq 154700)</td>
</tr>
<tr>
<td>(B_3)</td>
<td>0.2</td>
<td>6</td>
<td>0.2</td>
<td>6</td>
<td>2212</td>
<td>14300</td>
</tr>
<tr>
<td>(B_4(100, 10))</td>
<td>0.8</td>
<td>202</td>
<td>5.4</td>
<td>2212</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>(B_4(100, 1000))</td>
<td>0.7</td>
<td>202</td>
<td>3564.3</td>
<td>201202</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>(B_4(5000, 100))</td>
<td>23.2</td>
<td>10002</td>
<td>3429.3</td>
<td>1010102</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>(B_5)</td>
<td>38.2</td>
<td>3006</td>
<td>501.0</td>
<td>34799</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Time in ms, some benchmarks were custom-crafted, others from prior papers, \(B_5\) had open guards. \(B_4\) was a parametrized example, where first component relates to size of PDTA, second to clock constraints.

\(^1\)https://github.com/karthik-314/PDTA_Reachability.git
Experiments and comparison

- Implemented tool\(^1\) on top of the Open Source tool TChecker.
- Tried two ways of pruning
  - Simulation within trees and equivalence across roots.
  - Equivalence everywhere
- Also compared region based approach from [AGKS17]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>(\leq_{LU}) Time</th>
<th>(\leq_{LU}) # nodes</th>
<th>(~_{LU}) Time</th>
<th>(~_{LU}) # nodes</th>
<th>Region Time</th>
<th>Region # nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B_1)</td>
<td>0.2</td>
<td>17</td>
<td>0.2</td>
<td>17</td>
<td>235.6</td>
<td>4100</td>
</tr>
<tr>
<td>(B_2)</td>
<td>20.0</td>
<td>5252</td>
<td>20.7</td>
<td>5252</td>
<td>T.O.</td>
<td>(\geq 154700)</td>
</tr>
<tr>
<td>(B_3)</td>
<td>0.2</td>
<td>6</td>
<td>0.2</td>
<td>6</td>
<td>1043.8</td>
<td>14300</td>
</tr>
<tr>
<td>(B_4(100, 10))</td>
<td>0.8</td>
<td>202</td>
<td>5.4</td>
<td>2212</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>(B_4(100, 1000))</td>
<td>0.7</td>
<td>202</td>
<td>3564.3</td>
<td>201202</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>(B_4(5000, 100))</td>
<td>23.2</td>
<td>10002</td>
<td>3429.3</td>
<td>1010102</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>(B_5)</td>
<td>38.2</td>
<td>3006</td>
<td>501.0</td>
<td>34799</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Time in ms, some benchmarks were custom-crafted, others from prior papers, \(B_5\) had open guards. \(B_4\) was a parametrized example, where first component relates to size of PDTA, second to clock constraints.

Simulation-based Zone algorithm was always as good and often much better.

\(^1\)https://github.com/karthik-314/PDTA_Reachability.git
Experiments and comparison

- Implemented tool\(^1\) on top of the Open Source tool TChecker.
- Tried two ways of pruning
  - Simulation within trees and equivalence across roots.
  - Equivalence everywhere
- Also compared region based approach from [AGKS17]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$\leq_{LU}$ Time</th>
<th>$\leq_{LU}$ # nodes</th>
<th>$\sim_{LU}$ Time</th>
<th>$\sim_{LU}$ # nodes</th>
<th>Region Time</th>
<th>Region # nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>0.2</td>
<td>17</td>
<td>0.2</td>
<td>17</td>
<td>235.6</td>
<td>4100</td>
</tr>
<tr>
<td>$B_2$</td>
<td>20.0</td>
<td>5252</td>
<td>20.7</td>
<td>5252</td>
<td>T.O.</td>
<td>$\geq 154700$</td>
</tr>
<tr>
<td>$B_3$</td>
<td>0.2</td>
<td>6</td>
<td>0.2</td>
<td>6</td>
<td>1043.8</td>
<td>14300</td>
</tr>
<tr>
<td>$B_4(100, 10)$</td>
<td>0.8</td>
<td>202</td>
<td>5.4</td>
<td>2212</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>$B_4(100, 1000)$</td>
<td>0.7</td>
<td>202</td>
<td>3564.3</td>
<td>201202</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>$B_5$</td>
<td>23.2</td>
<td>10002</td>
<td>3429.3</td>
<td>1010102</td>
<td>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>$B_4(5000, 100)$</td>
<td>38.2</td>
<td>3006</td>
<td>501.0</td>
<td>34799</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Time in ms, some benchmarks were custom-crafted, others from prior papers, $B_5$ had open guards. $B_4$ was a parametrized example, where first component relates to size of PDTA, second to clock constraints.

Simulation-based Zone algorithm was always as good and often much better.

\(^1\)https://github.com/karthik-314/PDTA_Reachability.git
Simulations can prune branches but equivalences are good for roots!
Simulations can prune branches but equivalences are good for roots!

A few concluding remarks
Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

- Lifts from well-nested reachability to general reachability
Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

1. Lifts from well-nested reachability to general reachability
2. Works with any strongly finite simulation... not just LU-abstraction.
   - Using so-called $G$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

1. Lifts from well-nested reachability to general reachability
2. Works with any strongly finite simulation... not just LU-abstraction.
   - Using so-called $G$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
   - Other simulations and extrapolations [BBLP06, HSW12]
Simulations can **prune branches** but **equivalences are good for roots!**

### A few concluding remarks

1. Lifts from well-nested reachability to general reachability
2. Works with any strongly finite simulation... not just LU-abstraction.
   - Using so-called $G$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblance to Liveness in TA [Tri09, LOD$^+$13, HSTW20].
Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

1. Lifts from well-nested reachability to general reachability
2. Works with any strongly finite simulation... not just LU-abstraction.
   - Using so-called $G$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblance to Liveness in TA [Tri09, LOD$^+$13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.
Simulations can **prune branches** but equivalences are **good for roots!**

A few concluding remarks

1. Lifts from well-nested reachability to general reachability
2. Works with any strongly finite simulation... not just LU-abstraction.
   - Using so-called $G$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblance to Liveness in TA [Tri09, LOD$^+$13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.

Even more future work

- Can we extend the algorithm to handle ages on the stack?
Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

1. Lifts from well-nested reachability to general reachability

2. Works with any strongly finite simulation... not just LU-abstraction.
   - Using so-called $G$-abstraction [GMS19] will allow handling diagonal guards in PDTA.

   - Other simulations and extrapolations [BBLP06, HSW12]

4. Interesting link and resemblance to Liveness in TA [Tri09, LOD+13, HSTW20].

5. Witness generation not completely obvious, due to fix pt computation, but can be done.

Even more future work

- Can we extend the algorithm to handle ages on the stack?
- Can we get some real benchmarks, e.g., Boolean programs with timers?
Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

1. Lifts from well-nested reachability to general reachability
2. Works with any strongly finite simulation... not just LU-abstraction.
   - Using so-called $G$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblance to Liveness in TA [Tri09, LOD$^+$13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.

Even more future work

- Can we extend the algorithm to handle ages on the stack?
- Can we get some real benchmarks, e.g., Boolean programs with timers?

– Thanks!
Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman.
Dense-timed pushdown automata.

Rajeev Alur and David L. Dill.
Automata for modeling real-time systems.

Timed systems through the lens of logic.

S. Akshay, Paul Gastin, and Shankara Narayanan Krishna.
Analyzing Timed Systems Using Tree Automata.
LMCS, Volume 14, Issue 2, 2018.

Revisiting underapproximate reachability for multipushdown systems.

S. Akshay, Paul Gastin, Shankara Narayanan Krishna, and Ilias Sarkar.
Towards an efficient tree automata based technique for timed systems.

Gerd Behrmann, Patricia Bouyer, Kim G Larsen, and Radek Pelánek.
Lower and upper bounds in zone-based abstractions of timed automata.

Uppaal 4.0.
2006.
Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana.
On the automatic verification of systems with continuous variables and unbounded discrete data structures.

Uppaal¡xa tool suite for automatic verification of real-time systems.

Lorenzo Clemente and Slawomir Lasota.
Timed pushdown automata revisited.

Lorenzo Clemente and Slawomir Lasota.
Reachability relations of timed pushdown automata.

Lorenzo Clemente, Slawomir Lasota, Ranko Lazic, and Filip Mazowiecki.
Timed pushdown automata and branching vector addition systems.

Paul Gastin, Sayan Mukherjee, and B. Srivathsan.
Fast algorithms for handling diagonal constraints in timed automata.

Frédéric Herbreteau and Gerald Point.
Tchecker.
Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz.
Why liveness for timed automata is hard, and what we can do about it.

Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz.
Better abstractions for timed automata.

Alfons Laarman, Mads Chr. Olesen, Andreas Engelbrecht Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol.
Multi-core emptiness checking of timed büchi automata using inclusion abstraction.

Kim G Larsen, Paul Pettersson, and Wang Yi.
Uppaal in a nutshell.

Paul Pettersson and Kim G Larsen.
Uppaal2k.

Stavros Tripakis.
Checking timed büchi automata emptiness on simulation graphs.