Efficient Algorithms 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

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

SNR @ Confest Sept 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 (Previous talk!)

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 (Previous talk!)

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?
A natural extension combining Time and Recursion

- Introduced in [BER94], just after Timed automata [AD90].
- PDTA = Timed automata + (pushdown) stack!
Pushdown timed automata (PDTA)

A natural extension combining Time and Recursion

- Introduced in [BER94], just after Timed automata [AD90].
- PDTA = Timed automata + (pushdown) stack!

Many theoretical results, variants and extensions

- For instance, [TW10, AAS12, CL15, AGK18, CLLM17, AGJK19, CL21]
Pushdown timed automata (PDTA)

A natural extension combining Time and Recursion
- Introduced in [BER94], just after Timed automata [AD90].
- PDTA = Timed automata + (pushdown) stack!

Many theoretical results, variants and extensions
- For instance, [TW10, AAS12, CL15, AGK18, CLLM17, AGJK19, CL21]
- But very few implementations: [AGKS17, AGKR20].
A natural extension combining Time and Recursion
- Introduced in [BER94], just after Timed automata [AD90].
- PDTA = Timed automata + (pushdown) stack!

Many theoretical results, variants and extensions
- For instance, [TW10, 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.

\[
q_0 \xrightarrow{y \leq 2, \text{push}_a} q_1 \xrightarrow{x \geq 1, \{x\}} q_2 \xrightarrow{\text{pop}_a} q_3
\]
Our problem statement

is 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?
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.
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.

Prototype implementation built on Open source tool, TChecker.
Challenges in implementing the above algorithm.
Experimental results and comparisons.
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.
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, \( Z_0 \)
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

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

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

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

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

- Zone graph is defined on nodes, i.e., \((\text{state, Zone})\) pairs

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

- Reachability using Zone graph construction is sound, and complete, but non-terminating.
Recall: Getting a finite Zone graph using simulations
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 \(q_0\)).
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 \(q_0\)).

\[
(q_0, Z_2) \preceq_{q_0} (q_0, Z_1) \quad \preceq_{q_0} (q_0, Z_1)
\]

Don’t explore
Recall: Getting a finite Zone graph using simulations

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

**Strongly Finite Simulation**

- \((q_0, Z_2) \preceq_{q_0} (q_0, Z_1)\) (Behaviour of \(Z_2\) captured by \(Z_1\) at \(q_0\)).
- In 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 \((q_i, Z_i) \preceq_{q_i} (q_j, Z_j), (q_j, Z_j) \preceq_{q_i} (q_i, Z_i)\)
Recall: Getting a finite Zone graph using simulations

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

\[(q_0, Z_3) \rightarrow (q_1, Z_3) \rightarrow (q_2, Z_3) \rightarrow \ldots \]

Strongly Finite Simulation

- \((q_0, Z_2) \preceq_{q_0} (q_0, Z_1)\) (Behaviour of \(Z_2\) captured by \(Z_1\) at \(q_0\)).
- In 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 \((q_i, Z_i) \preceq_{q_i} (q_j, Z_j), (q_j, Z_j) \preceq_{q_i} (q_i, Z_i)\)

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

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

Strongly Finite Simulation

- \( (q_0, Z_2) \preceq_{q_0} (q_0, Z_1) \) (Behaviour of \( Z_2 \) captured by \( Z_1 \) at \( q_0 \)).
- In 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 \( (q_i, Z_i) \preceq_{q_i} (q_j, Z_j), (q_j, Z_j) \preceq_{q_j} (q_i, Z_i) \)

Strongly finite simulations guarantee finite zone graph preserving soundness, completeness!

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

Strongly Finite Simulation

- \((q_0, Z_2) \preceq_{q_0} (q_0, Z_1)\) (Behaviour of \(Z_2\) captured by \(Z_1\) at \(q_0\)).
- In 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 \((q_i, Z_i) \preceq_{q_i} (q_j, Z_j), (q_j, Z_j) \preceq_{q_j} (q_i, Z_i)\)

Strongly finite simulations guarantee finite zone graph preserving soundness, completeness!

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

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

\[
\begin{array}{c}
(q_0, Z_0) \\
(q_1, Z_0) \\
(q_1, (y - x = 1)) \\
\end{array}
\]

\[
\begin{array}{c}
(x = 1, \{x\}) \\
(q_0, Z_0) \\
(q_1, Z_0) \\
(q_1, (y - x = 1)) \\
\end{array}
\]
Recall: Getting a finite Zone graph using simulations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[ \xrightarrow{(x = 1, \{x\})} \]
Recall: Getting a finite Zone graph using simulations

Modify the re-write rule based saturation algorithm

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

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

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

\[ \text{Trans} \]
Recall: Getting a finite Zone graph using simulations

Modify the re-write rule based saturation algorithm

\[
S := \{ (q_0, Z_0) \} \quad \text{(start)}
\]
\[
(q, Z) \in S \quad q \xrightarrow{g \in R} q' \quad Z' = R(g \land Z) \neq \emptyset
\]
\[
S := S \cup \{ (q', Z') \}, \quad \text{unless} \quad \exists (q', Z'') \in S, \quad Z' \preceq q' Z''
\]

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.

From TA to PDTA
From TA to PDTA

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).
From TA to PDTA

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 $S$. The initial state is $(q_0, Z_0)$. 

\[ (q_0, Z_0) \]
We start with the initial node and explore as before as long as we see internal transitions (no push-pop).
Viewing well-nested reachability in PDTA

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

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

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

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

\[
\text{Start}
\]

\[
\text{Internal}
\]
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 $\mathcal{G}$!

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

\[
\begin{array}{c}
(q, Z) \in \mathcal{G} \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = R(g \land Z') \neq \emptyset \\
S_{(q, Z)} := S_{(q, Z)} \cup \{(q'', Z'')\}
\end{array}
\]
Reachability rules for PDTA

\[
\begin{align*}
\mathcal{G} & := \{(q_0, Z_0)\}, \quad S(q_0, Z_0) := \{(q_0, Z_0)\} & \text{Start} \\
(q, Z) \in \mathcal{G} & \quad (q', Z') \in S(q, Z) & q' \xrightarrow{g, \text{nop}, R} q'' & Z'' = R(g \land Z') \neq \emptyset & \text{Internal} \\
S(q, Z) := S(q, Z) \cup \{(q'', Z'')\}. & \\
\end{align*}
\]

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

\[
\begin{align*}
(q, Z) \in \mathcal{G} & \quad (q', Z') \in S(q, Z) & q' \xrightarrow{g, \text{push}, R} q'' & Z'' = R(g \land Z') \neq \emptyset & \text{Push} \\
\mathcal{G} := \mathcal{G} \cup \{(q'', Z'')\}, \quad S(q'', Z'') = \{(q'', Z'')\}. & \\
\end{align*}
\]
Reachability rules for PDTA

Start

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

\( (q, Z) \in S \) \hspace{1cm} \( (q', Z') \in S(q, Z) \) \hspace{1cm} \( q' \xrightarrow{\text{nop}, R} q'' \) \hspace{1cm} \( Z'' = R(g \land Z') \neq \emptyset \)

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

Internal

\( (q, Z) \in S \) \hspace{1cm} \( (q', Z') \in S(q, Z) \) \hspace{1cm} \( q' \xrightarrow{\text{push}_a, R} q'' \) \hspace{1cm} \( Z'' = R(g \land Z') \neq \emptyset \)

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

Push

\( (q, Z) \in S \) \hspace{1cm} \( (q', Z') \in S(q, Z) \) \hspace{1cm} \( q' \xrightarrow{g, \text{push}_a, R} q'' \) \hspace{1cm} \( Z'' = R(g \land Z') \neq \emptyset \)

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

Pop

\( (q, Z) \in S \) \hspace{1cm} \( (q', Z') \in S(q, Z) \) \hspace{1cm} \( q' \xrightarrow{g_1, \text{push}_a, R_1} q'' \) \hspace{1cm} \( Z'' = R(g_1 \land Z') \neq \emptyset \)

\[ (q'', Z'') \in S \] \hspace{1cm} \( (q_1', Z_1') \in S(q'', Z'') \) \hspace{1cm} \( q_1' \xrightarrow{g_1', \text{pop}_a, R_1} q_2 \) \hspace{1cm} \( Z_2 = R_1(g_1' \land Z_1') \neq \emptyset \)

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

Finally, when we see pop, we continue exploring tree where corresponding push happened.
Reachability rules for PDTA

\[ \begin{align*}
\emptyset & := \{(q_0, Z_0)\}, \quad S(q_0, Z_0) := \{(q_0, Z_0)\} \\
(q, Z) \in \emptyset & \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{nop}, R} q'' \quad Z'' = R(g \wedge Z') \neq \emptyset \quad \text{Start} \\
S(q, Z) & := S(q, Z) \cup \{(q'', Z'')\}, \\
(q, Z) \in \emptyset & \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{push}, a, R} q'' \quad Z'' = R(g \wedge Z') \neq \emptyset \quad \text{Internal} \\
S(q, Z) & := S(q, Z) \cup \{(q'', Z'')\}, \\
(q, Z) \in \emptyset & \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g, \text{push}, a, R} q'' \quad Z'' = R(g \wedge Z') \neq \emptyset \quad \text{Push} \\
S(q, Z) & := S(q, Z) \cup \{(q'', Z'')\}, \\
(q', Z') \in \emptyset & \quad (q'_1, Z'_1) \in S(q', Z') \quad q'_1 \xrightarrow{g_1, \text{pop}, R_1} q_2 \quad Z_2 = R_1(g_1 \wedge Z'_1) \neq \emptyset \quad \text{Pop} \\
S(q, Z) & := S(q, Z) \cup \{(q_2, Z_2)\}
\end{align*} \]

- 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

Let's consider the Zone graph with states $(q_0, Z_0)$, $(q_1, Z_1)$, and $(q_2, Z_2)$. We apply transitions with labels "push_a" and "push_b".

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

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

- 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

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

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

\[
(q', Z') \triangleleft (q'_1, Z'_1)
\]
The problem with simulation & soundness

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

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

\[ \cdots \text{push}_a (q, Z) \cdots \text{pop}_a (q_f, Z_f) \cdots \text{push}_b (q_1, Z_1) \cdots \text{pop}_b (q_2, Z_2) \cdots \]

\[ \cdots \text{push}_a (q, Z) \rightarrow (q', Z') \cdots (q_1', Z_1') \cdots \text{push}_b (q_1, Z_2) \cdots \]

\[ \cdots \text{pop}_b (q_2', Z_2') \cdots \text{pop}_a (q_f', Z_f') \cdots \]
The problem with simulation & soundness

\[ (q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \rightarrow (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \xrightarrow{\text{pop}_a} (q_1', Z_1') \xrightarrow{\text{pop}_b} (q_2', Z_2') \xrightarrow{\text{pop}_b} (q_1', Z_2') \xrightarrow{\text{pop}_a} (q_f, Z_f) \]
The problem with simulation & soundness

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

Not Sound!

\[(q_1, Z_1) \rightarrow (q_1', Z_1') \xrightarrow{\text{pop}_b} (q_2', Z_2') \xrightarrow{\text{pop}_a} (q_f, Z_f)\]
The problem with simulation & soundness

\[(q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \rightarrow (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \not\rightarrow (q_1, Z_2) \xrightarrow{\text{pop}_b} (q'_1, Z'_1) \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[(q_0, Z_0) \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \rightarrow (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \not\rightarrow (q_1, Z_2) \xrightarrow{\text{pop}_b} (q'_1, Z'_1) \]

So how do we fix it?

\[(q_1, Z_1) \rightarrow (q'_1, Z'_1) \xrightarrow{\text{pop}_b} (q'_2, Z'_2) \xrightarrow{\text{pop}_a} (q_f, Z_f)\]
The problem with simulation & soundness

Use equivalence!
The problem with simulation & soundness

Use equivalence!
The problem with simulation & soundness

Use equivalence!

\[
\begin{align*}
(q_0, Z_0) & \rightarrow (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \rightarrow (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \rightarrow (q'_1, Z''_1) \\
| & | | & | & | & | & | & | \\
& & & & & & & & \\
& (q_1, Z_1) & \xrightarrow{\gamma} & (q_1, Z_2) & \xrightarrow{\gamma} & (q'_1, Z''_1) \\
& & & & & & & & \\
& & & & & & & & \\
& (q_2, Z'_2) & \xrightarrow{\text{pop}_b} & (q'_1, Z'_1) & \xrightarrow{\text{pop}_a} & (q, Z) \\
& & & & & & & & \\
& (q_f, Z_f) & \xrightarrow{\text{pop}_b} & (q_1, Z_2) & \xrightarrow{\text{pop}_a} & (q_f, Z_f)
\end{align*}
\]
The problem with simulation & soundness

Use equivalence!

Thus,

- Checking equivalence to prune at roots gives a sound and complete procedure.
- The enumeration will terminate since the simulation is “strongly finite”.
Rules for PDTA to regain finiteness

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

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

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

Start

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

Internal

\( (q, Z) \in S \quad (q', Z') \in S_{(q, Z)} \quad q' \xrightarrow{g, push_a, R} q'' \quad Z'' = R(g \land Z') \sim_{q''} Z_1 \)

\( (q'', Z_1) \in S \quad (q'_1, Z'_1) \in S_{(q'', Z_1)} \quad q'_1 \xrightarrow{g_1, pop_a, R_1} q_2 \quad Z_2 = R_1(g_1 \land Z'_1) \neq \emptyset \)

Pop

\( S_{(q, Z)} := S_{(q, Z)} \cup \{(q_2, Z_2)\} \)

Push

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

\( S := S \cup \{(q'', Z'')\} \quad S_{(q'', Z'')} = \{(q'', Z'')\} \)
Rules for PDTA to regain finiteness

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

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

**Start**

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

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

**Internal**

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

\( S(q, Z) := S(q, Z) \cup \{(q_2, Z_2)\} \), unless \( \exists (q_2, Z'_2) \in S(q, Z), \quad Z_2 \preceq_{q_2} Z'_2 \)

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

**Push**

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

\( (q, Z) \in \mathcal{S} \quad (q', Z') \in S(q, Z) \quad q' \xrightarrow{g_1,\text{pop}_a,\text{R}_1} q_2 \quad Z_2 = R_1(g_1 \land Z'_1) \neq \emptyset \)

**Pop**

\( S(q, Z) := S(q, Z) \cup \{(q_2, Z_2)\} \), unless \( \exists (q_2, Z'_2) \in S(q, Z), \quad Z_2 \preceq_{q_2} Z'_2 \)

\( (q', Z') \in S(q, Z) \quad q' \xrightarrow{g_1,\text{pop}_a,\text{R}_1} q_2 \quad Z_2 = R_1(g_1 \land Z'_1) \neq \emptyset \)

**Internal**
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

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

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

Pop

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

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

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

Push

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

Main Theorem

This set of rules is sound, complete & terminating for well-nested control-state reachability in PDTA.
Implementation and Experiments

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

- The rules only give a fix pt saturation algorithm.
- To implement it efficiently, we needed to
  1. Come up with a good data structure.
  2. Decide on order of exploration.
  3. Avoid/reduce revisiting explored nodes.

---

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

---
Implementations and Experiments

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

- The rules only give a fix pt saturation algorithm.
- To implement it efficiently, we needed to
  1. Come up with a good data structure.
  2. Decide on order of exploration.
  3. Avoid/reduce revisiting explored nodes.

Comparisons

- Tried two ways of pruning
  - Simulation within trees and equivalence across roots.
  - Equivalence everywhere
- Region based approach from [AGKS17]

---

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

S. Akshay, IIT Bombay  Efficient Algorithms for Reachability in Pushdown Timed Automata  SNR@Confest Sept 2022
Implementation and Experiments

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

- The rules only give a fix pt saturation algorithm.
- To implement it efficiently, we needed to
  1. Come up with a good data structure.
  2. Decide on order of exploration.
  3. Avoid/reduce revisiting explored nodes.

Comparisons

- Tried two ways of pruning
  - Simulation within trees and equivalence across roots.
  - Equivalence everywhere
- Region based approach from [AGKS17]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>≤LU Time</th>
<th>≤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>≥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>OoM</td>
<td>OoM</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

S. Akshay, IIT Bombay  Efficient Algorithms for Reachability in Pushdown Timed Automata  SNR@Confest Sept 2022
Implementation and Experiments

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

- The rules only give a fix pt saturation algorithm.
- To implement it efficiently, we needed to
  1. Come up with a good data structure.
  2. Decide on order of exploration.
  3. Avoid/reduce revisiting explored nodes.

Comparisons

- Tried two ways of pruning
  - Simulation within trees and equivalence across roots.
  - Equivalence everywhere
- Region based approach from [AGKS17]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>≤LU Time</th>
<th>≤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>≥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)(1000, 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>OoM</td>
<td>OoM</td>
</tr>
<tr>
<td>B(_5)</td>
<td></td>
<td></td>
<td></td>
<td></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
Conclusion

A few concluding remarks
A few concluding remarks

1. Lifts from well-nested reachability to general reachability
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.
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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.

   - Simulation across contexts?
   - Other simulations and extrapolations [BBLP06, HSW12]?
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.
   - Simulation across contexts?
   - Other simulations and extrapolations [BBLP06, HSW12]?
4. Link to Liveness in TA [Tri09, LOD$^{+}$13, HSTW20].
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.
   - Simulation across contexts?
   - Other simulations and extrapolations [BBLP06, HSW12]?
4. Link to Liveness in TA [Tri09, LOD$^+$13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.
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.
   - Simulation across contexts?
   - Other simulations and extrapolations [BBLP06, HSW12]?
4. Link to Liveness in TA [Tri09, LOD\(^+\)13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.

Future work

- How to handle ages on the stack?
Conclusion

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.
   - Simulation across contexts?
   - Other simulations and extrapolations [BBLP06, HSW12]?
4. Link to Liveness in TA [Tri09, LOD$^+$13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.

Future work

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

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.
   - Simulation across contexts?
   - Other simulations and extrapolations [BBLP06, HSW12]?
4. Link to Liveness in TA [Tri09, LOD$^+$13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.

Future work

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

– Thanks!
References


References

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

Uppaal jxa 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.

S. Akshay, IIT Bombay
Efficient Algorithms for Reachability in Pushdown Timed Automata
SNR@Confest Sept 2022
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.

Ashutosh Trivedi and Dominik Wojtczak.
Recursive timed automata.