Modeling and checking robustness of communicating autonomous vehicles

Johan Arcile, Raymond Devillers, Hanna Klaudel, Witold Klaudel, Bożena Woźna-Szcześniak

Modeling and checking robustness of communicating autonomous vehicles

Johan Arcile[1], Raymond Devillers[2], Hanna Klaudel[1],
Witold Klaudel[3], and Bożena Woźna-Szcześniak[4]

[1] IBISC, University of Evry, 23 bd de France, 91037 Evry, France,
{johan.arcile,hanna.klaudel}@ibisc.univ-evry.fr
[2] ULB, Bruxelles, Belgium, rdevil@ulb.ac.be
[3] Renault SA, 1 av. du Golf, 78084 Guyancourt, France, witold.klaudel@renault.com
[4] IMCS, Jan Długosz University in Częstochowa. Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland b.wozna@ajd.czest.pl

Abstract This paper presents a method for the validation of communicating autonomous vehicles (CAVs) systems. The approach focuses on the formal modeling of CAVs by means of timed automata, to allow the formal analysis through model-checkers of the vehicle behavior, including their fault tolerance due to various kinds of injected faults. We also present our case studies results, based on implementations of our CAVs' model in UPPAAL.

Keywords: Timed Automata, Model-Checking, Communicating Vehicles

1 Introduction

Autonomous vehicles are rational and sophisticated entities (agents) that, as the term already suggests, act autonomously (having different informations and/or diverging interests) across open and distributed environments. A *communicating* autonomous vehicles (CAVs) system is both a multi–agent system [6] and a real-time system [2]. More precisely, a CAV system is a network of vehicles that communicate to fulfill a mission as fast as possible while complying to the code rules and avoiding conflicts. Moreover, its correctness also depends on respecting a set of time constraints.

The CAVs systems have the potential to substantially affect safety, congestion, energy consumption, and, ultimately, will likely reduce crashes and pollution. Thus, the reliability of CAVs systems is a key concern of policymakers [3]. It is therefore appealing to formally verify the core CAVs behaviors in order to be confident in the integration of autonomous vehicles into the road traffic.

Formal modeling and verification of CAVs systems require not only the definition of both the vehicle states and the road (the environment) but also a specification of interactions between vehicles and an expressive property language. The resulting model of a CAVs system should be accurate enough to capture spatial and time aspects of the original system and also should provide a formal basis for verification of properties like effectiveness of overtakings, the resulting arrival order, and the impossibility of collisions (safety). The property language should be easy and appropriate for expressing the needed properties and applying automatic verification techniques for them.

Model checking [4] is a widely used automatic method for system verification. It provides algorithmic means for determining whether an abstract model – representing a hardware or software project (e.g., a CAVs system) – satisfies a formal specification expressed as a temporal logic formula. Moreover, if the property does not hold, the method identifies a counterexample execution that shows the source of the problem.

The main objective of this paper is to present our ongoing work on formal modeling and model checking of a communicating autonomous vehicles (CAVs) system, focusing in particular on the robustness of the decision policy. The modeling formalism that we use in our approach to specify the behavior of CAVs is a network of Timed Automata [2] with the synchronous semantics. We have chosen this formalism, because it is the most well-established model for the specification and verification of distributed real-time system design. Moreover, the automata formalism is a standard supported by several verification tools, e.g., the model checker UPPAAL [5].

The timed automata formalism allows us to create a clear and concise abstract model of the considered CAVs system. The formalism allows for assessing a vehicle decision policy robustness through a fault injection, and allows for applying model checking algorithms, in particular the algorithms designed for timed properties expressed in the temporal logic TCTL [1]. To the best of our knowledge, this is the first application of model checking to validate CAVs systems.

2 Modeling

The systems we consider are composed of several lanes forming a portion of a motorway on which several vehicles can move in the same direction, each one realizing some goal (mission). Each vehicle is defined by constants such as its length or breaking capacities. The vehicles are provided with various perception sensors allowing to observe the behavior of their neighbors. They are also able to communicate informations, which are not always directly observable such as their intention of lane change, which may be available immediately or after some delay. A vehicle behavior consists in performing, at its own frequency:

- a computation allowing to make a decision on the immediate action to be performed (*i.e.*, execution of the controller algorithm) followed by
- the communication of its intention (and possibly when the maneuver could start if not immediately) to all vehicles around (at least to those, which are capable to receive this information).

Received information coming from the other vehicles is stored in the vehicle's database and used when running the controller algorithm. The controller algorithm computes predictions of trajectories of the neighbors allowing in turn to compute the instructions for itself in terms of acceleration or lane change, all this of course has to be done safely and obeying traffic laws.

On this basis, we build an abstract but realistic enough model in timed automata that may be analyzable with various temporal logic queries in a reasonable[5] time.

In our framework, we consider a road section composed of one or more unidirectional lanes the vehicles move on, and we store at any time the coordinates (position) of each vehicle on it. The current position (x,y) (as well as the initial one) is expressed as the distance from the beginning of the road section (x) and from the road border (y). Due to verification purposes requiring discrete values domains, these distances are expressed using discrete values, but precise enough to satisfactorily model the vehicle progression on the road. As a consequence, the position of a vehicle is a point on a two-dimensional orthogonal grid on which the longitudinal and lateral gaps between adjacent points may be different.

A vehicle's mission is defined as a sequence of positions the vehicle has to visit. More precisely, we consider a sequence of sets of positions to be reached, with the requirement to reach at least one position of each set. Possible maneuvers a vehicle may launch in order to fulfill its mission are: keeping/changing lane and accelerating/decelerating or keeping its speed.

Each vehicle state is described by a record containing its position (the vehicle's center), its current speed, its direction (to the left, to the right or no change), and its acceleration (taken from a given range of possibilities, negative, positive and null). Its behavior is modeled by a timed automaton in the form of a loop which sets at some frequency the vehicle's acceleration and current direction. These indications are the result of the vehicle's controller algorithm, which can be different for each vehicle.

The vehicles' movements are modeled by another timed automaton, the environment, which computes and updates at every sample period all vehicles speeds and positions, according to the indications given by the vehicle's controllers.

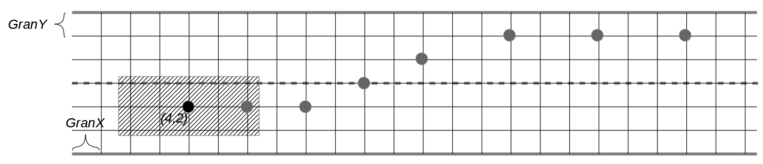


Figure 1. A two lanes road section discretized in abscissa with a granularity Gran_x and in ordinate with Gran_y. A vehicle (dashed rectangle) is centered in $(4,2)$ and its predicted timed trajectory $(6,2);(8,2);(10,3);(12,4);(15,5);(18,5);(21,5)$ for seven next time periods is shown by grey dots.

For our experiments, we developped a controller in a way that the goal of a vehicle is to reach positions of its mission as fast as possible while complying to the code rules and avoiding conflicts with other vehicles. To do so, the vehicle computes timed

[5] a few minutes, or hours for very complex systems and queries (but recall this is the time needed to analyze models off-line, not the time for a vehicle to react on the road)

trajectories of its neighbors from their positions, speeds and intentions, and chooses its own trajectory regarding to their.

Vehicle's predicted trajectories are represented using well chosen timed abstractions. In order to optimize the state space, the trajectories are never stored but they may be computed on demand from the information about vehicles, which has to be as compact as possible. Thus, the timed trajectories are represented as sequences of positions the vehicle will reach at some dates up to some time horizon (typically 10 s), as shown in Fig. 1.

3 Discussion on modeling choices and parameters

Our objective is to obtain a realistic modeling of a communicating autonomous vehicles system that allows their model checking in a reasonable time. Since the available tools that are able to formally verify such systems can only handle integer data structures, we have to propose a satisfactory discretization for the various physical quantities describing the vehicle behaviors. Moreover, the complexity of the verification procedures rapidly increases with the range of these integer variables, while the accuracy of the models requests an adequate granularity.

Our discrete representation of a vehicle's state will thus be encoded with the following discrete variables: its (longitudinal and lateral) positions x and y, its forward speed v with the corresponding forward acceleration a, and the direction of the vehicle $D \in \{-1, 0, 1\}$, corresponding respectively to a lateral move to the right, no change and to the left. With the exception of D, we will denote by Gran_a, Gran_v, Gran_x, Gran_y the granularity of these quantities, *i.e.*, the gap between two consecutive values (this allows to normalize the data as integers), and by N (with the corresponding subscripts) the size of the needed data structure, called a *range*.

The updates of the forward speed and position values after a period are expressed by the usual formulas. In order to adapt the data structures to the verification constraints, we want to make the system parametric depending on the value of the time period, called the *sample*, *i.e.*, the period with which the system is observed. Thus, to describe the system let us consider the following parameters and constants:

- S is the sample period, expressed in seconds (written as s);
- L is the length and R is the width of the road segment, in meters (written as m);
- V_{min} is the min and V_{max} is the max value of (longitudinal) speed expressed in meters per second (written as m/s);
- A_{min} is the min and A_{max} is the max value of acceleration expressed in meters per second squared (written as m/s/s);
- Gran_a is the granularity of the acceleration expressed in meters per second squared;
- W is the maximal value of the lateral speed expressed in meters per second.

This yields the acceleration range $\mathsf{N}_a = 1 + (A_{max} - A_{min})/\mathsf{Gran}_a$; we assume that 0 is one of the possible accelerations, and that A_{max}/Gran_a as well as A_{min}/Gran_a are integers. The acceleration is then expressed as $a = A \cdot \mathsf{Gran}_a$, longitudinal speed as $v = V \cdot \mathsf{Gran}_v$, longitudinal position as $x = X \cdot \mathsf{Gran}_x$, and lateral position as $y = Y \cdot \mathsf{Gran}_y$, where A, V, X, Y are integers.

Now, we are able to compute the granularities and ranges for the longitudinal and lateral positions and speeds, as well as give their normalized updates:

– For longitudinal speed, the update after one sample is $v' = v + a \cdot \mathsf{S}$. For a given granularity Gran_v, this lead to $V' = V + (A \cdot \mathsf{S} \cdot \mathsf{Gran}_a)/\mathsf{Gran}_v$ (in normalized variables). We choose a granularity $\mathsf{Gran}_v = \mathsf{Gran}_a \cdot \mathsf{S}$ (which does not introduce new losses) leading to a normalized speed update $V' = V + A$. The resulting range is $\mathsf{N}_v = 1 + (V_{max} - V_{min})/(\mathsf{Gran}_a \cdot \mathsf{S})$, using a ceiling function if the division does not provide an integer.
– For the longitudinal position, the update after one sample is $x' = x + v \cdot \mathsf{S} + a \cdot \mathsf{S}^2/2$. For a given granularity G_x this leads to $X' = X + ((V \cdot \mathsf{S} \cdot \mathsf{Gran}_v) + (A \cdot \mathsf{S}^2/2 \cdot \mathsf{Gran}_a))/G_x = X + (2V + A)(\mathsf{Gran}_a \cdot \mathsf{S}^2/2)/G_x$ (in normalized variables). In order to avoid losses, we should choose the granularity $G_x = \mathsf{Gran}_a \cdot \mathsf{S}^2/2$. However this granularity will usually be too small (for instance, if $\mathsf{Gran}_a = 1$ and $\mathsf{S} = 10^{-1}$, we get a granularity of 5 mm), leading to a huge range. In order to avoid a state space explosion, we shall thus approximate x with a precision of $\mathsf{Gran}_x = p \cdot G_x$, with an adequate parameter p. Hence, the normalized update of x becomes $X' = X + (2V + A)/p$ (rounded) and the corresponding range is $\mathsf{N}_x = \mathsf{L}/\mathsf{Gran}_x$. In order to choose a suitable p, we may observe that Gran_x is the maximal longitudinal loss of precision we may face during a sample. Hence, we may introduce the (normalized, maximal) loss of precision[6] during one second $\mathsf{Norm}_x = \mathsf{Gran}_x/\mathsf{S} = p \cdot \mathsf{Gran}_v/2$. Norm_x may be fixed independently from the constants of the program and we get the corresponding $\mathsf{Gran}_x = \mathsf{Norm}_x \cdot S$ and $p = 2 \cdot \mathsf{Norm}_x/\mathsf{Gran}_v$.
– For the lateral position update after one sample we have $y' = y + W \cdot \mathsf{S} \cdot D$, leading to a granularity $\mathsf{Gran}_y = W \cdot \mathsf{S}$ and the range $\mathsf{N}_y = \mathsf{R}/\mathsf{Gran}_y$. In the normalized variables, this becomes $Y' = Y + D$.

The size E of the state space per vehicle, due to the data structure, is obtained by multiplying all the above variable ranges, while it becomes E^n for n vehicles. Fortunately, the formal tools do not necessarily construct the whole state space to analyze such systems. Of course, the complexity of the verification procedures also depends on the number of clocks used in the model and on the number of time zones they generate, as well as on the degree of non determinism present in the specification and the difficulty to solve the queries.

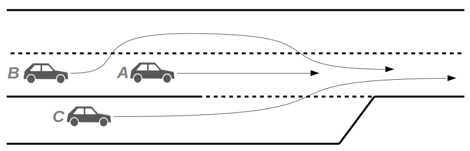
4 Experiments with the model

We illustrate our approach on a scenario implying three vehicles evolving on a three-lane highway section of $\mathsf{L} = 500$ m long and $\mathsf{R} = 10.5$ large. One of the lanes is a junction lane, which starts at the beginning of the section, joins the right lane after 200 m and ends 200 m later. The constraints defining the velocity of the vehicles are $V_{min} = 0$ m/s and $V_{max} = 40$ m/s (= 144 km/h); $A_{min} = -5$ m/s/s and $A_{max} = 3$ m/s/s, and $W = 1$ m/s. We fix $\mathsf{Gran}_a = 1$ m/s/s and the environment sample $\mathsf{S} = 0.1$ s (10 Hz). Such an S allows to monitor the system's behavior in a satisfactory way and such a Gran_a yields a sufficient number of acceleration choices for the vehicle's controllers.

With such parameters and the granularity guaranteeing no further loss of information, *i.e.*, $G_x = 0.005$ m, the complexity of the data structure per vehicle is of the order

[6] Actually two times the loss of precision thanks to rounding.

of 10^{11} and it becomes $(10^{11})^3$ for our 3 cars scenario, which is problematic for the verification tools. We then fix a reasonable normalized accuracy $\mathsf{Norm}_x = 1$ and get $p = 2 \cdot q/\mathsf{Gran}_v = 20$. Therefore, the granularity on x becomes $\mathsf{Gran}_x = p \cdot G_x = 0.1$ m and this approximation allows to divide the state space by $p^3 = 8000$ while not significantly impacting the behavior of the model.



Query\Variant	Base case	B-slowed	C-slowed	C-moved
Exists B overtakes A	T	F	T	T
Always B overtakes A	F	F	F	F
Exists C overtakes A	T	T	F	T
Always C overtakes A	T	T	F	F
Exists B overtakes C	F	F	T	T
Always B overtakes C	F	F	T	T

Figure 2. Top: initial situation of the Base Case (BC) with $x_A = 50$, $x_B = 0$, $x_C = 20$, $v_A = 20$, $v_B = 35$ and $v_C = 30$; B-slowed is BC with $v_B = 30$, C-slowed is BC with $v_C = 25$ and C-moved is BC with $x_C = 15$; bottom: queries and results.

We aim at studying the vehicle's controllers decisions (working at a period of 0.11 s) and their impact on each others. The vehicles A, B and C are initially positioned as shown in Figure 2 (top). They are assumed to have initially a null acceleration and their mission is to reach the right lane at the end of the road.

For a number of initial situations, we consider a series of properties – expressed as CTL formulae [4] – to check the effectiveness of overtakings, the resulting arrival order, and of course the possibility of collisions. The table in the bottom of Figure 2 gathers the results of the model checking for the base case defined above and a few of its variants.

For the Base Case, we can observe that it is possible that B overtakes A, which is slower, but this does not happen in every possible path. We can also observe that C always overtakes A but is not overtaken by B, meaning that C always arrives first. For B-slowed, the only difference with the Base Case is that B can no longer overtake A, independently of the order in which the vehicles take their decisions. For C-slowed, we can see that C now never overtakes A and is always overtaken by B, which means that C always lets the two other go before changing lane. For C-moved, we can see that the only difference with C-slowed is that C always waits for B to go first but then there are some paths where it goes before A and some where it doesn't. To explain this behavior, a hypothesis could be that C always waits for B, and when A is overtaken by B, C also overtakes A. We can check this with another query $(B \text{ overtakes } A) \rightarrow (C \text{ overtakes } A)$, which is indeed true. Finally, there is no path in any scenario that allows collision.

Despite the high number of states, UPPAAL handles most of the queries in less than 10 seconds on an office laptop. Unfortunately, in some cases such as for example C-slowed, some queries needed to explore a large part of the state space and took more than 4 minutes, meaning that the approximation on the position was necessary.

5 Experiments with the model after fault injection

The scenarios analyzed in section 4 were assumed to take place in a perfect environment, where each vehicle had a perfect knowledge of the other vehicles, thanks to its sensors and the communication devices, with no loss of informations (but the basic discretization). It resulted in behaviors that were always safe (*i.e.*, the controllers always made safe choices). Here, we address the assessment of robustness of the system by injecting various kinds of faults. Concretely, we will degrade the environment by modifying the knowledge of some vehicles, representing either defective sensors or communication problems, such as for example a larger latency. We shall thus consider the same properties as before on the following degraded versions of the base case:

– Var-Null: the other vehicles' acceleration is interpreted as 0;
– V-Up: the other vehicles' speed is multiplied by $3/2$;
– V-Down: the other vehicles' speed is multiplied by $2/3$;
– X-Up: the other vehicles' longitudinal position is increased by 2 m;
– X-Down1: the other vehicles' longitudinal position is decreased by 5 m;
– X-Down2: the other vehicles' longitudinal position is decreased by 10 m.

Those faults impact the trajectory that each vehicle computes in order to represent the behavior of its neighbors. Table 1 gathers the model checking results for those versions.

Query\Variant	Var-Null	V-Up	V-Down	X-Up	X-Down1	X-Down2
Exists B overtakes A	F	T	F	F	T	T
Always B overtakes A	F	F	F	F	F	T
Exists C overtakes A	T	T	T	T	T	T
Always C overtakes A	T	T	T	T	T	T
Exists B overtakes C	F	T	F	F	F	F
Always B overtakes C	F	F	F	F	F	F
Exists collision	F	T	F	T	F	T

Table 1. Meaning of the queries with the corresponding results for each kind of fault.

For Var-Null, where each vehicle is unable to know the value of the other vehicles acceleration, and V-Down, where they use a speed value lower than the reality, we observe that B does not succeed in overtaking A anymore, but there is no collision. X-Up shows the same behavior for B, but a collision is then possible. The trace given by the model checker shows that B crashes with A while trying to overtake it. For V-Up, where the speed value of any other vehicle is increased, we observe that B sometimes overtakes C, and a collision is also possible. The trace given by the model checker shows that B crashes with A without even trying to change lane: considering a false speed value, the controller decides that there is no need to change lane, until it becomes too late to prevent the collision. Finally, X-Down1, where the vehicles' sensors use a longitudinal position decreased by 5 meters shows a behavior identical to the base case, while X-Down2, where the estimated positions are decreased by 10 meters, shows that B always overtakes A but a collision may happen. The trace shows that the problem occurs at the junction, between C and the vehicle on the right lane. Indeed, when the vehicles on each side of the junction are actually side by side, they perceive each other 10 m behind and so decide they have enough room to pass, which leads to a collision.

Based on these results, we can say that, in this particular situation, the cases that often cause problems to our controller are those where the speed or position estimated values of the other vehicles are increased in such a way that the vehicle underestimates how close to collision it is. Of course, doing these checks for a single initial situation does not allow us to determine a general trend about the robustness of our controller. However, applying the same process on various situations, one might be able to evaluate the level of faults a given controller can handle without leading to a collision.

6 Conclusion

We proposed an efficient formal modeling of a complex real time multi-agent system in order to obtain guarantees of its good functioning and robustness. The inherent size of the system made it impossible to address the problem in a direct and naive way, but a suitable discretization and approximation allowed to use model checking exhaustive techniques while limiting the state explosion.

We illustrated the usage of such a modeling to assess the performances of decision making of autonomous communicating vehicles. To do so we implemented controllers and checked their properties, first when evolving in a somehow perfect environment and then in a modified faulty environment. The obtained results show the differences and limits of the considered decision policy, pointing out vulnerabilities. The developed framework is parametric and may easily be adapted to study various kinds of scenarios with a fixed a priori precision. It allows to check a large range of properties, including of course the absence of collisions.

In the future, we plan to extend this approach to cooperative vehicles in order to assess various cooperation strategies in order to improve the safety and increase the fluidity of the traffic, such as negociations between vehicles. Also, implementing obstacles and human controlled vehicles would be interesting in our approach to quantify decision policy robustness. These extensions would certainly require further abstractions and better exploit system's symmetries.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
2. R. Alur and D. Dill. Automata for modelling real-time systems. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.
3. James M. Anderson, Nidhi Kalra, Karlyn Stanley, Paul Sorensen, Constantine Samaras, and Oluwatobi Oluwatola. *Autonomous Vehicle Technology. A Guide for Policymakers.* Research Reports. RAND Corporation, 2016. ISBN: 978-0-8330-8398-2.

4. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
5. Uppaal. http://www.uppaal.org/, 2016.
6. M. Wooldridge. *An introduction to multi-agent systems - Second Edition*. John Wiley & Sons, 2009.