Verification of Timed Asynchronous Programs

Parosh Abdulla\textsuperscript{1}, Mohamed Faouzi Atig\textsuperscript{1}, Krishna S\textsuperscript{2}, Shaan Vaidya\textsuperscript{2}
Dec 13, FSTTCS 18, Ahmedabad

\textsuperscript{1} Uppsala University, Sweden
parosh@it.uu.se, mohamed_faouzi.atig@it.uu.se

\textsuperscript{2} IIT Bombay
krishnas@cse.iitb.ac.in, shaan@cse.iitb.ac.in
Table of contents

1. Introduction
2. Model
3. Verification Problems
4. Special subclass
5. Conclusion
Introduction
Asynchronous Programs

Widely used in building efficient and responsive software.
Asynchronous Programs

Widely used in building efficient and responsive software

Jobs broken up into tasks that are assigned to parallel threads
Asynchronous Programs

Widely used in building efficient and responsive software

Jobs broken up into tasks that are assigned to parallel threads

Asynchronous tasks stored in a buffer, can execute later
Asynchronous Programs

Widely used in building efficient and responsive software

Jobs broken up into tasks that are assigned to parallel threads

Asynchronous tasks stored in a buffer, can execute later

Asynchronous execution can lead to extremely intricate and unpredictable behaviours programs.
Most of the existing work on asynchronous programs considers the untimed version

[Sen, Viswanathan ’06] introduces multiset pushdown systems for recursive asynchronous programs

[Fang et al ’16] introduce timed task automata which are extensions of task automata\(^1\) which have states associated with tasks and on triggering, it is added to a queue

In [Ganty, Majumdar ’09], they consider timed constraints on tasks but the model is different from ours

They show that the safety checking for their model is undecidable

\(^1\)Fersman et al. 2007; Norstrom et al. 1999; Fersman et al. 2002
Main contribution

N-Multiset Timed Automata

- Control State Reachability: Decidable
- Configuration Reachability: Undecidable

Stateless & Time-independent

- Control State Reachability: PSPACE-complete
- Configuration Reachability: Decidable
Model
Timed Automata

\[
\ell_1 \xrightarrow{x_1 \in [0, 1)} \ell_2 \xrightarrow{\{x_1\}} \ell_3 \rightarrow \ell_4
\]
Multiset Timed Automata

\[ \ell_1 \xrightarrow{1?\beta} \ell_2 \xrightarrow{1!1(\kappa[2])} \ell_3 \xrightarrow{1!1(\beta)} \ell_4 \]

\[ \ell_2 \xleftarrow{x_1 \in [0, 1]} \]

\[ \ell_1 \xleftarrow{\{x_1\}} \]

\[ \ell_4 \xleftarrow{1!1(\beta)} \]
N - Multiset Timed Automata
N - Multiset Timed Automata

1

\[
1 \quad l_1 \xrightarrow{1?\beta} l_2 \xrightarrow{1!2(\kappa[2])} l_3 \xrightarrow{1!3(\beta)} l_4
\]

\[
\begin{align*}
\{x_1\} & \quad x_1 \in [0, 1] \\
\end{align*}
\]

2

\[
2 \quad l_5 \xrightarrow{\text{nop}_2} l_6 \xrightarrow{2?\zeta} l_7 \xrightarrow{2!3(\zeta[2])} l_8
\]

\[
\{x_2\} \quad \{x_2\} \\
\]

\[
2!2(\zeta)
\]

\[
2?\zeta
\]

\[
\{x_2\}
\]

\[
2!3(\zeta[2])
\]
N - Multiset Timed Automata

1

2

3

\begin{align*}
\ell_5 & \overset{\text{nop}_2}{\longrightarrow} \ell_6 \\
\ell_6 & \overset{2?\zeta}{\longrightarrow} \ell_7 \\
\ell_7 & \overset{2!3(\zeta[2])}{\longrightarrow} \ell_8 \\
\end{align*}

\begin{align*}
\ell_1 & \overset{1?\beta}{\longrightarrow} \ell_2 \\
\ell_2 & \overset{1!2(\kappa[2])}{\longrightarrow} \ell_3 \\
\ell_3 & \overset{1!3(\beta)}{\longrightarrow} \ell_4 \\
\end{align*}

\begin{align*}
\ell_9 & \overset{3!3(\zeta[1])}{\longrightarrow} \ell_{10} \\
\ell_{10} & \overset{3?\zeta}{\longrightarrow} \ell_{11} \\
\ell_{11} & \overset{3!2(\zeta)}{\longrightarrow} \ell_{12} \\
\end{align*}
Configuration of N-MTA

Configurations of the Timed Automata
(state + clock valuation)

+ Multiset Configurations
Verification Problems
Two Problems

Control State
Reachability
Two Problems

Control State Reachability

Configuration Reachability
Two Problems

Control State Reachability
Can a particular tuple of states be reached?

Configuration Reachability
Two Problems

Control State Reachability
Can a particular tuple of states be reached?

Configuration Reachability
Can a particular tuple of states with empty multisets be reached?
Claim. Control State Reachability is decidable for N-MTA
Idea. Reduction to the coverability problem for Timed Petri Nets with read-arcs (RTPN)
Timed Petri Nets with read arcs
Reduction from N-MTA to RTPN

States \[\leftrightarrow\] one place each

To simulate the state of the automata, only one place per automata is marked at a time
Reduction from N-MTA to RTPN

States $\leftrightarrow$ one place each

Clocks $\leftrightarrow$ one place each
Reduction from N-MTA to RTPN

States $\iff$ one place each

Clocks $\iff$ one place each

Bags $\iff$ multiple places for each

A place for each multiset, for each action, for each (integer) deadline value: $\{0, 1, \ldots, d_{max}, \infty\}$
Reduction from N-MTA to RTPN

States $\leftrightarrow$ one place each

Clocks $\leftrightarrow$ one place each

Bags $\leftrightarrow$ multiple places for each

Transitions $\leftrightarrow$ Normal arcs

When picking a task, check deadline as constraint on arc
Reduction from N-MTA to RTPN

<table>
<thead>
<tr>
<th>States</th>
<th>↔</th>
<th>one place each</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clocks</td>
<td>↔</td>
<td>one place each</td>
</tr>
<tr>
<td>Bags</td>
<td>↔</td>
<td>multiple places for each</td>
</tr>
<tr>
<td>Transitions</td>
<td>↔</td>
<td>Normal arcs</td>
</tr>
<tr>
<td>Clock Constraints</td>
<td>↔</td>
<td>Read arcs</td>
</tr>
</tbody>
</table>
Claim. Configuration Reachability is undecidable for N-MTA

Idea. Reduction from the reachability problem for a 2-counter machine
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

- States of 1-MTA simulate states of the 2-counter machine
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

- States of 1-MTA simulate states of the 2-counter machine
- Two types of tasks: number of tasks in bag simulates that particular counter value
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

• States of 1-MTA simulate states of the 2-counter machine
• Two types of tasks: number of tasks in bag simulates that particular counter value
• Increment / Decrement - add or consume a task

Zero tests guessed correctly if empty pending tasks at the end
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

• States of 1-MTA simulate states of the 2-counter machine
• Two types of tasks: number of tasks in bag simulates that particular counter value
• Increment / Decrement - add or consume a task
• Zero test for counter 1 -
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

- States of 1-MTA simulate states of the 2-counter machine
- Two types of tasks: number of tasks in bag simulates that particular counter value
- Increment / Decrement - add or consume a task
- Zero test for counter 1 -
  - Check \( x = 0 \)
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

• States of 1-MTA simulate states of the 2-counter machine
• Two types of tasks: number of tasks in bag simulates that particular counter value
• Increment / Decrement - add or consume a task
• Zero test for counter 1 -
  • Check $x = 0$
  • Consume all tasks $2[0]$ and add tasks $2[1]$
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

- States of 1-MTA simulate states of the 2-counter machine
- Two types of tasks: number of tasks in bag simulates that particular counter value
- Increment / Decrement - add or consume a task
- Zero test for counter 1 -
  - Check \( x = 0 \)
  - Consume all tasks 2[0] and add tasks 2[1]
  - Change state while checking \( x = 0 \)
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

- States of 1-MTA simulate states of the 2-counter machine
- Two types of tasks: number of tasks in bag simulates that particular counter value
- Increment / Decrement - add or consume a task
- Zero test for counter 1 -
  - Check $x = 0$
  - Consume all tasks 2[0] and add tasks 2[1]
  - Change state while checking $x = 0$
  - Check $x = 1$ and reset clock
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

- States of 1-MTA simulate states of the 2-counter machine
- Two types of tasks: number of tasks in bag simulates that particular counter value
- Increment / Decrement - add or consume a task
- Zero test for counter 1 -
  - Check $x = 0$
  - Consume all tasks 2[0] and add tasks 2[1]
  - Change state while checking $x = 0$
  - Check $x = 1$ and reset clock
  - Consume all tasks 2[1] and add as many 2[0] back
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

• States of 1-MTA simulate states of the 2-counter machine
• Two types of tasks: number of tasks in bag simulates that particular counter value
• Increment / Decrement - add or consume a task
• Zero test for counter 1 -
  • Check $x = 0$
  • Consume all tasks 2[0] and add tasks 2[1]
  • Change state while checking $x = 0$
  • Check $x = 1$ and reset clock
  • Consume all tasks 2[1] and add as many 2[0] back
  • Check $x = 0$, go to normal execution
Reduction from 2-counter machines

Reduce reachability of 2-counter machines to configuration reachability of a 1-MTA

- States of 1-MTA simulate states of the 2-counter machine
- Two types of tasks: number of tasks in bag simulates that particular counter value
- Increment / Decrement - add or consume a task
- Zero test for counter 1 -
  - Check $x = 0$
  - Consume all tasks 2[0] and add tasks 2[1]
  - Change state while checking $x = 0$
  - Check $x = 1$ and reset clock
  - Consume all tasks 2[1] and add as many 2[0] back
  - Check $x = 0$, go to normal execution

- Zero tests guessed correctly if empty pending tasks at the end
Special subclass
Stateless and Time-Independent

Stateless
Stateless and Time-Independent

Stateless

Time-independent
Stateless and Time-Independent

**Stateless**
Unique state per automata for picking up a task

**Time-independent**
Stateless and Time-Independent

Stateless
Unique state per automata for picking up a task

Time-independent
Clocks reset before picking up a task
Claim. Control State Reachability is PSPACE-complete for stateless & time-independent N-MTA
Claim. Control State Reachability is PSPACE-complete for stateless & time-independent N-MTA

Idea.

PSPACE-hardness since N-MTA subsume timed automata
**Claim.** Control State Reachability is PSPACE-complete for stateless & time-independent N-MTA

**Idea.**

- PSPACE-hardness since N-MTA subsume timed automata
- Reduction to a PSPACE-complete problem: coverability of 1-safe RTPNs
Claim. Control State Reachability is PSPACE-complete for stateless & time-independent N-MTA

Idea.

PSPACE-hardness since N-MTA subsume timed automata

Reduction to a PSPACE-complete problem: coverability of 1-safe RTPNs

- Number of relevant tasks in any bag at any point of time in the run is bounded
Claim. Control State Reachability is PSPACE-complete for stateless & time-independent N-MTA

Idea.

PSPACE-hardness since N-MTA subsume timed automata

Reduction to a PSPACE-complete problem: coverability of 1-safe RTPNs

- Number of relevant tasks in any bag at any point of time in the run is bounded
- Once bound is established, construct 1-safe RTPN
Run. Sequence of time-elapse and discrete transitions
Given a run $\sigma$, define $\sigma_i$ as the sequence of transitions corresponding to a particular automata $i$. 
Bounding the number of relevant tasks

**Block** in $\sigma_i$. Transitions following picking up a task before picking up the next task.
Bounding the number of relevant tasks

**Block** in $\sigma_i$. Transitions following picking up a task before picking up the next task

Label all transitions with its block label
Bounding the number of relevant tasks

Relevance of tasks
• Since we only care about the final control state that is reached, if a block does not affect the final state of any automata, we can ignore it
Relevance of tasks

• Since we only care about the final control state that is reached, if a block does not affect the final state of any automata, we can ignore it.

• If a control state can be reached starting from a configuration, it can also be reached starting from a *larger* configuration.
Relevance of tasks

• Since we only care about the final control state that is reached, if a block does not affect the final state of any automata, we can ignore it.
• If a control state can be reached starting from a configuration, it can also be reached starting from a larger configuration.
• Starting backwards, one can construct a dependency graph i.e. which blocks affect the final state.
Bounding the number of relevant tasks

- At any point in the run, the total number of relevant tasks in all bags $\leq N$
• At any point in the run, the total number of relevant tasks in all bags $\leq N$
• This is because the relevance of blocks is based on which task added which task which eventually resulted in the final control state
• At any point in the run, the total number of relevant tasks in all bags $\leq N$
• This is because the relevance of blocks is based on *which task added which task* which eventually resulted in the final control state
• There cannot be more than one task in the bag whose blocks resulted in the final control state of the same automata
Reduction to 1-safe RTPN

Similar to the previous construction of RTPN but ...
Similar to the previous construction of RTPN but ...

Since the number of relevant tasks is bounded, we have multiple 1-safe places to store them
Similar to the previous construction of RTPN but ...

Since the number of relevant tasks is bounded, we have multiple 1-safe places to store them

While picking a task, can pick from any of the $N$ copies
Reduction to 1-safe RTPN

Similar to the previous construction of RTPN but ...

Since the number of relevant tasks is bounded, we have multiple 1-safe places to store them

While picking a task, can pick from any of the \( N \) copies

While adding a task, non-deterministically add to any one of the multiple places
Similar to the previous construction of RTPN but ...

Since the number of relevant tasks is bounded, we have multiple 1-safe places to store them.

While picking a task, can pick from any of the $N$ copies.

While adding a task, non-deterministically add to any one of the multiple places.

Choose to not add a task non-deterministically (guess it to be not relevant).
Claim. Configuration Reachability is decidable for stateless & time-independent N-MTA

Idea.

WQO over the configurations of the N-MTA

Karp-Miller style algorithm for reachability
• Potentially infinite configurations (state + clock valuation + multiset config)

• Regions form a WQO
• Potentially infinite configurations (state + clock valuation + multiset config)

• But if they agree on the integral parts of the clocks, ages of tasks and the ordering of fractional parts of the clocks, ages of tasks: very similar!
• Potentially infinite configurations (state + clock valuation + multiset config)
• But if they agree on the integral parts of the clocks, ages of tasks and the ordering of fractional parts of the clocks, ages of tasks: very similar!

Region = (clocks + tasks with fractional part = 0) + (clocks + tasks with smallest fractional part) + (clocks + tasks with second smallest fractional part) + ... + (clocks + tasks with ages larger than the max value)
Regions

- Potentially infinite configurations (state + clock valuation + multiset config)
- But if they agree on the integral parts of the clocks, ages of tasks and the ordering of fractional parts of the clocks, ages of tasks: very similar!
- Regions form a WQO

Region = (clocks + tasks with fractional part = 0) + (clocks + tasks with smallest fractional part) + (clocks + tasks with second smallest fractional part) + ... + (clocks + tasks with ages larger than the max value)
Algorithm for decidability

- Start with initial region and add it to a set
- Pick an unmarked region from the set, add its successors to the set and mark the current region
- In the set, at any point, if there is a region larger than another region in the set, remove it

Termination guaranteed from the WQO property of regions

Works because if an empty multiset can be reached from a configuration, it can also be reached from a smaller configuration
Statelessness

2-MTA which is not stateless
2-MTA which is not stateless

\[ c_1 = (\ell_1, \ell_6, \{\beta_1, \beta_3\}, \emptyset) \preceq (\ell_1, \ell_6, \{\beta_1, \beta_2, \beta_3\}, \emptyset) = c_2 \]
Statelessness

2-MTA which is not stateless

\[ c_1 = (l_1, l_6, \{\beta_1, \beta_3\}, \emptyset) \preceq (l_1, l_6, \{\beta_1, \beta_2, \beta_3\}, \emptyset) = c_2 \]

From \( c_2 \), one can reach \((l_4, l_6, \emptyset, \emptyset)\), but not from \( c_1 \)
Time-independence

2-MTA which is not time independent
Time-independence

\[ c_1 = \left( (s_1, 0), s_6 \right), \{ (\beta_1, 0, \infty), (\beta_3, 0, \infty) \}, \emptyset \]
Time-independence

$\{x\}$ $x \in [1, \infty)$ $\beta_2$

$S_1$ $S_3$

$\beta_1 \quad x = 0 \quad x \in [1, 1]$ $\beta_3 \quad x \in (0, 1)$

$S_2$ $S_4$

2-MTA which is not time independent

$c_1 = (((S_1, 0), S_6), \{(\beta_1, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset)$

$c_2 = (((S_1, 0), S_6), \{(\beta_1, 0, \infty), (\beta_2, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset)$
2-MTA which is not time independent

\[ c_1 = (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset) \]

\[ c_2 = (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_2, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset) \]

\[ c_1 \preceq c_2 \]

(((s_1, 0), s_6), \emptyset, \emptyset) is reachable from \(c_2\) but not from \(c_1\).
Conclusion
Conclusion

N-Multiset Timed Automata

- Control State Reachability: Decidable
- Configuration Reachability: Undecidable
- Stateless & Time-independent
  - Control State Reachability: PSPACE-complete
  - Configuration Reachability: Decidable
Future Work

Priority for tasks
Future Work

Priority for tasks

Recursive programs: Timed pushdown automata
Future Work

Priority for tasks

Recursive programs: Timed pushdown automata

Schedulability?
Questions?