“Formal verification of autonomous vehicle platooning”

Maryam Kamali, et all.

Science of Computer Programming 148 (2017) 88–106
http://dx.doi.org/10.1016/j.scico.2017.05.006

Software Verification and Validation

PhD student: Lincoln Herbert Teixeira
Professor: Dr. István Majzik

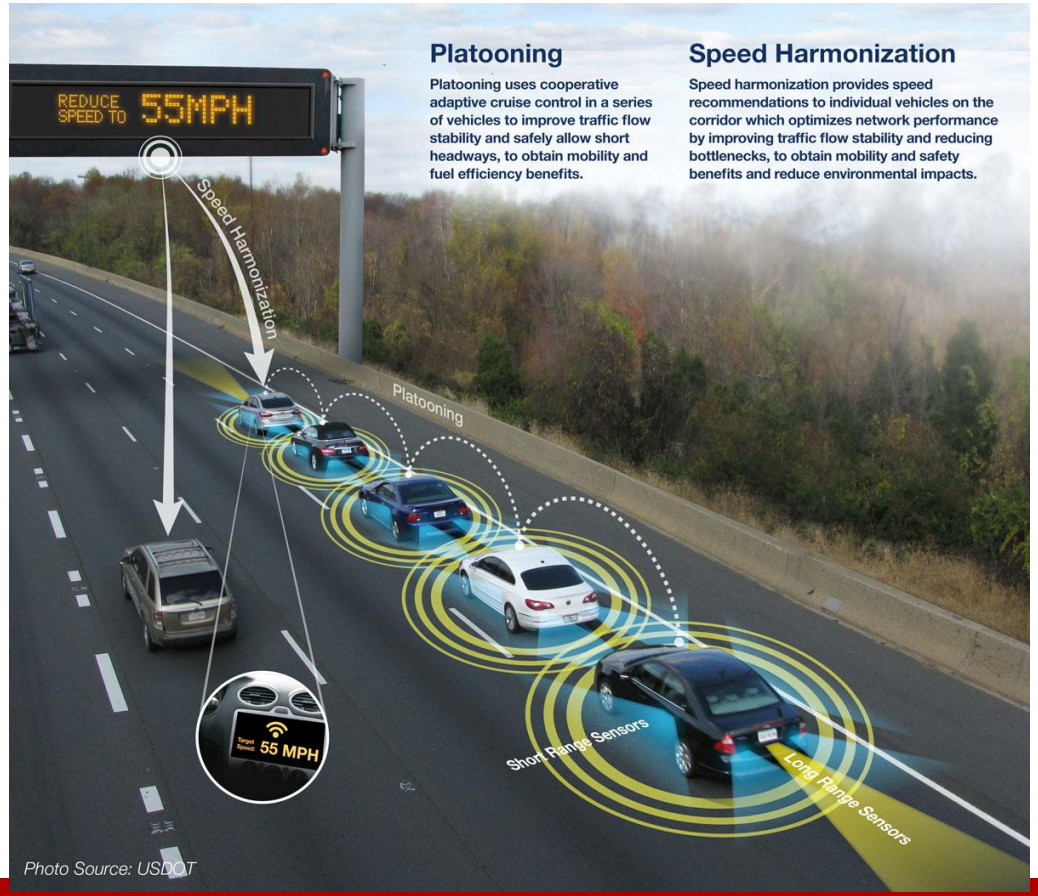
November 29th, 2018

Outline

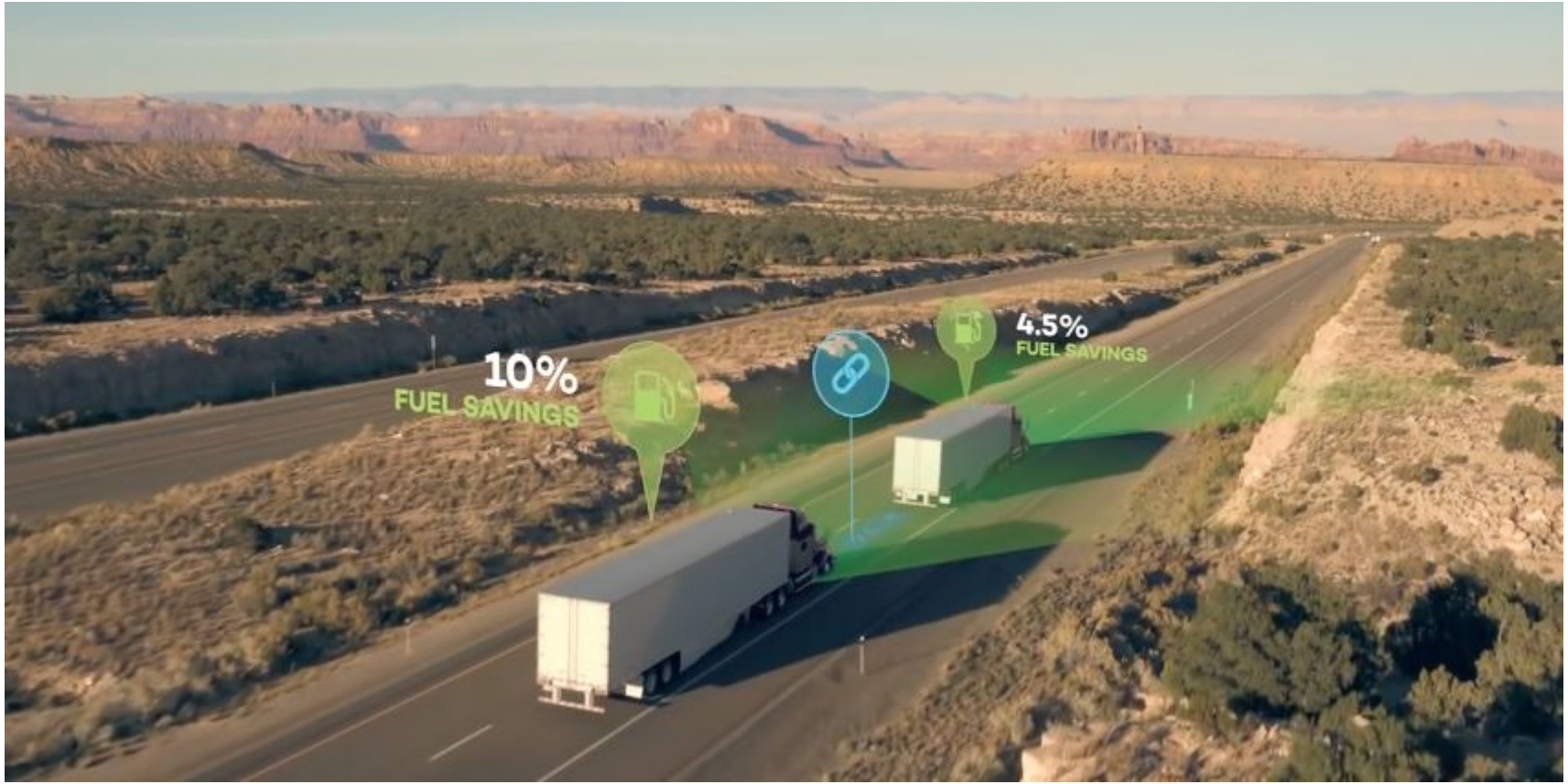
Overview

1. Introduction
2. Automotive Platoons
3. Agent-based development of automotive platoon
4. Verification
5. Concluding remarks

What is a Platoon?



Fluel saving



Security

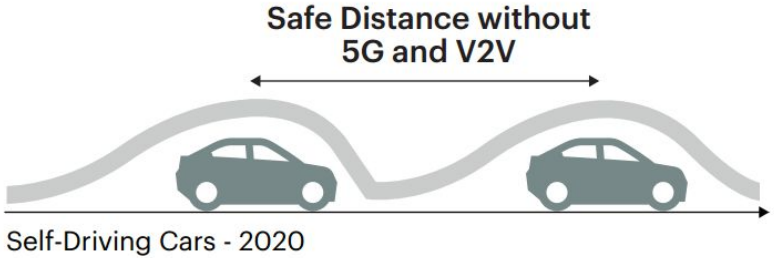
Platooning

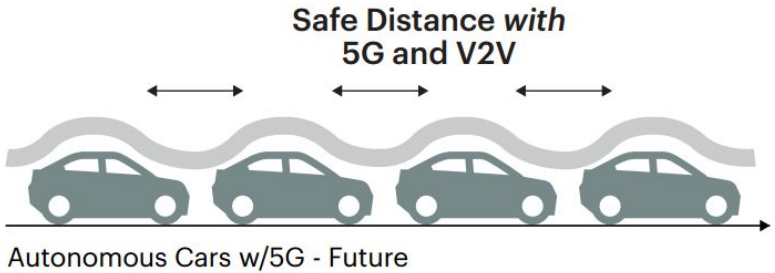
Platooning ist ein System für den Strassenverkehr, bei dem mehrere Fahrzeuge über WLAN elektronisch aneinandergekoppelt sind. Alle im «Platoon» (Englisch für militärischen «Zug») fahrenden Fahrzeuge folgen einander in minimalem Abstand. Gesteuert werden sie vom Fahrzeug an der Spitze. Platooning kann für den strassengebundenen Personen- und Güterverkehr in Frage kommen. Aus wirtschaftlichen Gründen ist die Entwicklung aber vor allem für den Güterverkehr von Interesse.

Communication

Smart Car Convoys

Fuel Savings of 25%[9]





1 - Introduction

- “Driverless cars” is not fully autonomous yet.
- Legal constraints, such as the Vienna Convention, responsible human.
- Platoons: each vehicle autonomously follows the one in front of it, with the lead vehicle in the platoon/convoy being controlled by a professional human driver.
- Safety and efficiency of vehicles on very congested roads;
- Needs to communicate with each others - V2V;

V2V Communication

- **Lower level** (control system) to adjust each vehicle's **position** in the lanes and the **spacing** between the vehicles;
- **Higher levels**, to communicate joining **requests**, leaving requests, or **commands** dissolving the platoon;
- A traditional approach is used to implement the software for each vehicle in terms of **hybrid** (and hierarchical) **control systems** and to analyse this using hybrid systems techniques.

Contribution

- ISO 26262 **standard**: functional **safety** management in automotive applications - **V&V**;
- Fulfilled in **design** development and validation;
- Verifiable agent-based architecture for development of **safe** automotive platooning;
- New combined verification techniques for autonomous systems developed based on **hybrid agent architectures**;
- Show the **applicability of verification** techniques in the development of platooning;

2 - Automotive Platoons

- Enable road vehicles to travel as a **group**, lead by a professional driver;
- The follow vehicles are **controlled autonomously**;
- Vehicles equipped with **low-level** control system(speed/ steering)
- Lead vehicle **coordinate** the platoon: setting parameters, creates certificates to join an leave.

Formal automotive platoons requirements

- Understand the functioning of platooning **protocols**;
- Verify essential properties such as the functional **correctness** and **liveness** of the **protocols**;
- A vehicle can join and leave a platoon **if, and only if**, the whole platooning remains **safe**;

Joining the platoon

A new vehicle can join a platoon with different control strategies:

- sends a **joining request** and position to the leader;
- the maximum l**ength is checked** and in **normal operation**, i.e., no other active are happening;
- if position is in front of vehicle X and the **maximum length is ok**, the leader sends an **“increase space”** command to vehicle **X**, and when the leader is informed that enough spacing has been created, it sends back an **agreement** to the joining vehicle;
- the joining vehicle **changes its lane**;
- its **automatic speed controller** is enabled;
- when the vehicle is close enough to the preceding vehicle, its **automatic steering controller** is enabled and acknowledgement;
- a “**decrease space**” command to vehicle X, and **acknowledgement**.

Safe join operation

The requirements decision-making components of automotive platoon:

1 - A vehicle must only initiate joining a platoon, once it has **received confirmation** from the leader.
2 - Before autonomous control is enabled, a joining vehicle must **approach the preceding vehicle**, in the **correct lane**.
3 - Automatic steering controller **must only be enabled** once the joining vehicle is sufficiently close to the preceding vehicle.

Leaving the platton

- a member **sends leaving request** to the leader and **waits** for authorization;
- when **maximum spacing** has been achieved, the vehicle **switches** both its speed and steering controller to **‘manual’**;
- If receipt of **‘leave’ authorization**, the vehicle increases its space from the preceding vehicle; **changes its lane**;
- the vehicle sends an **acknowledgement** to the leader.

3 - Agent-based development of automotive platoon

- **High-level decision-making**: Gwendolen agent code;
- Control system is provided as a Simulink model;
- To **simulation** (TORCS) and testing of protocols for validation;
- The assumption that the physical model in the simulation is representative of real world vehicle dynamics;
- An interface between TORCS and MATLAB/Simulink has been developed;
- **Safety** analysis of systems;

Gwendolen programming language

Is a declarative logic-programming language incorporating explicit representations of goals, beliefs, and plans;
Gwendolen uses many syntactic conventions from **BDI** (**Belief-Desire-Intention**) agent languages.

Gwendolen syntax

Belief-Desire-Intention (BDI) model

perform goal	+!g [perform]	adding a new goal to perform some actions
achievement goal	+!g [achieve]	adding a new goal and continuously attempting the plans associated with the goal until the agent has acquired the belief *g*
believe	+b	adding a new belief *b*
disbelieve	-b	removing a belief *b*
on-hold believe	*b	waiting for belief *b* to be true
plan	trigger : guard ← body	*trigger* is the addition of a goal or a belief; *guard* gives conditions under which the plan can be executed; *body* is a stack of actions that should be performed
guard condition	B x	checking if belief x is perceivable
guard condition	G x	checking if goal x has been added
action	perf(x)	action x causing the execution of x

+!g indicates the addition of the goal g;
+b indicates the addition of the belief b;
- b indicates the removal of the belief;
* b indicates that execution of a plan is suspended until the belief, b, becomes true

A follower vehicle's code

Listing 1 A follower vehicle's code.

```
Reasoning Rules
joining(X, Y):- name(X), platoon_ok

Plans
+! joining(X, Y) [achieve]: {B name(X) , ~B join_agreement(X, Y)}
    <- +!speed_contr(0) [perform], +!steering_contr(0) [perform],
       .send(leader, :tell, join_req(X, Y)), *join_agreement(X, Y);

+! joining(X, Y) [achieve]: {B name(X), B join_agreement(X, Y),
    ~B changed_lane, ~G set_spacing(Z) [achieve]}
    <- +!speed_contr(0) [perform], +!steering_contr(0) [perform],
       perf(changing_lane(1)), *changed_lane;

+! joining(X, Y) [achieve]: {B name(X), B join_agreement(X, Y),
    B changed_lane, ~B speed_contr, ~ B steering_contr,
    ~B joining_distance, ~G set_spacing(Z) [achieve]}
    <- +!speed_contr(1) [perform], *joining_distance;

+! joining(X, Y) [achieve]: {B name(X), B join_agreement(X, Y),
    B changed_lane, B speed_contr, ~B steering_contr,
    B joining_distance, ~G set_spacing(Z) [achieve]}
    <- +!steering_contr(1) [perform];

+! joining(X, Y) [achieve]: {B name(X), B join_agreement(X, Y),
    B changed_lane, ~B speed_contr, ~B steering_contr,
    B joining_distance, ~G set_spacing(Z) [achieve]}
    <- +!speed_contr(1) [perform], +!steering_contr(1) [perform];

+! joining(X, Y) [achieve]: {B name(X), B join_agreement(X, Y),
    B changed_lane, B speed_contr, B steering_contr,
    B joining_distance, ~B platoon_m, ~G set_spacing(Z) [achieve]}
    <- .send(leader,:tell,message(X,joined_succ),
       *platoon_m, +platoon_ok;
```

Syntax
0 - manual
1 - automatic
* wait to be true
~ not
+ add
- remove

High-level plan: decision-making agent for the joining procedure for a follower vehicle

4 - Verification

- Verification task: agent behaviour and real-time requirement of the system;
- Agent behaviour: AJPF (Java PathFinder model checker - only program model-checker for rational agents);
- Real-time requirements: UPPAAL model checker (timed automata);
- Validation of verification;

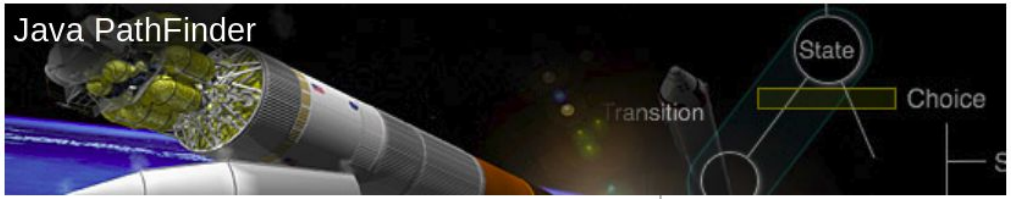
Java PathFinder

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

+ABOUT NASA | +LATEST NEWS | +MULTIMEDIA | +MISSIONS | +WORK FOR NASA

- + NASA Home
- + Ames Home
- + Intelligent Systems Division
- + Robust Software Engineering
- + Verification and Validation

Java PathFinder



Java PathFinder

A Model Checker for Java Programs

Recipient of NASA's 2003 TGIR Award for Engineering Innovation

JPF is now available at SourceForge *(Non-NASA Link!)*

The majority of work carried out in the formal methods community throughout the last three decades has (for good reasons) been devoted to special languages designed to make it easier to experiment with mechanized formal methods such as theorem provers and model checkers. We believe it is time for the formal methods community to shift some of its attention towards the analysis of programs written in modern programming languages.

Researchers

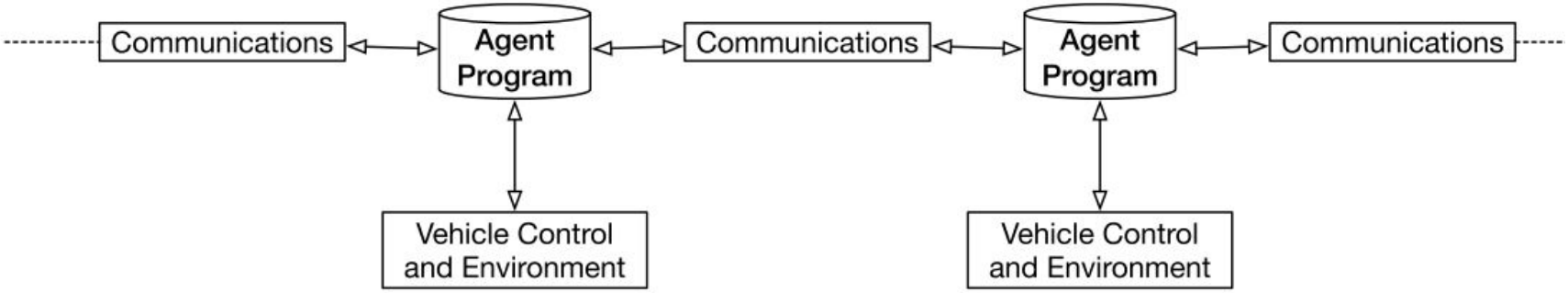
- Peter Mehlitz
- Corina Pasareanu
- Dimitra Giannakopoulou
- Masoud Mansouri-Samani

Past Researchers

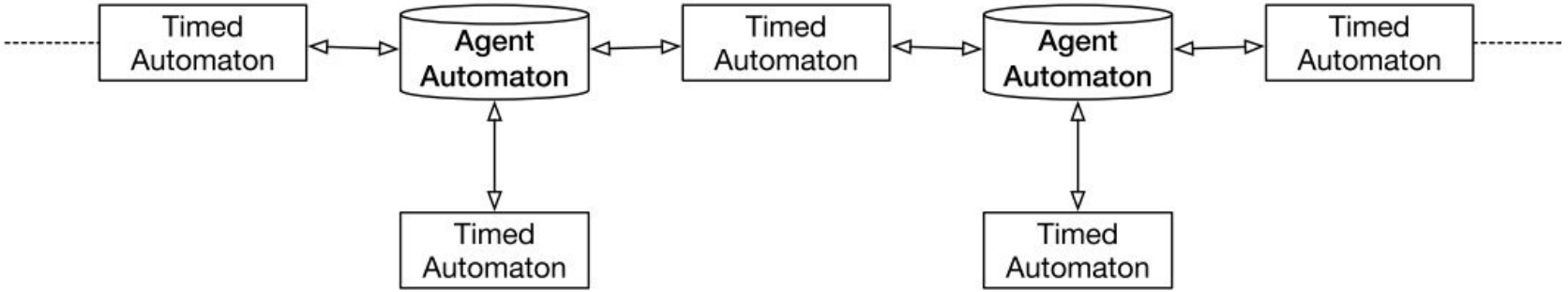
- Willem Visser (Seven)
- John Penix (Google)
- Klaus Havelund (JPL)

Student Interns

Verification Methodology



which, abstracts to an overarching formal model:

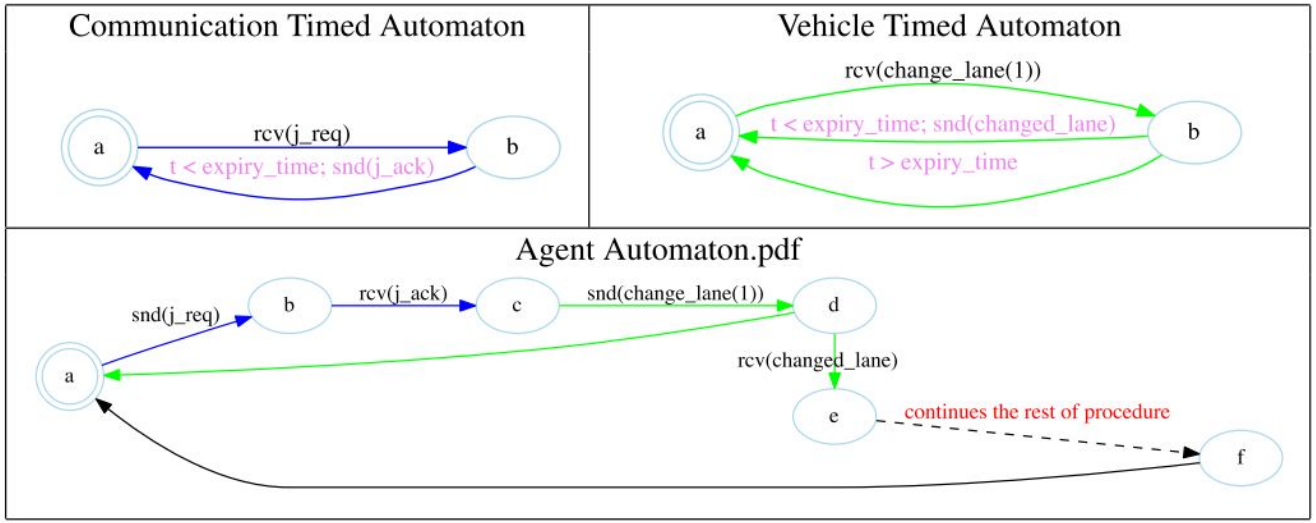


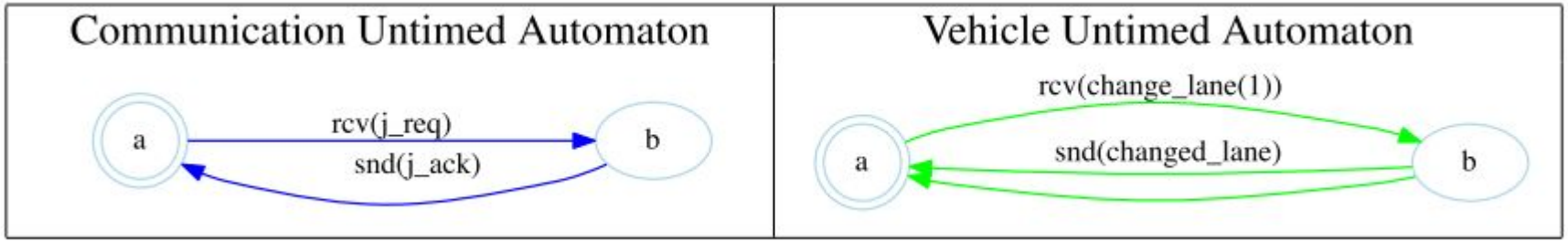
Architecture

Verification Methodology

Abstraction time to untimed automaton over-approximation

Automata of the Platooning





An example of timed and untimed automata of the platooning.

From agent model to untimed automata

- Generate a state model of an agent model through the **AJPF model checker**;
- In every state we have a set of beliefs, intentions, actions, and messages which can represent either the internal state of the agent, or the input/output information from/to the environment;
- The environment is an abstract representation of a vehicle and other agents, in this context;
- For automaton generation only the input/output information is required;
- Remove superfluous information from the state model and merge equivalent states, as described in Algorithm 1;

The generated state model of the agent code consists of 1396 states and 2361 transitions

Agent model algorithm

Algorithm 1 Agent model to untimed automaton algorithm.

translateAgentToAutomaton(S, E)
Input: a state model A, representing as a graph $G = (S, E)$
Output: a minimised state model A', representing as (S''', E''')
$(S', E) =$ StripInternalBeliefs(S, E)
$(S'', E'') =$ RemoveEmpty$(S', E))$
$(S''', E''') =$ MatchAndMergeDuplicates(S'', E'')
function StripInternalBeliefs(S, E)
  $S' = \emptyset$
  for each $s \in S$ **do**
    $s' = s$ with all *internal* beliefs and intentions removed
    $s'' = s'$ with all remaining beliefs and messages converted to propositions
    $S' = S' \cup \{s''\}$
  end for
  return (S', E)
end function
function RemoveEmpty(S, E)
  $S' = S,\ E' = E$
  for each $s \in S'$ **do**
    if $\mathbf{prop}(s) = \emptyset$ **then**
      $E' = (E' \cup \{(x, y) \mid x \in \mathbf{in}(s) \wedge y \in \mathbf{out}(s)\})$
        $\setminus(\{(x, s) \mid x \in \mathbf{in}(s)\} \cup \{(s, y) \mid y \in \mathbf{out}(s)\})$
      $S' = S' \setminus \{s\}$
    end if
  end for
  return (S', E')
end function

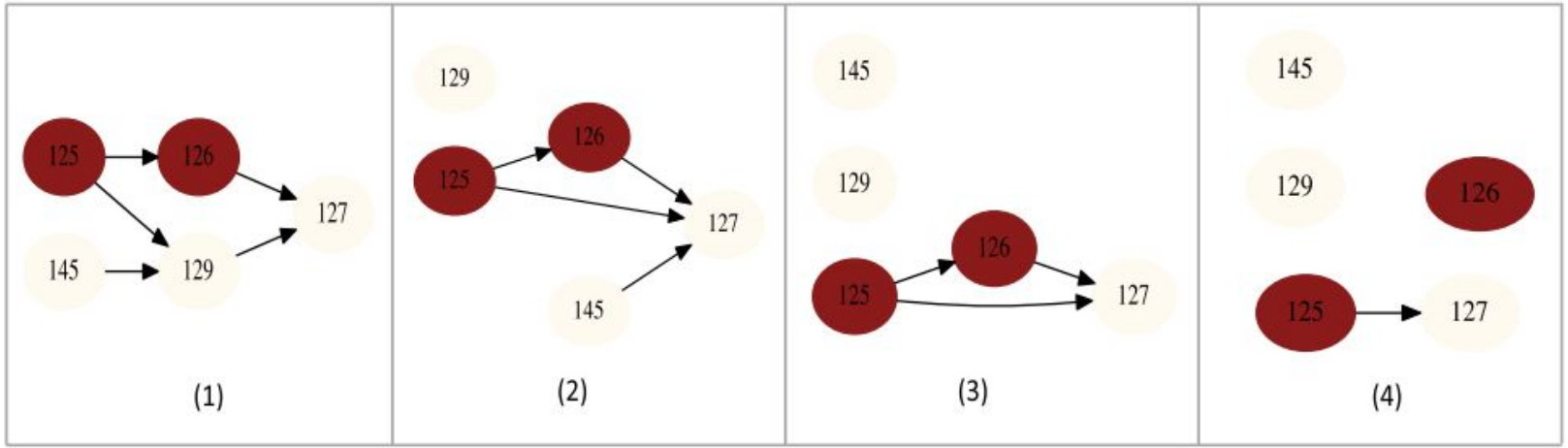
function MatchAndMergeDuplicates(S, E)
  $S' = S,\ E' = E$
  for each $s_i \in S'$ **do**
    for each $s_j \in S' \setminus \{s_i\}$ **do**
      if $\mathbf{out}(s_i) = \mathbf{out}(s_j)$ and $\mathbf{prop}(s_i) = \mathbf{prop}(s_j)$ **then**
        $E' = (E' \cup \{(s_{in}, s_i) \mid s_{in} \in \mathbf{in}(s_j)\})$
          $\setminus(\{(s_j, s_{out}) \mid s_{out} \in \mathbf{out}(s_j)\} \cup \{(s_{in}, s_j) \mid s_{in} \in \mathbf{in}(s_j)\})$
        $S' = S' \setminus \{s_j\}$
      end if
    end for
  end for
  return (S', E')
end function

To remove superfluous information from the state model and merge equivalent states;
This reduces the number of states and transitions;
States converges to a size of 34.

Agent model algorithm

Algorithm 1 Agent model to untimed automaton algorithm.

1: **translateAgentToAutomaton**(S, E)
Input: a state model A, representing as a graph $G = (S, E)$
Output: a minimised state model A', representing as (S''', E''')
2: $(S', E) =$ StripInternalBeliefs(S, E)
3: $(S'', E'') =$ RemoveEmpty$(S', E))$
4: $(S''', E''') =$ MatchAndMergeDuplicates(S'', E'')
5: **function** StripInternalBeliefs(S, E)
6: $S' = \emptyset$
7: **fo**
8:
9:
10:
11: **e**
12: **r**
13: **end**
14: **func**
15:
16: **f**
17:
18:
19:
20:
21: **end for**
22: **return** (S', E')
23: **end function**



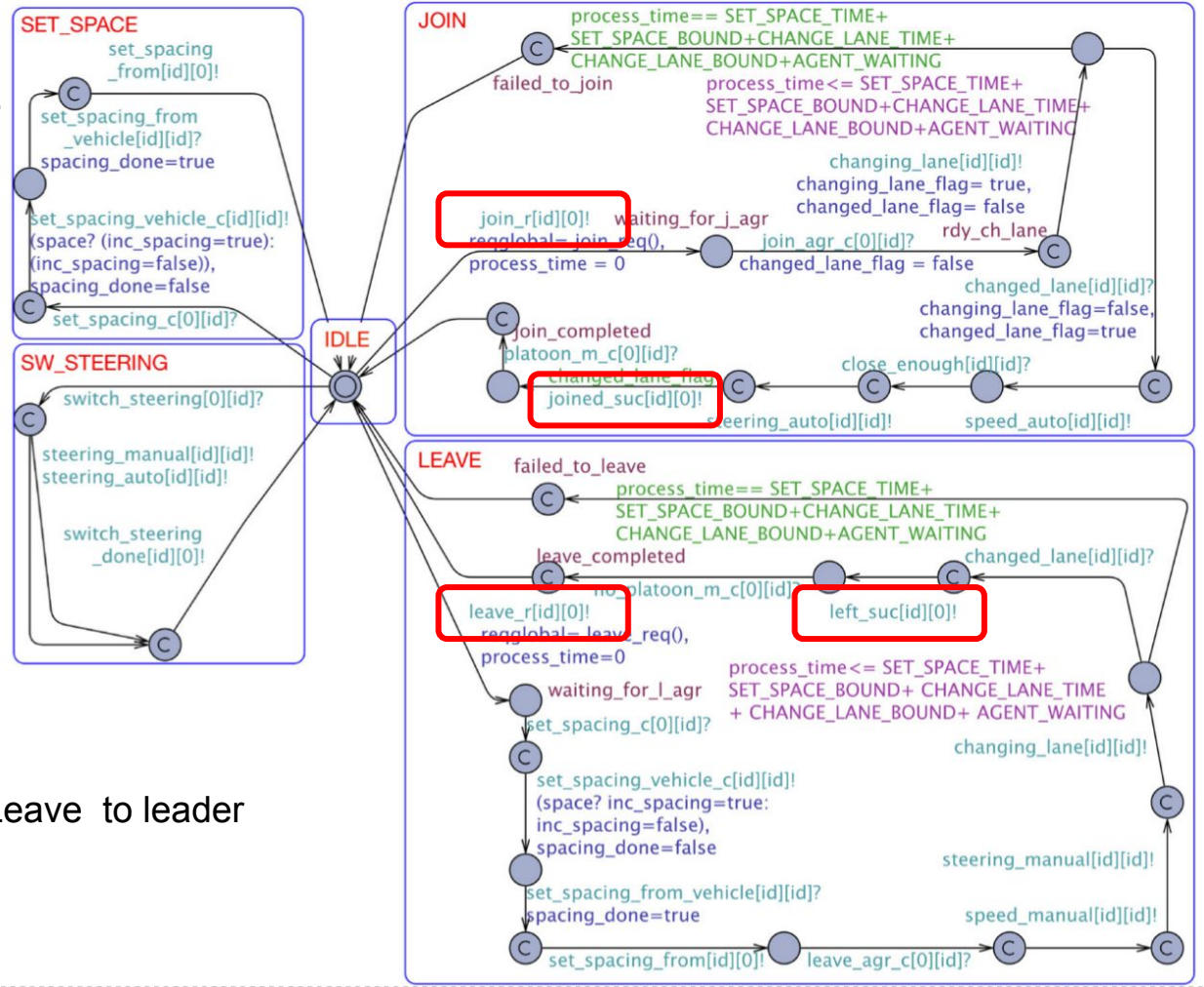
$(s_j)\})$

33: **end for**
34: **return** (S', E')
35: **end function**

To remove superfluous information from the state model and merge equivalent states;
This reduces the number of states and transitions;

Example: Uppal agent automaton

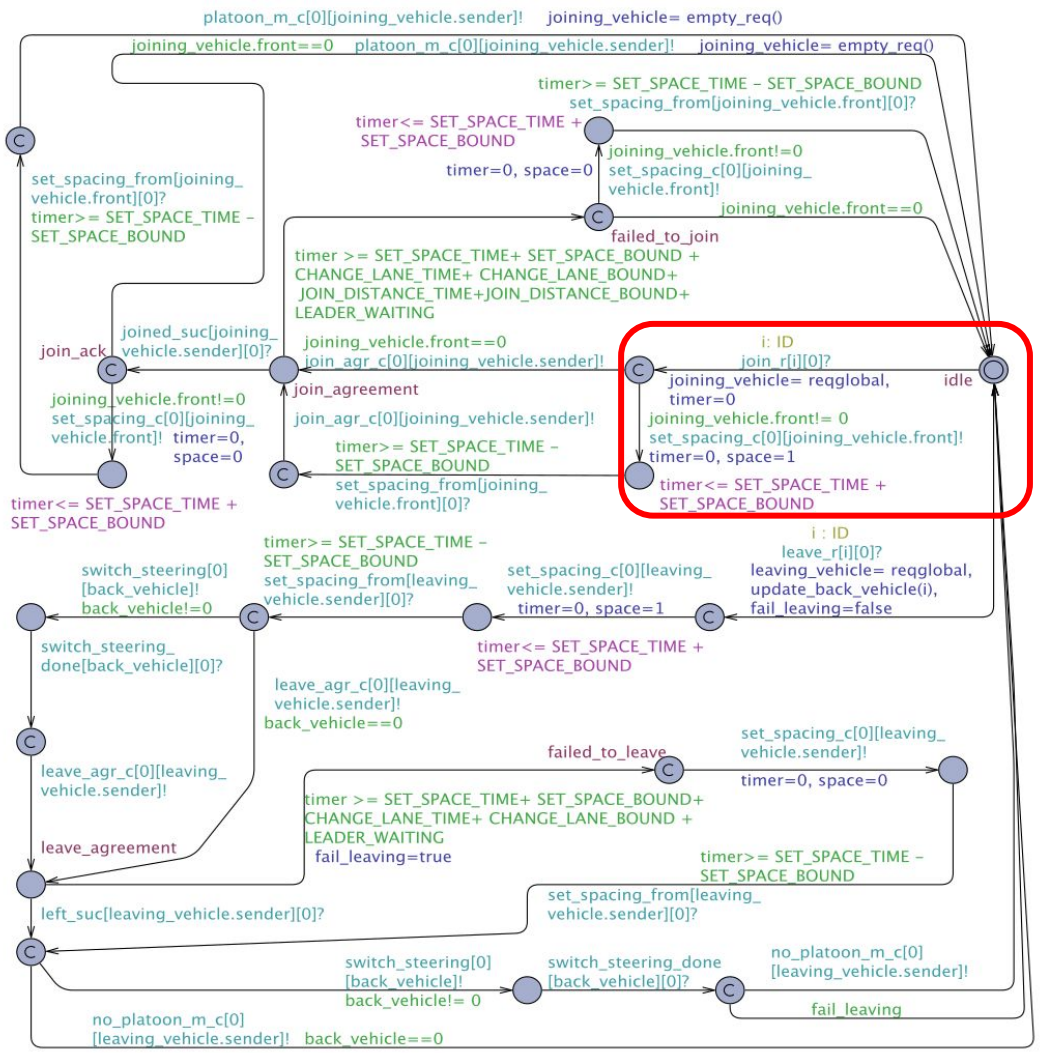
29 locations extracted from the output of previous Algorithm



- Unicast sending Join/Leave to leader
- Receive acknowledge

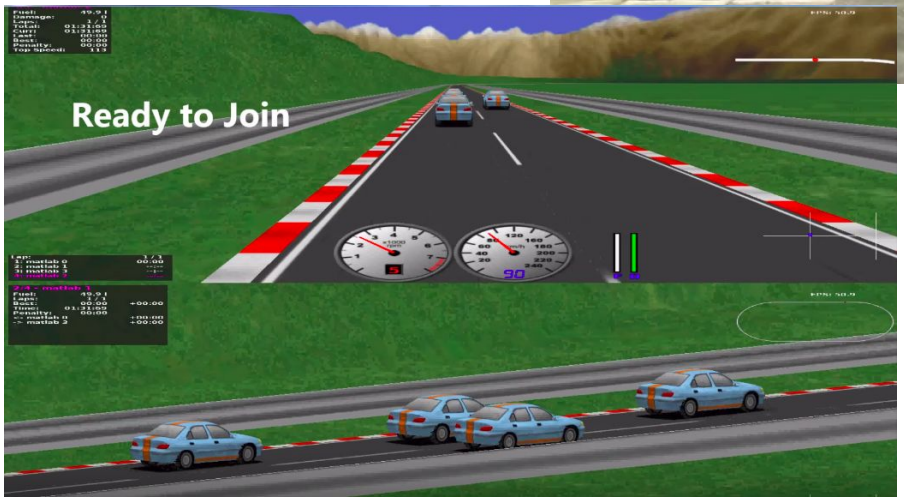
Example: Leader automaton

- Unicast synchronization channels to communicate with the agents



Validating the verification

- It is necessary because of the time abstraction;
- Failure found;



TORCS Simulator

Jaguar outdoor rover physical vehicle

Concluding remarks

The verification of safety considerations for automotive platooning:

- Non-trivial **hybrid autonomous systems**, with each vehicle mixing feedback controllers and agent decision-making;
- **Strong requirement** to verify the actual code used in the implementation, rather than extracting a **formal model of the program's behaviour** - this leads on to program model-checking, which is resource intensive;
- **Rational agents** are essential for capturing high-level autonomy and, as there are no other practical systems able to model-check temporal and modal properties of complex **BDI** agents, we are led to AJPF;
- AJPF is very **resource intensive** (12 hours for some agent properties) and cannot be practically used to verify whole system properties of automotive platooning;
- Timing considerations can be crucial.

Thank you!