Verification of Timed Asynchronous Programs

Parosh Abdulla[1], Mohamed Faouzi Atig[1], Krishna S[2], **Shaan Vaidya**[2]

Dec 13, FSTTCS 18, Ahmedabad

1 Uppsala University, Sweden
parosh@it.uu.se, mohamed_faouzi.atig@it.uu.se

2 IIT Bombay
krishnas@cse.iitb.ac.in, shaan@cse.iitb.ac.in

Table of contents

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.

Related Work

Most of the existing work on asynchronous programs considers the untimed version

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

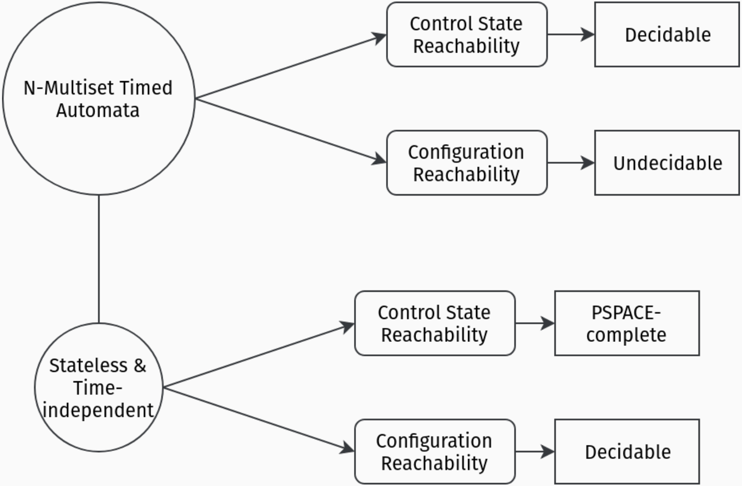
[Fang et al ’16] introduce timed task automata which are extensions of task automata[1]which have states associated with tasks and on triggering, it is added to a queue

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

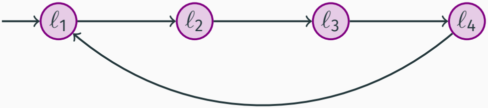
[1]Fersman et al. 2007; Norstrom et al. 1999; Fersman et al. 2002

Main contribution

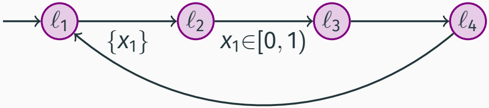


Model

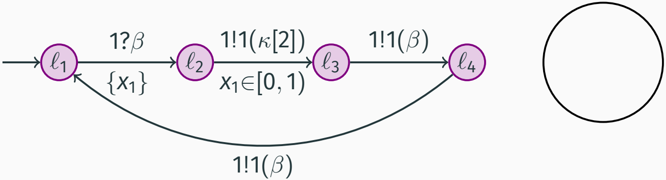
Automata



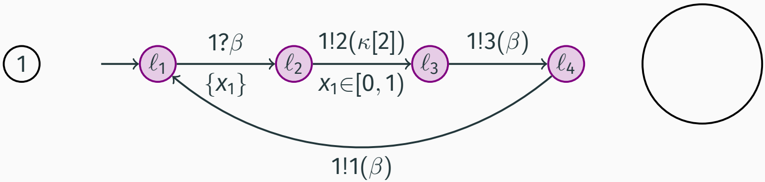
Timed Automata



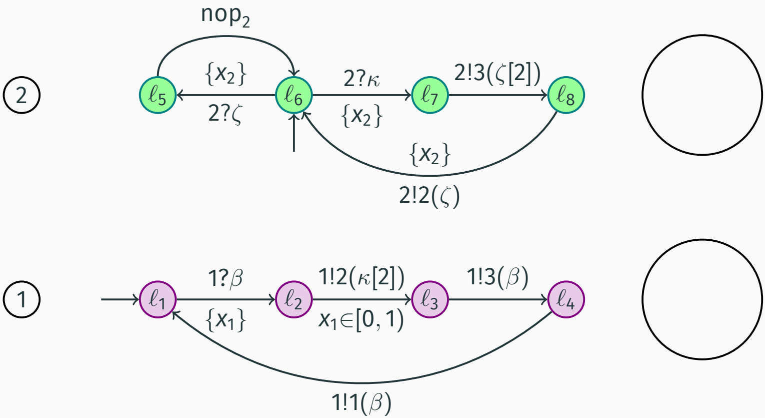
Multiset Timed Automata



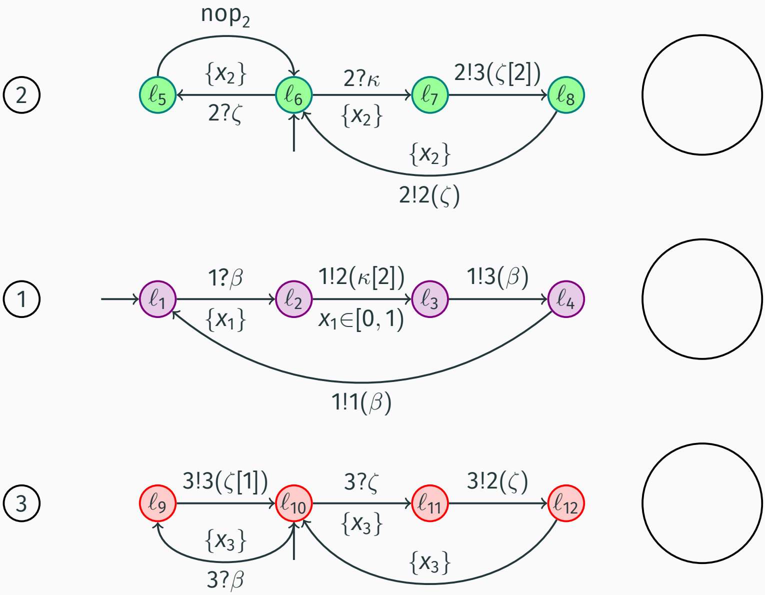
N - Multiset Timed Automata



N - Multiset Timed Automata



N - Multiset Timed Automata



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?

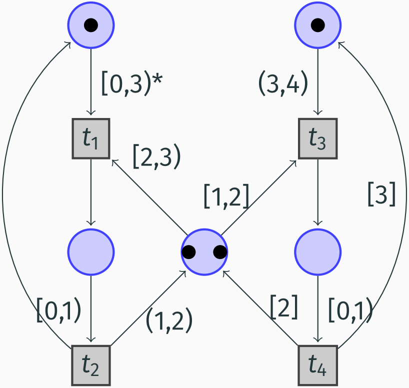
Configuration Reachability

Can a particular tuple of states with empty multisets be reached?

Control State Reachability

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 $\longleftrightarrow$ 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	$\longleftrightarrow$	one place each
Clocks	$\longleftrightarrow$	one place each

Reduction from N-MTA to RTPN

States	$\longleftrightarrow$	one place each
Clocks	$\longleftrightarrow$	one place each
Bags	$\longleftrightarrow$	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	$\longleftrightarrow$	one place each
Clocks	$\longleftrightarrow$	one place each
Bags	$\longleftrightarrow$	multiple places for each
Transitions	$\longleftrightarrow$	Normal arcs

When picking a task, check deadline as constraint on arc

Reduction from N-MTA to RTPN

States	$\longleftrightarrow$	one place each
Clocks	$\longleftrightarrow$	one place each
Bags	$\longleftrightarrow$	multiple places for each
Transitions	$\longleftrightarrow$	Normal arcs
Clock Constraints	$\longleftrightarrow$	Read arcs

Configuration Reachability

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

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

Control State Reachability

Claim. Control State Reachability is PSPACE-complete for stateless & time-independent N-MTA

Control State Reachability

Claim. Control State Reachability is PSPACE-complete for stateless & time-independent N-MTA
Idea.

PSPACE-hardness since N-MTA subsume timed automata

Control State Reachability

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

Control State Reachability

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

Control State Reachability

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

Bounding the number of relevant tasks

Run. Sequence of time-elapse and discrete transitions

Bounding the number of relevant tasks

Given a run σ, define σ_i as the sequence of transitions corresponding to a particular automata i

Block in σ_i. Transitions following picking up a task before picking up the next task

Bounding the number of relevant tasks

Block in σ_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

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 $\leq N$

Bounding the number of relevant tasks

- 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

Bounding the number of relevant tasks

- 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 ...

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

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

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

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
- Choose to not add a task non-deterministically (guess it to be not *relevant*)

Configuration Reachability

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

Regions

- Potentially infinite configurations (state + clock valuation + multiset config)

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

- 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!
-

$$\begin{aligned}\text{Region} = &\ (\text{clocks} + \text{tasks with fractional part} = 0)\\ &+ (\text{clocks} + \text{tasks with smallest fractional part})\\ &+ (\text{clocks} + \text{tasks with second smallest fractional part})\\ &+ \dots\\ &+ (\text{clocks} + \text{tasks with ages larger than the } \textit{max} \text{ value})\end{aligned}$$

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!
-

$$\begin{aligned}\text{Region} &= (\text{clocks} + \text{tasks with fractional part} = 0)\\ &+ (\text{clocks} + \text{tasks with smallest fractional part})\\ &+ (\text{clocks} + \text{tasks with second smallest fractional part})\\ &+ \dots\\ &+ (\text{clocks} + \text{tasks with ages larger than the } \textit{max} \text{ value})\end{aligned}$$

- Regions form a WQO

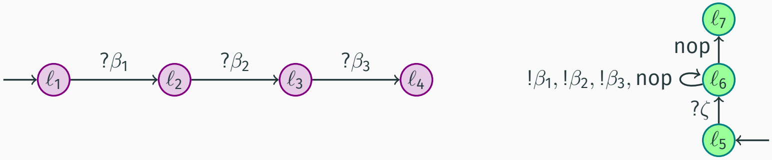
Algorithm for decidability

- 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

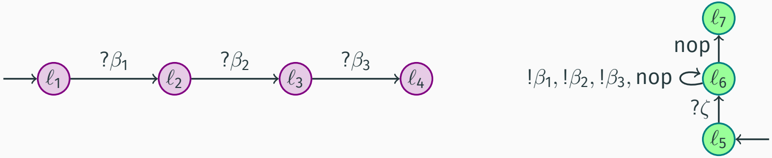
Works because if an empty multiset can be reached from a configuration, it can also be reached from a smaller configuration

Statelessness



2-MTA which is not stateless

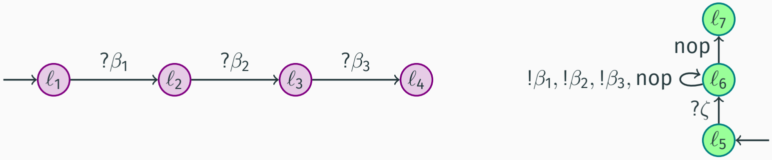
Statelessness



$\boxed{\text{2-MTA which is not stateless}}$

$$\mathfrak{c}_1 = (\ell_1, \ell_6, \{\beta_1, \beta_3\}, \emptyset) \preceq (\ell_1, \ell_6, \{\beta_1, \beta_2, \beta_3\}, \emptyset) = \mathfrak{c}_2$$

Statelessness

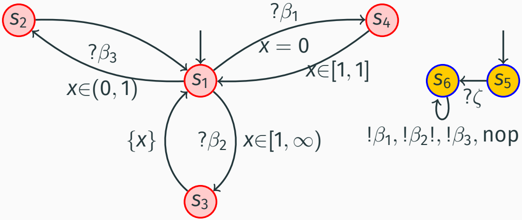


2-MTA which is not stateless

$\mathfrak{c}_1 = (\ell_1, \ell_6, \{\beta_1, \beta_3\}, \emptyset) \preceq (\ell_1, \ell_6, \{\beta_1, \beta_2, \beta_3\}, \emptyset) = \mathfrak{c}_2$

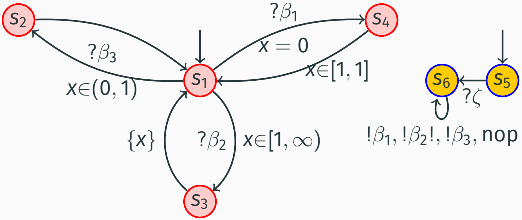
From $\mathfrak{c}_2$, one can reach $(\ell_4, \ell_6, \emptyset, \emptyset)$, but not from $\mathfrak{c}_1$

Time-independence



2-MTA which is not time independent

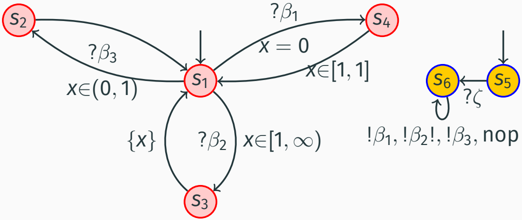
Time-independence



2-MTA which is not time independent

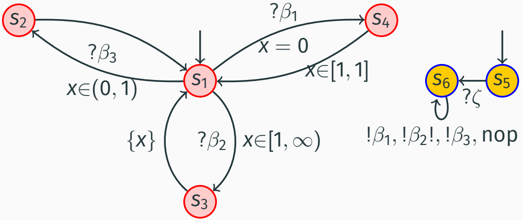
$$\mathfrak{c}_1 {=} (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset)$$

Time-independence



2-MTA which is not time independent

$$\mathfrak{c}_1 = (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset)$$
$$\mathfrak{c}_2 = (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_2, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset)$$



2-MTA which is not time independent

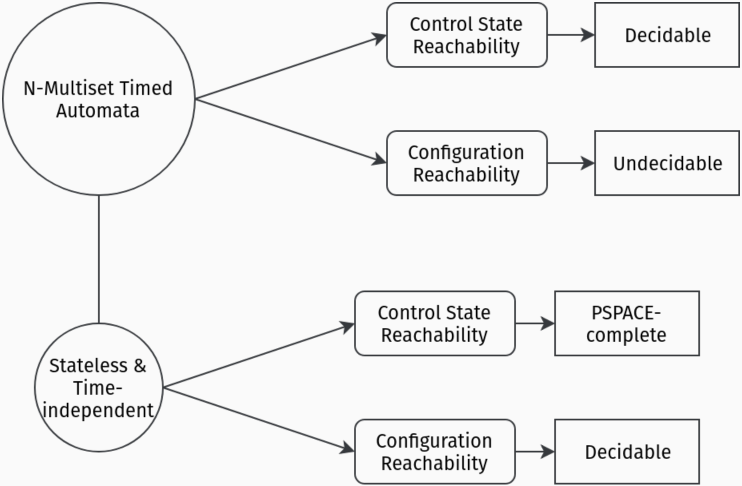
$\mathfrak{c}_1 = (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset)$

$\mathfrak{c}_2 = (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_2, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset)$ $\mathfrak{c}_1 \preceq \mathfrak{c}_2$

$(((s_1, 0), s_6), \emptyset, \emptyset)$ is reachable from $\mathfrak{c}_2$ but not from $\mathfrak{c}_1$.

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

Priority for tasks

Recursive programs: Timed pushdown automata

Future Work

Priority for tasks

Recursive programs: Timed pushdown automata

Schedulability?

Questions?