Analyzing Timed Systems Using Tree Automata

S Akshay\(^1\), Paul Gastin\(^2\) and Krishna Shankara Narayanan\(^1\)

\(^1\) Dept of CSE, IIT Bombay, India,
\(^2\) LSV, ENS Cachan, France.

CONCUR 2016, Quebec
26 Aug 2016
Timed automata and timed runs

$q_1 \xrightarrow{x \leq 3, a, x := 0} q_2$
Timed automata and timed runs

\[ q_1 \xrightarrow{x \leq 3 \atop a, x := 0} q_2 \xrightarrow{y \geq 4 \atop b, y := 0} q_3 \xrightarrow{x \leq 2 \atop a, x := 0} q_2 \]
Timed automata and timed runs

\[ x \leq 3 \quad a, x := 0 \]
\[ y \geq 4 \quad b, y := 0 \]
\[ x \leq 2 \quad a, x := 0 \]
Timed automata and timed runs

The timed language $L_T(A) = \text{set of such good timed words}$

Emptiness problem: Given $A$, is $L_T(A) = \emptyset$?
Emptiness for timed automata

A well-studied problem with a now standard approach
Emptiness for timed automata

A well-studied problem with a now standard approach

- Timed automata: Region construction [Alur-Dill’90], and many optimizations since...
Emptiness for timed (pushdown) automata

A well-studied problem with a now standard approach

- **Timed automata**: Region construction [Alur-Dill’90], and many optimizations since...
- **Timed pushdown automata:**
Emptiness for timed (pushdown) automata

A well-studied problem with a now standard approach

- **Timed automata**: Region construction [Alur-Dill’90], and many optimizations since...

- **Timed pushdown automata**: Lifting region construction – [Bouajjani et al. ’94], [Abdulla et al. 2012]
Emptiness for timed (pushdown) automata

A well-studied problem with a now standard approach

- **Timed automata**: Region construction [Alur-Dill’90], and many optimizations since...
- **Timed pushdown automata**: Lifting region construction – [Bouajjani et al. ‘94], [Abdulla et al. 2012]
- An orthogonal approach: [Clemente-Lasota 2015]
Emptiness for timed (pushdown) automata

A well-studied problem with a now standard approach

- **Timed automata**: Region construction [Alur-Dill’90], and many optimizations since...

- **Timed pushdown automata**: Lifting region construction – [Bouajjani et al. ’94], [Abdulla et al. 2012]

- Common feature:
  - represent behaviors as timed words and,
  - use abstractions to reduce to finite automata over words
Emptiness for timed (pushdown) automata

A well-studied problem with a now standard approach

- **Timed automata**: Region construction [Alur-Dill’90], and many optimizations since...
- **Timed pushdown automata**: Lifting region construction – [Bouajjani et al. ’94], [Abdulla et al. 2012]
- Common feature:
  - represent behaviors as timed words and,
  - use abstractions to reduce to finite automata over words

Our point-de-depart

- represent behaviors as graphs with timing constraints
- use tree interpretations to reduce to tree automata
Emptiness for timed (pushdown) automata

A well-studied problem with a now standard approach

- Timed automata: Region construction [Alur-Dill’90], and many optimizations since...

- **Timed pushdown automata**: Lifting region construction – [Bouajjani et al. ’94], [Abdulla et al. 2012]

- Common feature:
  - represent behaviors as timed words and,
  - use abstractions to reduce to finite automata over words

Our point-de-depart

- represent behaviors as graphs with timing constraints
- use tree interpretations to reduce to tree automata
  - A higher level and more powerful formalism
  - Yields simpler proofs for more complicated systems
  - A new technique which does not depend on regions/zones
Outline

1. Timed behaviours as graphs
2. Checking realizability
3. Interpreting graphs into trees
4. Bounding the (split-)width of graphs
5. Conclusion & future work
Abstracting paths of a timed system as graphs
Abstracting paths of a timed system as graphs
Abstracting paths of a timed system as graphs

\[ x \leq 3, \quad a, x := 0 \]
\[ y \geq 4, \quad b, y := 0 \]
\[ x \leq 2, \quad a, x := 0 \]

There are edges between states labeled with inequalities and assignments, indicating transitions in the timed system.
Abstracting paths of a timed system as graphs

set of such time-constrained graphs, TC-words = $\mathcal{L}_{TCW}(A)$. 
Abstracting paths of a timed system as graphs

- set of such time-constrained graphs, TC-words = $\mathcal{L}_{TCW}(A)$.
- What are some properties of such graphs?
Abstracting paths of a timed system as graphs

- set of such time-constrained graphs, TC-words = $\mathcal{L}_{TCW}(\mathcal{A})$.
- What are some properties of such graphs?
- What is the link between $\mathcal{L}_{TCW}(\mathcal{A})$ and $\mathcal{L}_T(\mathcal{A})$?
TC-words and their relation to timed words

Properties of TC-words and timed words

1. Not all (linearly-ordered) graphs are TC-words
TC-words and their relation to timed words

Properties of TC-words and timed words

1. Not all (linearly-ordered) graphs are TC-words
TC-words and their relation to timed words

Properties of TC-words and timed words

1. Not all (linearly-ordered) graphs are TC-words

This graph cannot be generated by any timed automaton.
TC-words and their relation to timed words

Properties of TC-words and timed words

1. Not all (linearly-ordered) graphs are TC-words

This graph cannot be generated by any timed automaton.
TC-words and their relation to timed words

Properties of TC-words and timed words

1. Not all (linearly-ordered) graphs are TC-words

This graph cannot be generated by any timed automaton.
But, it can be generated by a timed pushdown automaton!
TC-words and their relation to timed words

Properties of TC-words and timed words

1. Not all (linearly-ordered) graphs are TC-words

2. A TC-word can be realized by (infinitely) many timed words
TC-words and their relation to timed words

Properties of TC-words and timed words

1. Not all (linearly-ordered) graphs are TC-words
2. A TC-word can be realized by (infinitely) many timed words
3. However, a TC-word may be realized by no timed word too!
Realizability of TC-words

- **Realization** of a TC-word is a timed word satisfying constraints.
- A TC-word is **realizable** if it has a timed word realization.
Realizability of TC-words

- **Realization** of a TC-word is a timed word satisfying constraints.
- A TC-word is **realizable** if it has a timed word realization.
- Recall: for a timed system $\mathcal{A}$, $\mathcal{L}_{TCW}(\mathcal{A})$ denotes the set of TC-words accepted by it.
Realizability of TC-words

• **Realization** of a TC-word is a timed word satisfying constraints.
• A TC-word is **realizable** if it has a timed word realization.
• Recall: for a timed system $\mathcal{A}$, $\mathcal{L}_{TCW}(\mathcal{A})$ denotes the set of TC-words accepted by it.

Difference between $\mathcal{L}_{TCW}(\mathcal{A})$ and $\mathcal{L}_T(\mathcal{A})$:

$\mathcal{L}_{TCW}(\mathcal{A})$ is over a finite alphabet, while $\mathcal{L}_T(\mathcal{A})$ is not.
Realizability of TC-words

- **Realization**: of a TC-word is a timed word satisfying constraints.
- A TC-word is **realizable** if it has a timed word realization.
- Recall: for a timed system $\mathcal{A}$, $\mathcal{L}_{TCW}(\mathcal{A})$ denotes the set of TC-words accepted by it.

**Theorem**: $\mathcal{L}_{T}(\mathcal{A}) = \text{Realizations}(\mathcal{L}_{TCW}(\mathcal{A}))$
Realizability of TC-words

- **Realization** of a TC-word is a timed word satisfying constraints.
- A TC-word is **realizable** if it has a timed word realization.
- Recall: for a timed system \( \mathcal{A} \), \( L_{TCW}(\mathcal{A}) \) denotes the set of TC-words accepted by it.

**Theorem:** \( L_T(\mathcal{A}) = \text{Realizations}(L_{TCW}(\mathcal{A})) \)

**The Emptiness problem**

For a given timed (pushdown) automaton \( \mathcal{A} \), \( L_T(\mathcal{A}) \neq \emptyset \) iff there exists a realizable TC-word in \( L_{TCW}(\mathcal{A}) \).
Realizability of TC-words

- **Realization** of a TC-word is a timed word satisfying constraints.
- A TC-word is **realizable** if it has a timed word realization.
- Recall: for a timed system $\mathcal{A}$, $\mathcal{L}_{TCW}(\mathcal{A})$ denotes the set of TC-words accepted by it.

**Theorem:** $\mathcal{L}_T(\mathcal{A}) = \text{Realizations}(\mathcal{L}_{TCW}(\mathcal{A}))$

**The Emptiness problem**

For a given timed (pushdown) automaton $\mathcal{A}$, $\mathcal{L}_T(\mathcal{A}) \neq \emptyset$ iff there exists a realizable TC-word in $\mathcal{L}_{TCW}(\mathcal{A})$.

Thus, the question is: how to reason about these graphs?
Checking realizability of a single TC-word
Checking realizability of a single TC-word
Checking realizability of a single TC-word

A simple exercise

A TC-word is realizable iff its directed graph has no negative cycle.
The Emptiness problem

For a given timed (pushdown) automaton $\mathcal{A}$,

Does there exist a TC-word in $\mathcal{L}_{\mathcal{T}_{\text{CW}}} (\mathcal{A})$, whose directed graph has no negative cycle?

- How to reason about the set of graphs $\mathcal{L}_{\mathcal{T}_{\text{CW}}} (\mathcal{A})$?
The Emptiness problem

For a given timed (pushdown) automaton $\mathcal{A}$,

Does there exist a TC-word in $\mathcal{L}_{TCW}(\mathcal{A})$, whose directed graph has no negative cycle?

If we can show that:

1. Graphs have a bounded-width.
2. Each property is expressible in MSO.
The Emptiness problem

For a given timed (pushdown) automaton $\mathcal{A}$, Does there exist a TC-word in $\mathcal{L}_{TCW}(\mathcal{A})$, whose directed graph has no negative cycle?

- If we can show that:
  1. Graphs have a bounded-width.
  2. Each property is expressible in MSO.
     - Graphs are well-formed
     - Graphs define an abstract path in the given timed system.
     - Graphs are realizable, i.e., no negative weight cycle.
The Emptiness problem

For a given timed (pushdown) automaton $\mathcal{A}$,

Does there exist a TC-word in $\mathcal{L}_{TCW}(\mathcal{A})$, whose directed graph has no negative cycle?

- If we can show that:
  1. Graphs have a bounded-width.
  2. Each property is expressible in MSO.
     - Graphs are well-formed
     - Graphs define an abstract path in the given timed system.
     - Graphs are realizable, i.e., no negative weight cycle.

- Then, by Courcelle’s theory, we obtain a finite tree automaton (by interpreting the graphs into trees).
The Emptiness problem

For a given timed (pushdown) automaton $\mathcal{A}$,

Does there exist a TC-word in $\mathcal{L}_{TCW}(\mathcal{A})$, whose directed graph has no negative cycle?

- If we can show that:
  1. Graphs have a bounded-width.
  2. Each property is expressible in MSO.
     - Graphs are well-formed
     - Graphs define an abstract path in the given timed system.
     - Graphs are realizable, i.e., no negative weight cycle.

- Then, by Courcelle’s theory, we obtain a finite tree automaton (by interpreting the graphs into trees).

Same strategy as [Madhusudan & Parlato’11, Aiswarya et al ’12] for untimed pushdown systems.
The Emptiness problem

For a given timed (pushdown) automaton \( A \),

Does there exist a TC-word in \( L_{TCW}(A) \), whose directed graph has no negative cycle?

- If we can show that:
  1. Graphs have a bounded-width.
  2. Each property is expressible in MSO.
     - Graphs are well-formed
     - Graphs define an abstract path in the given timed system.
     - Graphs are realizable, i.e., no negative weight cycle.

- Then, by Courcelle’s theory, we obtain a finite tree automaton (by interpreting the graphs into trees).

We show

- Step 1: graphs from \( T(PD)A \) have a bounded (split-)width.
The Emptiness problem

For a given timed (pushdown) automaton $\mathcal{A}$,

Does there exist a TC-word in $\mathcal{L}_{TCW}(\mathcal{A})$, whose directed graph has no negative cycle?

Graphs from timed systems are different!

We show

- \textbf{Step 1:} graphs from $T(PD)\mathcal{A}$ have a bounded (split-)width.
The Emptiness problem

For a given timed (pushdown) automaton \( \mathcal{A} \),

Does there exist a TC-word in \( L_{T_{CW}}(\mathcal{A}) \), whose directed graph has no negative cycle?

- If we can show that:
  1. Graphs have a bounded-width.
  2. Each property is expressible in MSO.

    - Graphs are well-formed
    - Graphs define an abstract path in the given timed system.
    - Graphs are realizable, i.e., no negative weight cycle.

- Then, by Courcelle’s theory, we obtain a finite tree automaton (by interpreting the graphs into trees).

We show

- Step 1: graphs from T(PD)A have a bounded (split-)width.
The Emptiness problem

For a given timed (pushdown) automaton \( \mathcal{A} \),

Does there exist a TC-word in \( \mathcal{L}_{TCW}(\mathcal{A}) \), whose directed graph has no negative cycle?

- If we can show that:
  1. Graphs have a bounded-width.
  2. Each property is expressible in MSO.
     - Graphs are well-formed
     - Graphs define an abstract path in the given timed system.
     - Graphs are realizable, i.e., no negative weight cycle.

- Then, by Courcelle’s theory, we obtain a finite tree automaton (by interpreting the graphs into trees).

We show

- Step 1: graphs from T(PD)A have a bounded (split-)width.
The Emptiness problem

For a given timed (pushdown) automaton $\mathcal{A}$,

Does there exist a TC-word in $L_{TCW}(\mathcal{A})$, whose directed graph has no negative cycle?

- If we can show that:
  1. Graphs have a bounded-width.
  2. Each property is expressible in MSO.
     - Graphs are well-formed
     - Graphs define an abstract path in the given timed system.
     ? Graphs are realizable, i.e., no negative weight cycle.

- Then, by Courcelle’s theory, we obtain a finite tree automaton (by interpreting the graphs into trees).

We show

- Step 1: graphs from $T(PD)\mathcal{A}$ have a bounded (split-)width.
- Step 2: directly build a finite bottom-up tree automaton.
Complexity bounds

- Step 1: Bound on (split-)width for timed (pushdown) systems
- Step 2: Directly building the tree automaton allows us to get tight complexity bounds.
Complexity bounds

- Step 1: Bound on (split-)width for timed (pushdown) systems
- Step 2: Directly building the tree automaton allows us to get tight complexity bounds.

Main results

1. For timed automaton $A$ with clocks $X$, all simple TC-words of $A$ have (split-)width $K \leq |X| + 4$. 
Complexity bounds

- Step 1: Bound on (split-)width for timed (pushdown) systems
- Step 2: Directly building the tree automaton allows us to get tight complexity bounds.

Main results

1. For timed (pushdown) automaton $\mathcal{A}$ with clocks $X$, all simple TC-words of $\mathcal{A}$ have (split-)width $K \leq |X| + 4 (4|X| + 6)$. 
Complexity bounds

- Step 1: Bound on (split-)width for timed (pushdown) systems
- Step 2: Directly building the tree automaton allows us to get tight complexity bounds.

Main results

1. For timed (pushdown) automaton $A$ with clocks $X$, all simple TC-words of $A$ have (split-)width $K \leq |X| + 4 (4|X| + 6)$.
2. We can build a tree automaton of size exponential in $K^2$ to check realizability (details in paper).
Complexity bounds

- Step 1: Bound on (split-)width for timed (pushdown) systems
- Step 2: Directly building the tree automaton allows us to get tight complexity bounds.

Main results

1. For timed (pushdown) automaton $\mathcal{A}$ with clocks $X$, all simple TC-words of $\mathcal{A}$ have (split-)width $K \leq |X| + 4 (4|X| + 6)$.
2. We can build a tree automaton of size exponential in $K^2$ to check realizability (details in paper).
3. Corollary: PSPACE (Exptime) emptiness for timed (pushdown) automata.
Complexity bounds

- Step 1: Bound on (split-)width for timed (pushdown) systems
- Step 2: Directly building the tree automaton allows us to get tight complexity bounds.

Main results

1. For timed (pushdown) automaton $A$ with clocks $X$, all simple TC-words of $A$ have (split-)width $K \leq |X| + 4 (4|X| + 6)$.
2. We can build a tree automaton of size exponential in $K^2$ to check realizability (details in paper).
3. Corollary: PSPACE (Exptime) emptiness for timed (pushdown) automata.

Lift to timed multi-pushdown systems with bounded rounds

- Easy generalization, new decidability result & complexity too!
Step 0: Simplifying the TC-words

We first break TC-words into “simpler” graphs, so that each node has only one upper/lower time constraint attached to it.
Step 0: Simplifying the TC-words

We first break TC-words into “simpler” graphs, so that each node has only one upper/lower time constraint attached to it. For example,
Step 0: Simplifying the TC-words

- We first break TC-words into “simpler” graphs, so that each node has only one upper/lower time constraint attached to it. For example,

```
\[\text{is converted to:}\]
```

![Diagram showing the simplification process of TC-words into simpler graphs]

```
Step 0: Simplifying the TC-words

- We first break TC-words into “simpler” graphs, so that each node has only one upper/lower time constraint attached to it. For example,

  ![Original Graph](image1)

  is converted to:

  ![Simplified Graph](image2)

- To maintain atomicity, we use a single extra clock & add a constraint to each event:

  ![Atomicity Constraint](image3)
Step 1: Split-width for timed systems

Now, define split game (see [Aiswarya et. al.’12, ’15])...
Step 1: Split-width for timed systems

Now, define split game (see [Aiswarya et. al.’12, ’15])...

- Eve tries to disconnect the graph by cutting process edges.
- Positions are simple TC-words \textbf{with holes}.
Step 1: Split-width for timed systems

Now, define split game (see [Aiswarya et. al.’12, ’15])...

- Eve tries to disconnect the graph by cutting process edges.
- Positions are simple TC-words with holes.
- Adam chooses which connected component to continue.
Step 1: Split-width for timed systems

Now, define split game (see [Aiswarya et. al.’12, ’15])...

- Game ends at atomic nodes (no process edges left).
Step 1: Split-width for timed systems

Now, define split game (see [Aiswarya et. al.’12, ’15])...

- Width of such a split simple TC-word = no. of blocks in it.
- Cost of play = max width of split TC-word seen along play.
- Split-width = min cost that Eve can achieve.
Step 1: Split-width for timed systems

- To bound: split-width of any well-formed simple TC-word, i.e., graph from a timed (pushdown) automaton.
Step 1: Split-width for timed systems

- To bound: split-width of any well-formed simple TC-word, i.e., graph from a timed (pushdown) automaton.
Step 1: Split-width for timed systems

- To bound: split-width of any **well-formed** simple TC-word, i.e., graph from a timed (pushdown) automaton.
- Let’s play the game...
Split-width for timed automata
Split-width for timed automata
Split-width for timed automata
Split-width for timed automata
For any TC-word of a timed automaton

In any move of the game, we have:

- Each hole is attached to last reset of a clock, holes only widen!
Split-width for timed automata

For any TC-word of a timed automaton

In any move of the game, we have:

- Each hole is attached to last reset of a clock, holes only widen!
- Thus, no. of blocks $\leq$ No. of clocks + 4.
Split-width for timed pushdown automata
Split-width for timed pushdown automata
Split-width for timed pushdown automata
Split-width for timed pushdown automata
For any TC-word of a timed pushdown automaton

In any move of the game, we have:

- Number of blocks $\leq 4 \cdot \text{No. of clocks} + 6$. 
Conclusion and Future work

A new recipe for analyzing timed systems. Given $\mathcal{A}$,
Conclusion and Future work

A new recipe for analyzing timed systems. Given $\mathcal{A}$,

1. Write behaviors as graphs with timing constraints $\mathcal{L}_{TCW}(\mathcal{A})$. 

Future work
- Concurrent recursive timed programs
- MSO definability of realizability
- Going beyond emptiness. What about model-checking?
Conclusion and Future work

A new recipe for analyzing timed systems. Given $\mathcal{A}$,

1. Write behaviors as graphs with timing constraints $\mathcal{L}_{TCW}(\mathcal{A})$.
2. Show a bound on width of graphs in $\mathcal{L}_{TCW}(\mathcal{A})$. 

A common framework for timed, pushdown, multi-pushdown automata with bounded rounds.

Robust framework: diagonal guards, etc.

Future work

Concurrent recursive timed programs
MSO definability of realizability
Going beyond emptiness. What about model-checking?
Conclusion and Future work

A new recipe for analyzing timed systems. Given \( \mathcal{A} \),

1. Write behaviors as graphs with timing constraints \( \mathcal{L}_{TCW}(\mathcal{A}) \).
2. Show a bound on width of graphs in \( \mathcal{L}_{TCW}(\mathcal{A}) \).
3. Interpret graphs into trees and reduce to a tree automaton \( \mathcal{B} \).
A new recipe for analyzing timed systems. Given $\mathcal{A}$,

1. Write behaviors as graphs with timing constraints $\mathcal{L}_{TCW}(\mathcal{A})$.
2. Show a bound on width of graphs in $\mathcal{L}_{TCW}(\mathcal{A})$.
3. Interpret graphs into trees and reduce to a tree automaton $\mathcal{B}$ s.t., $\text{Realizations}(\mathcal{L}_{TCW}(\mathcal{A})) = \emptyset$ iff $\mathcal{L}(\mathcal{B}) = \emptyset$. 

Conclusion and Future work

A common framework for timed, pushdown, multi-pushdown automata with bounded rounds.

Robust framework: diagonal guards, etc.

Future work

Concurrent recursive timed programs

MSO definability of realizability

Going beyond emptiness. What about model-checking?
Conclusion and Future work

A new recipe for analyzing timed systems. Given $\mathcal{A}$,

1. Write behaviors as graphs with timing constraints $\mathcal{L}_{TCW}(\mathcal{A})$.
2. Show a bound on width of graphs in $\mathcal{L}_{TCW}(\mathcal{A})$.
3. Interpret graphs into trees and reduce to a tree automaton $\mathcal{B}$ s.t., $\text{Realizations}(\mathcal{L}_{TCW}(\mathcal{A})) = \emptyset$ iff $\mathcal{L}(\mathcal{B}) = \emptyset$.

- A common framework for timed, pushdown, multi-pushdown automata with bounded rounds.
A new recipe for analyzing timed systems. Given $\mathcal{A}$,

1. Write behaviors as graphs with timing constraints $\mathcal{L}_{TCW}(\mathcal{A})$.
2. Show a bound on width of graphs in $\mathcal{L}_{TCW}(\mathcal{A})$.
3. Interpret graphs into trees and reduce to a tree automaton $\mathcal{B}$ s.t., $\text{Realizations}(\mathcal{L}_{TCW}(\mathcal{A})) = \emptyset$ iff $\mathcal{L}(\mathcal{B}) = \emptyset$.

- A common framework for timed, pushdown, multi-pushdown automata with bounded rounds.
- Robust framework: diagonal guards, etc.
Conclusion and Future work

A new recipe for analyzing timed systems. Given $\mathcal{A}$,

1. Write behaviors as graphs with timing constraints $\mathcal{L}_{TCW}(\mathcal{A})$.
2. Show a bound on width of graphs in $\mathcal{L}_{TCW}(\mathcal{A})$.
3. Interpret graphs into trees and reduce to a tree automaton $\mathcal{B}$ s.t., $\text{Realizations}(\mathcal{L}_{TCW}(\mathcal{A})) = \emptyset$ iff $\mathcal{L}(\mathcal{B}) = \emptyset$.

- A common framework for timed, pushdown, multi-pushdown automata with bounded rounds.
- Robust framework: diagonal guards, etc.

Future work

- Concurrent recursive timed programs
- MSO definability of realizability
- Going beyond emptiness. What about model-checking?