

A Formal Approach to Autonomous Vehicle Coordination

Mikael Asplund, Atif Manzoor, Mélanie Bourroche,
Siobhán Clarke, Vinny Cahill
{asplunda, atif.manzoor, melanie.bourroche,
siobhan.clarke, vinny.cahill}@scss.tcd.ie

Lero - The Irish Software Engineering Research Centre
Distributed Systems Group
School of Computer Science and Statistics
Trinity College Dublin

Abstract. Increasing demands on safety and energy efficiency will require higher levels of automation in transportation systems. This involves dealing with safety-critical distributed coordination. In this paper we demonstrate how a Satisfiability Modulo Theories (SMT) solver can be used to prove correctness of a vehicular coordination problem. We formalise a recent distributed coordination protocol and validate our approach using an intersection collision avoidance (ICA) case study. The system model captures continuous time and space, and an unbounded number of vehicles and messages. The safety of the case study is automatically verified using the Z3 theorem prover.

1 Introduction

As the number of cars in the world crosses the 1 billion mark and the future travel needs of the world population keep increasing, we are paying an increasingly heavy price. Every year nearly 1.2 million people get killed in traffic [25], and as many die from urban pollution. Moreover, transportation stands for 23% of the total emissions of carbon dioxide in the European Union [11].

Better software allows us to make cars smarter, safer, and more efficient, thereby ameliorating some of the adverse effects of car-based transport. Modern cars are equipped with a wide range of sensors and driver assistance systems and there are already a number of self-driving cars that are being tested by the major automotive companies as well as Google. The fact that the state of Nevada passed legislation allowing driver-less vehicles to operate on public roads can be seen as a sign of the momentum in the industry at the moment. Previously unsolved problems such as accurate positioning and reliable object detection now have credible solutions. The next big challenge is to enable efficient *coordination* among smart vehicles to further increase the safety and efficiency of the traffic.

Collisions in intersections constitute 45% of all traffic personal car injury accidents [27], so there is a clear need for collision avoidance systems. Having a centralised authority for each intersection that directs the traffic can be a good

alternative to having traffic lights. However, the large majority intersections do not even have traffic lights today. It would not be cost effective to put a central manager to all these unmanaged crossings, a fully distributed solution will be needed. On the other hand, distributed coordination is a non-trivial problem. A dynamic environment where cars move at high speed and where communication is unreliable and subject to interference creates many challenges. Yet any solution to such a distributed coordination problem must be able to guarantee safety.

In this paper we propose to utilise the strengths of automated reasoning tools to tackle the problem of safe distributed coordination. We show how a coordination problem can be formalised in a constraint specification language called SMT-lib [3] and verified with the Z3 theorem prover [9]. The novelty of our approach lies in employing a fully automated theorem prover to a distributed coordination problem involving explicit message passing, continuous time and space as well as an unbounded number of cars. Our focus is not on new verification methods for hybrid systems, but rather on the application of formal methods to a coordination approach and how to verify safety of a collaborative vehicular application. Our longer term objective is to incorporate the basic building blocks introduced in this paper in a general tool for modelling and verifying vehicular applications. To evaluate the feasibility of our approach we model an intersection collision avoidance scenario, which is an instance of a distributed coordination problem. In summary, there are three main contributions of this paper.

- A formalisation of a distributed coordination protocol.
- A constraint-based modelling approach for collaborative vehicular applications.
- A simple but realistic case study demonstrating the usefulness of our approach.

The rest of this paper is organised as follows. Section 2 provides a formal description of the coordination problem and the CwoRIS protocol. The intersection collision avoidance case study is presented in Section 3 followed by Section 4, outlining the verification and proof strategies. Section 5 contains related work and finally, Section 6 concludes the paper.

2 Distributed Coordination as Constraint Verification

We now proceed to formalise the distributed coordination problem. We begin by giving an overview of our approach, then go on to describe how we model the communication channel before describing our formalisation of the CwoRIS coordination protocol.

2.1 Overview

Consider the problem of designing a software subsystem for a car (we use the more general notion of entity, or sometimes vehicle) that can affect the steering and speed of the entity and that takes its decision based on communication with surrounding vehicles. Examples of such systems are collaborative adaptive cruise

control, advanced collision avoidance systems, and lane merging applications. Our aim is to prove such a system safe by proving that a specific entity A will not collide with another object.

Figure 1 shows an overview of how our system model is constructed. It is composed of a “core” automaton and a set of time-dependent constraints.

With core automaton, we refer to the state transitions involving variables specific to entity A . We now proceed to provide a more formal description of the system model and how we represent it as a SMT problem. We model the system as a tuple M as described below.

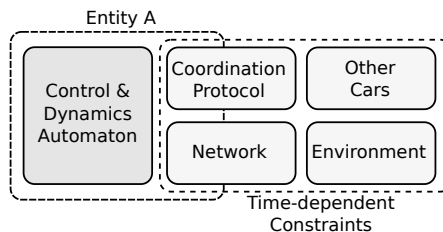


Fig. 1. System Model Overview

$$M = (\mathbf{E}, \mathbf{M}, \mathcal{S}, \mathcal{I}, T, \mathcal{F}, \mathcal{C})$$

\mathbf{E} - a set of entities (i.e., the vehicles in the system)

\mathbf{M} - a set of messages

\mathcal{S} - a set of states

$\mathcal{I} \subset \mathcal{S}$ - set of initial states

$T : \mathcal{S} \times \mathcal{S} \rightarrow \mathbf{Bool}$ - A transition function

\mathcal{F} - a finite set of uninterpreted functions

\mathcal{C} - a finite set of constraints

Note that the sets $\mathbf{E}, \mathbf{M}, \mathcal{S}, \mathcal{I}$ can all be infinite, thereby allowing us to model an unbounded number of cars and messages. The set of uninterpreted functions (or predicates), \mathcal{F} , provides the semantics for the states. The allowed domains and ranges of the functions are real numbers (time), integers, and any of the sets in our model. An example of an uninterpreted function that we use in our model is $x : \mathbf{E} \times \mathbb{R} \rightarrow \mathbb{R}$ which denotes the x position of an entity at some given time point.

The constraints in \mathcal{C} provide us with a way to describe the properties of the environment and other assumptions that we need to make. The constraints apply over the same domains as the uninterpreted functions, \mathcal{F} , and may also contain quantifiers. An example of a constraint (which we do not use) could be $\forall e \in \mathbf{E}, t \in \mathbb{R} : x(e, t) \leq 3.0$, which would require the x position of all entities to be less than 3.0 at all times.

We let the states in \mathcal{S} and the transition function T denote the state and behaviour of the specific entity A . The behaviour of other entities in the system is modelled using constraints in \mathcal{C} . This allows us to provide a more detailed internal model of a single entity, and model other entities using assumptions on their observable behaviour (including communication).

Finally, consider the transition function $T(i, j)$, where i and j are states, which is used to characterise the behaviour of entity A . We encode the hybrid automaton of A as a transition function that alternates between timed and non-timed transitions. Let $\delta : \mathcal{S} \rightarrow \mathbf{Bool}$ (we write δ^i) be an uninterpreted function,

Table 1. Communication Predicates

Predicate	Type	Description
$sent(m)$	Bool	message m was sent
$received(m, e)$	Bool	m was received by entity e at some point in time
$source(m)$	E	the sender of m
$sendtime(m)$	\mathbb{R}	the send time of m
$receivetime(m, e)$	\mathbb{R}	when m was received by entity e (if m is never received by e , this can have any value)
$isAck(m)$	Bool	True if message is an acknowledgement
$getReq(m)$	M	if m is an acknowledgement message, this denotes the message that m acknowledges

denoting whether the next transition should be a timed transition or not. Then we can define T as:

$$T(i, j) \equiv (T_D(i, j) \wedge \neg\delta^i \wedge \delta^j) \vee (T_C(i, j) \wedge \delta^i \wedge \neg\delta^j)$$

Where T_D is the transition function for non-timed (discrete) transitions and T_C for timed transitions (continuous).

2.2 Communication

We now proceed to introduce a subset of \mathcal{F} relating to message passing. These are the basic concepts that we use to formally reason about communication in the system. Table 1 lists the predicates, the resulting type and a description of each. It might be worth pointing out a couple of things. First, the sent and received predicates do not have a time parameter. Thus, the semantics is that if a message m is sent at any time, then $sent(m)$ is true. To check whether a message had been sent at some given time t , this can be expressed as: $sent(m) \wedge (t \geq sendtime(m))$. Finally, messages can be either request messages or acknowledgements to requests. Thus the last two predicates are used to determine the message type and to identify the request associated with a given acknowledgement message.

We now describe the constraints relating to the basic communication properties. There are three constraints that have to be satisfied. First, any message that has been received by some entity must have been sent.

$$\forall m \in \mathbf{M}, e \in \mathbf{E} : received(m, e) \Rightarrow sent(m)$$

Second, the reception time of a message m at entity e must be strictly greater than the send time of the message.

$$\forall m \in \mathbf{M}, e \in \mathbf{E} : receivetime(m, e) > sendtime(m)$$

Finally, we need some consistency checks for when an acknowledgement can be sent. The following constraints states that for all acknowledgement that have been sent three conditions must be met, (1) it must correspond to a received message, (2), the received message cannot be an acknowledgement, and (3) the

acknowledgement must have been sent after receiving the request.

$$\begin{aligned} \forall m \in \mathbf{M} : & (sent(m) \wedge isAck(m)) \Rightarrow \\ & (received(getReq(m), source(m)) \\ & \wedge \neg isAck(getReq(m)) \\ & \wedge receivetime(getReq(m), source(m)) \leq sendtime(m)) \end{aligned}$$

The above set of predicates and constraints provides some very basic elements of communication, which can easily be provided by any communication interface in a real application. However, in order to solve the coordination problem, we need to make an additional assumption on membership information. For this purpose we assume the existence of an *active area* in which entity A operates and that all entities within the active area are known to each other (i.e., essentially a perfect membership protocol). The membership information allows an entity to decide whether a message it has sent has reached all other entities in the area. While solving the membership problem using purely communication is recognised as a difficult problem [6], it can be solved in the vehicular domain with the aid of ranging sensors as shown by Slot and Cahill [23].

2.3 Distributed Coordination

We base our formalisation of distributed coordination on previous work by Bourouche [5] and Sin et al. [22]. The basic idea behind this model is that vehicles do not need to fully agree on a shared state in order to achieve safe coordination. Instead, the basic concept is that of *responsibility*. Each entity has a responsibility to ensure that certain safety criteria are met. If an entity is not able to ensure that its planned actions are compatible with those of other entities in the environment, it must adapt its behaviour accordingly (e.g. by stopping). The key aspect of this approach is that an entity does not need to agree on the behaviour of other entities in the system. While this might sound trivial, it is actually a step away from approaches where first all entities reach a distributed agreement on the course of actions to take, allowing greater flexibility.

In the CwoRIS protocol by Sin et al. [22], the responsibility requirement is implemented with the means of *resources*. A resource corresponds to a physical area of the road. An entity should not enter a resource without having made sure that it has exclusive access to the resource. While space does not allow a full description and explanation of the rationale of the CwoRIS protocol, we provide a brief intuition of how it works. Note that for the purpose of this formalisation we have made some simplifying assumptions compared to the original protocol. We allow only a single resource, requests are not allowed to be updated, and a sent request is assumed to be immediately received by the sender of the message, and no new entities enter the active area during the negotiation. These simplifications do not have a big impact on the core logic of the protocol, and we expect that removing these restrictions from the formalisation is a straightforward process. Table 2 describes the predicates that we use in the coordination mechanism.

Table 2. CwoRIS predicates

Predicate	Type	Description
$hasRequest(e, t)$	Bool	Entity e has an active request at time t
$c(e, t)$	M	Current request of entity e at time t
$start(m)$	\mathbb{R}	Resource request start time
$end(m)$	\mathbb{R}	Resource request end time
$prio(e)$	\mathbb{Z}	Priority of an entity
$valid(m)$	Bool	Message m is a valid request
$vtime(m)$	\mathbb{R}	Time when m was validated
$conflict(m, m')$	Bool	Requests m and m' are in conflict
$accepted(m, e)$	Bool	Message m is accepted by entity e
$hasResource(e, t)$	Bool	Entity e has the resource at time t

In essence the CwoRIS protocol works by entities sending out requests to access a shared resource, after which $hasRequest$ becomes true for the sender entity, and the current request is referred to as $c(e, t)$. Each request has a $start$ and end time and each entity has a unique priority¹. If an entity has received an acknowledgement from all other entities in the area and not received any conflicting request from an entity with a higher priority, the request is considered to be *valid*. A *conflict* is said to occur between two requests if their request times overlap.

When sending out a new request, a node must make sure that the request it sends does not conflict with any previously received request that it has accepted. A message m is *accepted* by entity e if it is received by e , and one of three conditions hold

- e does not have a request when receiving m
- m does not conflict with the current request of e
- e has a strictly lower priority than the sender of m

Thus a message from a lower priority entity can be ignored by an entity with a higher priority. Note that two entities cannot ignore each others requests since both cannot have a higher priority than the other. Finally, node $hasResource$ at time t if and only if it has a valid request for that resource and the time interval of the request covers t . We state this last constraint formally as it is the main interface to the other components in the system.

$$hasResource(e, t) \equiv hasRequest(e, t) \wedge valid(c(e, t)) \wedge vtime(c(e, t)) < t \\ \wedge start(c(e, t)) \leq t \leq end(c(e, t))$$

This concludes our description of the coordination protocol. Naturally, most of the above description is rather textual rather than formal. We refer to the full model² for the exact constraints.

¹ Uniqueness can be achieved through e.g. globally unique IPv6 addresses that are part of the future communication standard for vehicular applications, and should also take into account relative proximity to the intersection.

² Available at <http://code.google.com/p/smtica/>

3 Case study

To demonstrate the applicability of our approach we have chosen a basic intersection scenario to model and validate. We first outline the general scenario and our assumptions, and then describe how the states and transition functions for the vehicle automaton are defined.

3.1 Scenario

We consider a four way intersection as depicted in Figure 2. The intersection is not equipped with a central traffic control mechanism such as a traffic light, so vehicles need to coordinate their actions to avoid collisions. The figure shows entity A approaching the intersection. For simplicity we have aligned the roads with the x and y axes respectively, and assumed that entity A will not turn. Thus, it will only need to travel in the x direction

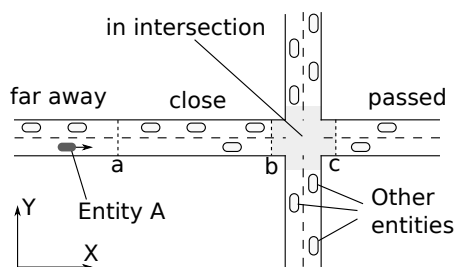


Fig. 2. Intersection Scenario

To tackle a wider range of road geometries one needs to transform coordinate system of the vehicle along the road (i.e., using longitudinal and lateral directions). Allowing the vehicle to turn can be easily incorporated in the model. There are four conceptual regions for this entity in relation to the intersection, “far away” when the x position is less than some specified value a , “close” when $a \leq x \leq b$, “in intersection”, when $b \leq x \leq c$, and passed when $x \geq c$.

In our model, we have chosen to put as few restrictions on the allowed behaviour of the system as possible. However, some restrictions are necessary to prove the desired safety properties. Since the actual behaviour of a car is more restricted than our model of it, by proving that the wider envelope is safe, it follows that a restricted subset of the behaviour will also be safe.

We further assume that all entities use the CwoRIS resource reservation protocol to negotiate access to the intersection, and that if another entity is in the intersection then it must be in the active area given by the membership protocol. Apart from the assumption that entities keep in lane, the positions $x(e, t)$ of entities e other than entity A , are only restricted in the sense that if entity e is in the intersection, it must have the resource. For entity A this is not assumed, but proven to hold as explained in Section 4.2.

3.2 Core automaton

We now proceed to describe the core automaton (the states \mathcal{S} , the initial states \mathcal{I} , and the transition function T) that encodes the behaviour of entity A . The

Table 3. State variables

Continuous state variables			Discrete state variables		
Predicate	Type	Description	Predicate	Type	Description
t^i	\mathbb{R}	time at state i	l^i	\mathbf{L}	location
x^i	\mathbb{R}	x position in state i	v_t^i	\mathbb{R}	intended target speed
y^i	\mathbb{R}	y position in state i	P^i	\mathbf{Bool}	will pass
v^i	\mathbb{R}	speed in state i			

logic of the vehicle is quite straightforward and roughly based on the intersection collision avoidance application described by Sin et al. [22].

Table 3 contains the continuous and discrete variables (that actually encode the discrete states), and Figure 3 shows a graphical representation of the discrete state transitions. The continuous state variables are time, x and y position and speed. The discrete variables are as follows. The location $l \in \mathbf{L} = \{\text{farAway, close, inInter, passed}\}$ denotes the logical location of the vehicles in relation to the intersection. The intended target speed v_t denotes the reference value to which the vehicle tries to adapt its speed. The Boolean variable P denotes an internal decision corresponding to whether the vehicle intends to pass the intersection in the near future.

Now consider Figure 3 which shows the discrete states and transitions of the core automaton (where all states have implicit self-loops). Initially, the vehicle is considered to be far away from the intersection, but when the x position of the vehicle passed the proximity point a , its state will change. There are two possibilities, either the entity has acquired the resource and will have it for a sufficiently long time to pass the intersection (we denote this *willHaveResource*), in which case it will set P (will pass) to true and prepare to cross the intersection. Otherwise, the vehicle must break (set target speed $v_t = 0$), and wait until a resource is acquired.

Once the entity has secured the resource it will need to maintain a minimum speed (v_{min}) while close to or in the intersection. When the entity passes $x = b$,

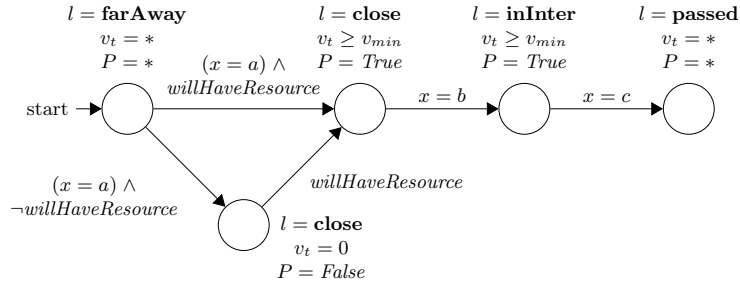


Fig. 3. Automaton for the behaviour of vehicle A

it is considered to be in the intersection, until it passes point $x = c$, after which it sets its location to “passed”.

Having covered the logical control of the vehicle, we now turn to a simple model of its physical characteristics. This is defined by the continuous transition function $T_C(i, j)$ (where i and j are states), which is a conjunction of criteria on the allowed evolution (or flow) of the continuous variables.

$$T_C(i, j) \equiv \text{move}(i, j) \wedge \text{speed}(i, j) \wedge \text{duration}(i, j) \wedge (t^i < t^j) \wedge \text{consts}(i, j) \wedge \text{inv}(i, j)$$

The allowed movement ($\text{move}(i, j)$) of the vehicle is defined below. This movement formula assumes that the average speed during the duration of the continuous transition is equal to the mean of the start and end speeds (v^i and v^j). This is true if for example the acceleration is constant during that time.

$$\text{move}(i, j) \equiv \left(x^j = x^i + \frac{v^i + v^j}{2} (t^j - t^i) \right)$$

The speed change during a continuous transition is controlled by the minimum absolute acceleration parameter a . In line with letting the behaviour of the car to be unrestricted unless required to prove the safety of the system we do not limit the maximum acceleration. Note that this does not mean that we assume vehicles to have unbounded acceleration, but rather that as long as the speed change is within the envelope we are able to prove system safety. Formally, the allowed speed change is expressed as follows.

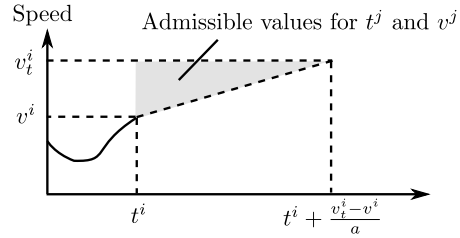


Fig. 4. Speed

$$\begin{aligned} \text{speed}(i, j) \equiv & ((v_t^i = v^i) \wedge (v^j = v^i)) \vee \\ & ((v_t^i < v^i) \wedge (v_t^i \leq v^j \leq v^i - a(t^j - t^i))) \vee \\ & ((v_t^i > v^i) \wedge (v^i + a(t^j - t^i) \leq v^j \leq v_t^i)) \end{aligned}$$

If the duration of the continuous transition is long enough, the above formula will cause the resulting speed to pass the intended target speed. Therefore, we add a restriction on the duration of a continuous transition when there is a speed change:

$$\text{duration}(i, j) \equiv (v_t^i = v^i) \vee \left(t^j \leq t^i + \frac{|v_t^i - v^i|}{a} \right)$$

The easiest way to understand the above formulae is through figure 4. It shows the case where the target speed is higher than the increased speed. The gray area shows the admissible values for t^j and v^j .

The fourth criterion ($t^i < t^j$) in T_C states that a timed transition must increment the clock, since otherwise it would be possible to have an infinite amount

of transitions without any time passing. The $const(i, j)$ criterion simply requires all discrete variables to stay constant during the timed transition. Finally, with the invariant criterion inv , we introduce a restriction which is merely for sake of the reducing the search space. There is no algorithm for solving general non-linear arithmetic constraints, and with the model description so far Z3 returns *unknown* when asked for satisfiability. We were thus forced to restrict the search space by requiring that the speed variable v to be a multiple of $0.5m/s$. Each speed step can be seen as modelling a $0.5m/s$ wide range of the actual vehicle speed³. Note that this must be done with some care. Specifically, one must make sure that it does not lead to dead ends in the automaton from which there are no outgoing transitions, as we show in the next section.

4 Verification

Having described our model we now outline our efforts to verify safety properties of our model. Let $\mathcal{R} \subset \mathcal{S}$ be the set of states that are reachable from \mathcal{I} with a finite sequence of transitions. Our objective is to show that all states in \mathcal{R} fulfil some safety property $safe^i$. This predicate should exclude the possibility of vehicle A colliding with any other entity within the area, so we let $safe^i \equiv safeDist^i$, where:

$$\begin{aligned}
 safeDist^i \equiv \forall e \in \mathbf{E} : & (e = A) \vee \neg inArea(e) \\
 & \vee (|x(A, t^i) - x(e, t^i)| > X_{min}) \\
 & \vee (|y(A, t^i) - y(e, t^i)| > Y_{min})
 \end{aligned} \tag{1}$$

Note that we specify the minimum allowed distance individually for the x and y dimensions simply because the constraint solver we used could not cope with a proper euclidean distance constraint. We leave it to future work to find a way around this limitation. Moreover, we only include the vehicles in the active area to reduce the verification complexity. Having defined the safety predicate we now want to prove that all reachable states are safe: $M \models \forall i \in \mathcal{R} : safe^i$

Unfortunately, this formulation is not very suitable for automatic verification; we first have to transform the problem into a more tractable one. To do this we employ as basic variant of k-induction and manual invariant strengthening.

4.1 Safety by induction

Proving safety using induction and a SAT solver was introduced by Sheeran et al. [21] and is naturally extended to SMT solvers. The basic idea is to prove safety of the system by induction, using paths of length K as the base case. By testing increasingly larger values for K , this method will eventually provide an answer for finite system representations. In the case of our model we use $K = 2$. Using only the first state as the base case is not enough since both a

³ This can be seen as a discretisation of the speed variable, but does not restrict the possible values for the other continuous variables.

continuous and a discrete transition is needed to ensure that the next state is one that could reasonably occur for this system. Starting with the base case that all initial states and all successors to the initial states are safe:

$$M \models \forall i \in \mathcal{I}, j \in \mathcal{S} : safe^i \wedge (T(i, j) \Rightarrow safe^j)$$

We then formulate the inductive step, that if two successive states are safe, then the third successor must also be a safe state.

$$M \models \forall i, j, k \in \mathcal{S} : (safe^i \wedge safe^j \wedge T(i, j) \wedge T(j, k)) \Rightarrow safe^k$$

Recall that to prove these formulae we try to assert their negation in Z3. When the solver concludes that the negated formula is unsatisfiable, we know that the formula is a consequence of the model M , and that all reachable states are safe. If, on the other hand, the solver finds a solution to the constraint problem there are two possibilities. Either the system is not safe, in which case the variable assignment that satisfies the negation of what we want to prove provides us with an example of how the system can enter an unsafe state. This provides useful information for debugging the formulation of the model.

The other case is worse. The fact that the inductive step is false does not necessarily mean that the system is unsafe. If the solver finds a case where the safe states i and j lead to an unsafe state k , but i and j are not reachable states, the counterexample is of no use. In this case, the system might or might not be safe; we have no way of knowing. Increasing K does not help in our case since we do not enforce maximal progress of timed transitions. Moreover, the safe *safeDist* predicate only considers the positions of the vehicles. Thus, to prove safety we need to replace the safety criterion with a stronger invariant that also ensures proper speed and resource allocation.

4.2 Safety Invariants

The problem of finding invariants is often the key of automated theorem proving. Fortunately, in our case, the invariants are fairly straightforward to the problem at hand. Moreover, we do not consider it to be a problem that these need to be defined manually. When designing a system for automotive safety, there will be a large number of criteria in the system specification and these definitely fall within this range. Apart from the *safeDistⁱ* property (equation (1)) we add two more invariants. The first *hasResInInter* requiring that if the entity is in the intersection, then it must (1) have acquired the resource (2) have this resource for a sufficient amount of time (3) have decided to pass the intersection and (4) have a minimum target speed v_{min} .

Finally, the predicate *safeSpeed* limits the maximum speed that the entity can have when being close to the intersection, but not having acquired the resource. Or, similar to the above, the entity has decided to pass, to have a minimum target speed. Figure 5 shows a graphical representation of the maximum speed before an intersection. When the vehicle is far away from the intersection, there is a fixed maximum speed. However, when the vehicle approaches the intersection, if

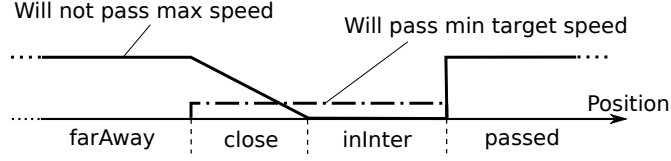


Fig. 5. SafeSpeed

it has not decided to pass, it must start to slow down. The final safety invariant is then defined as $safe^i \equiv safeDist^i \wedge hasResInInter^i \wedge safeSpeed^i$.

With these additions and the induction scheme outlined in the previous subsection we were able to prove the safety of all reachable states using the Z3 solver.

4.3 Deadlock freedom

The final step in our verification process is to ensure that the model is sound in the sense that we have not made it overly restrictive. In particular, it should always be possible to transition to a new state. If the model is stuck, it means that we have made an error. One possible approach to show this is to use the same inductive reasoning as for proving safety.

$$M \models \forall i, j \in \mathcal{S} : T(i, j) \Rightarrow \exists k \in \mathcal{S} : T(j, k)$$

However, when feeding the negation of this formula to Z3 it returns “unknown”. It turns out that one of the core reasons for this is that time must increase for a continuous transition ($t^j > t^i$). Unfortunately we cannot just remove this criterion, since it is required to prove safety. Instead we found another solution to this problem, based on *constructing* a successor to every state.

We introduced a successor function $succ : \mathcal{S} \rightarrow \mathcal{S}$ that for each state returns a new state to which there is a valid transition. The successor function can be derived without major effort from the definition of the transition function. Since $succ$ is always guaranteed to give an output for every input state, we can prove freedom from deadlock by proving the following formula.

$$M \models \forall i, j : T(i, j) \Rightarrow T(j, succ(j))$$

The reason for having an antecedent ($T(i, j)$) is that this ensures that the state variables in state j are not in themselves contradictory.

4.4 Final remarks

In addition to the above, we asserted basic properties such as that no two entities both believe that they had the resource and that there is a sequence of transitions in which A can pass the intersection. We did not formally prove progress of the model, but this would also be an important aspect for a model checking tool. We believe that our approach can be extended to handle this aspect provided

stronger assumptions on the underlying communication system, this is currently work in progress. The entire model is composed of 825 lines of SMT-lib code (including comments), and the verification by Z3 took 14 seconds on a Dell optiplex 990 with a 3.4 GHz Intel Core i7 processor. According to the statistic outputs by Z3 109MB of memory was consumed and 965k equations were added by the constraint solver in the process.

5 Related work

There is a rich field of research on verification of hybrid systems, see Alur [2] for a nice historic overview. Thanks to the foundational research on basic theories for hybrid automata and satisfiability [8, 12], there are now a number of very powerful verification tools available. Our focus is on the application of such automatic formal verification tools on distributed coordination problems. Several works such as [14, 24] use SMT solvers to verify real-time communication protocols but do not consider mobility and spatial safety constraints. The problem of how autonomous traffic agents (or robots) should avoid collisions has also been treated formally with manual proof strategies. For example, Damm et al. [7] present a proof rule for collision freedom of two vehicles. Such work is crucial for the understanding of the basic characteristics of the coordination problem, but can be difficult to directly translate in to a model which is machine verifiable.

Our approach to traffic management is based on a coordination scheme where a physical resource is allocated using a distributed coordination protocol. However, the collision avoidance can also be assured with the help of other abstractions. If a central authority can be deployed as in the case of the European Train Control System (ETCS), it is enough to verify that the agent does not go outside the boundary given by the manager [13, 29]. Collision avoidance between two entities has also been studied in the context of air traffic management [26, 15].

Another approach to ensuring collision freedom is to verify that the trajectories of the different entities do not intersect. Clearly, such an approach requires very sophisticated reasoning about the differential equations relating the vehicle movements. Althoff et al. [1] use reachability analysis to prove safety of evasive manoeuvres. Strong results can be shown with deductive methods as shown by Platzer [19]. This approach has been applied to platooning [16], air traffic management [20], and intersection collision avoidance [17]. While this method allows more powerful model of the vehicle dynamics than what was possible to verify in our model, verifying properties with a deductive approach often require manual interaction. For example, the safety of the intersection control application [17] required in total over 800 interactive steps to complete. Moreover, this study assumes the existence of a stop light, and does not explicitly model communication.

Autonomous intersection management has been extensively explored in the intelligent transportation community [10, 22, 28], though usually not with a focus on proving correctness. Naumann et al. [18] consider a formal model of the scenario, but it is based on a discrete set of locations for each car. The Comhordú

coordination scheme on which the coordination approach presented here is based was formalised by Bhandal et al. [4] using a process algebraic approach.

6 Conclusions

In this paper we have presented a formalisation of the distributed coordination problem encountered by intelligent vehicles while contending for the same physical resource. We formalised a coordination protocol and an intersection collision avoidance case study in the SMT-lib language and proved system safety using the Z3 theorem prover.

We can draw two conclusions from this work. First, the responsibility approach to distributed coordination is a suitable abstraction for formal reasoning on system safety. The core of this approach is that every entity is responsible for making sure that it does not enter an unsafe state with respect to any other entity. This can be contrasted with the other approaches where consensus is required between all nodes, decisions are made by a central manager, or where each pair of nodes negotiates independently, all of which seem problematic from a scalability point of view.

The second conclusion is that automatic verification of collaborative vehicular applications with the help of SMT solvers is at least plausible. We have encountered some cases where the model could not be verified, and increasing the detail and scale of the model would certainly enlarge this problem. However, there are certainly domain-specific approximations that can be made to alleviate some of these problems. Our next step is to generalise our specific case study to construct a tool that allows high level models of applications for smart vehicles to be automatically verified using an underlying formal reasoning engine. This includes dealing with more general physical environment models (e.g., multiple intersections). Another interesting direction is to explore more detailed formal models of the membership protocol.

7 Acknowledgement

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

1. M. Althoff, D. Althoff, D. Wollherr, and M. Buss. Safety verification of autonomous vehicles for coordinated evasive maneuvers. In *IEEE Intelligent Vehicles Symposium, IV*, 2010. doi: 10.1109/IVS.2010.5548121.
2. R. Alur. Formal verification of hybrid systems. In *Proceedings of the ninth ACM international conference on Embedded software*, EMSOFT. ACM, 2011. doi: 10.1145/2038642.2038685.
3. C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2010.
4. C. Bhandal, M. Bouroche, and A. Hughes. A process algebraic description of a temporal wireless network protocol. In *Proceedings of the Fourth International Workshop on Formal Methods for Interactive Systems*, 2011.

5. M. Bourroche. *Real-Time Coordination of Mobile Autonomous Entities*. PhD thesis, Dept. of Computer Science, Trinity College Dublin, 2007.
6. T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *fifteenth annual ACM symposium on Principles of distributed computing (PODC)*. ACM Press, 1996. doi: 10.1145/248052.248120.
7. W. Damm, H. Hungar, and E.-R. Olderog. Verification of cooperating traffic agents. *International Journal of Control*, 79(5), 2006. doi: 10.1080/00207170600587531.
8. L. De Moura and N. Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54, 2011. doi: <http://doi.acm.org/10.1145/1995376.1995394>.
9. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer Berlin / Heidelberg, 2008. doi: 10.1007/978-3-540-78800-3_24.
10. K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *J. Artif. Int. Res.*, 31(1):591–656, 2008.
11. European Commission. Eu energy and transport in figures, 2010. http://ec.europa.eu/energy/publications/statistics/statistics_en.htm, accessed Jan 2012.
12. T. Henzinger. The theory of hybrid automata. In *Logic in Computer Science, 1996. LICS '96. Proceedings., Eleventh Annual IEEE Symposium on*, 1996. doi: 10.1109/LICS.1996.561342.
13. C. Herde, A. Eggers, M. Franzle, and T. Teige. Analysis of hybrid systems using hysat. In *Third International Conference on Systems (ICONS)*, 2008. doi: 10.1109/ICONS.2008.17.
14. J. Huang, J. Blech, A. Raabe, C. Buckl, and A. Knoll. Static scheduling of a time-triggered network-on-chip based on SMT solving. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 509–514, 2012.
15. C. Livadas, J. Lygeros, and N. Lynch. High-level modeling and analysis of the traffic alert and collision avoidance system (tcas). *Proceedings of the IEEE*, 88(7), 2000. doi: 10.1109/5.871302.
16. S. Loos, A. Platzer, and L. Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In M. Butler and W. Schulte, editors, *FM 2011: Formal Methods*, volume 6664 of *Lecture Notes in Computer Science*, pages 42–56. Springer Berlin / Heidelberg, 2011. doi: 10.1007/978-3-642-21437-0_6.
17. S. M. Loos and A. Platzer. Safe intersections: At the crossing of hybrid systems and verification. In *14th International IEEE Conference on Intelligent Transportation Systems, ITSC*, 2011. doi: 10.1109/ITSC.2011.6083138.
18. R. Naumann, R. Rasche, J. Tacke, and C. Tahedi. Validation and simulation of a decentralized intersection collision avoidance algorithm. In *IEEE Conference on Intelligent Transportation System, ITSC*, 1997. doi: 10.1109/ITSC.1997.660579.
19. A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2), 2008. doi: 10.1007/s10817-008-9103-8.
20. A. Platzer and E. M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In A. Cavalcanti and D. Dams, editors, *Proceedings of the 16th International Symposium on Formal Methods, FM*. Springer, 2009. doi: 10.1007/978-3-642-05089-3_35.
21. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In W. Hunt and S. Johnson, editors, *Formal Methods in Computer-Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 127–144. Springer Berlin / Heidelberg, 2000. doi: 10.1007/3-540-40922-X_8.
22. M. L. Sin, M. Bourroche, and V. Cahill. Scheduling of dynamic participants in real-time distributed systems. In *30th IEEE Symposium on Reliable Distributed Systems, SRDS*, 2011. doi: 10.1109/SRDS.2011.37.
23. M. Slot and V. Cahill. A reliable membership service for vehicular safety applications. In *IEEE Intelligent Vehicles Symposium, IV*, 2011. doi: 10.1109/IVS.2011.5940487.
24. W. Steiner and B. Dutertre. SMT-based formal verification of a TTEthernet synchronization function. In S. Kowalewski and M. Roveri, editors, *Formal Methods for Industrial Critical Systems*, volume 6371 of *Lecture Notes in Computer Science*, pages 148–163. Springer Berlin / Heidelberg, 2010. doi: 10.1007/978-3-642-15898-8_10.
25. The World Bank. Road safety. <http://www.worldbank.org/transport/roads/safety.htm>, 2011. Accessed December 2011.
26. C. Tomlin, G. Pappas, and S. Sastry. Conflict resolution for air traffic management: a study in multiagent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4), 1998. doi: 10.1109/9.664154.
27. Traffic Accident Causation in Europe (TRACE) FP6-2004-IST-4. Deliverable 1.3 road users and accident causation., 2009.
28. R. Verma and D. Vecchio. Semiautonomous multivehicle safety. *Robotics Automation Magazine, IEEE*, 18(3), 2011. doi: 10.1109/MRA.2011.942114.
29. A. Zimmermann and G. Hommel. A train control system case study in model-based real time system design. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003. doi: 10.1109/IPDPS.2003.1213234.