Layered controller synthesis for dynamic multi-agent systems

Emily Clement[1 3] Nicolas Perrin-Gilbert[1] Philipp Schlehuber-Caissier[2]

[1]Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR, F-75005 Paris, France
[2]EPITA Research Laboratory
[3]Universite Paris Cité, CNRS, IRIF, Paris, France

October 9 2023

Introduction

Dynamic multi-agent system's verification

Our objectives

- A running example

https://perso.eleves.ens-rennes.fr/people/Emily.Clement/Videos/example_episodes/ex_0.mp4

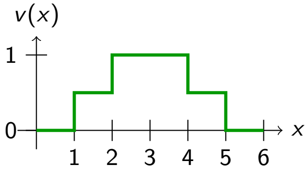
	Timed Automata	Reinforcement Learning
Model	Abstract representation (acceleration)	
Waekness	Time of execution	Combinatorial or Continuous aspects

- Our assumptions
  - **Our control:** the speed of (all) cars.

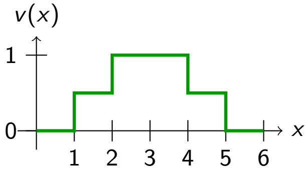
Our layered approach

- Our assumptions
  - **Our control:** the speed of (all) cars.
  - **Goal:** reach goals while avoiding collisions between agents.

Our layered approach

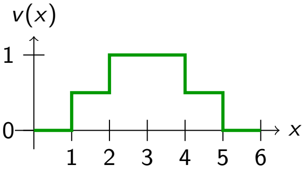
- Our assumptions
  - **Our control:** the speed of (all) cars.
  - **Goal:** reach goals while avoiding collisions between agents.
  - **Speed:** 

Our layered approach

- Our assumptions
  - ▷ **Our control:** the speed of (all) cars.
  - ▷ **Goal:** reach goals while avoiding collisions between agents.
  - ▷ **Speed:** 

  - ▷ **Paths of cars:** fixed trajectories, fixed finals & initial positions.

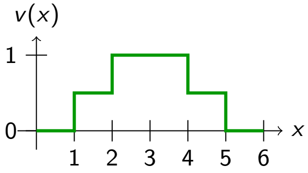
Our layered approach

- Our assumptions
  - **Our control:** the speed of (all) cars.
  - **Goal:** reach goals while avoiding collisions between agents.
  - **Speed:** 

  - **Paths of cars:** fixed trajectories, fixed finals & initial positions.
  - **Trajectories:** we abstract from the curves of the trajectories.

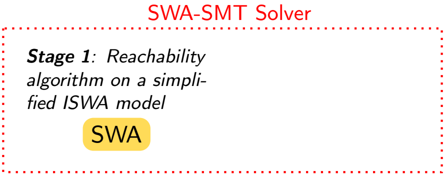
Our layered approach

- Our assumptions
  - **Our control:** the speed of (all) cars.
  - **Goal:** reach goals while avoiding collisions between agents.
  - **Speed:**



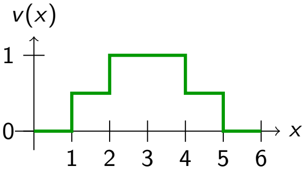
  - **Paths of cars:** fixed trajectories, fixed finals & initial positions.
  - **Trajectories:** we abstract from the curves of the trajectories.

- Our contribution: Three-layered Controller synthesis



Our layered approach

- Our assumptions
  - **Our control:** the speed of (all) cars.
  - **Goal:** reach goals while avoiding collisions between agents.
  - **Speed:**



  - **Paths of cars:** fixed trajectories, fixed finals & initial positions.
  - **Trajectories:** we abstract from the curves of the trajectories.

- Our contribution: Three-layered Controller synthesis

SWA-SMT Solver

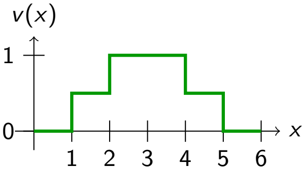
Stage 1: *Reachability algorithm on a simplified ISWA model*

SWA

Stage 2: *Refine the model of the speed*

SMT

Our layered approach

- Our assumptions
  - **Our control:** the speed of (all) cars.
  - **Goal:** reach goals while avoiding collisions between agents.
  - **Speed:** 

  - **Paths of cars:** fixed trajectories, fixed finals & initial positions.
  - **Trajectories:** we abstract from the curves of the trajectories.
- Our contribution: Three-layered Controller synthesis

SWA-SMT Solver

Stage 1: *Reachability algorithm on a simplified ISWA model*

SWA

Stage 2: *Refine the model of the speed*

SMT

RL training

Generate *a dataset for random initial positions*

Dataset

Stage 3: *Train an RL algorithm with our dasaset*

RL

SWA-SMT solver

SWA solver

Stage 1: Reachability algorithm on system of ISWA

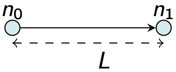
SWA

***Stage 2**: Model the accelleration and deceleration*

SMT

Model for a car traffic

- A point in $\mathbb{R}^2$: a node 

- A section $s_{[n_0, n_1], L}$ of the road: 

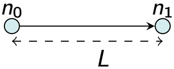
Model for a car traffic

- A point in $\mathbb{R}^2$: a node n_0
- A section $s_{[n_0, n_1], L}$ of the road: $n_0 \xrightarrow{} n_1$, $\underbrace{}_{L}$
- A path: $p_0 : n_0 \to n_1 \to n_3 \to n_4 \to n_6 \to n_{11}$

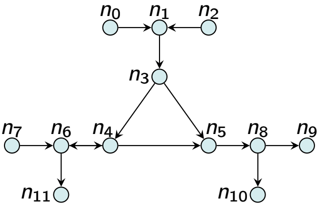
Model for a car traffic

- A point in $\mathbb{R}^2$: a node n_0
- A section $s_{[n_0, n_1], L}$ of the road: $n_0 \xrightarrow{} n_1$, $\underbrace{\longleftrightarrow}_{L}$
- A path: $p_0 : n_0 \to n_1 \to n_3 \to n_4 \to n_6 \to n_{11}$
- Car: (position, speed, trajectory)

- A point in $\mathbb{R}^2$: a node n_0
- A section $s_{[n_0,n_1],L}$ of the road:



- A path: $p_0 : n_0 \rightarrow n_1 \rightarrow n_3 \rightarrow n_4 \rightarrow n_6 \rightarrow n_{11}$
- Car: (position, speed, trajectory)
- A car traffic: c_0, c_1, c_2 are each assigned paths p_0, p_1, p_2:



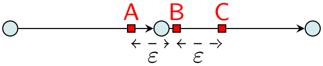
$p_0 : n_0 \rightarrow n_1 \rightarrow n_3 \rightarrow n_4 \rightarrow n_6 \rightarrow n_{11}$

$p_1 : n_2 \rightarrow n_1 \rightarrow n_3 \rightarrow n_5 \rightarrow n_8 \rightarrow n_{10}$

$p_2 : n_7 \rightarrow n_6 \rightarrow n_4 \rightarrow n_5 \rightarrow n_8 \rightarrow n_9$

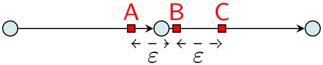
Rules to model collision avoidance

- #1: security distance when driving in the same direction and between neighbouring sections

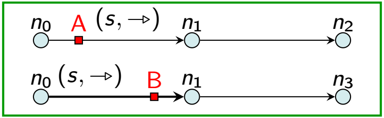


Rules to model collision avoidance

- #1: security distance when driving in the same direction and between neighbouring sections

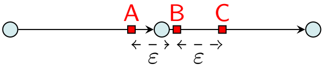


- #2: cars cannot share a section if driving in **opposite** direction

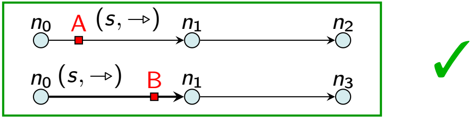


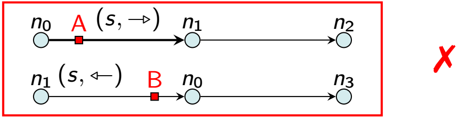
✓

•#1: security distance when driving in the same direction and between neighbouring sections

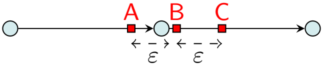


•#2: cars cannot share a section if driving in **opposite** direction

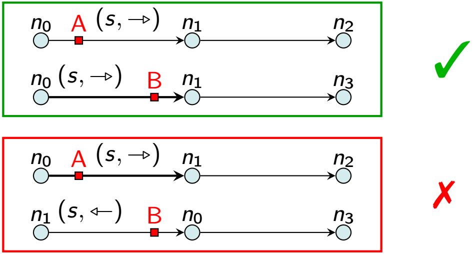




- #1: security distance when driving in the same direction and between neighbouring sections



- #2: cars cannot share a section if driving in **opposite** direction



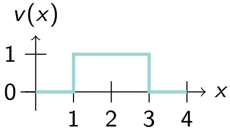
- #3: No Overtaking between cars

What type of Timed Automata to use to model this?

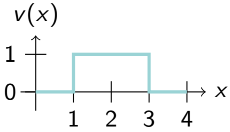
- Needs
  - **Clocks of TA**: Monitor each car's progress.

What type of Timed Automata to use to model this?

- Needs
  - **Clocks of TA**: Monitor each car's progress.
  - **Stopwatch Timed Automata**:

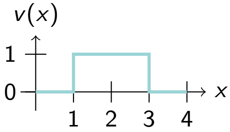


What type of Timed Automata to use to model this?

- Needs
  - ▷ **Clocks of TA**: Monitor each car's progress.
  - ▷ **Stopwatch Timed Automata**: 

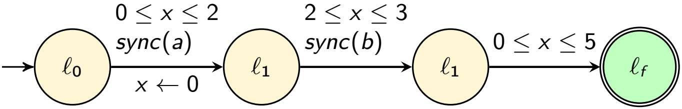
  - ▷ **Synchronised action**: Compute distance between each cars.

What type of Timed Automata to use to model this?

- Needs
  - **Clocks of TA**: Monitor each car's progress.
  - **Stopwatch Timed Automata**: 

  - **Synchronised action**: Compute distance between each cars.
  - **FiFo channels**: A car cannot overtake another car.

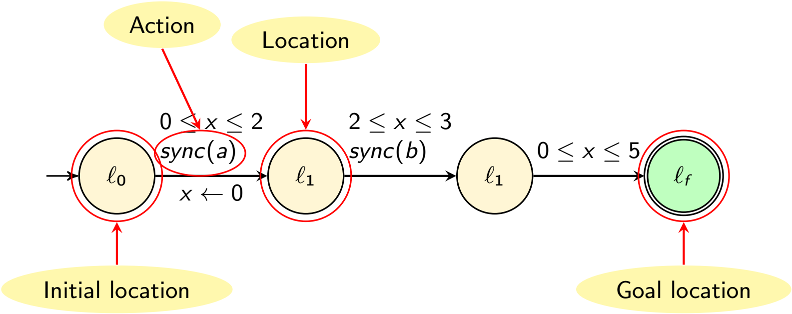
- A classic timed automaton



- Variants

What is a Timed Automaton (and its variants) ?

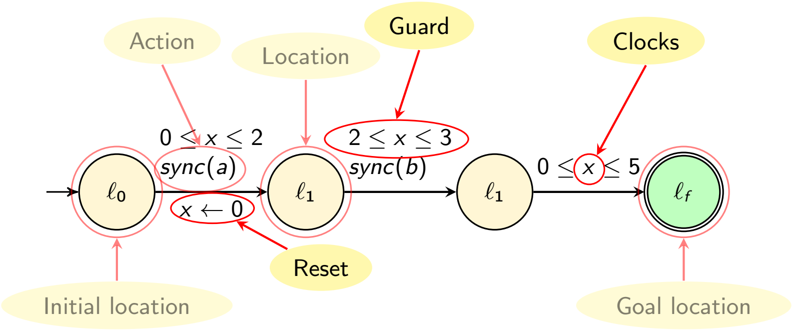
- A classic timed automaton



- Variants

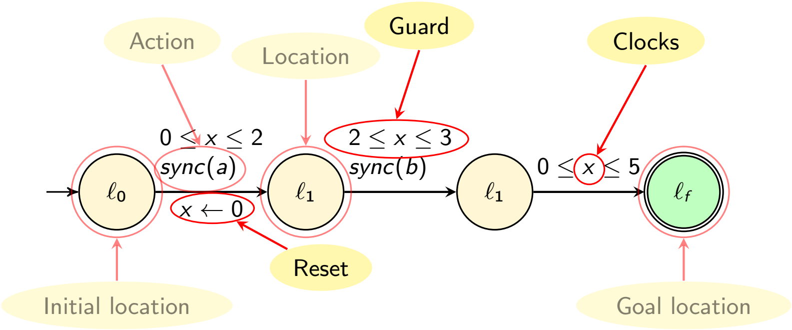
What is a Timed Automaton (and its variants) ?

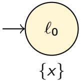
- A classic timed automaton



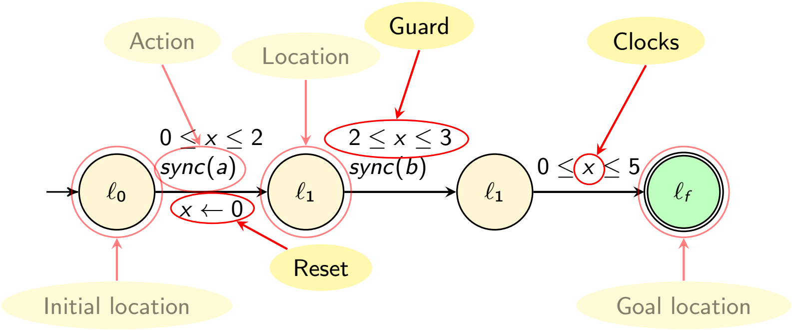
- Variants

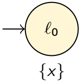
- A classic timed automaton



- Variants
  - ▷ Stopwatch  : clock x is stopped in location ℓ_0.

- A classic timed automaton

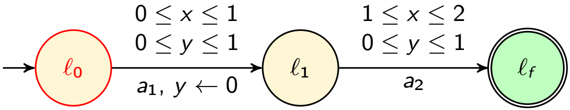


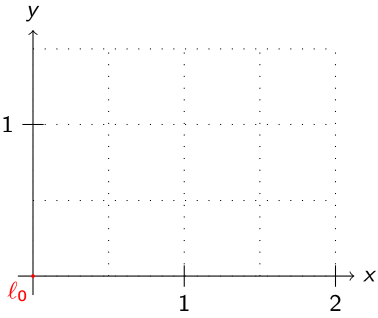
- Variants
  - ▷ Stopwatch 

 : clock x is stopped in location ℓ_0.
  - ▷ Channels: FiFo queue of symbols (actions) to be pushed/read

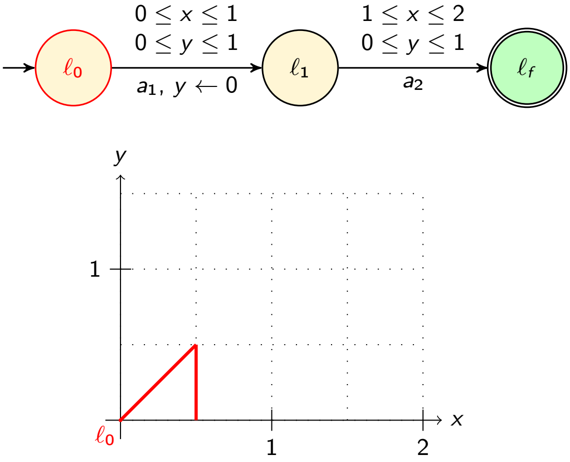
A run of a timed automaton

- A run of a two-clock Timed Automaton

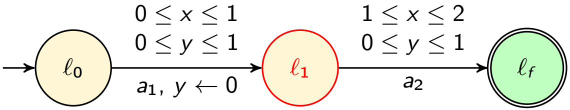


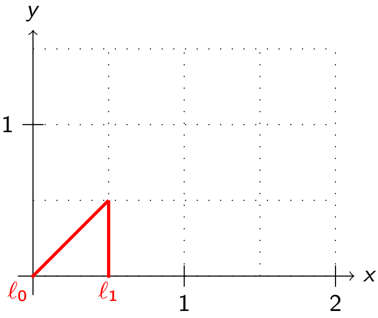


- A run of a two-clock Timed Automaton



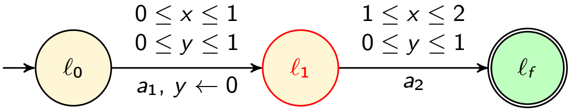
- A run of a two-clock Timed Automaton

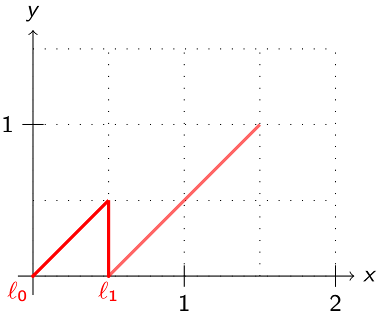




A run of a timed automaton

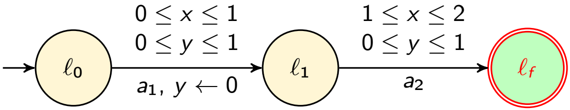
- A run of a two-clock Timed Automaton

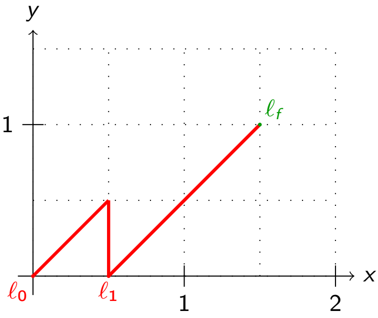




A run of a timed automaton

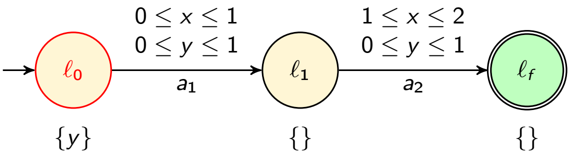
- A run of a two-clock Timed Automaton

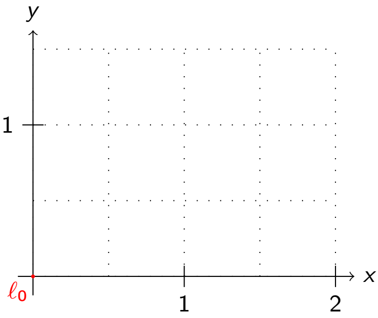




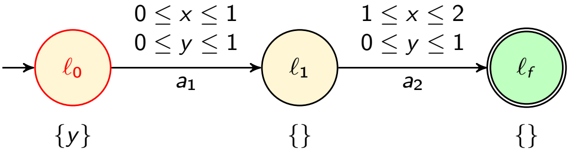
A run of a timed automaton

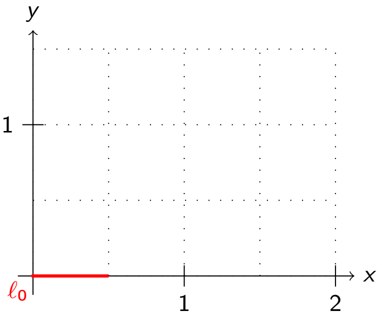
- A run with a (two-clock) stopwatch timed automaton (ISWA)





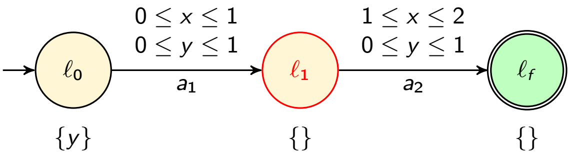
- A run with a (two-clock) stopwatch timed automaton (ISWA)

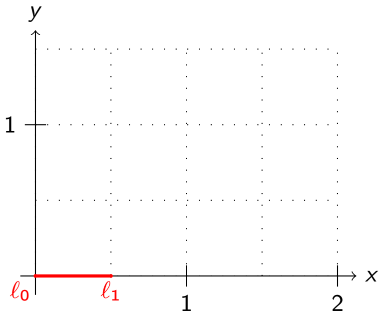




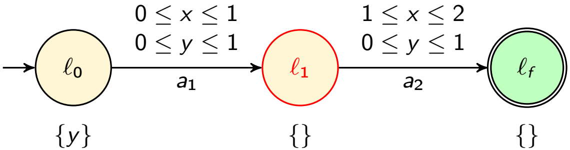
A run of a timed automaton

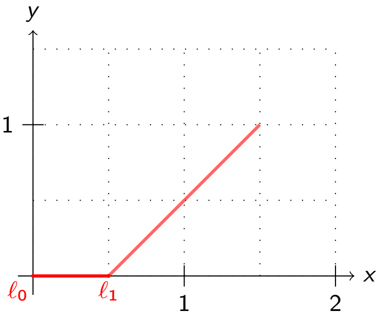
- A run with a (two-clock) stopwatch timed automaton (ISWA)





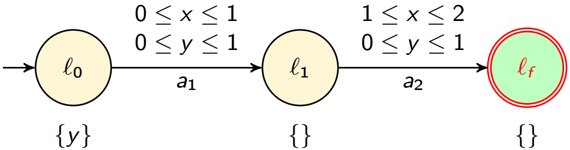
- A run with a (two-clock) stopwatch timed automaton (ISWA)

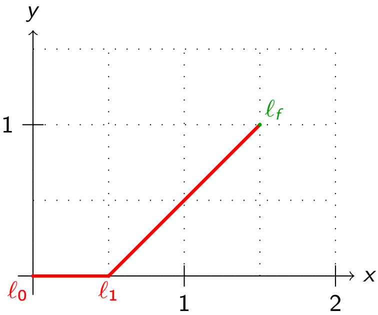




A run of a timed automaton

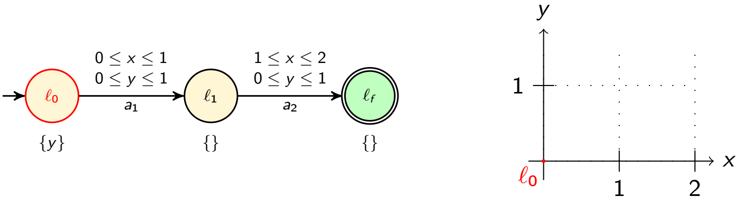
- A run with a (two-clock) stopwatch timed automaton (ISWA)





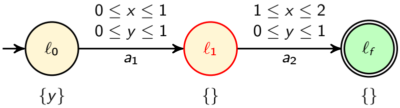
Initialized Stopwatch Timed Automata with bounded channels

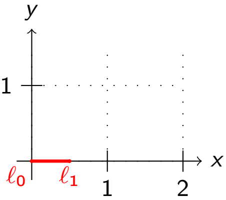
- Example of a two-clocks Stopwatch Timed Automata



Initialized Stopwatch Timed Automata with bounded channels

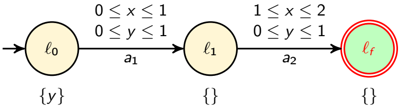
- Example of a two-clocks Stopwatch Timed Automata

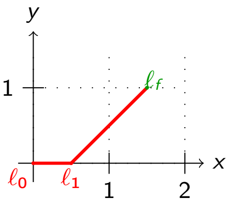




Initialized Stopwatch Timed Automata with bounded channels

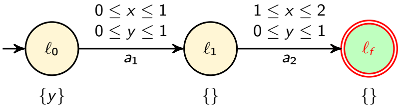
- Example of a two-clocks Stopwatch Timed Automata

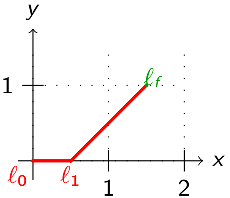




Initialized Stopwatch Timed Automata with bounded channels

- Example of a two-clocks Stopwatch Timed Automata

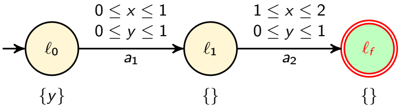


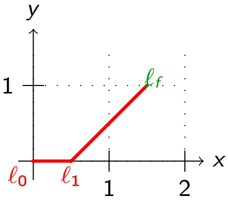


  - ▷ Reachability is Undecidable in general cases.

Initialized Stopwatch Timed Automata with bounded channels

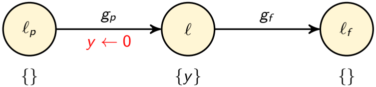
- Example of a two-clocks Stopwatch Timed Automata





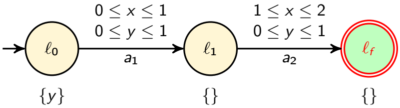
  - ▷ Reachability is Undecidable in general cases.

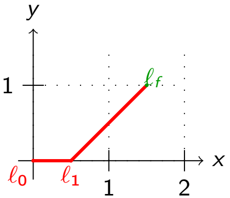
- Initialized Stopwatch Timed Automata
  - ▷ Reset the stopped clock in the previous or following transition:



  - ▷ **Reachability** becomes Decidable for this fragment of SWA.

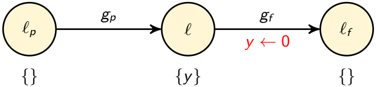
- Example of a two-clocks Stopwatch Timed Automata





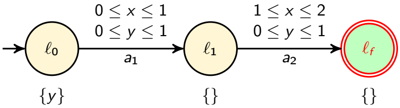
  - ▷ Reachability is Undecidable in general cases.

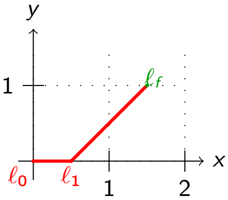
- Initialized Stopwatch Timed Automata
  - ▷ Reset the stopped clock in the previous or following transition:



  - ▷ **Reachability** becomes Decidable for this fragment of SWA.

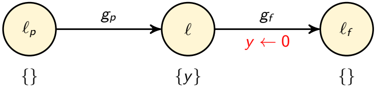
- Example of a two-clocks Stopwatch Timed Automata





  - Reachability is Undecidable in general cases.

- Initialized Stopwatch Timed Automata
  - Reset the stopped clock in the previous or following transition:

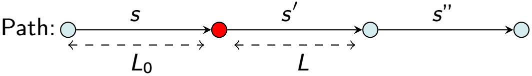


  - **Reachability** becomes Decidable for this fragment of SWA.

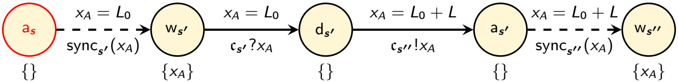
- Bounded channels
  - Channels: FiFo queue of symbols (actions) to be pushed/read

Model the car progress

- Car A progress along its paths



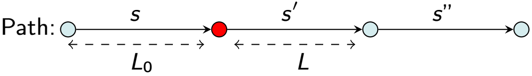
- Car A Timed automaton:



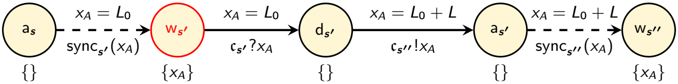
  - **Clock** x_A: distance travelled along its paths
  - **Stopwatches** $\{x_A\}$: the car A stops instantly.
  - **Channels** $\mathfrak{c}_{s'}!x_A/\mathfrak{c}_{s'}?x_A$: respect the order of cars in a section $s \Rightarrow$ no over-taking.
  - **Intersection**: use classical synchronized action to activate *intersection automata*

Model the car progress

- Car *A* progress along its paths

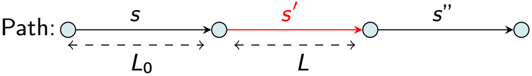


- Car *A* Timed automaton:

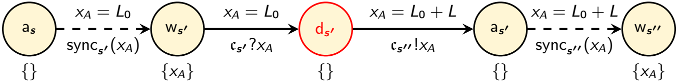


- ▷ **Clock** x_A: distance travelled along its paths
- ▷ **Stopwatches** $\{x_A\}$: the car *A* stops instantly.
- ▷ **Channels** $\mathfrak{c}_{s'}!x_A/\mathfrak{c}_{s'}?x_A$: respect the order of cars in a section $s \Rightarrow$ no overtaking.
- ▷ **Intersection**: use classical synchronized action to activate *intersection automata*

- Car A progress along its paths



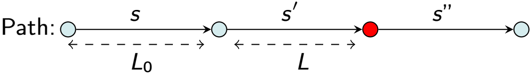
- Car A Timed automaton:



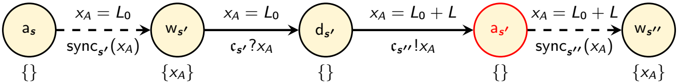
  - **Clock** x_A: distance travelled along its paths
  - **Stopwatches** $\{x_A\}$: the car A stops instantly.
  - **Channels** $\mathfrak{c}_{s'}!x_A/\mathfrak{c}_{s'}?x_A$: respect the order of cars in a section $s \Rightarrow$ no overtaking.
  - **Intersection**: use classical synchronized action to activate *intersection automata*

Model the car progress

- Car A progress along its paths



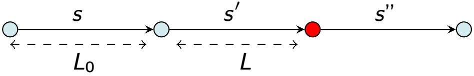
- Car A Timed automaton:



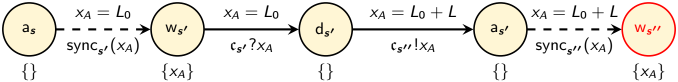
- ▷ **Clock** x_A: distance travelled along its paths
- ▷ **Stopwatches** $\{x_A\}$: the car A stops instantly.
- ▷ **Channels** $\mathfrak{c}_{s'}!x_A/\mathfrak{c}_{s'}?x_A$: respect the order of cars in a section $s \Rightarrow$ no over-taking.
- ▷ **Intersection**: use classical synchronized action to activate *intersection automata*

Model the car progress

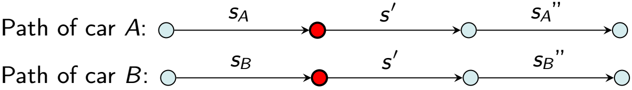
- Car A progress along its paths

Path: 

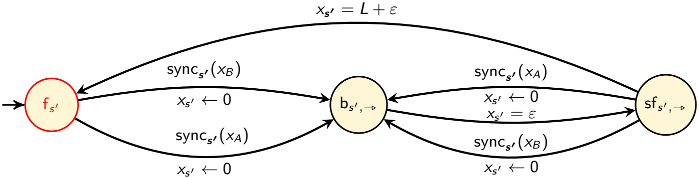
- Car A Timed automaton:



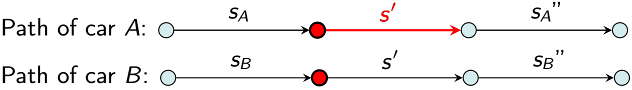
  - **Clock** x_A: distance travelled along its paths
  - **Stopwatches** $\{x_A\}$: the car A stops instantly.
  - **Channels** $\mathfrak{c}_{s'}!x_A/\mathfrak{c}_{s'}?x_A$: respect the order of cars in a section $s \Rightarrow$ no overtaking.
  - **Intersection**: use classical synchronized action to activate *intersection automata*



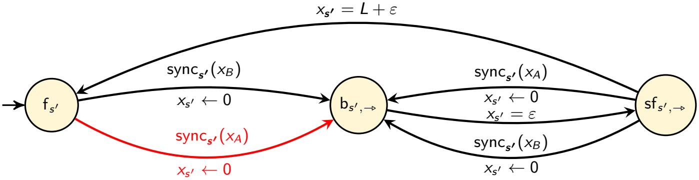
- Intersection automaton



Model distance between cars: intersection



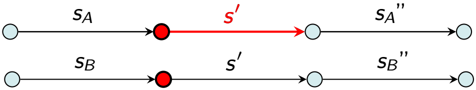
- Intersection automaton



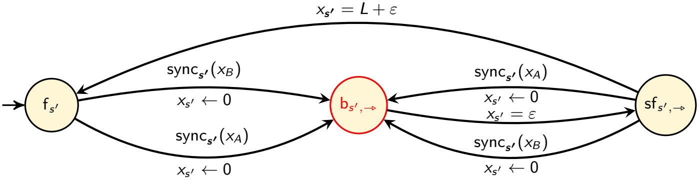
Model distance between cars: intersection

Path of car A:

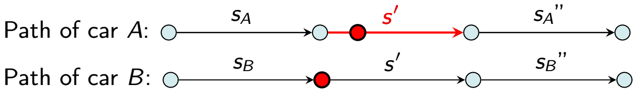
Path of car B:



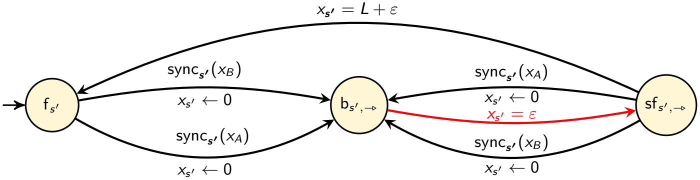
- Intersection automaton

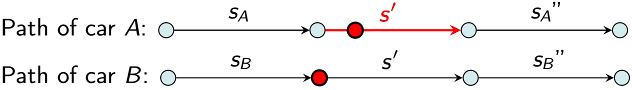


Model distance between cars: intersection

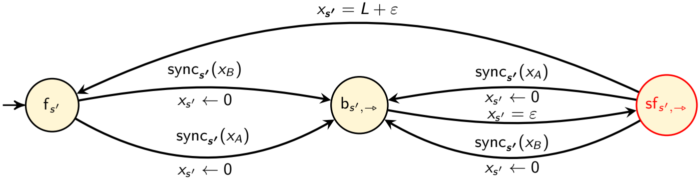


- Intersection automaton

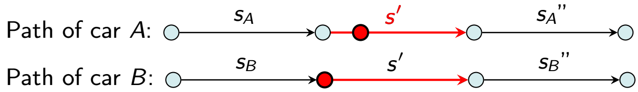




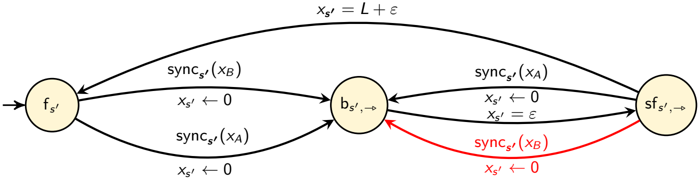
- Intersection automaton

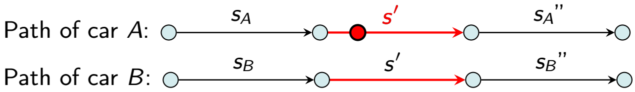


Model distance between cars: intersection

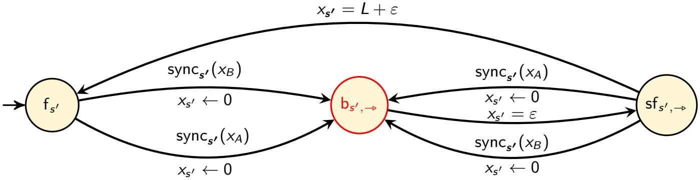


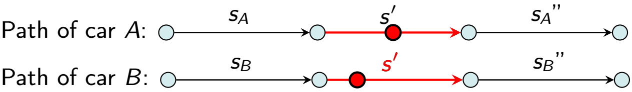
- Intersection automaton



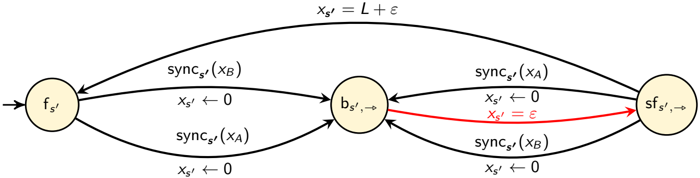


- Intersection automaton

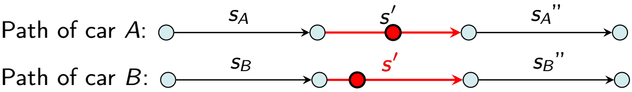




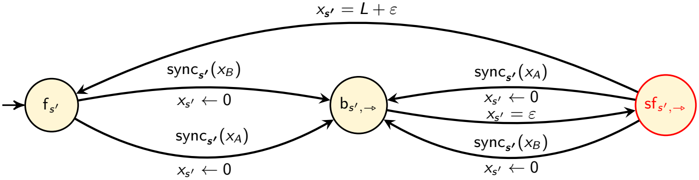
- Intersection automaton



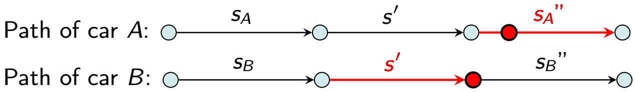
Model distance between cars: intersection



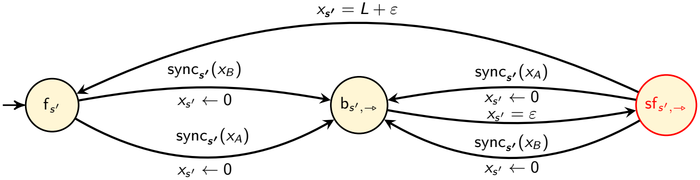
- Intersection automaton

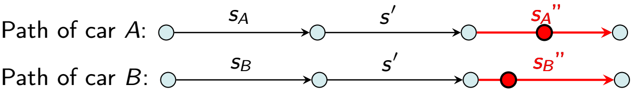


Model distance between cars: intersection

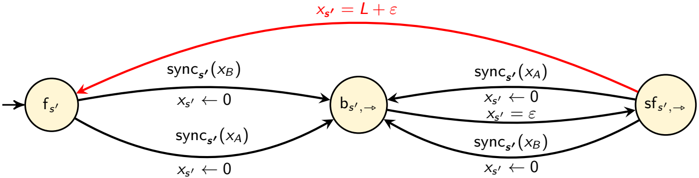


- Intersection automaton



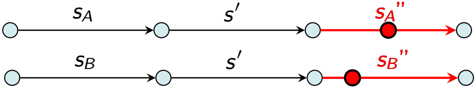


- Intersection automaton

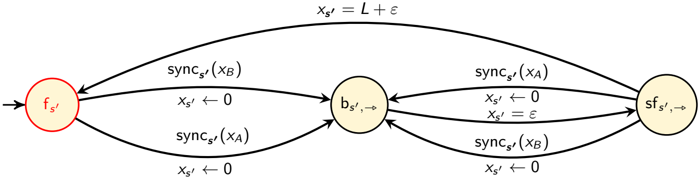


Model distance between cars: intersection

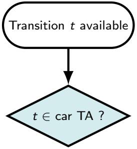
Path of car A:
Path of car B:



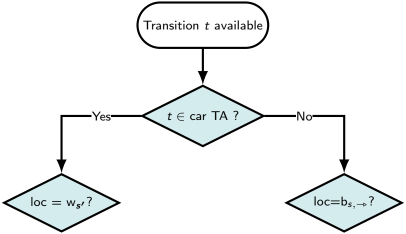
- Intersection automaton



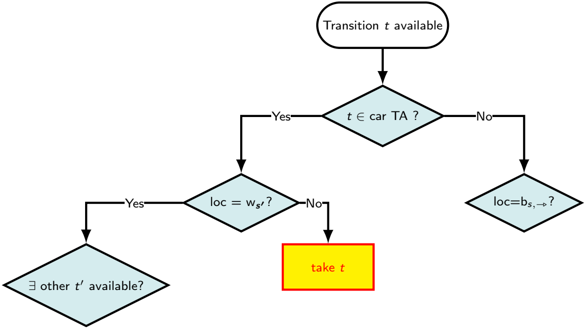
Our Algorithm: a DFS with an optimised succ function



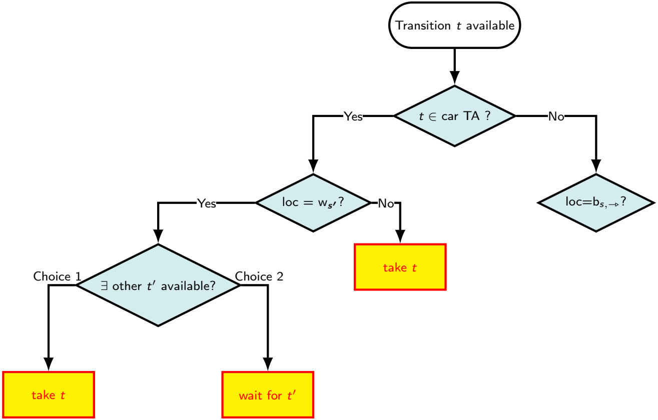
Our Algorithm: a DFS with an optimised succ function



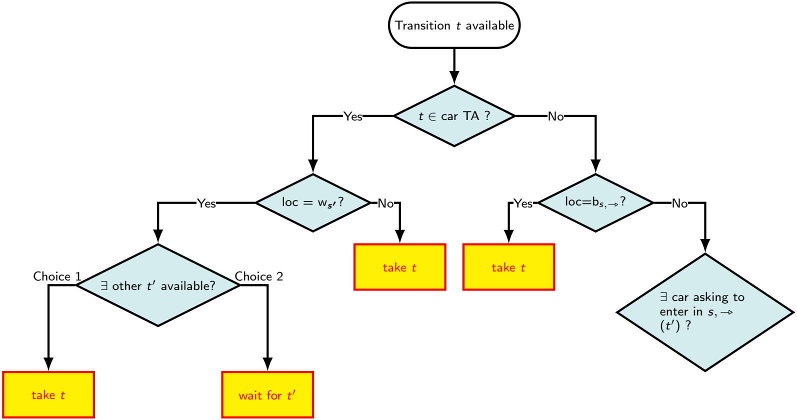
Our Algorithm: a DFS with an optimised succ function



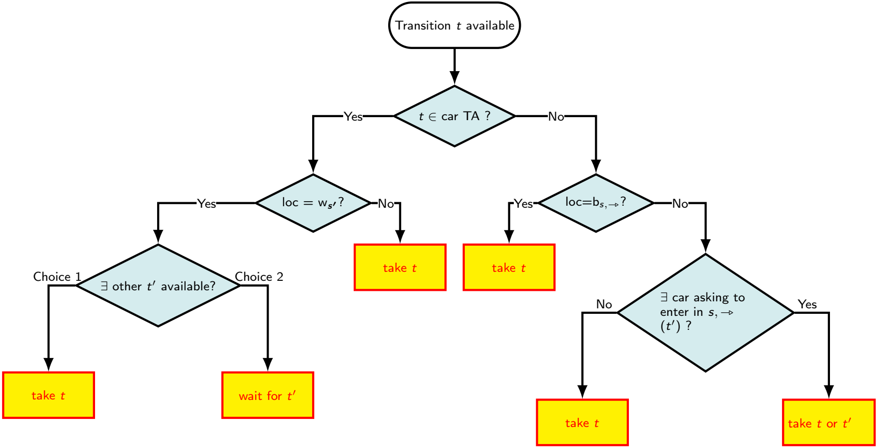
Our Algorithm: a DFS with an optimised succ function



Our Algorithm: a DFS with an optimised succ function



Our Algorithm: a DFS with an optimised succ function



SWA-SMT solver

SMT solver

Stage 1: Reachability algorithm on system of ISWA

SWA

***Stage 2**: Model the accelleration and deceleration*

SMT

Why use of SMT solver?

DFS algorithm

Stage 1: *Reachability algorithm on a simplified ISWA model*

SWA

Solved: combinatorial aspect of the problem. Results: Important events and their relative order

Drawback: A very abstract model of speed

SMT Solver

Stage 2: *Refine the model of the speed*

SMT

RL training

Generate *a dataset for random initial positions*

Dataset

Stage 3: *Train an RL algorithm with our dasaset*

RL

Why use of SMT solver?

DFS algorithm

Stage 1: *Reachability algorithm on a simplified ISWA model*

SWA

Solved: combinatorial aspect of the problem.
Results: Important events and their relative order

Drawback: A very abstract model of speed

SMT Solver

Stage 2: *Refine the model of the speed*

SMT

RL training

Generate *a dataset for random initial positions*

Dataset

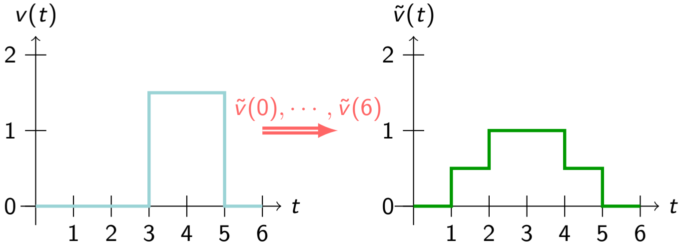
Stage 3: *Train an RL algorithm with our dasaset*

RL

- SMT solver
  - The **continuous** aspect of the problem
  - Introduce a more **realistic** model of speed

- A constant piecewise affine function
  - A more realistic model that takes into account the **dynamic of the system**
  - **Different** car speeds
  - **Bounds** on deceleration and acceleration

$$
\begin{array}{lcl}
v_i(t) & \Rightarrow & \tilde{v}_i(0), \cdots, \tilde{v}_i(k-1) \\
x(t) & \Rightarrow & \tilde{x}_i(k) = \sum_{l=0}^{k-1} \tilde{v}_i(l)
\end{array}
$$



- New positions/speeds
  - $\tilde{x}_i(k) = \sum_{l=0}^{k-1} \tilde{v}_i(l)$
  - $\tilde{v}_i(0), \cdots, \tilde{v}_i(k-1)$

How to preserve security distance ?

- New positions/speeds
  - $\tilde{x}_i(k) = \sum_{l=0}^{k-1} \tilde{v}_i(l)$
  - $\tilde{v}_i(0), \cdots, \tilde{v}_i(k-1)$

- Example of SMT solver's inequalities

For each step k :

  - $\tilde{v}_i(k) - d_{\max} \leq \tilde{v}_i(k+1) \leq \tilde{v}_i(k) + a_{\max}$

- New positions/speeds
  - ▷ $\tilde{x}_i(k) = \sum_{l=0}^{k-1} \tilde{v}_i(l)$
  - ▷ $\tilde{v}_i(0), \cdots, \tilde{v}_i(k-1)$
- Example of SMT solver's inequalities

For each step k :

- ▷ $\tilde{v}_i(k) - d_{\max} \leq \tilde{v}_i(k+1) \leq \tilde{v}_i(k) + a_{\max}$
- ▷ $0 \leq \tilde{v}_i(k) \leq v_{\max}$

RL training

Generate *a dataset for random initial positions*

Dataset

Stage 3*: Train an RL algorithm with our dasaset*

RL

Why use of SMT solver?

DFS algorithm

SMT Solver

RL training

***Stage 1**: Reachability algorithm on a simplified ISWA model*

SWA

Solved: combinatorial aspect of the problem.
Results: Important events and their relative order

Drawback: A very abstract model of speed

***Stage 2**: Refine the model of the speed*

SMT

A more realistic model of speed
Results: traces that takes into account the dynamical aspect of the problem
Drawback: runtime execution

SWA-SMT solver

***Generate** a dataset for random initial positions*

Dataset

***Stage 3**: Train an RL algorithm with our dasaset*

RL

Drawback: our problem has both combinatorial and continuous aspects
Goal: get an intuition from dataset to avoid unsuccessful choices

Why use of SMT solver?

DFS algorithm

Stage 1: Reachability algorithm on a simplified ISWA model

SWA

Solved: combinatorial aspect of the problem.
Results: Important events and their relative order

Drawback: A very abstract model of speed

SMT Solver

Stage 2: Refine the model of the speed

SMT

A more realistic model of speed
Results: traces that takes into account the dynamical aspect of the problem
Drawback: runtime execution

SWA-SMT solver

RL training

Generate a dataset for random initial positions

Dataset

Stage 3: Train an RL algorithm with our dasaset

RL

Drawback: our problem has both combinatorial and continuous aspects
Goal: get an intuition from dataset to avoid unsuccessful choices

- RL training dataset
  - Create random initial positions/speeds for cars
  - Generate traces with the SWA-SMT solver

- Markov Decision Process
  - **Deterministic running example**: deterministic transition function.

Model

- Markov Decision Process
  - **Deterministic running example**: deterministic transition function.
  - **State** s_i. For each section s, if a car c is in s: $v_{i,c}, \mathsf{pos}_{i,c}, \mathsf{id}_\mathsf{c}, 1$

- Markov Decision Process
  - **Deterministic running example**: deterministic transition function.
  - **State** s_i. For each section s, if a car c is in s: $v_{i,c}, \mathrm{pos}_{i,c}, \mathrm{id}_c, 1$
  - **Action** act_i: $(\mathrm{acc}_{i,c})_{c \in \mathbf{Cars}}$

$$\underset{i}{(\mathrm{pos}_{i,c}, v_{i,c})} \dashrightarrow \underset{i+1}{(\mathrm{pos}_{i,c} + v_i, v_{i,c} + \mathrm{acc}_{i,c})}$$

Model

- Markov Decision Process
  - **Deterministic running example**: deterministic transition function.
  - **State** s_i. For each section s, if a car c is in s: $v_{i,c}, \mathrm{pos}_{i,c}, \mathrm{id}_c, 1$
  - **Action** act_i: $(\mathrm{acc}_{i,c})_{c \in \mathrm{Cars}}$

$$\underset{i}{(\mathrm{pos}_{i,c}, v_{i,c})} \dashrightarrow \underset{i+1}{(\mathrm{pos}_{i,c} + v_i, v_{i,c} + \mathrm{acc}_{i,c})}$$

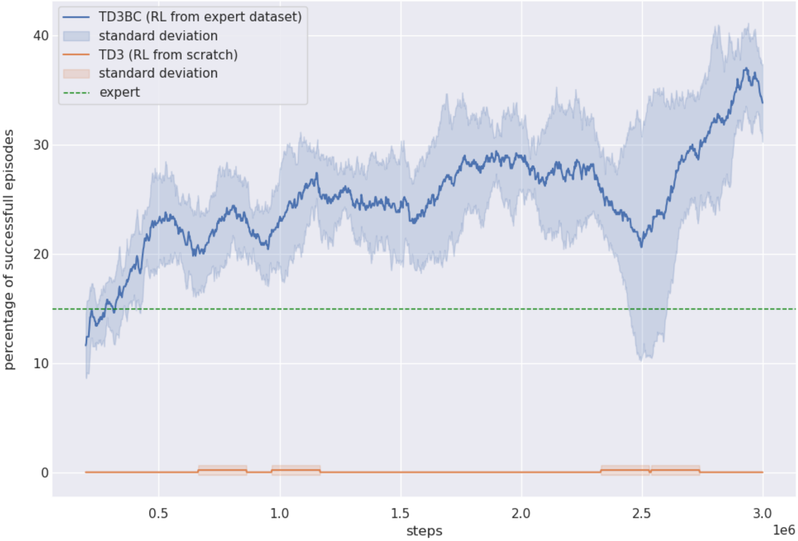
  - **Trajectories** s_i, Obs_i, act_i

- Markov Decision Process
  - **Deterministic running example**: deterministic transition function.
  - **State** s_i. For each section s, if a car c is in s: $v_{i,c}, \mathsf{pos}_{i,c}, \mathsf{id}_c, 1$
  - **Action** act_i: $(\mathsf{acc}_{i,c})_{c \in \mathsf{Cars}}$

$$\underset{i}{(\mathsf{pos}_{i,c}, v_{i,c})} \dashrightarrow \underset{i+1}{(\mathsf{pos}_{i,c} + v_i, v_{i,c} + \mathsf{acc}_{i,c})}$$

  - **Trajectories** s_i, Obs_i, act_i
  - **Reward**:
    - $+2000$ if goals are achieved
    - -100 if distance rules are not respected
    - $\nearrow$ with speed
    - $\nearrow$ with the increase of distance between cars

Results with SWA-SMT solver, post SWA-SMT solver RL and single RL training



DFS algorithm

Stage 1: *Reachability algorithm on a simplified ISWA model*

SWA

Solved: combinatorial aspect of them problem.
Results: Important events and their relative order

Drawback: A very abstract model of speed

SMT Solver

Stage 2: *Refine the model of the speed*

SMT

A more realistic model of speed
Results: traces that takes into account the dynamical aspect of the problem
Drawback: runtime execution

SWA-SMT solver

RL training

Generate *a dataset for random initial positions*

Dataset

Stage 3: *Train an RL algorithm with our dasaset*

RL

Drawback: our problem has both combinatorial and continuous aspects
Method: get an intuition from dataset to avoid unsuccessful choices
MDP model to reward short-time episode and distance between cars

Conclusion

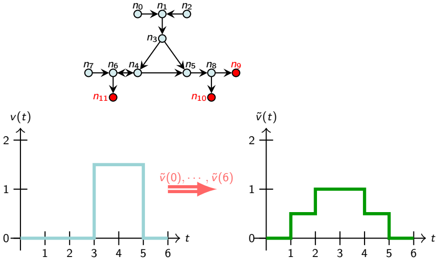
- SWA-SMT Solver

Automata-based model

Efficient algorithm
Abstract model with unrealistic speed model

Piecewise-affine speed graph

Bounded accelleration and deceleration
Different speed
SMT solver to model and solve the distance constraints



Conclusion

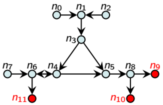
- SWA-SMT Solver

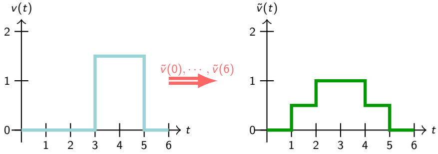
Automata-based model

Efficient algorithm
Abstract model with unrealistic speed model

Piecewise-affine speed graph

Bounded accelleration and deceleration
Different speed
SMT solver to model and solve the distance constraints





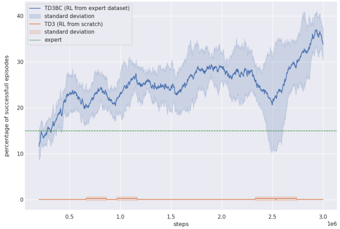
- RL training

Dataset

Trace generated with SWA-SMT solver
Random positions & speeds

Performance of RL (helped with SWA-SMT solver)

Better than single RL
Better than SWA-SMT solver
Runtime: $\sim$ *2 days*

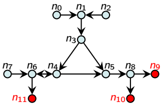


Conclusion

- SWA-SMT Solver

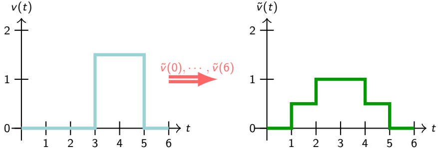
Automata-based model

Efficient algorithm
Abstract model with unrealistic speed model



Piecewise-affine speed graph

Bounded accelleration and deceleration
Different speed
SMT solver to model and solve the distance constraints



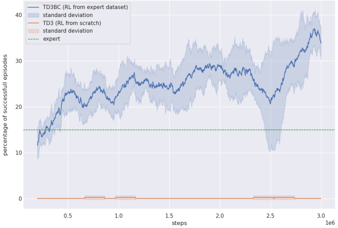
- RL training

Dataset

Trace generated with SWA-SMT solver
Random positions & speeds

Performance of RL (helped with SWA-SMT solver)

Better than single RL
Better than SWA-SMT solver
Runtime: $\sim$ *2 days*



- Future work: Decentralized multi-agent systems