Can Zones be used for reachability in Pushdown Timed Automata?

S. Akshay

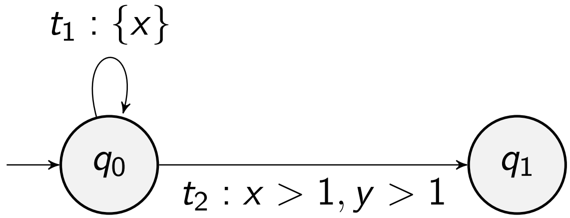
Dept of CSE, Indian Institute of Technology Bombay, India

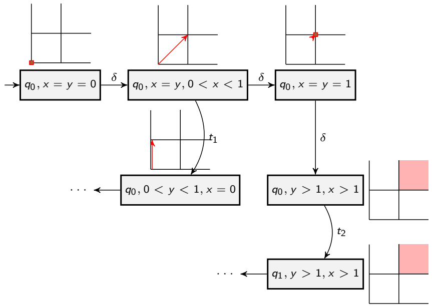
Joint work with Paul Gastin, Karthik R. Prakash
Based on work that appeared in CAV'21.

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

TickTac meeting March 2022

Modeling Timed Systems using Automata

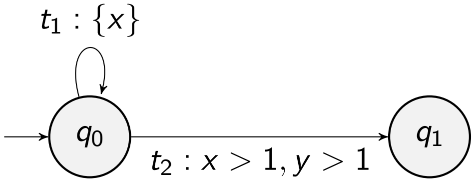


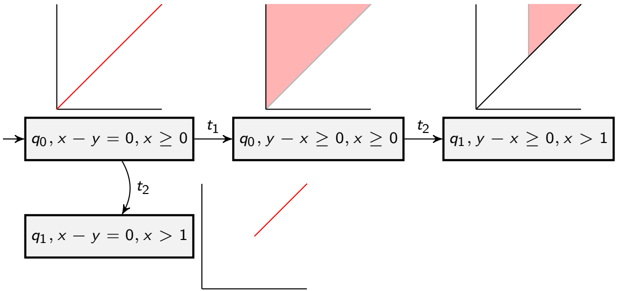


The timed automaton model

- 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

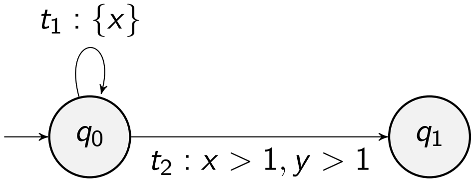


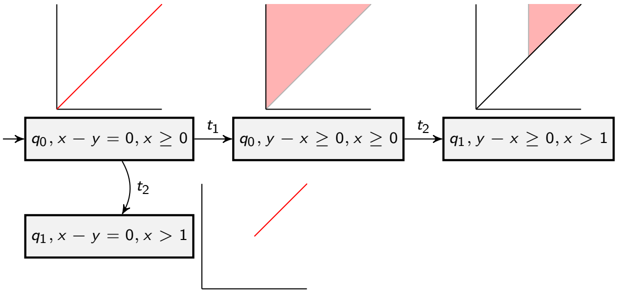


Zone based abstractions of Timed automata

- 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



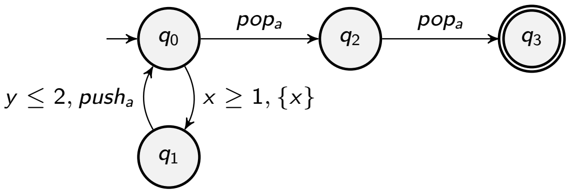


Zone based abstractions of Timed automata

- 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?

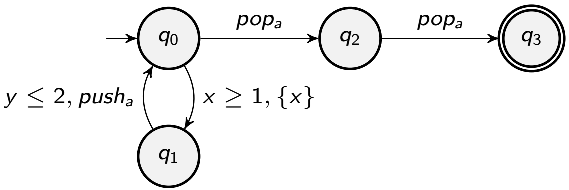
Pushdown timed automata (PDTA)



A natural extension combining Time and Recursion

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

Pushdown timed automata (PDTA)



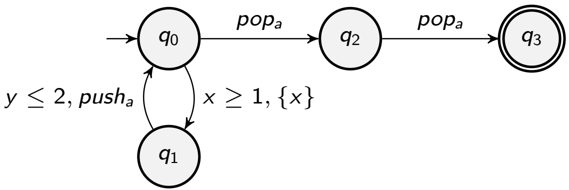
A natural extension combining Time and Recursion

- 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)



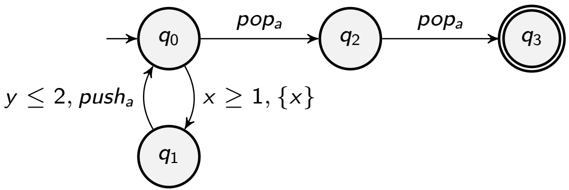
A natural extension combining Time and Recursion

- 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

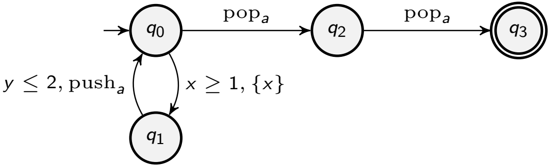
- Introduced in [BER94], just after Timed automata!
- PDTA = Timed automata + (pushdown) stack!

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?!

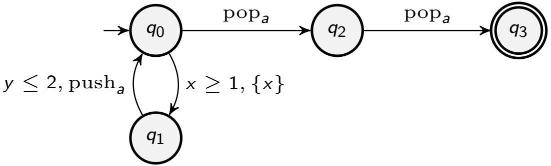
Our problem statement



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.

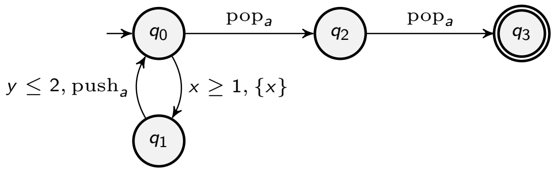
Our problem statement



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

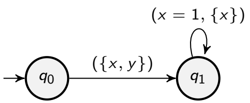
1. Re-look at zone algorithms for TA, using re-write rules.
   - Strategies to prune: Simulations and equivalences
2. Pinpointing the difficulty in lifting simulations to PDTA
   - Why using simulations naively in PDTA instead of TA is not sound.
3. Refining the rules - New Zone algorithms for PDTA-reach!
   - A saturation algorithm for well-nested control state reachability in PDTA.
4. Prototype implementation built on Open source tool, TChecker.

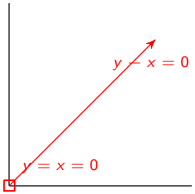
Outline of the talk

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.

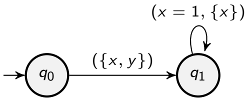
Outline of the talk

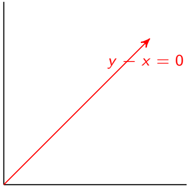
1. Re-look at zone algorithms for TA, using re-write rules.
   - Strategies to prune: Simulations and equivalences
2. Pinpointing the difficulty in lifting simulations to PDTA
   - Why using simulations naively in PDTA instead of TA is not sound.
3. Refining the rules - New Zone algorithms for PDTA-reach!
   - 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.





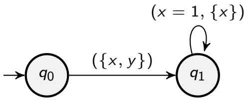
- Initial clock valuation: $(x = y = 0)$.
- Allowing time elapse: $(y - x = 0, x \geq 0)$
  - $\overrightarrow{(x = y = 0)} = (y - x = 0 \land x \geq 0)$ is the initial zone, Z_0

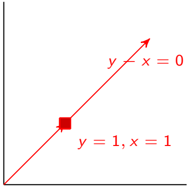




- Initial clock valuation: $(x = y = 0)$.
- Allowing time elapse: $(y - x = 0, x \geq 0)$
  - $\overrightarrow{(x = y = 0)} = (y - x = 0 \wedge x \geq 0)$ is the initial zone, Z_0
- From zone Z, when we fire transition $t = (g, R)$, we get

Recall: Zones in Timed automata



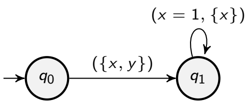


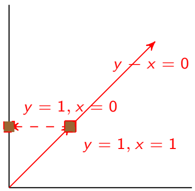
$$y - x = 0 \wedge x \geq 0 \wedge x = 1$$

- Initial clock valuation: $(x = y = 0)$.
- Allowing time elapse: $(y - x = 0, x \geq 0)$
  - $\overrightarrow{(x = y = 0)} = (y - x = 0 \wedge x \geq 0)$ is the initial zone, Z_0
- From zone Z, when we fire transition $t = (g, R)$, we get

$$Z \wedge g$$

Recall: Zones in Timed automata



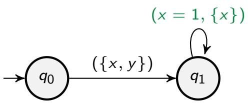


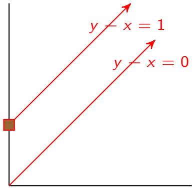
$[\{x\}](x = 1, y = 1)$

- Initial clock valuation: $(x = y = 0)$.
- Allowing time elapse: $(y - x = 0, x \geq 0)$
  - $\overrightarrow{(x = y = 0)} = (y - x = 0 \wedge x \geq 0)$ is the initial zone, Z_0
- From zone Z, when we fire transition $t = (g, R)$, we get

$$[R](Z \wedge g)$$

Recall: Zones in Timed automata



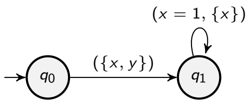


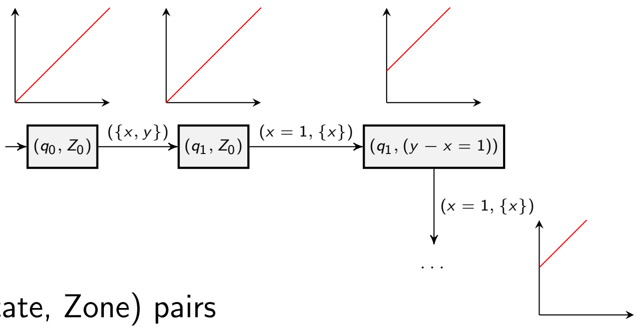
$\overrightarrow{(y = 1, x = 0)}$

- Initial clock valuation: $(x = y = 0)$.
- Allowing time elapse: $(y - x = 0, x \geq 0)$
  - $\overrightarrow{(x = y = 0)} = (y - x = 0 \wedge x \geq 0)$ is the initial zone, Z_0
- From zone Z, when we fire transition $t = (g, R)$, we get

$$Z' = \overrightarrow{[R](Z \wedge g)}$$

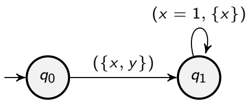
Recall: Zone based Reachability in Timed Automata

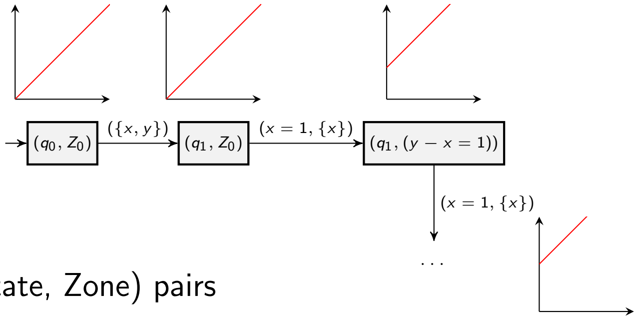




- Zone graph is defined on nodes, i.e., (state, Zone) pairs

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





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

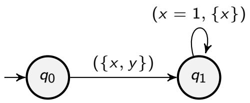
$$(q, Z) \xrightarrow{t} (q', Z') \text{ if } t = (q, g, R, q'), Z' = \overrightarrow{[R](Z \wedge g)}$$

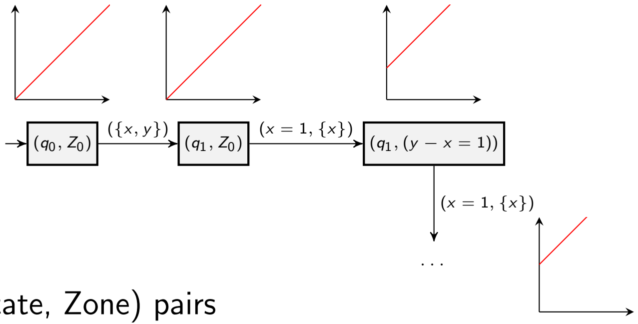
First re-look: We view this as a fix pt computation

$$\frac{}{S := \{(q_0, Z_0)\}}\ \text{start}$$

$$\frac{(q, Z) \in S \qquad q \xrightarrow{g,R} q' \qquad Z' = \overrightarrow{R(g \wedge Z)} \neq \emptyset}{S := S \cup \{(q', Z')\}}\ \text{Trans}$$

Recall: Zone based Reachability in Timed Automata



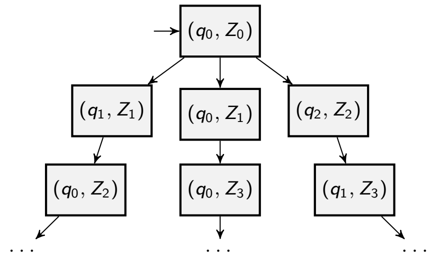


- Zone graph is defined on nodes, i.e., (state, Zone) pairs

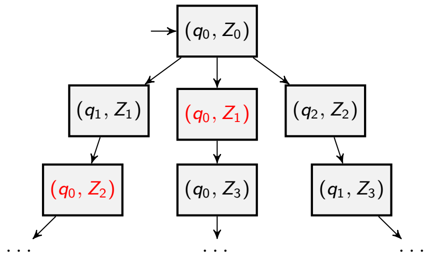
$$(q, Z) \xrightarrow{t} (q', Z') \text{ if } t = (q, g, R, q'), Z' = \overrightarrow{[R](Z \wedge g)}$$

- Reachability using Zone graph construction is sound, and complete, but non-terminating.

Recall: Getting a finite Zone graph using simulations



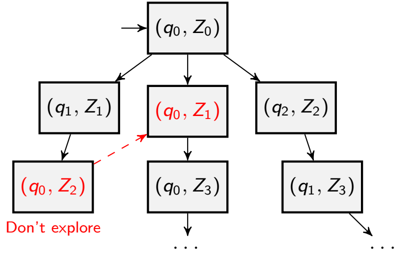
Recall: Getting a finite Zone graph using simulations



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

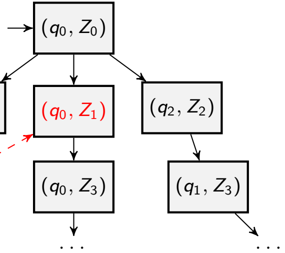


Simulation

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

$$\begin{array}{ccc}
(q_0, Z_2) & \preceq_{q_0} & (q_0, Z_1) \\
{*}\downarrow & & {*}\downarrow \\
(q_n, Z_n) & \preceq_{q_n} & (q_n, Z_n')
\end{array}$$

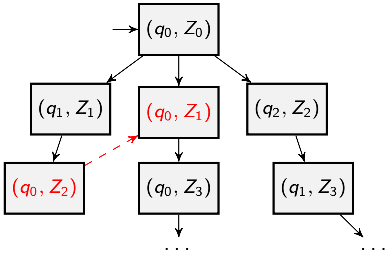
Recall: Getting a finite Zone graph using simulations

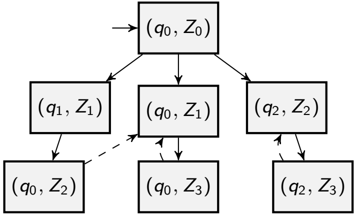


Finite Simulation

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



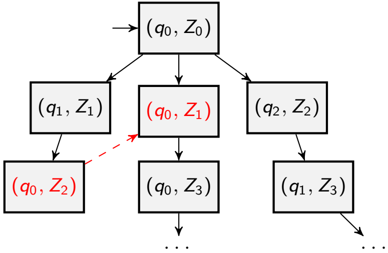


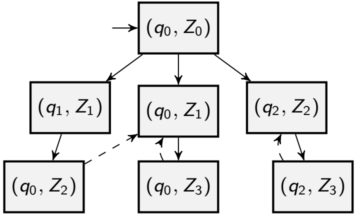
Finite Simulation

- $(q_0, Z_2) \preceq_{q_0} (q_0, Z_1)$ (Behaviour of Z_2 captured by Z_1 at q_0).
- For any infinite sequence of nodes $(q_0, Z_0), (q_1, Z_1), \cdots$, 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!

Recall: Getting a finite Zone graph using simulations





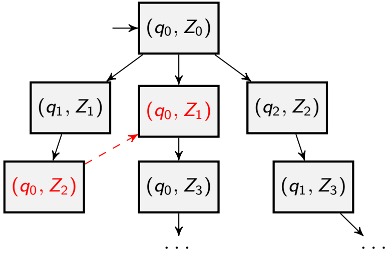
Finite Simulation

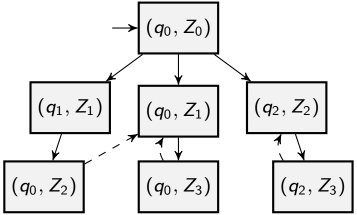
- $(q_0, Z_2) \preceq_{q_0} (q_0, Z_1)$ (Behaviour of Z_2 captured by Z_1 at q_0).
- For any infinite sequence of nodes $(q_0, Z_0), (q_1, Z_1), \cdots$, there must exist $j < i$, s.t., $q_i = q_j$ and $(q_i, Z_i) \preceq_{q_i} (q_j, Z_j)$

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





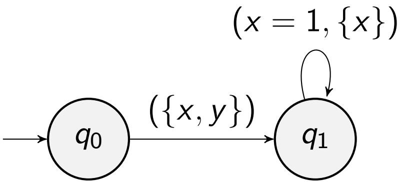
Finite Simulation

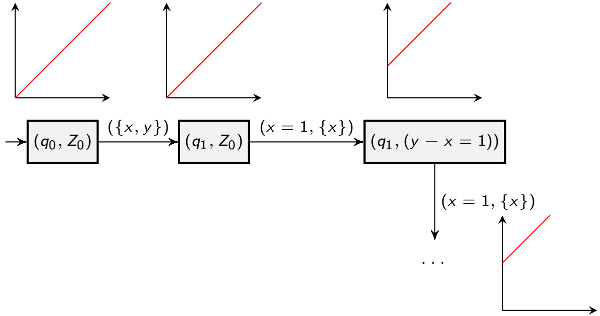
- $(q_0, Z_2) \preceq_{q_0} (q_0, Z_1)$ (Behaviour of Z_2 captured by Z_1 at q_0).
- For any infinite sequence of nodes $(q_0, Z_0), (q_1, Z_1), \cdots$, there must exist $j < i$, s.t., $q_i = q_j$ and $(q_i, Z_i) \preceq_{q_i} (q_j, Z_j)$

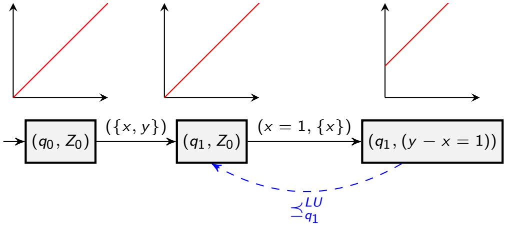
Finite simulations guarantee finite zone graph preserving soundness, completeness!

- There are many known finite simulations, e.g., *LU*-abstraction [BBLP06].

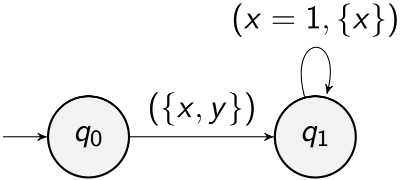
Recall: Getting a finite Zone graph using simulations

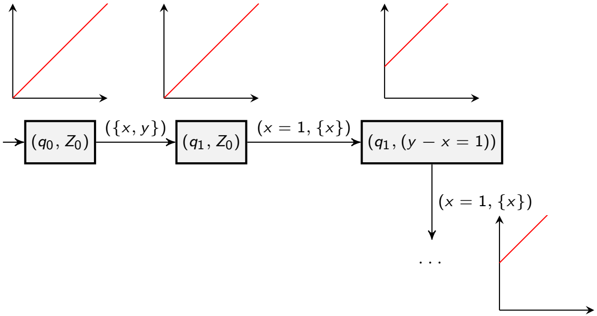


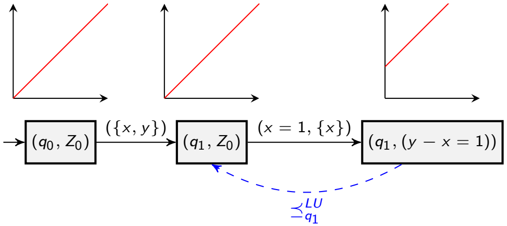




Recall: Getting a finite Zone graph using simulations

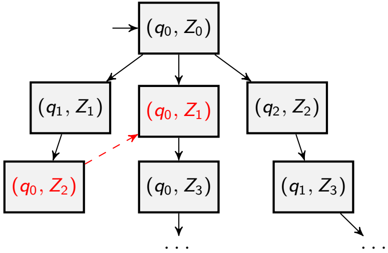


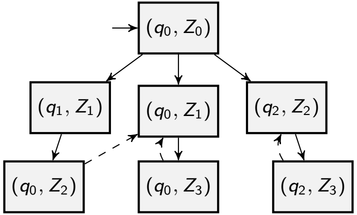




We don't need to know any more details.

We only care that such finite simulations exist!



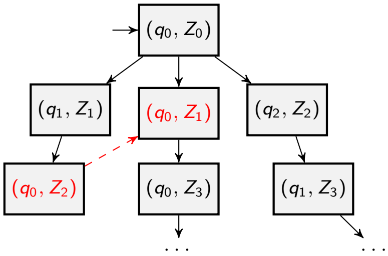


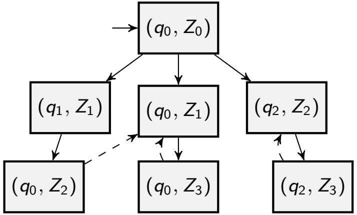
A modified re-write rule based saturation algorithm

$$\frac{}{S := \{(q_0, Z_0)\}}\ \text{start}$$

$$\frac{(q, Z) \in S \qquad q \xrightarrow{g,R} q' \qquad Z' = \overrightarrow{R(g \wedge Z)} \neq \emptyset}{S := S \cup \{(q', Z')\},\ \text{unless } \exists (q', Z'') \in S,\ Z' \preceq_{q'} Z''}\ \text{Trans}$$

Recall: Getting a finite Zone graph using simulations





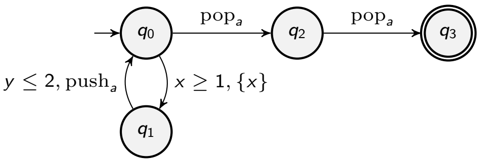
A modified re-write rule based saturation algorithm

$$\frac{}{S := \{(q_0, Z_0)\}} \text{ start}$$

$$\frac{(q, Z) \in S \qquad q \xrightarrow{g,R} q' \qquad Z' = \overrightarrow{R(g \wedge Z)} \neq \emptyset}{S := S \cup \{(q', Z')\}, \text{ unless } \exists (q', Z'') \in S,\ Z' \preceq_{q'} Z''} \text{ Trans}$$

This algorithm is sound, complete and terminating for computing set of reachable nodes in TA.

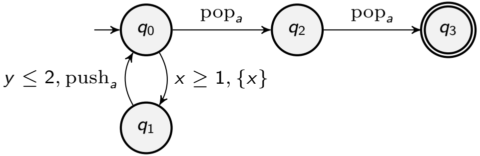
From TA to PDTA



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.

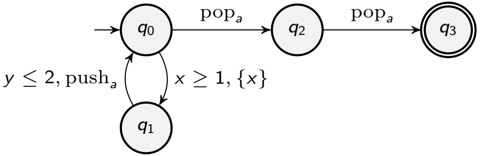
From TA to PDTA



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).

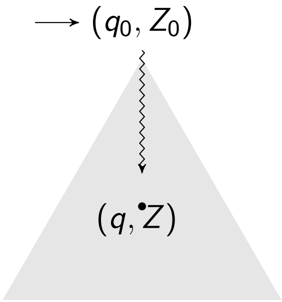
Let us try the same approach as above!

Viewing well-nested reachability in PDTA

$\longrightarrow (q_0, Z_0)$

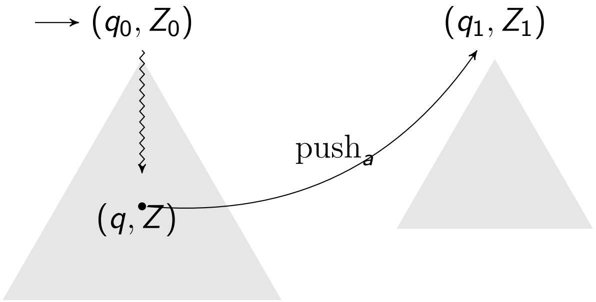
- We start with the initial node

Viewing well-nested reachability in PDTA



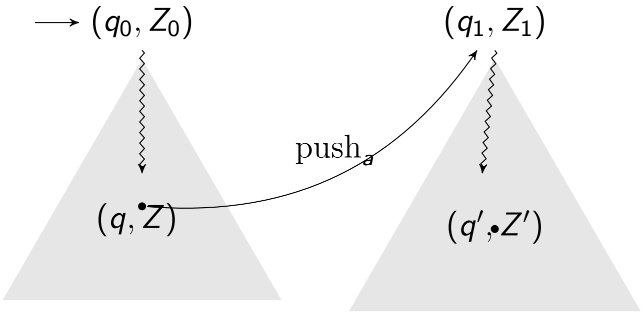
- We start with the initial node and explore as before as long as we see internal transitions (no push-pop).

Viewing well-nested reachability in PDTA



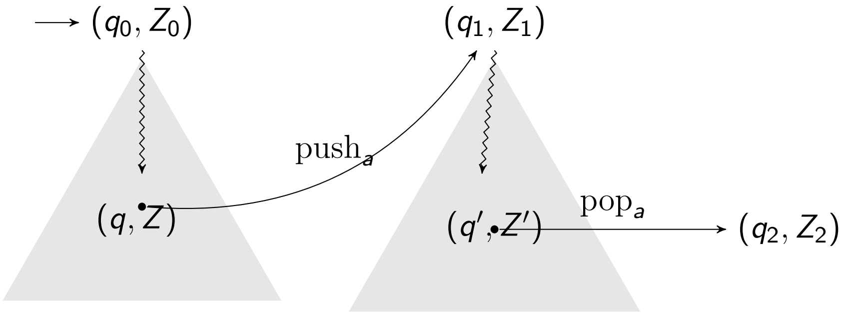
- When we see a Push, we start a new tree/context!

Viewing well-nested reachability in PDTA



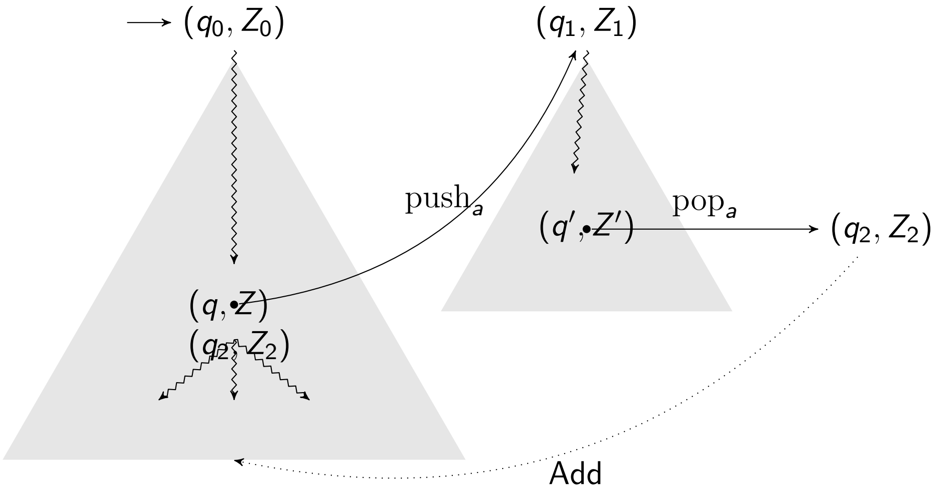
- When we see a Push, we start a new tree/context!
- Continue as long as we only see internal transitions.

Viewing well-nested reachability in PDTA



- Continue as long as we only see internal transitions.
- When we see a "matching" Pop transition,

Viewing well-nested reachability in PDTA



- 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 $S_{(q_0,Z_0)}$.

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

$$\frac{(q', Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'', Z'')\},}\ \text{Internal}$$

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 $\mathfrak{S}$!

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

$$\frac{(q, Z) \in \mathfrak{S} \qquad (q', Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'', Z'')\},}\ \text{Internal}$$

Reachability rules for PDTA

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'',Z'')\},}\ \text{Internal}$$

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

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{push}_a,R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{\mathfrak{S} := \mathfrak{S} \cup \{(q'',Z'')\},\ S_{(q'',Z'')} = \{(q'',Z'')\}}\ \text{Push}$$

Reachability rules for PDTA

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'',Z'')\},}\ \text{Internal}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{push}_a,R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{\mathfrak{S} := \mathfrak{S} \cup \{(q'',Z'')\},\ S_{(q'',Z'')} = \{(q'',Z'')\}}\ \text{Push}$$

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

$$\frac{\begin{array}{llll}(q,Z) \in \mathfrak{S} & (q',Z') \in S_{(q,Z)} & q' \xrightarrow{g,\mathrm{push}_a,R} q'' & Z'' = \overrightarrow{R(g \wedge Z')} \\ (q'',Z'') \in \mathfrak{S} & (q'_1,Z'_1) \in S_{(q'',Z'')} & q'_1 \xrightarrow{g_1,\mathrm{pop}_a,R_1} q_2 & Z_2 = \overrightarrow{R_1(g_1 \wedge Z'_1)} \neq \emptyset\end{array}}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q_2,Z_2)\}}\ \text{Pop}$$

Reachability rules for PDTA

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

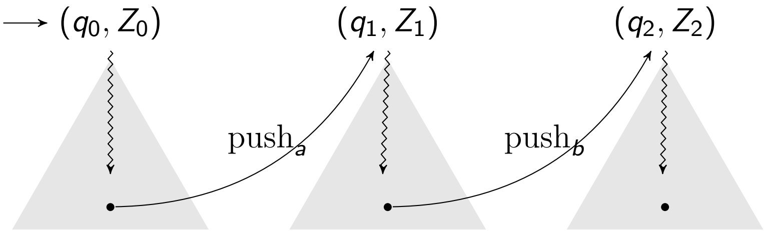
$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'',Z'')\},}\ \text{Internal}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{push}_a,R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{\mathfrak{S} := \mathfrak{S} \cup \{(q'',Z'')\},\ S_{(q'',Z'')} = \{(q'',Z'')\}}\ \text{Push}$$

$$\frac{\begin{array}{cccc} (q,Z) \in \mathfrak{S} & (q',Z') \in S_{(q,Z)} & q' \xrightarrow{g,\mathrm{push}_a,R} q'' & Z'' = \overrightarrow{R(g \wedge Z')} \\ (q'',Z'') \in \mathfrak{S} & (q_1',Z_1') \in S_{(q'',Z'')} & q_1' \xrightarrow{g_1,\mathrm{pop}_a,R_1} q_2 & Z_2 = \overrightarrow{R_1(g_1 \wedge Z_1')} \neq \emptyset \end{array}}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q_2,Z_2)\}}\ \text{Pop}$$

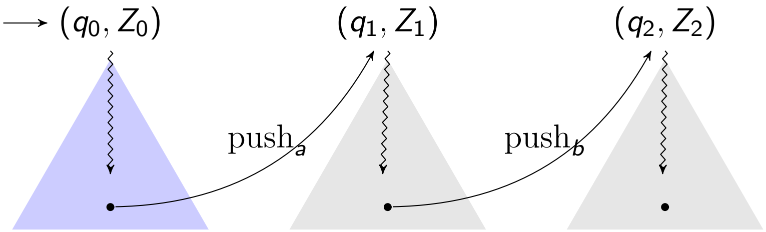
- This set of rules is sound and complete for well-nested control-state reachability in PDTA.
- Issue: But it is not terminating!

How to handle Push-Pop in the Zone graph



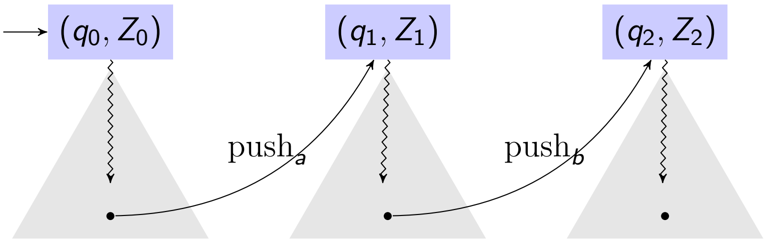
- Two sources of infinity!

How to handle Push-Pop in the Zone graph



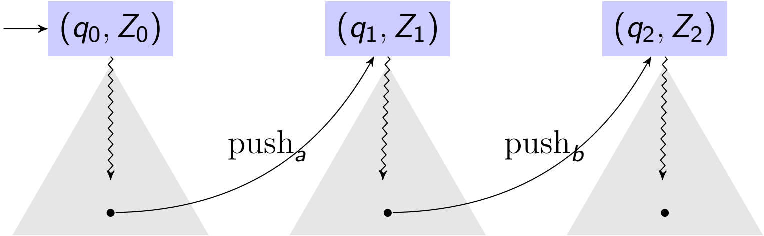
- Two sources of infinity!
  - Number of nodes in a tree

How to handle Push-Pop in the Zone graph



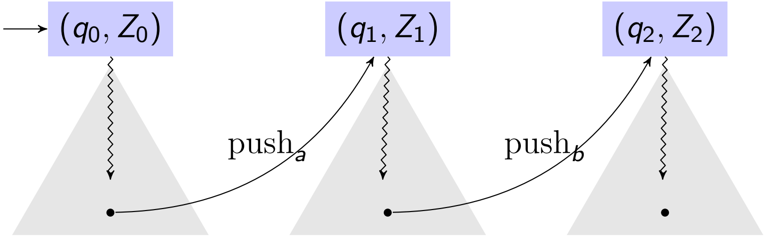
- Two sources of infinity!
  - Number of nodes in a tree
  - Number of root nodes, since each push starts tree at new root!

How to handle Push-Pop in the Zone graph



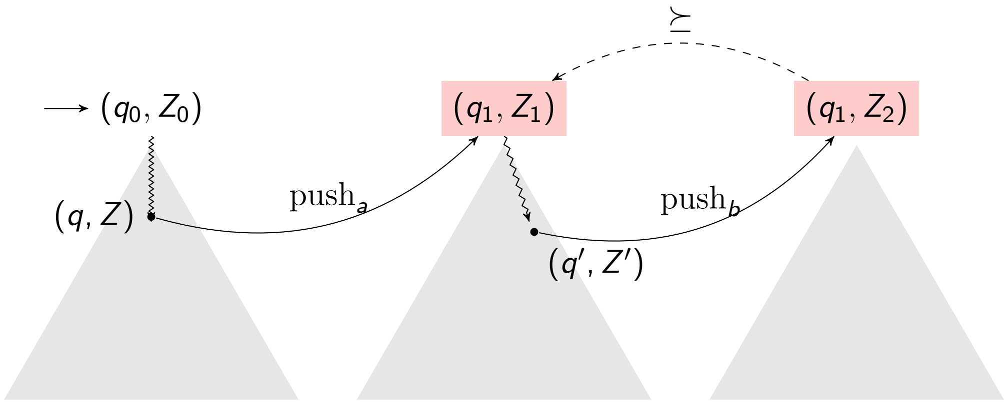
- Two sources of infinity!
  - Number of nodes in a tree
  - Number of root nodes, since each push starts tree at new root!
- Simulation inside a tree (i.e., within each tree) handles the first.

How to handle Push-Pop in the Zone graph

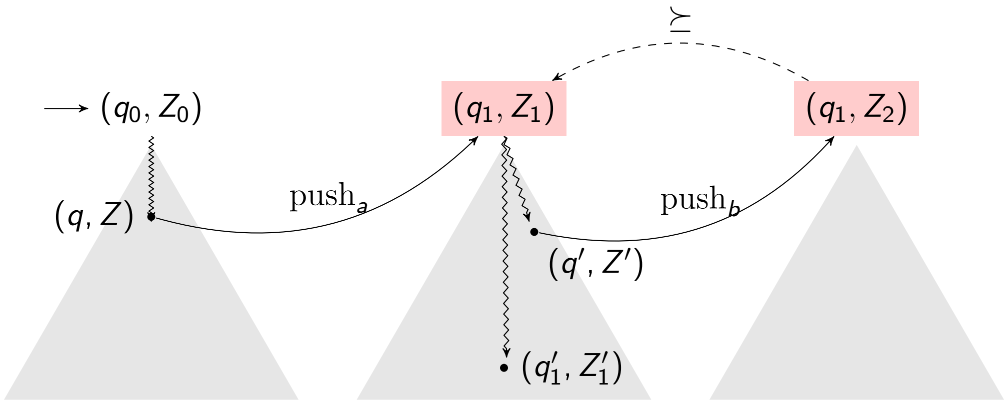


- Two sources of infinity!
  - Number of nodes in a tree
  - Number of root nodes, since each push starts tree at new root!
- Simulation inside a tree (i.e., within each tree) handles the first.
- But not the second! We lose soundness...

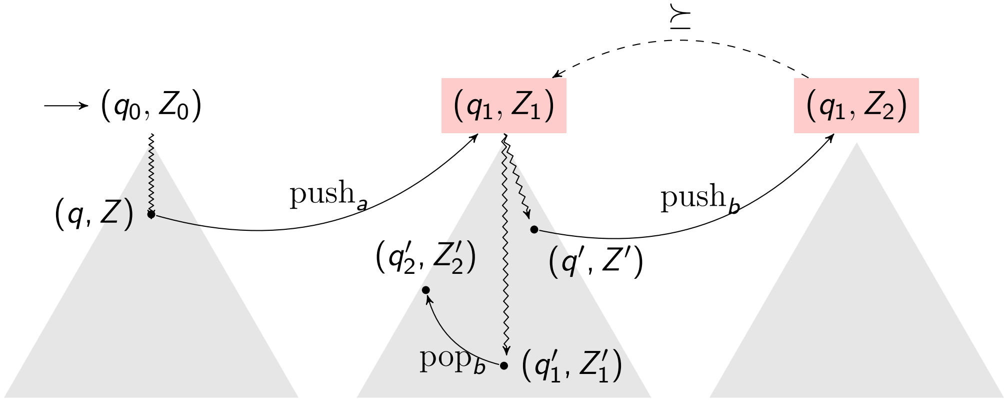
The problem with simulation & soundness



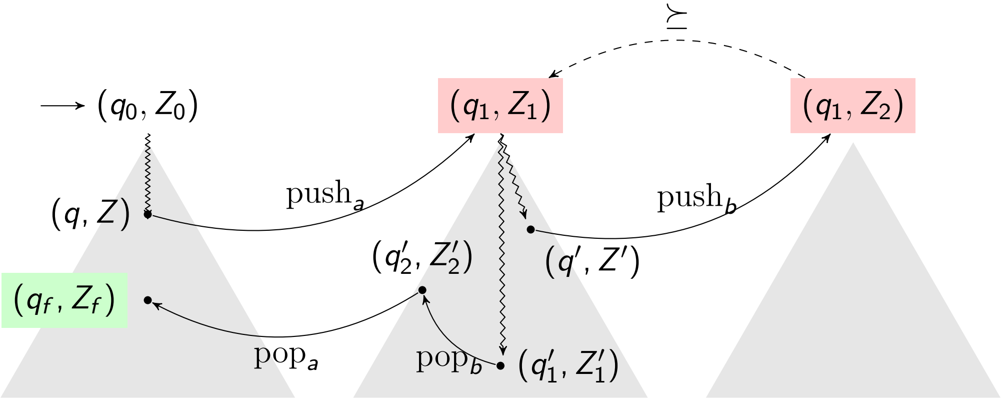
The problem with simulation & soundness



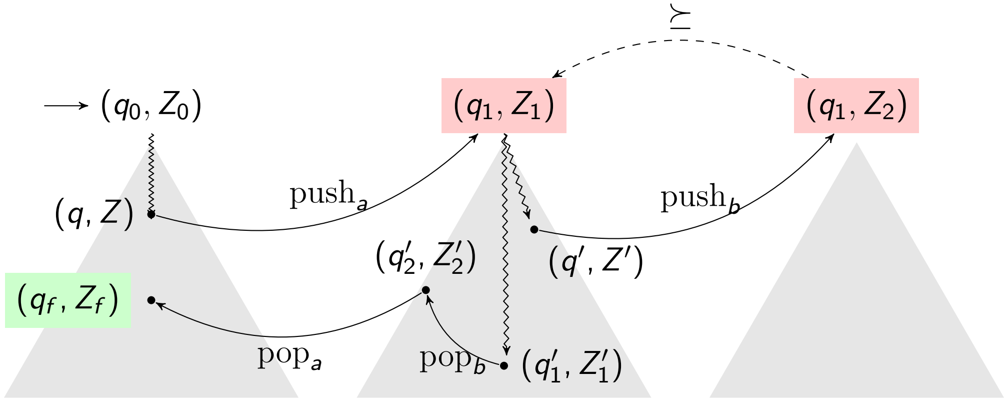
The problem with simulation & soundness



The problem with simulation & soundness



The problem with simulation & soundness

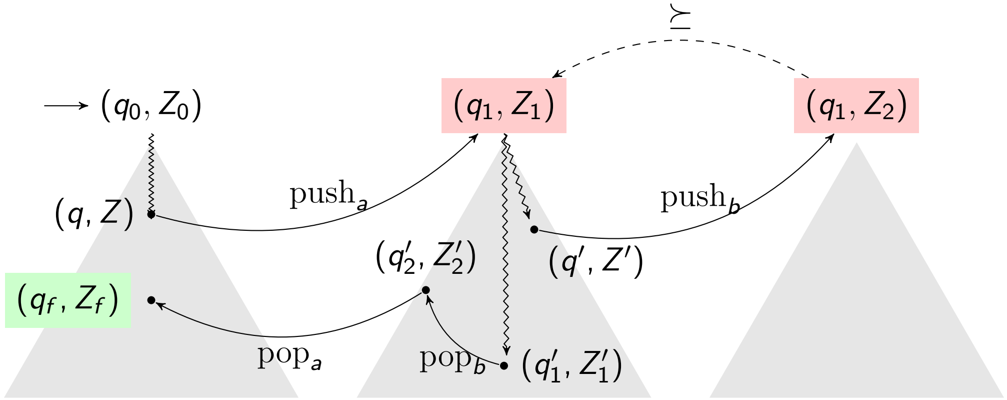


$$(q_0, Z_0) \to (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \to (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2)$$

$$\rotatebox{90}{$\succeq$}$$

$$(q_1, Z_1) \to (q'_1, Z'_1) \xrightarrow{\text{pop}_b} (q'_2, Z'_2) \xrightarrow{\text{pop}_a} (q_f, Z_f)$$

The problem with simulation & soundness

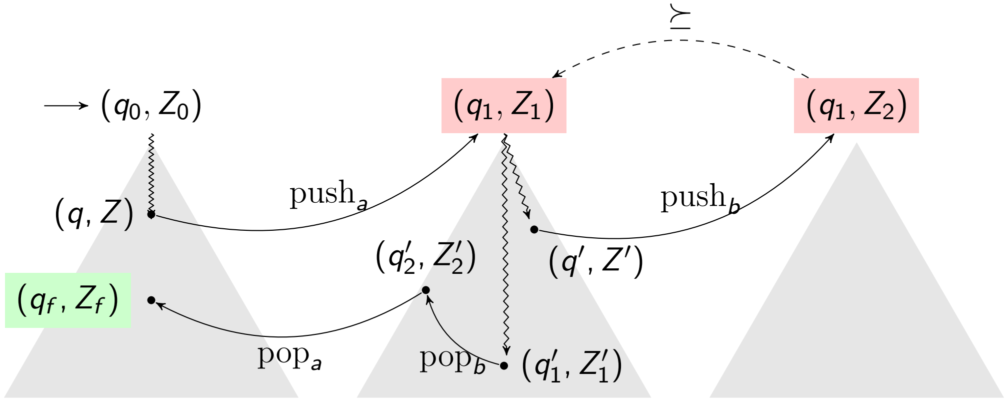


$$(q_0, Z_0) \to (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \to (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \not\to$$

Not Sound!

$$\begin{array}{c} \curlyeqsucc \\ \end{array}$$

$$(q_1, Z_1) \to (q'_1, Z'_1) \xrightarrow{\text{pop}_b} (q'_2, Z'_2) \xrightarrow{\text{pop}_a} (q_f, Z_f)$$



$$(q_0, Z_0) \to (q, Z) \xrightarrow{\mathrm{push}_a} (q_1, Z_1) \to (q', Z') \xrightarrow{\mathrm{push}_b} (q_1, Z_2) \not\to$$

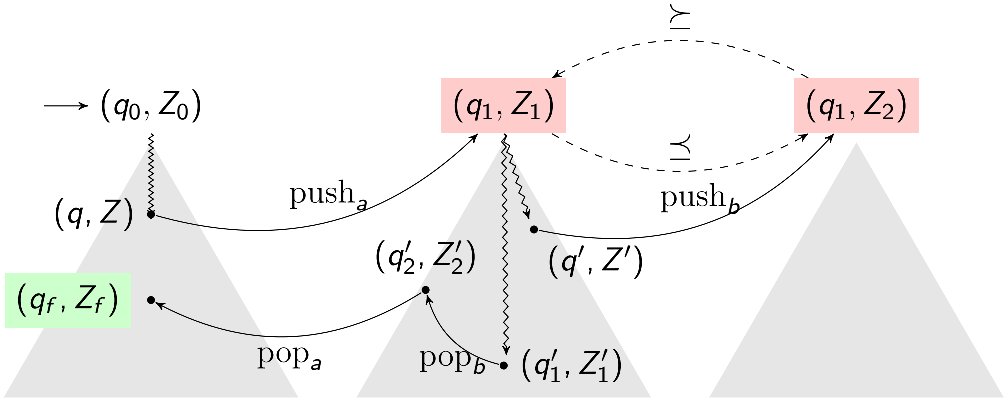
Not Sound!

$$\succeq$$

So how do we fix it?

$$(q_1, Z_1) \to (q'_1, Z'_1) \xrightarrow{\mathrm{pop}_b} (q'_2, Z'_2) \xrightarrow{\mathrm{pop}_a} (q_f, Z_f)$$

The problem with simulation & soundness



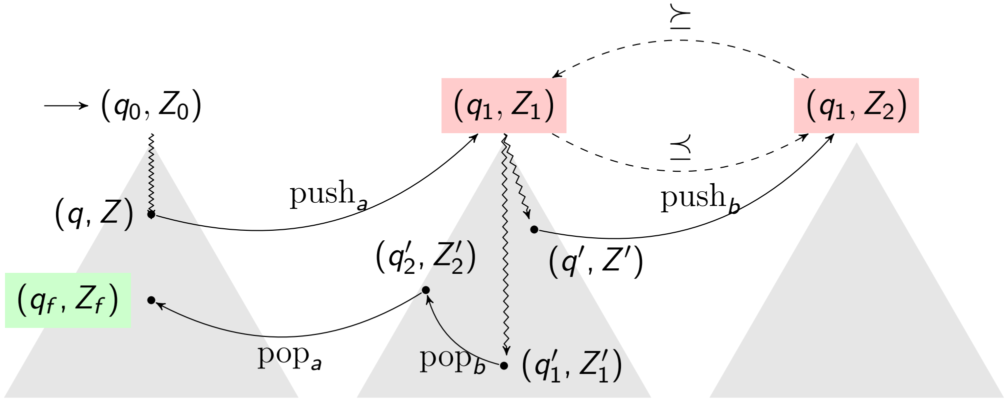
$$(q_0, Z_0) \to (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \to (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2)$$

$$\preceq \quad \succeq$$

Use equivalence!

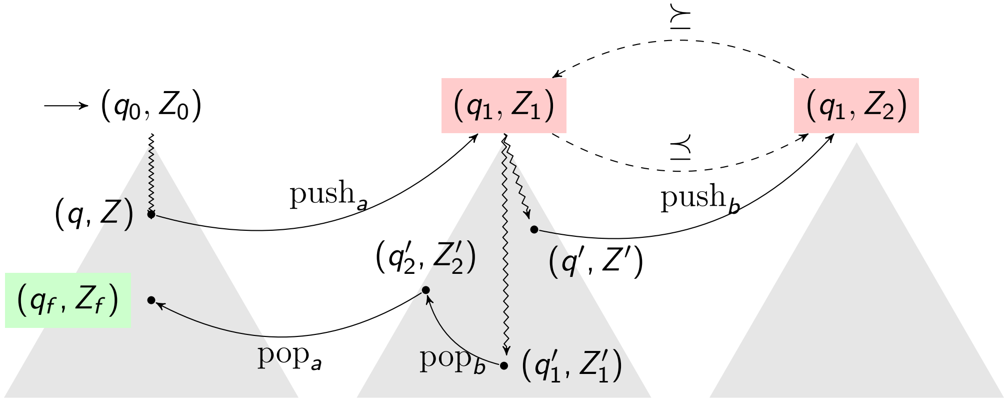
$$(q_1, Z_1) \to (q'_1, Z'_1) \xrightarrow{\text{pop}_b} (q'_2, Z'_2) \xrightarrow{\text{pop}_a} (q_f, Z_f)$$

The problem with simulation & soundness



$$
\begin{array}{ccccccccccccc}
(q_0, Z_0) & \to & (q, Z) & \xrightarrow{\text{push}_a} & (q_1, Z_1) & \to & (q', Z') & \xrightarrow{\text{push}_b} & (q_1, Z_2) & \to & (q_1', Z_1'') & & \\
 & & & & & & & & \rotatebox{90}{$\preceq$}\ \rotatebox{-90}{$\preceq$} & & \rotatebox{90}{$\preceq$}\ \rotatebox{-90}{$\preceq$} & & \\
 & & & & & & & & (q_1, Z_1) & \to & (q_1', Z_1') & \xrightarrow{\text{pop}_b} & (q_2', Z_2') \xrightarrow{\text{pop}_a} (q_f, Z_f)
\end{array}
$$

The problem with simulation & soundness



$$\begin{array}{l} (q_0, Z_0) \to (q, Z) \xrightarrow{\text{push}_a} (q_1, Z_1) \to (q', Z') \xrightarrow{\text{push}_b} (q_1, Z_2) \to (q_1', Z_1'')\ldots \\ \hspace{8em} \rotatebox{90}{$\preceq$} \ \rotatebox{90}{$\succeq$} \hspace{3em} \rotatebox{90}{$\preceq$} \ \rotatebox{90}{$\succeq$} \\ \hspace{8em} (q_1, Z_1) \to (q_1', Z_1') \xrightarrow{\text{pop}_b} (q_2', Z_2') \xrightarrow{\text{pop}_a} (q_f, Z_f) \end{array}$$

The solution: equivalence leads to soundess

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

The solution: equivalence leads to soundess

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

The solution: equivalence leads to soundess

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

The solution: equivalence leads to soundess

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

- Thus one solution is to require equivalence between roots.
  - $(q, Z) \sim_q (q, Z')$ if $(q, Z) \preceq_q (q, Z') \wedge (q, Z') \succeq_q (q, Z)$
  - 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!

The solution: equivalence leads to soundess

- Thus one solution is to require equivalence between roots.
  - $(q, Z) \sim_q (q, Z')$ if $(q, Z) \preceq_q (q, Z') \wedge (q, Z') \succeq_q (q, Z)$
  - 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!

The solution: equivalence leads to soundess

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

- Thus one solution is to require equivalence between roots.
  - $(q, Z) \sim_q (q, Z')$ if $(q, Z) \preceq_q (q, Z') \wedge (q, Z') \succeq_q (q, Z)$
  - 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

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

Rules for PDTA to regain finiteness

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'',Z'')\},\ \text{unless } \exists (q'',Z''') \in S_{(q,Z)},\ Z'' \preceq_{q''} Z'''}\ \text{Internal}$$

$$\frac{\begin{array}{cccc}(q,Z) \in \mathfrak{S} & (q',Z') \in S_{(q,Z)} & q' \xrightarrow{g,\mathrm{push}_a,R} q'' & Z'' = \overrightarrow{R(g \wedge Z')} \sim_{q''} Z_1 \\ (q'',Z_1) \in \mathfrak{S} & (q'_1,Z'_1) \in S_{(q'',Z_1)} & q'_1 \xrightarrow{g_1,\mathrm{pop}_a,R_1} q_2 & Z_2 = \overrightarrow{R_1(g_1 \wedge Z'_1)} \neq \emptyset\end{array}}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q_2,Z_2)\},\ \text{unless } \exists (q_2,Z'_2) \in S_{(q,Z)},\ Z_2 \preceq_{q_2} Z'_2}\ \text{Pop}$$

Rules for PDTA to regain finiteness

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'',Z'')\},\ \text{unless } \exists (q'',Z''') \in S_{(q,Z)},\ Z'' \preceq_{q''} Z'''}\ \text{Internal}$$

$$\frac{\begin{array}{cccc} (q,Z) \in \mathfrak{S} & (q',Z') \in S_{(q,Z)} & q' \xrightarrow{g,\mathrm{push}_a,R} q'' & Z'' = \overrightarrow{R(g \wedge Z')} \sim_{q''} Z_1 \\ (q'',Z_1) \in \mathfrak{S} & (q'_1,Z'_1) \in S_{(q'',Z_1)} & q'_1 \xrightarrow{g_1,\mathrm{pop}_a,R_1} q_2 & Z_2 = \overrightarrow{R_1(g_1 \wedge Z'_1)} \neq \emptyset \end{array}}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q_2,Z_2)\},\ \text{unless } \exists (q_2,Z'_2) \in S_{(q,Z)},\ Z_2 \preceq_{q_2} Z'_2}\ \text{Pop}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{push}_a,R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{\mathfrak{S} := \mathfrak{S} \cup \{(q'',Z'')\},\ S_{(q'',Z'')} = \{(q'',Z'')\}\ ,\ \text{unless } \exists (q'',Z''') \in \mathfrak{S},\ Z'' \sim_{q''} Z'''}\ \text{Push}$$

Rules for PDTA to regain finiteness

$$\frac{}{\mathfrak{S} := \{(q_0, Z_0)\},\ S_{(q_0,Z_0)} := \{(q_0, Z_0)\}}\ \text{Start}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{nop},R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q'',Z'')\},\ \text{unless } \exists (q'',Z''') \in S_{(q,Z)},\ Z'' \preceq_{q''} Z'''}\ \text{Internal}$$

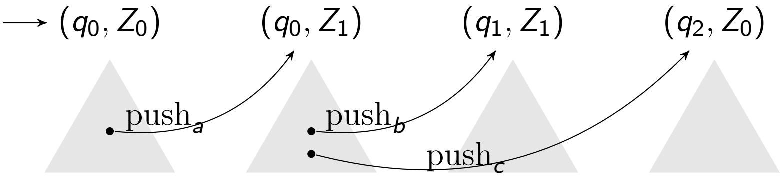
$$\frac{\begin{array}{cccc}(q,Z) \in \mathfrak{S} & (q',Z') \in S_{(q,Z)} & q' \xrightarrow{g,\mathrm{push}_a,R} q'' & Z'' = \overrightarrow{R(g \wedge Z')} \sim_{q''} Z_1 \\ (q'',Z_1) \in \mathfrak{S} & (q'_1,Z'_1) \in S_{(q'',Z_1)} & q'_1 \xrightarrow{g_1,\mathrm{pop}_a,R_1} q_2 & Z_2 = \overrightarrow{R_1(g_1 \wedge Z'_1)} \neq \emptyset\end{array}}{S_{(q,Z)} := S_{(q,Z)} \cup \{(q_2,Z_2)\},\ \text{unless } \exists (q_2,Z'_2) \in S_{(q,Z)},\ Z_2 \preceq_{q_2} Z'_2}\ \text{Pop}$$

$$\frac{(q,Z) \in \mathfrak{S} \qquad (q',Z') \in S_{(q,Z)} \qquad q' \xrightarrow{g,\mathrm{push}_a,R} q'' \qquad Z'' = \overrightarrow{R(g \wedge Z')} \neq \emptyset}{\mathfrak{S} := \mathfrak{S} \cup \{(q'',Z'')\},\ S_{(q'',Z'')} = \{(q'',Z'')\}\ ,\ \text{unless } \exists (q'',Z''') \in \mathfrak{S},\ Z'' \sim_{q''} Z'''}\ \text{Push}$$

Main Theorem

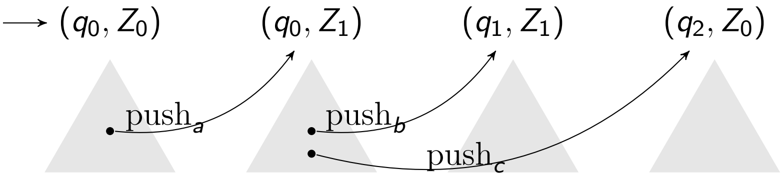
This set of rules is sound, complete & terminating for well-nested control-state reachability in PDTA.

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

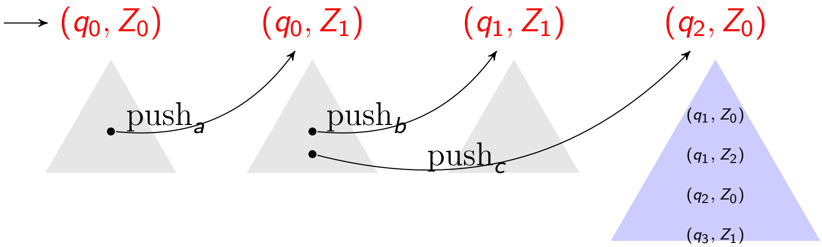


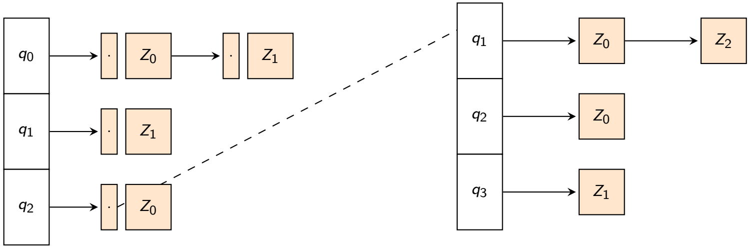
- The rules give a fix pt saturation algorithm.
- To implement it efficiently, we need to
  1. Come up with a good data structure.
  2. Decide on order of exploration
  3. Avoid/reduce revisiting explored nodes. (see paper)

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]

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

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]

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

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]

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

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

Simulations can prune branches but equivalences are good for roots!

Conclusion

Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

Conclusion

Simulations can prune branches but equivalences are good for roots!

A few concluding remarks

1. Lifts from well-nested reachability to general reachability

Conclusion

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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.

Conclusion

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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
3. Still lot of scope for optimizations/improvements.
    - Other simulations and extrapolations [BBLP06, HSW12]

Conclusion

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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
3. Still lot of scope for optimizations/improvements.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblence to Liveness in TA [Tri09, LOD$^+$13, HSTW20].

Conclusion

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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
3. Still lot of scope for optimizations/improvements.
    - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblence to Liveness in TA [Tri09, LOD^{+}13, HSTW20].
5. Witness generation not completely obvious, due to fix pt computation, but can be done.

Conclusion

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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
3. Still lot of scope for optimizations/improvements.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblence 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?

Conclusion

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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
3. Still lot of scope for optimizations/improvements.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblence 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?

Conclusion

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 $\mathcal{G}$-abstraction [GMS19] will allow handling diagonal guards in PDTA.
3. Still lot of scope for optimizations/improvements.
   - Other simulations and extrapolations [BBLP06, HSW12]
4. Interesting link and resemblence 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!

References I

Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman.
Dense-timed pushdown automata.
In Proc. of LICS'12, pages 35–¡V44, 2012.

Rajeev Alur and David L. Dill.
Automata for modeling real-time systems.
In Proc. of ICALP'90, volume 443 of LNCS, pages 322–335. Springer, 1990.

S. Akshay, Paul Gastin, Vincent Jugé, and Shankara Narayanan Krishna.
Timed systems through the lens of logic.
In Proc. of LICS, pages 1–13, 2019.

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

S. Akshay, Paul Gastin, Shankara Narayanan Krishna, and Sparsa Roychowdhury.
Revisiting underapproximate reachability for multipushdown systems.
In Proc. of TACAS'20, volume 12078, pages 387–404. Springer, 2020.

S. Akshay, Paul Gastin, Shankara Narayanan Krishna, and Ilias Sarkar.
Towards an efficient tree automata based technique for timed systems.
In Proc. of CONCUR, pages 39:1–39:15, 2017.

Gerd Behrmann, Patricia Bouyer, Kim G Larsen, and Radek Pelánek.
Lower and upper bounds in zone-based abstractions of timed automata.
STTT, 8(3):204–215, 2006.

Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks.
Uppaal 4.0.
2006.

References II

Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana.
On the automatic verification of systems with continuous variables and unbounded discrete data structures.
In International Hybrid Systems Workshop, pages 64–85. Springer, 1994.

Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi.
Uppaal¡xa tool suite for automatic verification of real-time systems.
In International Hybrid Systems Workshop, pages 232–243. Springer, 1995.

Lorenzo Clemente and Slawomir Lasota.
Timed pushdown automata revisited.
In Proc. of LICS, pages 738–¡V749, 2015.

Lorenzo Clemente and Slawomir Lasota.
Reachability relations of timed pushdown automata.
JCSS, 117:202–241, 2021.

Lorenzo Clemente, Slawomir Lasota, Ranko Lazic, and Filip Mazowiecki.
Timed pushdown automata and branching vector addition systems.
In Proc. of LICS, pages 1–12, 2017.

Paul Gastin, Sayan Mukherjee, and B. Srivathsan.
Fast algorithms for handling diagonal constraints in timed automata.
In Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I, volume 11561 of Lecture Notes in Computer Science, pages 41–59. Springer, 2019.

Frédéric Herbreteau and Gerald Point.
Tchecker.
Available at `https://github.com/fredher/tchecker`, 2019.

References III

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.

Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz.
Better abstractions for timed automata.
In Proc. of LICS, pages 375–384. IEEE Computer Society, 2012.

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 Proc. of CAV, volume 8044, pages 968–983. Springer, 2013.

Kim G Larsen, Paul Pettersson, and Wang Yi.
Uppaal in a nutshell.
STTT, 1(1-2):134–152, 1997.

Paul Pettersson and Kim G Larsen.
Uppaal2k.
Bulletin of the EATCS, 70(40–44):2, 2000.

Stavros Tripakis.
Checking timed büchi automata emptiness on simulation graphs.
ACM Trans. Comput. Log., 10(3):15:1–15:19, 2009.