Collision Avoidance in Intelligent Transport Systems: towards an Application of Control Theory

Béatrice Bérard†, Serge Haddad∗, Lom Messan Hillah‡, Fabrice Kordon‡ and Yann Thierry-Mieg‡

***Abstract*— Safety is a prevalent issue in Intelligent Transport Systems (ITS). To ensure such a vital requirement, methodologies must offer support for the careful design and analysis of such systems. Indeed these steps must cope with temporal and spatial constraints associated with mobility rules and parallelism which induce a high complexity. Here we handle the problem of unexpected and uncontrollable vehicles which significantly endanger the traffic. In this context, we propose to apply discrete control theory to a model of automatic motorway in order to synthesize a controller that handles collision avoidance. This approach includes two parts: the design of a formal model and an efficient implementation based on hierarchical decision diagrams.**

I.

The Intelligent Transport Systems (ITS) community tries to deal with the numerous challenges that arise when designing secure and reliable software dedicated to automatic transport systems.

Several recent ITS projects aim at providing assistance to drivers and deal with partially automated motorways. The community investigated first a fully automated infrastructure and vehicles approach (as in the PATH [1] project) in the 1990's. That approach was then abandoned in favour of a new line of research and development activities, more centered on safety strategies to ensure properties such as Collision Avoidance or Safety Margin for Assistance Vehicles [2].

This vision relies on cooperative systems where *"road operators, infrastructure, vehicles, their drivers and other road users will cooperate to deliver the most efficient, safe, secure and comfortable journeys"* [3]. Implementing such a system then follows a peer-to-peer organisation where each vehicle must fully cooperate in a time-constrained and safety-critical environment.

In that context, many projects are dealing with safety-oriented applications based on sensors, communication devices and protocols as well as distributed traffic management systems involving cooperation between the infrastructure and vehicles [4], [5], [6]. Thus, reliability, flexibility in the design as well as safety are primary issues. Such systems are even more complex to analyse than previous distributed systems. Consequently, there is a need for a specific methodology and tools to design and analyse them.

In this paper, we propose an approach for the modelling and analysis of an automatic motorway system that is emblematic of ITS problems. We formally specify a particular collision avoidance case study and show how to check whether a control strategy exists depending on the parameters (speed, safety distances, etc.). We cope with the high complexity of the state space thanks to Set Decision Diagrams (SDD).

The paper is structured the following way. Section II details the automatic motorway system. We introduce in section III a formal model for this case study. Then, section IV shows how we use the control theory to synthetize a controller when possible. SDDs and the corresponding encoding are presented in section V followed by some experimentation. In section VI, we conclude and give some perspectives to this work.

II. T

The complexity of Intelligent Transport Systems lies in some key characteristics. The intrinsic parallelism and dynamics which govern the interactions among the involved entities highlight the highly distributed nature of ITS. The entities that interact are vehicles and infrastructure equipments. Vehicles and the infrastructure are cooperative and their interactions are spatially constrained by their mobility. Typically, wireless communications take place within the kilometer range. Further, they are also temporally constrained.

These characteristics make ITS applications design-critical; thus safety and reliability are strong requirements on the models, especially during the validation stage. The models include control, computation and communication. In terms of control, key objectives are collision avoidance and throughput increase.

Current industrial development methodologies do not encompass sufficiently enough formal approaches for the qualitative analysis of ITS models. Vehicles are dynamic systems whose trajectories evolve according to physical parameters such as speed, acceleration, etc. Their state is a set of values, including their position. For qualitative analysis, the main issue in traffic management is the full comprehension of the decision-making process based on the knowledge of vehicles' states so that efficient algorithms can be designed. It can be carried out either by synthesizing a controller, or by the exhaustive exploration of the system's state space.

The exhaustive exploration of the system's state space allows one to understand its dynamics from the behavioral description of vehicles as presented in [7]. This approach relies on model checking, using state space symbolic representation techniques whereby large state spaces can be handled.

†Université Paris Dauphine, LAMSADE, CNRS UMR 7024
∗Ecole Normale Supérieure Cachan, LSV, CNRS UMR 8643
‡Université Pierre & Marie Curie, LIP6/MoVe, CNRS UMR 7606
contact email: berard@lamsade.dauphine.fr

This work has been partially supported by the ModelPlex European integrated project FP6-IP 034081 (Modeling Solutions for Complex Systems)

It is therefore natural in a control-oriented perspective to seek control strategies solutions for traffic management. To do so, we use control theory [8] to synthesize a controller for traffic management. Our main goal is collision avoidance.

The collision problem is illustrated by the case study depicted in Fig. 1. Vehicles are driving along a motorway section. We assume that all but one are fully cooperative; they are *controlled* and then immediately comply with the directives issued by the infrastructure to handle the traffic. One vehicle is *uncontrolled* (the black one in the figure); it means that its behavior only respects physical constraints (such as it cannot stop immediately and must respect inertial constraints). This feature represents either failures of the vehicle or unpredictable behavior of the driver.

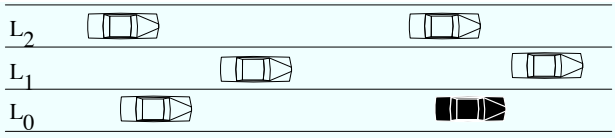

Fig. 1. The automatic motorway system

In this case study, the cooperative vehicles are handled by the *controller* for which we are seeking a collision avoidance strategy, whereas the uncontrolled ones are handled by the *environment*. The strategy must always be a successful one, so that collisions can be avoided whatever the behavior of the uncontrolled vehicle. There is at most one uncontrolled vehicle in the system.

That situation is now described by the following parameters, which will be formally defined in section III.

The motorway has a static and a dynamic part. The static part includes a set $L = \{L_i, 0 \leq i \leq n-1\}$ of lanes with $n \in \mathbf{N}^*$ the number of lanes and $l \in \mathbf{N}$ the length of the lanes. The dynamic part consists of a finite set A of the vehicles currently on the motorway.

Our model will represent discrete positions and moves for vehicles. Although hybrid automata could have been used to model the system, most verification and control problems are known to be undecidable in this framework [9]. Therefore we have chosen the discretization approach.

III. F

We model a section of a motorway, with the aim to avoid a crash between two or more vehicles due to an uncontrollable vehicle. A configuration of the system consists of the set of vehicles present in the section. The problem can be described as follows: given an initial configuration and a set of (bad) target configurations, can we design a controller that avoids reaching such configurations?

A. Elements of the system

*Definition 1 (***Section):** A section is composed of nl positions. A position is defined by its coordinates (x, y), where $y \in \{0, \ldots, n-1\}$ denotes the lane and $x \in \{0, \ldots, l-1\}$ denotes the horizontal position in the lane.

The following constants are associated with the section.

- $vmax$ (resp. $amax$) is the maximum speed (resp. acceleration) reachable by a vehicle. These values depend on hypotheses related to vehicles.
- $vmin$ is the minimum possible speed of a vehicle entering the section. This value depends on hypotheses related to the section.
- $dmin$ is the minimum possible delay between two successive entrances of vehicles in a given lane. This value depends on hypotheses related to the section.

We consider a synchronous behaviour where a change of states occurs every time unit.

*Definition 2 (***Vehicle):** A vehicle present in the section is a tuple $a = \langle a.x, a.y, a.v, a.c \rangle$ where:

- $(a.x, a.y)$ is the position of a;
- $a.v$ is the current speed of a; the speed is an integer denoting the number of positions visited during one time unit. It ranges over $\{0, \ldots, vmax\}$.
- $a.c \in B$ is a boolean denoting whether the vehicle is *controllable*.

*Definition 3 (***Configuration):** A configuration of the section is a tuple $s = \langle d, A \rangle$ where:

- d is an integer array indexed by the lanes such that $d[i]$ denotes the time elapsed since the last entrance on lane i when $d[i] < dmin$ and otherwise $d[i] = dmin$.
- A is a finite set of vehicles.

More formally, we define:

*Definition 4 (***Move of a vehicle):**
Let $a = \langle a.x, a.y, a.v, a.c \rangle$ be a vehicle, its new state is written as $\langle a.x', a.y', a.v', a.c' \rangle$, where:

- $a.x' = a.x + a.v$, *i.e.* the move is fixed by the current speed. When $a.x' \geq l$ then the vehicle is deleted.
- If $a.v > 0$ then $a.y' \in [a.y-1, a.y+1] \cap [0, n-1]$, otherwise $a.y' = a.y$. When its speed is not null a vehicle can non deterministically change lane.
- $a.v' \in [a.v - amax, a.v + amax] \cap [0, vmax]$. A vehicle can non deterministically change its speed according to the capacity of acceleration and speed while keeping a non negative speed.
- $a.c' = a.c$. The move of a vehicle does not change its status w.r.t. controllability.

B. Dimensioning the system

Let us now state some values to dimension the system. We consider a two-lane motorway (*i.e.* $n = 2$). The motorway section we study is 1 kilometer long and structured into 10 meters-long positions, thus $l = 100$. Taking safety distance into account, there are at most 25 vehicles on each lane.

A change of state as described in section III-C corresponds to one second, which is reasonable when considering human reaction in such situations. In a first approach, we choose rather large discretization steps: the maximal speed is $40m/s$ (this corresponds to $144km/h$), with integer speed values from 0 to 4.

Values for acceleration are -1, 0, 1. Thus, it allows a speed variation of $-10m/s$, $+10m/s$ or a stable speed at each round of the system's execution.

C. Evolution of the system

The system evolves from state to state. A change from state $s = \langle d, A\rangle$ to a state $s' = \langle d', A'\rangle$ consists of two successive half-moves:

- The first one is triggered by the environment and consists in (possibly) marking a vehicle as uncontrollable when there is none, inserting new vehicles and moving the uncontrollable vehicle if it exists.
- The second one is triggered by the controller and consists in moving the controlled vehicles.

Thus, the evolution scheme is described by a sequence $s \to s_1 \to s'$ of transitions which are more precisely defined below.

*Definition 5 (***Environment transition):** Let $s = \langle d, A\rangle$ be a configuration, then a new state $s_1 = \langle d_1, A_1\rangle$ can be reached by a transition from the environment by the following operations.

1) If no vehicle in A is uncontrollable then the environment may select $a \in A$ and performs $a.c = false$.
2) For every lane i, the environment performs $d_1[i] = \min(d[i] + 1, dmin)$. If $d_1[i] = dmin$ then it may insert a new vehicle a on lane i: $a.x = 0$, $a.y = i$, $a.v \in \{vmin, \ldots, vmax\}$, $a.c = true$ and reset the delay $d_1[i] = 0$.
3) If there is an uncontrollable vehicle a, its move is performed according to definition 4.

Observation. With no extra cost, we could add requirements on the positions at which a vehicle becomes uncontrollable. For readability sake, we stick to a simpler model.

*Definition 6 (***Controller transition):** Let $s_1 = \langle d_1, A_1\rangle$ be a configuration (obtained by a transition from the environment, as explained below), then a new configuration $s' = \langle d', A'\rangle$ can be reached by a transition from the controller by moving every controllable vehicle of A_1 according to definition 4.

*Definition 7 (***Complete transition):** Let $s = \langle d, A\rangle$ be a configuration, then a new configuration $s' = \langle d', A'\rangle$ can be reached by a *complete* transition if there exists an intermediate configuration $s_1 = \langle d_1, A_1\rangle$ such that there is a environment transition from s to s_1 and a controller transition from s_1 to s'. Furthermore the following consistency rules must hold. For every $a_1 \neq a_2 \in A \cap A'$,

1) $a_1.x' \neq a_2.x' \vee a_1.y' \neq a_2.y'$. Two vehicles do not share the same position.
2) If $a_1.x < a_2.x$ then either $a_1.x' < a_2.x'$ or ($a_2.x' \leq a_1.x'$ and $a_1.y = a_1.y'$ and $a_2.y = a_2.y'$ and $a_1.y \neq a_2.y$). If a_1 is behind a_2, then either a_1 stays behind a_2 or a_1 passes a_2 but then the two vehicles must stay on different lanes.
3) If $a_1.x = a_2.x$ then $a_1.y = a_1.y'$ and $a_2.y = a_2.y'$. If two vehicles are at the same horizontal position (but on different lanes as specified in condition 1), then they stay on their respective lane.

These consistency rules prevent risky situations that must be avoided. Examples of such situations are illustrated in figures 2 and 3. White vehicles correspond to the current state, grey vehicles correspond to the state at next timeframe. So, definition 7 ensures that any controlled move remains safe.

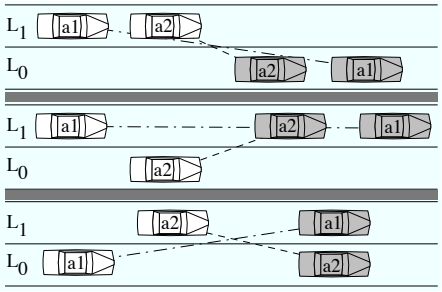

Fig. 2. Some risky situations avoided by rule 2 in definition 7

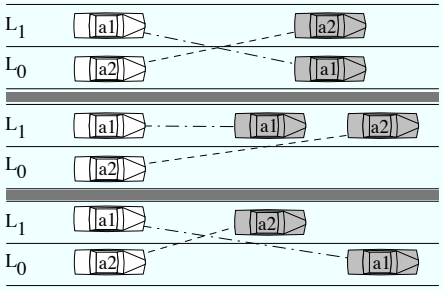

Fig. 3. Some risky situations avoided by rule 3 in definition 7

IV. C T A

First introduced by Ramadge and Wonham [8] at the end of the eighties, the problem of control was described for discrete event systems as follows: given a process P (the plant) and a set of admissible behaviours *Spec* (the specification), does there exist a controller (also called a supervisor) C such that all behaviours of the supervised system, denoted by $P||C$, are in *Spec* ? The synthesis problem is to construct such a controller if it exists.

The basic process P is an open system interacting with an environment. Thus, the alphabet of events is the disjoint union of two subsets containing the uncontrollable events (from the environment) and the controllable events (which can be handled by the controller). Moreover, the controller must react to any uncontrollable event.

Of course, many works have subsequently extended this theory, for instance replacing the discrete event system *Spec* by a formula from some logics and relating control theory to game theory [10]: the two players are the environment and the controller and the synthesis problem is solved by finding a winning strategy. Instead of looking for controllers with minimal state size (although not empty), which is proved NP-hard, other approaches try to find less restrictive controllers [11].

In our context, the specification is the set of behaviours that do not lead to a crash and we reformulate the problem in terms of reachable states.

Given a configuration s, we denote by $Succ_e(s)$ the set of successor configurations of s by an environment transition and $Succ_c(s)$ the set of successor configurations of s by a controller transition.

Let S be the set of all possible configurations. Then $S_{fail} = \{s \mid Succ_c(s) = \emptyset\}$ denotes the set of configurations for which the controller cannot avoid a crash between vehicles. We also set $S^* = S \setminus S_{fail}$.

We denote by s_0 the initial configuration, which consists of an empty section, ready to "receive" vehicles on its lanes.

Definition A strategy is a mapping f from S^* to S such that $f(s) \in Succ_c(s)$. For a strategy f, the reachability space $G_f(s_0) = G_f^c(s_0) \cup G_f^e(s_0)$ is defined inductively by:

- $s_0 \in G_f^c(s_0)$
- If $s \in G_f^c(s_0)$ then $\forall s' \in Succ_e(s), s' \in G_f^e(s_0)$
- If $s \in G_f^e(s_0) \cap S^*$ then $f(s) \in G_f^c(s_0)$

Definition A strategy f is *winning* if $G_f^e(s_0) \cap S_{fail} = \emptyset$.

We informally describe the standard algorithm to check whether a winning strategy exists.

1) We handle two sets of "bad" configurations $Sbad_e$ and $Sbad_c$. Initially $Sbad_e = S_{fail}$ and $Sbad_c = \emptyset$.
2) We extend these sets by the following rules.
   a Let s be a configuration such that $\forall s' \in Succ_c(s), s' \in Sbad_e$. Then $Sbad_c = Sbad_c \cup \{s\}$.
   b Let s be a configuration such that $\exists s' \in Succ_e(s), s' \in Sbad_c$. Then $Sbad_e = Sbad_e \cup \{s\}$.
3) We stop the algorithm either when $s_0 \in Sbad_c$ or when the rules are no more applicable. In the former case, a winning strategy does not exist while in the latter one it does.

Of course, while the algorithm is polynomial in the size of the system, the state space of the system is very large, so that we do not want to directly implement the algorithm. Instead, we now present a particular data structure based on SDD.

V. D D E

In this section we present how to explore the behaviors of the system using a symbolic representation based on decision diagrams. Decision diagrams (DD) are a compact data structure designed to represent large data sets. They allow model-checking of very complex systems [12]. They are based on a *shared decision tree* where each path represents a state of the system. Since the number of paths in a DD may be exponential w.r.t. the number of nodes used to represent them, the reduction factor associated with this technique is generally high. The effectiveness of DD is due to their canonical form, i.e. a set of values has a single DD representation, which is stored in a hashtable. Thanks to this characteristic, deciding if two sets stored as DD are equivalent is a constant complexity operation. Furthermore, using an operation cache, usual set-theoretic operations are computed with complexity at most quadratic to the number of nodes. Application specific operations (like a transition relation) also use a cache to have polynomial complexity w.r.t. the number of nodes.

Many variants of DD have been proposed in the literature. We use Set Decision diagrams (SDD) [13] which offer a hierarchical representation that increases sharing of similar subcomponents, and a flexible operation framework. This section describes SDD [13], then shows how to apply SDD to check controllability of the ITS.

A. Set Decision Diagrams

Set Decision Diagrams (SDD) are data structures for representing sequences of assignments of the form $\omega_1 \in s_1; \omega_2 \in s_2; \cdots \omega_n \in s_n$ where ω_i are variables and s_i are sets of values.

We assume no variable ordering, and the same variable can occur several times in an assignment sequence. We define the usual terminal 1 to represent accepting sequences. The terminal 0 is also introduced and represents the empty set of assignment sequences. In the following, Var denotes a set of variables, and for any ω in Var, $\mathrm{Dom}(\omega)$ represents the domain of ω *which may be infinite*.

Set Decision Diagram The set $\mathbf{S}$ of SDD is inductively defined by $\delta \in \mathbf{S}$ if:

- $\delta \in \{0, 1\}$ or
- $\delta = (\omega, \pi, \alpha)$ with:
  - $\omega \in Var$.
  - $\pi = s_0 \uplus \cdots \uplus s_n$ is a finite partition of $\mathrm{Dom}(\omega)$, i.e. $\forall i \neq j, s_i \cap s_j = \emptyset, s_i \neq \emptyset, n$ finite.
  - $\alpha : \pi \to \mathbf{S}$, such that $\forall i \neq j, \alpha(s_i) \neq \alpha(s_j)$.

We denote by $\omega \xrightarrow{s} \delta$, the SDD (ω, π, α) with $\pi = s \uplus (\mathrm{Dom}(\omega) \setminus s)$ $\alpha(s) = \delta, \alpha(\mathrm{Dom}(\omega) \setminus s) = 0$. By convention, when it exists, the element of the partition π that maps to the SDD 0 is not represented.

Despite its apparent simplicity, this definition supports complex data structures. For instance, variables of domain $\mathbf{S}$ can be defined, introducing hierarchy in the data structure.

SDD support standard set theoretic operations (union, intersection, set difference) for compatible SDD as well as a powerful and flexible mechanism to define user operations called inductive homomorphisms.

A small example showing the hierarchical representation is presented in the next section, in which we encode states of the ITS system using SDD.

B. Encoding the motorway problem

1) State representation: A state of the motorway is composed of the value for the entry delays for each lane, and of the positions, speed and control value of the vehicles.

We thus introduce a variable *delays* such that $\mathrm{Dom}(delays) = [0 \ldots dmin]^n$, i.e. a vector with number of lanes n entries which vary between 0 and $dmin$.

We also introduce a variable *vehicle* such that $\mathrm{Dom}(vehicle) = \{\langle x, y, v, c\rangle | x \in [0 \ldots l-1], y \in [0 \ldots n-1], v \in [0 \ldots vmax], c \in \mathbf{B}\}$ representing the state of a vehicle. The state of the set of vehicles is encoded as a sequence of *vehicle* variables, concatenated to the *delays* variable.

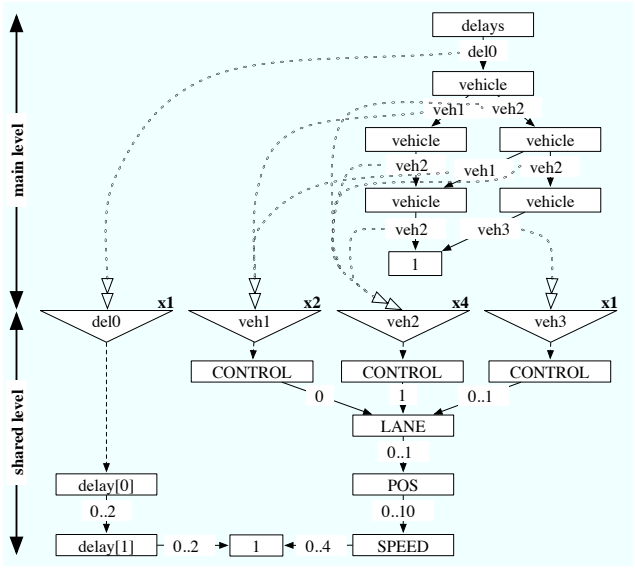

Fig. 4. SDD hierarchical encoding of the automatic motorway with the following values: $dmin = 2$, $vmax = 4$, $l = 11$, $n = 2$.

Figure 4 presents the SDD encoding of the ITS. We provide more detail later. But roughly speaking, the part above (main level) shows a set of states of the motorway, and the one below (shared level) represents the values which label the arcs of the first structure. As one can see, sharing is high as vehicle states (such as *veh*2) are reused several times (four times with our values for *veh*2).

2) State encoding and vehicle positions:

- **Ordering**: to obtain a more canonical form, since all vehicles are considered equivalent, there is no need to preserve identity or insertion order of vehicles. The state signature is thus sorted according to vehicle position: $a < b \equiv a.x < b.x \vee (a.x = b.x \wedge a.y < b.y)$. Since two vehicles cannot occupy the same position, this defines a total order over vehicles, allowing to define a canonical representation. A second positive aspect of this ordering is that it improves locality of actions, as conflicts between vehicle movements can be evaluated mostly by examining the states of the nearest vehicles.
- **Position**: encoding position information is a tricky issue, since naive encoding schemes are liable to produce exponential blowup in SDD representation size. The basic encoding scheme consists in giving for each vehicle its position w.r.t. the origin. But this introduces a strong dependency amongst the variables representing vehicle states, namely the successor b of a vehicle a will always verify $b.x \geq a.x$. If we consider the set encoding of all potentially reachable states S, we obtain a different successor node from the node encoding a for each value of $a.x$. We can thus roughly estimate at $l^{|A|-1}$ nodes ($|A|$ the number of vehicles) the size of the SDD needed to represent S, which is not manageable.
  A first idea to fight this effect consists in encoding for each vehicle a its position Δx *relative to the preceding vehicle*, or relative to the origin for the first vehicle instead of its absolute position $a.x$. This drastically improves the locality of dependencies. However we must be careful not to reintroduce a global variable dependency when we introduce the cutoff horizon. Indeed, the firing rule (definition 4, first item) requires to delete vehicles that have left the section, that is $a.x \geq l$. But if we strictly apply this rule, we obtain a new global dependency stating that the sum of Δx positions along any path (state) is at most $l-1$. This dependency is as costly to represent as in the first case (absolute positions).
  We therefore decided to represent a maximum number of vehicles rather than a total maximum distance cutoff criterion, with a new horizon cutoff criterion ensuring that any state with distance l or more between any two vehicles is not represented. This means we actually monitor a variable distance depending on the state considered. In any case the last vehicle ensures we monitor at least l positions, so overall we monitor at least $|A| + l$ positions and potentially $|A| \cdot l$.

The next step consists in defining the initial states of the controllability problem. We construct this set S_{fail} in the following manner :

1) We first construct a set of states in which all variables are unconstrained. This means lane delay variables vary within $0 \ldots dmin$, and for each vehicle $a = \langle x, y, v, c \rangle$ variable we allow $x \in \{0, \ldots, l-1\}, y \in \{0, \ldots, n-1\}, v \in \{0, \ldots, vmax\}$, and $c \in \mathbf{B}$.
   This produces a linear size SDD encoding a superset of all potentially reachable states.
2) We then prune from the structure paths such that more than one vehicle is uncontrolled, as by definition of the problem, there can be only one uncontrolled vehicle. This introduces a weak dependency on the variables, thus does not significantly increase representation size. Figure 4 presents the encoding obtained at this stage of the construction. At the top of the figure, SDD variables are represented: the variable "delays" contains the value of the delay variables for each lane, while each "vehicle" variable represents a vehicle state. The labels on the arcs of the structure are references to the SDD represented at the bottom of the figure. For instance, in the path $d \xrightarrow{del_0} v \xrightarrow{veh1} v \xrightarrow{veh2} v \xrightarrow{veh2} 1$ the first vehicle is uncontrolled (see *veh*1 at bottom), thus the two other vehicles must be controlled (they both are in states *veh*2 that fulfill this property).
3) We then remove states in which controlled vehicles occupy the same position, as the controller transition rules prevent this from happening. However, the uncontrolled vehicle can overlap another, as we are constructing the set of failure states, in which the environment has just finished playing, and it is the controller's turn to try and avoid a collision. This is done by enforcing a constraint stating that for any two successive controlled vehicle a and b, if $b.\Delta x = 0$ then $b.y > a.y$. The dependency introduced is not severe

thanks to its strong locality, and does not significantly increase the size of the SDD encoding.

This produces the set of all potentially reachable states that we will use to initiate our construction.

We need to select the states S_{fail} where the controller cannot make a move. To this end we use the fact that changing lanes can only increase the number of interference possibilities between vehicles, as by the consistency rules defined, it is not possible in a single step to both change lanes and overtake a vehicle.

We thus only consider moves in which the controller keeps each vehicle in its lane. We thus obtain a state based characterization of failure states: in any state for which the distance between a vehicle and the next vehicle on its lane is smaller than the difference in their speeds, a collision is unavoidable at this step. This constraint introduces more dependencies among variables, as it requires we control at most the n next vehicles (n number of lanes), to reach the next vehicle on the current lane.

We also need to control the sum of Δx positions of these n vehicles to compute the distance with the current vehicle a. When computing this sum, if it exceeds $a.v$ the test is considered successful, as $b.v \geq 0$. Consequently, the size of the encoding becomes sensitive to the number of lanes as well as the $vmax$ setting. However, these parameters typically have reasonably low values ($vmax = 4, n \leq 6$).

C. Assessment of the data structure

We have run some preliminary experiments to confirm that this encoding is viable. In this experience, we have computed the set S_{fail} as presented above. We let vary two parameters: the number of cars (50, 75, 100, 125, 150) and the number of lanes (2, 3, 4). Note that the realistic configuration given in section III-C corresponds to the smallest we processed. Execution was performed on a PC running Linux with: Core2duo, 3.4GHz, 3Gb.

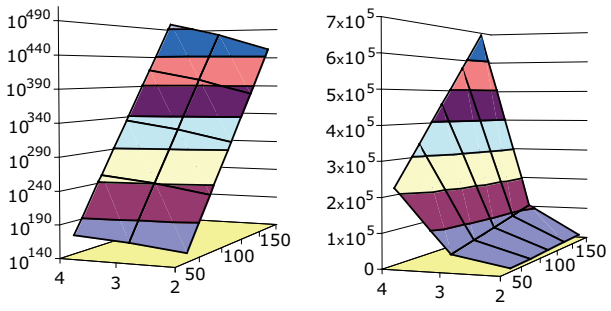

Fig. 5. Size of S_{fail} (left) and its corresponding SDD encoding (right).

Figure 5 shows the evolution of the computed number of states in S_{fail} (left) and of the number of SDD nodes required for its encoding (right). Computation of the largest configuration (150 cars, 4 lanes) took about 25 minutes and did not exhaust main memory. The ratio between the two figures is exponential.

This experiment is not yet complete as we have not defined a controller policy and therefore the backward firing required by control theory has not yet been implemented. However, it shows the feasibility of our approach to seek a winning control strategy with realistic values.

VI.

Our goal is to use control theory to find winning strategies for collision avoidance in an automatic motorway. Such a system is challenging due to the high number of entities and the complexity of their interactions. This dramatically raises the combinatorial explosion issue.

Prior to algorithmic issues, the first problem is to be able to store all possible configurations of the system.

In this paper, we elaborate a formal model to specify the automatic motorway. We also propose an efficient encoding technique to cope with the combinatorial explosion problem.

Experimentation shows that our approach is valid. We can store realistic configurations dealing with up to 150 vehicles dispatched over 4 lanes in a 1km long section.

The next step consists in implementing the search for winning strategies for collision avoidance.

R

[1] R. Horowitz and P. Varaiya, "Control design of an automated highway system," in *Proc. of the IEEE 88(7)*, 2000.

[2] Intelligent Vehicle Initiative, "Saving lives through advanced vehicle safety technology," US Department of Transportation, http://www.its.dot.gov, September 2005.

[3] P. Bly, "e-Safety - Co-operative Systems for Road Transport (IST Work Progamme 2005-2006)," European Commission, Tech. Rep., 2004.

[4] J.-M. Blosseville, "Driving assistance systems and road safety: State-of-the-art and outlook," in *Annals of Telecommunications - Intelligent Transportation Systems*, J. Ehrlich, Ed. GET-Lavoisier, March-April 2005, vol. 60, no. 3-4, pp. 281–298.

[5] R. Bishop, "Intelligent Vehicle R&D: a review and contrast of programs worldwide and emerging trends." in *Annals of Telecommunications - Intelligent Transportation Systems*, J. Ehrlich, Ed. GET-Lavoisier, March-April 2005, vol. 60, no. 3-4, pp. 228–263.

[6] Y. Robin-Jouan, J. Ehrlich, B. Guillaumin, M. Delarche, and M. Dutech, "Transport-specific communication services: Safety-based or critical applications for mobiles and cooperation with infrastructure networks," in *Annals of Telecommunications - Intelligent Transportation Systems*, J. Ehrlich, Ed., vol. 60, no. 3–4. GET-Lavoisier, March-April 2005, pp. 405–440.

[7] F. Bonnefoi, L. Hillah, F. Kordon, and G. Frémont, "An approach to model variations of a scenario: Application to Intelligent Transport Systems," in *Workshop on Modelling of Objects, Components, and Agents (MOCA'06)*, Turku, Finland, June 2006.

[8] P. Ramadge and W. Wonham, "The control of discrete event systems," in *Proceedings of the IEEE*, vol. 77, no. 1. IEEE Computer Society, 1989, pp. 81–98.

[9] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci.*, vol. 138, no. 1, pp. 3–34, 1995.

[10] A. Arnold, A. Vincent, and I. Walukiewicz, "Games for synthesis of controllers with partial observation," *Theor. Comput. Sci.*, vol. 303, no. 1, pp. 7–34, 2003.

[11] R. Su and W. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dynamic Systems*, vol. 14, no. 1, pp. 31–53, 2004.

[12] J. Burch, E. Clarke, and K. McMillan, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation (Special issue for best papers from LICS90)*, vol. 98, no. 2, pp. 153–181, 1992.

[13] J.-M. Couvreur and Y. Thierry-Mieg, "Hierarchical Decision Diagrams to Exploit Model Structure," in *Formal Techniques for Networked and Distributed Systems - FORTE 2005*, vol. 3731. LNCS: Springer Verlag, 2005, pp. 443–457.