Layered controller synthesis for dynamic multi-agent systems

Emily Clement\textsuperscript{1,3} Nicolas Perrin-Gilbert\textsuperscript{1} Philipp Schlehuber-Caissier\textsuperscript{2}

\textsuperscript{1}Sorbonne Université, CNRS, Institut des Systèmes Intelligents et de Robotique, ISIR, F-75005 Paris, France
\textsuperscript{2}EPITA Research Laboratory
\textsuperscript{3}Université Paris Cité, CNRS, IRIF, Paris, France

Septembre 19 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

<table>
<thead>
<tr>
<th></th>
<th>Timed Automata</th>
<th>Reinforcement Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Abstract representation (acceleration)</td>
<td></td>
</tr>
<tr>
<td>Weakness</td>
<td>Time of execution</td>
<td>Combinatorial or Continuous aspects</td>
</tr>
</tbody>
</table>
Our layered approach

- Our assumptions

\[ v(x) \]

\[ \begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array} \]

▷ Speed:

\[ \begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array} \]

- Our contribution: Three-layered Controller synthesis

SWA

Stage 1: Reachability algorithm on a simplified ISWA model

SMT

Stage 2: Refine the model of the speed

SWA-SMT Solver

Generate a dataset for random initial positions

Dataset

Stage 3: Train an RL algorithm with our dataset

RL

RL training
Our layered approach

- Our assumptions
  - **Speed:** $v(x)$
    - $v(x)$ is a step function with steps at $x = 1, 2, 3, 4, 5, 6$
    - $v(x) = 1$ for $x = 1, 2, 3, 4, 5, 6$
  
  - **Paths of cars:** fixed trajectories, fixed finals & initial positions.

Stage 1: Reachability algorithm on a simplified ISWA model
Stage 2: Refine the model of the speed
Stage 3: Train an RL algorithm with our dataset
Our layered approach

- Our assumptions

  \( v(x) \)

  ▶ **Speed:** 0, 1, 2, 3, 4, 5, 6

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

  ▶ **Trajectories:** we abstract from the curves of the trajectories.
Our layered approach

- Our assumptions

  - Speed: $v(x)$

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

  - Trajectories: we abstract from the curves of the trajectories.

  - Our control: the speed of (all) cars.
Our layered approach

- **Our assumptions**

  \[ v(x) \]

  ![Graph showing \( v(x) \)]

  ▶ **Speed:** 0

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

  ▶ **Trajectories:** we abstract from the curves of the trajectories.

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

  ▶ **Goal:** reach goals while avoiding collisions between agents.
Our layered approach

- **Our assumptions**
  - Speed: $v(x)$
  
    ![Graph showing speed](image)

  - **Speed:** 0, 1, 2, 3, 4, 5, 6

  - **Paths of cars:** fixed trajectories, fixed finals & initial positions.
  
  - **Trajectories:** we abstract from the curves of the trajectories.
  
  - **Our control:** the speed of (all) cars.
  
  - **Goal:** reach goals while avoiding collisions between agents.

- **Our contribution:** Three-layered Controller synthesis

  - **Stage 1:** Reachability algorithm on a simplified ISWA model
  
  - **SWA-SMT Solver**

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

  - **SWA**
Our layered approach

- Our assumptions

  \[ v(x) \]

  - **Speed:**
  
  - **Paths of cars:** fixed trajectories, fixed finals & initial positions.
  
  - **Trajectories:** we abstract from the curves of the trajectories.
  
  - **Our control:** the speed of (all) cars.
  
  - **Goal:** reach goals while avoiding collisions between agents.

- Our contribution: Three-layered Controller synthesis

  - **Stage 1:** Reachability algorithm on a simplified ISWA model
  - **Stage 2:** Refine the model of the speed

  - **Stage 3:** Train an RL algorithm with our dataset

SWA-SMT Solver
Our layered approach

- Our assumptions
  - Speed:
    - $v(x)$
    - $\text{Speed: } 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$
  - Paths of cars: fixed trajectories, fixed finals & initial positions.
  - Trajectories: we abstract from the curves of the trajectories.
  - Our control: the speed of (all) cars.
  - Goal: reach goals while avoiding collisions between agents.

- Our contribution: Three-layered Controller synthesis
  - SWA-SMT Solver
  - **Stage 1**: Reachability algorithm on a simplified ISWA model
  - **Stage 2**: Refine the model of the speed
  - **Stage 3**: Train an RL algorithm with our dataset
  - **Generate a dataset** for random initial positions
  - **Dataset**
SWA-SMT solver

SWA solver

Stage 1: Reachability algorithm on system of ISWA

Stage 2: Model the acceleration and deceleration
Rules to model collision avoidance

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

\[ \varepsilon \]

\[ A \quad B \quad C \]

\[ \varepsilon \quad \varepsilon \]
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

• #3: No Overtaking between cars
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
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

• #3: No Overtaking between cars
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:

```
  n0  n1
   ^
  L   
```
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:

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

▷ 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)
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:
- 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 $p_0, p_1, p_2$: 
What type of Timed Automata to use to model this?

- Needs

▷ **Stopwatch Timed Automata:**

\[ v(x) \]

\[ \begin{array}{c|c|c|c|c|}
1 & 2 & 3 & 4 & x \\
\hline
0 & 1 & 1 & 1 & 1 \\
\end{array} \]
What type of Timed Automata to use to model this?

- **Needs**
  - **Stopwatch Timed Automata:**
  - **Clocks of TA:** Monitor each car’s progress.
What type of Timed Automata to use to model this?

- **Needs**

  - **Stopwatch Timed Automata**: Monitor each car’s progress.
  - **Clocks of TA**: Monitor each car’s progress.
  - **Synchronised action**: Compute distance between each car.
What type of Timed Automata to use to model this?

- Needs
  - **Stopwatch Timed Automata:**
  - **Clocks of TA:** Monitor each car’s progress.
  - **Synchronised action:** Compute distance between each cars.
  - **FiFo channels:** A car cannot overtake another car.
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.

- Bounded channels

- Channels: FiFo queue of symbols (actions) to be pushed/read
Initialized Stopwatch Timed Automata with bounded channels

- Example of a two-clocks Stopwatch Timed Automata

\[ 0 \leq x \leq 1 \quad 0 \leq y \leq 1 \]

\[ 1 \leq x \leq 2 \quad 0 \leq y \leq 1 \]

Reachability is Undecidable in general cases.

- Initialized Stopwatch Timed Automata

  - Reset the stopped clock in the previous or following transition:

\[ \{ y \} \quad \{ \} \]

- Reachability becomes Decidable for this fragment of SWA.

- Bounded channels

  - Channels: FiFo queue of symbols (actions) to be pushed/read
Initialized Stopwatch Timed Automata with bounded channels

- Example of a two-clocks Stopwatch Timed Automata

\[
\begin{align*}
\ell_0 & \xrightarrow{a_1} \ell_1 & 0 \leq x \leq 1 & 0 \leq y \leq 1 \\
\ell_1 & \xrightarrow{a_2} \ell_f & 1 \leq x \leq 2 & 0 \leq y \leq 1
\end{align*}
\]

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)

Emily Clement

Layered controller synthesis for dynamic multi-agent systems
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.
Initialized Stopwatch Timed Automata with bounded channels

- Example of a two-clocks Stopwatch Timed Automata

\[
\begin{align*}
\ell_0 & \xrightarrow{a_1} \ell_1 \\
\{y\} & \quad & \{\} & \quad 0 \leq y \leq 1 \\
1 \leq x \leq 2 & \\
\ell_1 & \xrightarrow{a_2} \ell_f \\
\{\} & \quad & \{} & \\
0 \leq y \leq 1 & \\
\ell_f & \\
\end{align*}
\]

▷ Reachability is Undecidable in general cases.

- Initialized Stopwatch Timed Automata

▷ Reset the stopped clock in the previous or following transition:

\[
\begin{align*}
\ell_p & \xrightarrow{g_p} \ell \\
\{\} & \quad & \{y\} & \quad y \leftarrow 0 \\
\ell & \xrightarrow{g_f} \ell_f \\
\{\} & \\
\ell_f & \\
\end{align*}
\]

▷ 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**

  Path: $s \xleftrightarrow{L_0} s' \xleftrightarrow{L} s''$

- **Car A Timed automaton:**

  ![Timed automaton diagram]

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

  \[\begin{align*}
  a_s & \xrightarrow{x_A = L_0} w_{s'} & x_A = L_0 \\
  & \text{sync}_{s'}(x_A) & c_{s'}?x_A \\
  \{\} & \{x_A\} & \{\} \\
  \hline
  d_{s'} & \xrightarrow{x_A = L_0 + L} a_{s'} & x_A = L_0 + L \\
  & c_{s''}!x_A & \text{sync}_{s''}(x_A) \\
  \{\} & \{\} & \{x_A\} \\
  \hline
  w_{s''} & \xrightarrow{x_A = L_0 + L} & \\
  \end{align*}\]
Model the car progress

- **Car A progress along its paths**

  Path: $s$ \[\xrightarrow{L_0} \] $s'$ \[\xrightarrow{L} \] $s''$

- **Car A Timed automaton:**

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

Layered controller synthesis for dynamic multi-agent systems
Model the car progress

- **Car A progress along its paths**

  Path: $L_0 \xrightarrow{ss} L \xrightarrow{ss'} L$  

- **Car A Timed automaton:**

  ![Automaton Diagram]

  - **Clock** $x_A$: distance travelled along its paths
  - **Stopwatches** $\{x_A\}$: the car $A$ stops instantly.
  - **Channels** $c_{s'}!x_A/c_{s'}?x_A$: 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**

  Path: \[ s \overset{L_0}{\rightarrow} s' \overset{L}{\rightarrow} s'' \]

- **Car A Timed automaton:**

  - **Clock** \( x_A \): distance travelled along its paths
  - **Stopwatches** \( \{x_A\} \): the car A stops instantly.
  - **Channels** \( c_{s'}!x_A/c_{s'}?x_A \): 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

\[
\text{Path: } L_0 \xrightarrow{s} s' \xrightarrow{L} s''
\]

Car A Timed automaton:

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

Path of car $A$: $s_A \rightarrow s' \rightarrow s_A''$

Path of car $B$: $s_B \rightarrow s' \rightarrow s_B''$

- Intersection automaton

\[
x_{s'} = L + \epsilon
\]
Model distance between cars: intersection

Path of car A: \( s_A \rightarrow s' \rightarrow s''_A \)
Path of car B: \( s_B \rightarrow s' \rightarrow s''_B \)

- Intersection automaton

\[ x_{s'} = L + \varepsilon \]

- \( f_{s'} \)
- \( b_{s', \rightarrow} \)
- \( sf_{s', \rightarrow} \)

\( \text{sync}_{s'}(x_B) \)
\( x_{s'} \leftarrow 0 \)

\( \text{sync}_{s'}(x_A) \)
\( x_{s'} \leftarrow 0 \)

\( \text{sync}_{s'}(x_A) \)
\( x_{s'} \leftarrow 0 \)

\( \text{sync}_{s'}(x_B) \)
\( x_{s'} \leftarrow 0 \)
Model distance between cars: intersection

Path of car A: \( s_A \rightarrow s' \rightarrow s_A'' \)
Path of car B: \( s_B \rightarrow s' \rightarrow s_B'' \)

- Intersection automaton

\( x_{s'} = L + \varepsilon \)

\( \text{sync}_{s'}(x_B) \)
\( x_{s'} \gets 0 \)
\( \text{sync}_{s'}(x_A) \)
\( x_{s'} \gets 0 \)
\( \text{sync}_{s'}(x_A) \)
\( x_{s'} \gets 0 \)
\( \text{sync}_{s'}(x_B) \)
\( x_{s'} \gets 0 \)
Model distance between cars: intersection

Path of car A: $s_A \xrightarrow{s'} s''$

Path of car B: $s_B \xrightarrow{s'} s''$

- Intersection automaton

$$x_{s'} = L + \epsilon$$

Layered controller synthesis for dynamic multi-agent systems
Model distance between cars: intersection

Path of car A: $s_A \rightarrow s' \rightarrow s_A''$

Path of car B: $s_B \rightarrow s' \rightarrow s_B''$

- Intersection automaton

$$x_{s'} = L + \epsilon$$

**Layered controller synthesis for dynamic multi-agent systems**
Model distance between cars: intersection

Path of car A: $s_A \rightarrow s' \rightarrow s_A''$
Path of car B: $s_B \rightarrow s' \rightarrow s_B''$

• Intersection automaton

$x_{s'} = L + \varepsilon$

sync$_{s'}(x_B)$
$x_{s'} \leftarrow 0$

sync$_{s'}(x_A)$
$x_{s'} \leftarrow 0$

sync$_{s'}(x_A)$
$x_{s'} \leftarrow 0$

sync$_{s'}(x_B)$
$x_{s'} \leftarrow 0$

Layered controller synthesis for dynamic multi-agent systems
Model distance between cars: intersection

Path of car A: $s_A \rightarrow s' \rightarrow s''$
Path of car B: $s_B \rightarrow s' \rightarrow s''$

- Intersection automaton

\[ x_{s'} = L + \varepsilon \]

\[ \text{sync}_{s'}(x_B) \]
\[ x_{s'} \leftarrow 0 \]
\[ \text{sync}_{s'}(x_A) \]
\[ x_{s'} \leftarrow 0 \]

\[ \text{sync}_{s'}(x_B) \]
\[ x_{s'} \leftarrow 0 \]
\[ \text{sync}_{s'}(x_A) \]
\[ x_{s'} \leftarrow 0 \]

\[ \text{sync}_{s'}(x_B) \]
\[ x_{s'} \leftarrow 0 \]
\[ \text{sync}_{s'}(x_A) \]
\[ x_{s'} \leftarrow 0 \]
Model distance between cars: intersection

Path of car A: $s_A \rightarrow s' \rightarrow s_A''$
Path of car B: $s_B \rightarrow s' \rightarrow s_B''$

- Intersection automaton
Model distance between cars: intersection

Path of car $A$: $s_A \rightarrow s' \rightarrow s_A''$
Path of car $B$: $s_B \rightarrow s' \rightarrow s_B''$

- Intersection automaton

\[
x_{s'} = L + \varepsilon
\]

Layered controller synthesis for dynamic multi-agent systems
Model distance between cars: intersection

Path of car $A$: $s_A \rightarrow s' \rightarrow s_A''$

Path of car $B$: $s_B \rightarrow s' \rightarrow s_B''$

- Intersection automaton

\[
x_{s'} = L + \varepsilon
\]

Layered controller synthesis for dynamic multi-agent systems
Model distance between cars: intersection

Path of car $A$: $s_A \rightarrow s' \rightarrow s''$  

Path of car $B$: $s_B \rightarrow s' \rightarrow s''$  

- Intersection automaton

\[ x_{s'} = L + \varepsilon \]

Layered controller synthesis for dynamic multi-agent systems
Model distance between cars: intersection

Path of car A: \( S_A \rightarrow S' \rightarrow S_{A''} \)
Path of car B: \( S_B \rightarrow S' \rightarrow S_{B''} \)

- Intersection automaton

\[ x_{s'} = L + \varepsilon \]

\[ f_{s'} \rightarrow b_{s', \rightarrow} \rightarrow s_{f_{s'}, \rightarrow} \]

- \( x_{s'} \leftarrow 0 \)
- \( \text{sync}_{s'}(x_B) \)
- \( \text{sync}_{s'}(x_A) \)
- \( x_{s'} \leftarrow 0 \)
- \( \text{sync}_{s'}(x_B) \)
- \( \text{sync}_{s'}(x_A) \)
- \( x_{s'} \leftarrow 0 \)
- \( x_{s'} \leftarrow 0 \)
Our Algorithm: a DFS with an optimised succ function

Transition $t$ available

$t \in \text{car TA}?$
Our Algorithm: a DFS with an optimised succ function

Transition $t$ available

Yes

$t \in \text{car TA}$?

Yes

loc = $w_s$?

No

loc = $b_s$?

No

Choice 1

Yes

Choice 2

Yes
Our Algorithm: a DFS with an optimised succ function

Transition $t$ available

$t \in \text{car TA}？$

Yes

loc = $w$？

Yes

$\exists$ other $t'$ available？

No

No

loc = $b$？

$\exists$ other $t'$ available？

Take $t$
Our Algorithm: a DFS with an optimised succ function

Transition \( t \) available

- \( t \in \text{car TA?} \)
  - Yes
    - \( \text{loc = } w_s, ? \)
      - Yes
        - \( \exists \text{other } t' \text{ available?} \)
          - Choice 1: take \( t \)
          - Choice 2: take \( t \)
      - No
        - \( \text{wait for } t' \)
  - No
    - \( \text{loc= } b_s, ? \)
      - take \( t \)
Our Algorithm: a DFS with an optimised succ function

Transition $t$ available

Yes

$t \in \text{car TA}?$

Yes

loc = $w_s$?

Yes

$\exists$ other $t'$ available?

Yes

take $t$

No

wait for $t'$

No

$\exists$ car asking to enter in $s$, $\rightarrow$ ($t'$) ?

No

Choice 1

Choice 2

Yes

take $t$

No

take $t$

No

loc = $b_s$, $\rightarrow$ ?
Our Algorithm: a DFS with an optimised succ function

Transition $t$ available

Yes $t \in \text{car TA}$?

Yes $\exists \text{ other } t'$ available?

Choice 1

Choice 2

$\exists \text{ car asking to enter in } s, \rightarrow (t')$?

Yes $\text{take } t$

No $\text{take } t$ or $t$'

No $\text{take } t'$

No $\text{wait for } t'$

$\text{take } t$
SWA-SMT solver

Stage 1: Reachability algorithm on system of ISWA

Stage 2: Model the acceleration and deceleration

SWA  SMT
Why use of SMT solver?

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

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

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

**DFS algorithm**

- **SWA**

**SMT Solver**

- **SMT**

**RL training**

- **Dataset**
- **RL**

**Solved:** combinatorial aspect of the problem.

**Results:** Important events and their relative order

**Drawback:** A very abstract model of speed

**Generate a dataset** for random initial positions
Why use of SMT solver?

**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**

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

- **Dataset**

  - Generate a dataset for random initial positions

**SMT solver**

- The **continuous** aspect of the problem
- Introduce a more **realistic** model of speed
New model for speed graph

- 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

\[
\begin{align*}
  v_i(t) &\Rightarrow \tilde{v}_i(0), \ldots, \tilde{v}_i(k-1) \\
  x(t) &\Rightarrow \tilde{x}_i(k) = \sum_{l=0}^{k-1} \tilde{v}_i(l)
\end{align*}
\]
How to preserve security distance?

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

• Example of SMT solver’s inequalities
  For each step \( k \) :
  ▶ \( \tilde{v}_i(k) - d_{\text{max}} \leq \tilde{v}_i(k + 1) \leq \tilde{v}_i(k) + a_{\text{max}} \)
How to preserve security distance?

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

● Example of SMT solver’s inequalities
For each step \( k \):
  ▶ \( \ddot{v}_i(k) - d_{\text{max}} \leq \ddot{v}_i(k+1) \leq \ddot{v}_i(k) + a_{\text{max}} \)
  ▶ \( 0 \leq \ddot{v}_i(k) \leq v_{\text{max}} \)
**RL training**

*Generate a dataset for random initial positions*

*Stage 3: Train an RL algorithm with our dataset*
Why use of SMT solver?

**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

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

- **Dataset**
- **RL**
- Generate a dataset for random initial positions
- Drawback: our problem has both combinatorial and continuous aspects
- Goal: get an intuition from dataset to avoid unsuccessful choices
Why use of SMT solver?

---

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

- **SWA**
- **SMT**

**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

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

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

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

---

Emily Clement
Layered controller synthesis for dynamic multi-agent systems
• Markov Decision Process

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

**Model**

Layered controller synthesis for dynamic multi-agent systems
• **Markov Decision Process**
  
  ▶ **Deterministic running example**: deterministic transition function.
  
  ▶ **State** \( s_i \). For each section \( s \), if a car \( c \) is in \( s \): \( v_{i,c}, pos_{i,c}, id_c, 1 \)
  
  ▶ **Action** \( act_i \): \( (acc_{i,c})_{c \in \text{Cars}} \)

\[
(pos_{i,c}, v_{i,c}) \rightarrow (pos_{i,c} + v_i, v_{i,c} + acc_{i,c})
\]

\[
\begin{array}{l}
i \\
i + 1
\end{array}
\]
• Markov Decision Process

▷ **Deterministic running example**: deterministic transition function.

▷ **State** $s_i$. For each section $s$, if a car $c$ is in $s$: $v_{i,c}, \text{pos}_{i,c}, \text{id}_c, 1$

▷ **Action** $\text{act}_i$: $(\text{acc}_{i,c})_{c \in \text{Cars}}$

$$$(\text{pos}_{i,c}, v_{i,c}) \rightarrow (\text{pos}_{i,c} + v_{i,c}, v_{i,c} + \text{acc}_{i,c})$$$

▷ **Trajectories** $s_i, \text{Obs}_i, \text{act}_i$
Model

- Markov Decision Process
  - **Deterministic running example**: deterministic transition function.
  - **State** $s_i$. For each section $s$, if a car $c$ is in $s$: $v_{i,c}, pos_{i,c}, id_c, 1$
  - **Action** $act_i$: $(acc_{i,c})_{c \in \text{Cars}}$
    $$ (pos_{i,c}, v_{i,c}) \rightarrow_{i} (pos_{i,c} + v_{i,c}, v_{i,c} + acc_{i,c}) \rightarrow_{i+1} $$
  - **Trajectories** $s_i, \text{Obs}_i, act_i$
  - **Reward**:
    - $+2000$ if goals are achieved
    - $-100$ if distance rules are not respected
    - $\uparrow$ with speed
    - $\uparrow$ with the increase of distance between cars
Results with SWA-SMT solver, post SWA-SMT solver RL and single RL training
Steps of the layered method

**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 take into account the dynamical aspect of the problem.
- **Drawback:** Runtime execution.

**Stage 3: Train an RL algorithm with our dataset**
- **RL**
- **Generate a dataset for random initial positions**
- **Dataset**
- **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 acceleration 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
Conclusion

- **SWA-SMT Solver**

   Automata-based model
   
   *Efficient algorithm*
   *Abstract model with unrealistic speed model*

   Piecewise-affine speed graph
   
   *Bounded acceleration and deceleration*
   *Different speeds*
   *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: ~ 2 days*
Conclusion

- **SWA-SMT Solver**

  **Automata-based model**
  
  *Efficient algorithm*
  *Abstract model with unrealistic speed model*

  **Piecewise-affine speed graph**
  
  *Bounded acceleration 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**