To Err is Human: Designing Correct-by-Construction Driver Assistance Systems using Cognitive Modelling

Francisco Girbal Eiras

Linacre College, University of Oxford

Supervised by:

Dr. Morteza Lahijanian and Prof. Marta Kwiatkowska

A thesis submitted for the degree of

Master of Science in Computer Science

Trinity 2018

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Morteza Lahijanian, for his invaluable support and insight. Our meetings were essential to the progress of this dissertation and this project would not have been this complete without his ideas. I would also like to thank my co-supervisor, Professor Marta Kwiatkowska for giving me the opportunity to work on this project.

I would like to thank my academic and college supervisor, Professor Alessandro Abate, for his help and availability throughout the year. Finally, I would like to thank my family in general and in particular my sister, Constança, my dad, António, for always encouraging me to be my best self, and my mom, Ana, for her unconditional support and love.

Abstract

Research into safety in autonomous and semi-autonomous vehicles has, so far, largely been focused on testing and validation through simulation. Due to the fact that failure of these autonomous systems is potentially life-endangering, formal methods arise as a complementary approach.

This thesis studies the application of formal methods to the verification of different profiles of human drivers (built using the cognitive architecture ACT-R), and to the design of correct-by-construction Advanced Driver Assistance Systems (ADAS) using strategy synthesis. The situation considered is a 2-lane highway scenario and the interactions that arise for various profiles of drivers (e.g. follow, crash or overtake another vehicle).

The results show that, in this situation, the synthesised ADAS improve both safety and efficiency of the overall system when compared to the human driver alone.

The dissertation distinguishes itself from previous work done in this area due to the complex nature of the scenario considered and the efficient abstraction techniques introduced which yield compact finite state models. Additionally, it establishes insightful metrics for verification of the human driver model and synthesis of ADAS in driving situations, and paves the way for the establishment of a general framework to tackle similar driving scenarios.

Contents

Chapter 1

Introduction

Human safety is an extremely important aspect to consider when analysing possible transportation solutions. According to a study carried out by the National Highway Traffic Safety Administration in the USA, over 90% of all road accidents are mainly attributed to errors of human drivers, caused by distractions, fatigue, alcoholic influence, among other factors [1]. In an attempt to reduce these numbers, several car manufacturers have worked on solutions which minimise driver intervention through the introduction of autonomous features. Vehicles where these elements are present are ranked as autonomous vehicles of levels 2 or 3[1], and the features are generally described as Advanced Driver Assistance Systems (ADAS). Examples include Tesla's *Autopilot* and Ford's *Co-Pilot 360* [3, 4].

While ADAS are more practical for drivers and it would appear they are beneficial in terms of safety, there are two concerns regarding the way these systems are designed. Firstly, they are built by humans and, as such, the presence of software mistakes in the implementation is almost unavoidable, particularly due to the complexity of the systems in question. An example which supports this concern happened when, on March 23, 2018, a driver of a Tesla Model X died after the vehicle crashed into a highway divider in California with *Autopilot* engaged [5]. The accident was later attributed to a human error (i.e. a bug in the implementation) in *Autopilot* [6]. Secondly, existing ADAS lack the capability of understanding the human cognitive process; hence, they are unable to predict the actions drivers will take and adjust or suggest safe actions accordingly.

[1]Autonomous vehicles are ranked on six differing levels of autonomy, from 0 (not autonomous at all) to 5 (fully autonomous) [2].

In order to tackle these issues, formal verification - the act of proving or disproving the correctness of intended processes underlying a system with respect to a certain specification [7] - is appropriate to guarantee requirements are fulfilled and the systems function correctly, and cognitive modelling - the approximation of the human cognitive processes using appropriate frameworks [8] - is useful for the accurate modelling of the human driver behaviour.

1.1 Related Work

There are several approaches to the study of safety in the context of autonomous and semi-autonomous vehicles. As Nidhi *et al* pointed out in [9], a logical approach would be to test drive these vehicles in real world situations, create evaluation metrics, make observations on their performance, and apply statistical comparisons against the baseline that is the human driver. However, Nidhi *et al* concluded that this would be infeasible, as the data required to make any statistically relevant conclusions on the safety of the vehicles would take tens, if not hundreds, of years to collect. An alternative solution is through the modelling and simulation of the autonomous vehicle, using frameworks which allow testing under different conditions, as proposed in [10, 11, 12, 13]. Several companies developing autonomous cars have turned to this solution as a way of obtaining vast amounts of data on the security and reliability of their vehicles, in order to continuously improve the systems they have designed. Despite this, it is imperative to recognise the shortcomings of simulation in safety evaluation of complex driver assistance systems which could have life-endangering impact [14, 15]. In this context, formal verification arises as a complementary approach to simulation, leading to automation and precision in the testing process.

The techniques used in formal verification range from model checking, the process of verifying through exhaustive search whether a model meets a given specification, to strategy synthesis, which corresponds to obtaining a the set of actions to be taken which is guaranteed to satisfy one or multiple requirements (if they exist) [7, 16]. The strategies obtained through synthesis are, by definition, *correct-by-construction*, in the sense that they are outputted in the process as a result of the fact that they are provably correct up to the level of representation of the model [7, 16]. As such, they do not fall within the implementation mistakes that humans might. The process of strategy synthesis is widely used nowadays in software development [17, 18, 19].

Due to the benefits that formal verification brings to the field, many researchers have used these tools to evaluate control systems in autonomous vehicles, to verify models of human driver behaviour, and study cooperation within fleets of these vehicles [20, 21, 22, 23]. In [24], Nilsson *et al.* took the first step towards correct-by-construction ADAS by synthesising the control software module for adaptive cruise control from formal specifications given in Linear Temporal Logic. In [25], Lam modelled the behaviour of a distracted human driver using a cognitive architecture in a one-way street traffic light scenario (no other vehicles present) and studied how a driver assistance system could improve the safety of vehicle, following the work of [26, 27]. Despite these advances, there are many open questions in this area, particularly whether real data validates the models constructed and, therefore, results obtained; how to deal with more complex environments and interactions without getting into representation problems such as state explosion (with an increase of the complexity of the system to be modelled, the number of variables required rises and the state space exponentially increases); and whether or not it is possible to represent an arbitrarily generic situation in this context.

In the case of safety in the context of semi-autonomous vehicles, the mentioned techniques are constrained by the accurate modelling of the human driver. The first integrated approaches to this problem, proposed in [28, 29, 30], rely on modelling human behaviour through continuous controllers and specifically engineered frameworks. However, these methods lack the human behaviour variability attributed to the discrete nature of the control actions performed by the drivers. With this in mind, in [31] Salvucci *et al* proposed a proof-of-concept of the introduction of human constrains in the driver model through the use of the cognitive architecture Adaptive Control of Thought-Rational (ACT-R) - a framework for specifying computational behavioural models of human cognitive performance, embodying both the abilities (e.g. memory storage and recall or perception) and constraints (e.g. limited motor performance or memory decay) of the human system [26]. In [26], Salvucci proposed an updated version of the human driver model initially introduced in [31], which was improved using the advances in ACT-R.

1.2 Aims and Contributions

Following some of the work presented in the previous section, the aim of this project is to study the application of such techniques to the verification of models of different profiles of human drivers (built using a cognitive architecture) and design correct-by-construction advanced driver assistance systems using strategy synthesis under multi-objective properties.

This thesis extends the work of [20, 25, 26] by considering a 2-lane highway scenario and the interactions that arise for various profiles of drivers (e.g. follow, crash into or overtake another vehicle), with the primary goal of decreasing accident risk through the synthesis of an advanced driver assistance system. It takes the first steps towards the goal of establishing a general framework to tackle the design of ADAS for different driving situations. In the future, the created framework can be used to design and deploy the correct-by-construction cognitive modelling-based ADAS in semi-autonomous vehicles (as a short to medium term goal), and in the long term it could be used as a way to bootstrap fully autonomous solutions (where the human behaviour is simulated in the deployed system) which reduce accidents to a minimum and maximise the efficiency of the vehicles.

The scenario proposed distinguishes itself from previously studied ones due to the high complexity presented and the implementation of the human driver model using a cognitive architecture. Through this, the dissertation introduces novel strategies for accurate abstraction of existing models in order to generate finite state space Discrete Time Markov Chain (DTMC) and Markov Decision Process (MDP) representations. This thesis is also focused on multi-objective synthesis for the advanced driver assistance system, with safety and time efficiency being optimised (e.g. to avoid situations where the system will always slow down instead of overtaking another vehicle), contributing to the practical usefulness of the system for the driver of the vehicle.

1.3 Thesis Outline

The methodology of the dissertation is supported using several definitions and concepts from the literature which are presented in Chapter 2. Following this, Chapter 3 covers the methods applied to the construction of the human driver model, from the abstraction of Salvucci's integrated driver model to the establishment of properties to be verified on the model, as well as a graphical visualisation tool developed. In Chapter 4, the driver model is transformed and augmented with a driver assistance system. Several designs are studied and compared, with the best performing one being the subject of the in-depth analysis presented in Chapter 5. In addition to the performance evaluation of the ADAS, Chapter 5 is also composed of several examples of possible extensions of the scenario through the usage of different properties and assumptions. The thesis is concluded with an evaluation of the work developed and future directions that research in this area could follow.

Chapter 2

Literature Review and Definitions

This chapter presents the concepts and theory required to understand the methodology, rationale and contributions of this dissertation. It starts by defining a Cognitive Architecture, and goes into more detail about Adaptive Control of Thought - Rational, which is the underlying framework used in the Integrated Driver Modelling described in Section 2.2 and the basis for the system developed in Chapter 3. The chapter concludes with the introduction of several important concepts in the area of Probabilistic Model Checking, which are essential for modelling, verification and synthesis in this context.

2.1 Cognitive Architectures

Different definitions of a cognitive architecture can be found in the literature [32]. According to Sun, a cognitive architecture is a *"broadly-scoped, domain-generic computational cognitive model, capturing the essential structure and process of the mind, to be used for a broad, multiple-level, multiple-domain analysis of behavior"* [33]. In that sense, a cognitive architecture is no more than a formalised framework of the perception, memorisation, decision making, reasoning and execution of the human mind. It is important to note that this architecture should capture both the abilities (e.g. memory storage and recall, perception or motor action) and constraints (e.g. memory decay and limited motor performance) of the human system [26].

The idea of a cognitive architecture originated in the early 1970s. A survey by Korseruba and Tsotsos [32] revealed that, since then, an estimated 195 different cognitive architectures with different assumptions, structural organisations and implementations have been developed. Applications of these range from the fields of Robotics to Natural Language Processing [32]. One of the most well known of those architectures is Adaptive Control of Thought - Rational.

2.1.1 Adaptive Control of Thought - Rational (ACT-R)

Adaptive Control of Thought - Rational (or ACT-R as it is commonly known as) is a particular cognitive architecture first presented by John R. Anderson in 1983 [8] and further developed by Anderson *et al.* in [34].

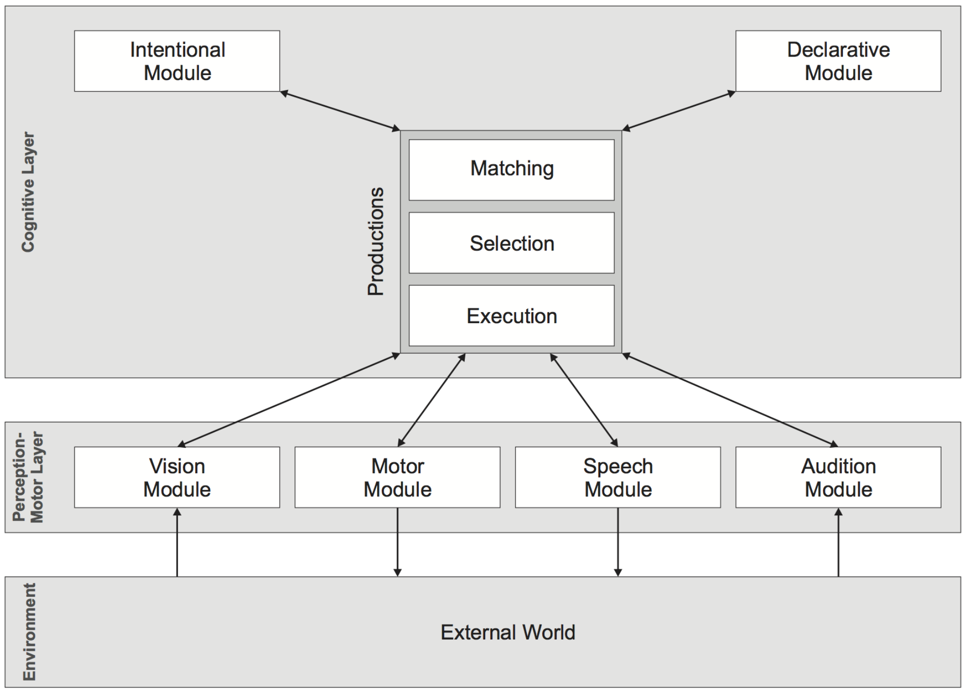


Figure 2.1 Overview of the ACT-R high level architecture (from [35]).

The architecture can be generally described as two distinct layers: a *perceptual-motor* layer and a *cognitive* layer, as presented in Figure 2.1. The perceptual-motor layer corresponds to the interface of the cognition with the environment (which plays a key role in ACT-R), being comprised of modules such as vision and motor actions. The cognitive layer is focused on memory, which can be divided into two different categories: *declarative* (consisting of factual knowledge and goals - e.g. *"The maximum* *driving speed in a typical UK motorway is 70 mph"* or *"Try get to point B"*) and *procedural* (consisting of rules/procedures - e.g. *"If the lead vehicle is going slowly, attempt an overtake"*) [35]. Furthermore, declarative memory is composed of *chunks*, which are smaller pieces of information, divided into categories and attributes, that form more complex memories. In the example *"The maximum driving speed in a typical UK motorway is 70 mph"*, `maximum driving speed in a typical UK motorway` is a category, whereas `70 mph` is an attribute. Another possible attribute for this category could be `112 km/h` (since $70mph = 112km/h$).

2.2 Integrated Driver Modelling in ACT-R

The first integrated approaches to this problem, proposed in [29, 36, 30], rely on modelling human behaviour through continuous controllers and specifically engineered frameworks. However, these methods lack the human behaviour variability attributed to the discrete nature of the control actions performed by the drivers. With this in mind, in [31] Salvucci *et al.* proposed a proof-of-concept of the introduction of human constrains in the driver model through the use of the cognitive architecture ACT-R. In [26], Salvucci proposed an updated version of the human driver model initially introduced in [31], which was improved using the advances in ACT-R (the model uses version 5.0 of the architecture) and re-designed elements (e.g. lane changing behaviour or distraction) resulting from validation performed by empirical studies.

The model Salvucci introduced in [26] was developed with a specific scenario in mind: *"(...) driving a standard midsize vehicle on a multilane highway with moderate traffic"* [26]. It consists of three distinct modules interacting in a sequential way: **control**, **monitoring**, and **decision making**. A high level schematic of the model can be found in Figure 2.2.

The **control** component manages both the lower level perception cues and the physical manipulation of the vehicle (e.g. steering, accelerating or breaking). The module can be divided into lateral (i.e. steering) and longitudinal (i.e. velocity) control, each modelled by a separate control law.

The lateral control is determined by the existence of two artefacts that the driver obtains using lower level perception cues: the *near point* and the *far point.* In this model, *"the near point represents the vehicle's current lane position, used to judge*

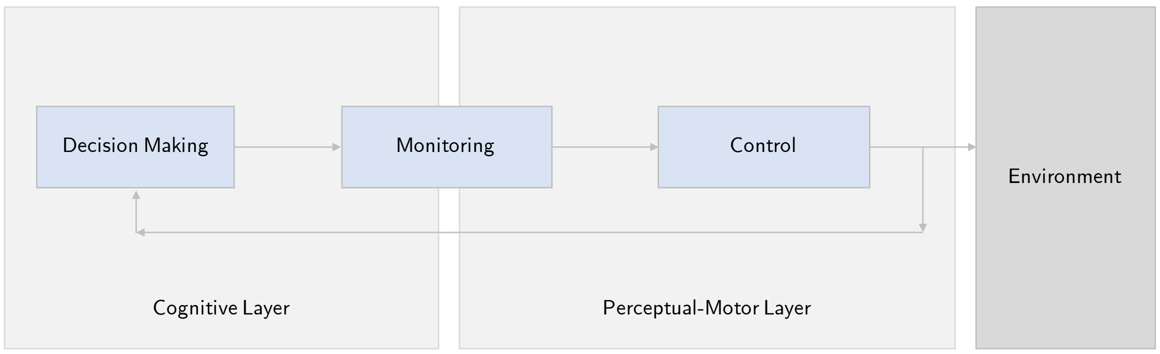


Figure 2.2 High level representation of the architecture presented in [26].

how close the vehicle is to the center of the roadway (...) [and] is characterized as a point in the center of the near lane visible in front of the vehicle, set at a distance of 10 m from the vehicle's center" while *"the far point indicates the curvature of the upcoming roadway, used to judge what the driver should execute to anticipate the upcoming curvature and remain in the current lane* [26]. At any cycle $i > 0$, the model works by using perception to determine the visual angles $\theta^{(i)}_{near}$ and $\theta^{(i)}_{far}$ and calculates:

$$\begin{aligned}\Delta\theta_{near} &= \theta^{(i)}_{near} - \theta^{(i-1)}_{near} \\ \Delta\theta_{far} &= \theta^{(i)}_{far} - \theta^{(i-1)}_{far}\end{aligned} \tag{2.1}$$

The control law for the steering angle φ can be defined as:

$$\Delta\varphi = k_{far}\Delta\theta_{far} + k_{near}\Delta\theta_{near} + k_I \min{(\theta^{(i)}_{near}, \theta_{\max})}\Delta t \tag{2.2}$$

with k_{far}, k_{near}, k_I and $\theta_{\max}$ defined as in [26].

The process for the longitudinal control is very similar. At any cycle $i > 0$, the model starts by encoding the position of the lead vehicle and calculating the time headway $thw^{(i)}_{car}$ to it. It then computes:

$$\Delta thw_{car} = thw^{(i)}_{car} - thw^{(i-1)}_{car} \tag{2.3}$$

The control law for the linear acceleration ψ can be written as:

$$\Delta\psi = k_{car}\Delta thw_{car} + k_{follow}(thw_{car} - thw_{follow})\Delta t \tag{2.4}$$

with k_{car}, k_{follow} and thw_{follow} defined as in [26].

These control laws can be applied in order to model the behaviour of the driver in terms of lane keeping and curve negotiation, and generalise to lane changing actions in a straightforward manner. In order to initiate a lane change, the driver just starts following the near and far points of the destination lane instead of the current lane, as Salvucci and Liu presented in [30].

The **monitoring** component maintains situational awareness through the awareness of the position of other vehicles around the driver's vehicle. It accomplishes this through a random-sampling, with probability $p_{monitor}$, of one of four areas: either the left or right lane, and either forward or backward (with equal probability). After deciding which area to sample (if any), the model uses visual perception to determine whether a vehicle is present or not. If it is, the distance, lane and direction are saved in ACT-R's declarative memory. The value of $p_{monitor}$ is defined in [26].

Finally, the **decision making** module uses the information gathered in the monitoring and control stage (if any is available) to determine which tactical decision should be taken. In this model, the decision corresponds to determining when and where a lane change should occur. Salvucci describes this process in [26] as: *"If the driver's vehicle is in the right lane, the model checks the current time headway to the lead vehicle (if any); if the lead car time headway drops below a desired time headway thw_{pass}, the model decides to change lanes to pass the vehicle. If the driver vehicle is in the left lane, the model checks instead simply for the presence of a lead vehicle. If there is a lead vehicle, the model remains in the left lane (because this vehicle is also passing other vehicles); otherwise it decides to [try to] change lanes to return to the right lane"*. This process is summarised in the flowchart presented in Figure 2.3, and the value of thw_{pass} is defined in [26].

Even after this decision to change lanes is taken, the model must then determine whether it is actually safe to do so. In order to verify that it is, the model initially attempts to recall, from the declarative memory, the nearest vehicle to it in the destination lane. If one is recalled at a distance closer than d_{safe}, then the change is aborted. If not, then the vehicle performs a safety monitoring of the destination lane.

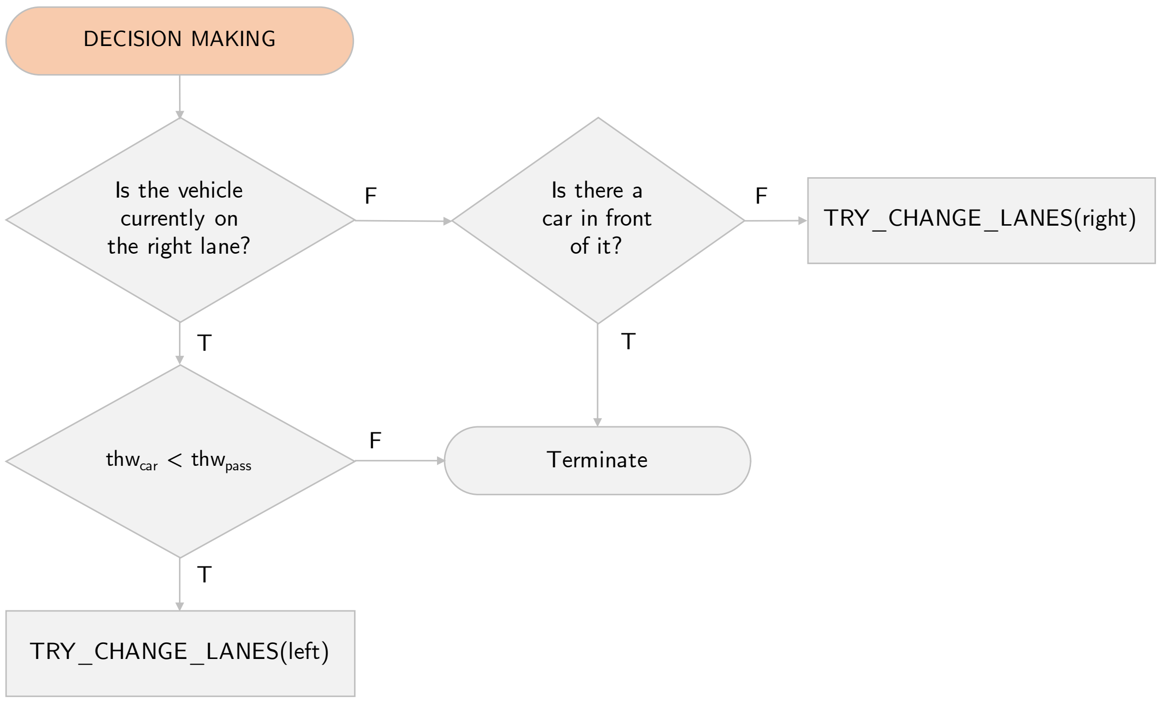

DECISION MAKING
Is the vehicle currently on the right lane?
F
Is there a car in front of it?
F
TRY_CHANGE_LANES(right)
T
T
thw$_{car}$ < thw$_{pass}$
F
Terminate
T
TRY_CHANGE_LANES(left)

Figure 2.3 Flowchart of part of the decision making process of the model.

If this monitoring does not observe a vehicle at a distance closer than d_{safe}, then the vehicle initiates the execution of the lane change. The parameter d_{safe} is as defined in [26].

2.3 Probabilistic Model Checking

Model checking is a formal verification technique consisting of the process of automatically verifying, through exhaustive search, whether a model meets a given specification [7]. A model in this context is considered to be a labelled finite state-transition system, with states corresponding to configurations of the system and transitions between states representing the evolution between the said configurations.

Probabilistic model checking is a generalisation of model checking for the automated verification of systems that exhibit probabilistic behaviour [37, 7]. It should be noted that this is not to be confused with *probabilistic verification*, which amounts to partial state-space exploration [7]. Within this section, important concepts of probabilistic model checking related to modelling of systems, and verification and synthesis of models will be explained in detail.

2.3.1 Markov Chains and Decision Processes

Probabilistic model checking begins with the appropriate modelling of the system in a stochastic finite state-transition representation. There are several ways to do this according to the type of system to be modelled.

In *discrete-time Markov Chains* (DTMCs), all choices (i.e. transitions) are probabilistic [7]. While this is useful in some cases, others require nondeterminism, which is where *Markov Decision Processes* (MDPs) come in. In MDPs, nondeterministic probabilistic choices coexist. Essentially, a DTMC is an MDP with a uniquely determined probabilistic distribution [7]. Both DTMCs and MDPs can be augmented with reward structures, which correspond to real positive values assigned to states, transitions or actions that can be interpreted as bonuses or costs [7].

The formal definitions of DTMCs, MDPs and reward structures follow (adapted from [7, 37, 38]).

Definition 1. Discrete-time Markov Chain (DTMC)

A *discrete-time Markov chain* is a tuple $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ such that:

- S is a countable, non-empty set of states
- $\mathbf{P} : S \times S \to [0, 1]$ is a transition probability function such that for all states $s \in S$:
$$\sum_{s' \in S} \mathbf{P}(s, s') = 1 \tag{2.5}$$
- $p_{init} : S \to [0, 1]$ is the initial state distribution, such that:
$$\sum_{s \in S} p_{init}(s) = 1 \tag{2.6}$$
- AP is a set of atomic propositions, and
- $L : S \to 2^{AP}$ is a labelling function.

$\mathcal{M}$ is denoted *finite* if S and AP are finite sets. ■

Definition 2. Markov Decision Process (MDP)

A *Markov decision process* is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ such that:

- S is a countable, non-empty set of states
- Act is a set of actions
- $\mathbf{P} : S \times Act \times S \to [0,1]$ is a transition probability function such that for all states $s \in S$ and actions $\alpha \in Act$:

$$\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\} \tag{2.7}$$

- $p_{init} : S \to [0,1]$ is the initial state distribution, such that:

$$\sum_{s \in S} p_{init}(s) = 1 \tag{2.8}$$

- AP is a set of atomic propositions, and
- $L : S \to 2^{AP}$ is a labelling function.

An action $\alpha \in Act$ is enabled in a state s if, and only if, $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$. Let $Act(s)$ denote the set of actions enabled in a state s. It must be that for all states $s \in S$, $Act(s) \neq \emptyset$. ■

Definition 3. Reward Structures for DTMCs and MDPs

A *reward structure* for a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ is a tuple $\mathcal{C} = (\rho, \iota)$ such that:

- $\rho : S \to \mathbb{R}_{\geq 0}$ is a state reward function
- $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a transition reward function

A *reward structure* for a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ is a tuple $\mathcal{C} = (\rho, \iota)$ such that:

- $\rho : S \to \mathbb{R}_{\geq 0}$ is a state reward function

- $\iota : S \times Act \rightarrow \mathbb{R}_{\geq 0}$ is an action reward function

■

There are some additional concepts related to DTMCs and MDPs which are relevant to the understanding of temporal logics and model checking. These are the concepts of paths in Markov chains or decision processes, and adversary in a Markov decision process (adapted from [7, 37]).

Definition 4. Paths in DTMCs and MDPs

An *(infinite) path* π in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ is defined as an infinite sequence of states $\pi = s_0 \rightarrow s_1 \rightarrow s_2...$ (also written as $\pi = s_0 s_1 s_2...$) such that $\forall i \geq 0 : \mathbf{P}(s_i, s_{i+1}) > 0$.

A *finite path* ρ in the discrete-time Markov chain $\mathcal{M}$ is defined as the prefix of an infinite path π, $\rho = s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_n$ for $n > 0$.

An *(infinite) path* π in a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ is defined as an infinite sequence of states and actions $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2...$ (also written as $\pi = s_0 a_0 s_1 a_1 s_2...$) such that $\forall i \geq 0 : \mathbf{P}(s_i, a_i, s_{i+1}) > 0$.

A *finite path* ρ in the Markov decision process $\mathcal{M}$ is defined as the prefix of an infinite path π, $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n$ for $n > 0$.

In both Markov chains and decision processes, the set of all infinite paths is denoted by $Paths(\mathcal{M})$, while the set of finite paths is denoted by $Paths_{fin}(\mathcal{M})$ ■

Definition 5. Adversary (alternatively Strategy or Scheduler)

An *adversary* for an MDP $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ is a function $\sigma : S^+ \rightarrow Act$ such that $\sigma(s_0 s_1 ... s_n) \in Act(s_n)$ for all $s_0 s_1 ... s_n \in S^+$.

An adversary σ is called memoryless if, and only if, for any $\pi_1 = s_0 s_1 ... s_n$ and $\pi_2 = s'_0 s'_1 ... s_n$ it is true that $\sigma(\pi_1) = \sigma(\pi_2) = \sigma(s_n)$. ■

2.3.2 Temporal Logics

In order to verify properties in models, there needs to be a way of expressing those properties formally and unequivocally. While some reachability and reward-based properties come directly from the definition of DTMCs, MDPs and reward structures [37], more interesting and complex properties require the use of a temporal logic. A propositional temporal logic is essentially an extension of propositional logic by temporal operators [7]. In this context, one can consider time as being *linear* or *branching*. In linear time, at every moment in time there must only exist a single successor moment, whereas from the branching perspective, there may exist multiple ones, forming a tree-like structure of alternatives. Linear Temporal Logic (LTL) is a temporal logic which assumes the first, while Computation Tree Logic (CTL) is based on the branching perspective. Probabilistic Computational Tree Logic (PCTL) is an extension of CTL that takes into account a probabilistic operator.

Given the definition of a path in a MC and an MDP, it is possible to define the syntax for LTL and PCTL, as shown below (adapted from [7, 37]).

Definition 6. Syntax and Semantics of LTL

An *LTL formulae* φ over a set of atomic propositions AP can formed according to the following grammar:

$$\varphi ::= \texttt{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathrm{X}\varphi \mid \varphi_1 \mathrm{\ U\ } \varphi_2 \tag{2.9}$$

where $a \in AP$, φ_i are path formulas, X is the next operator and U is the until operator as defined below for Markov chains and decision processes.

For a given path $\pi = s_0 s_1 s_2 ...$ in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $\pi \models \texttt{true}$ always.
- $\pi \models a$ if, and only if, $a \in L(s_0)$.
- $\pi \models \varphi_1 \wedge \varphi_2$ if, and only if, $\pi \models \varphi_1 \wedge \pi \models \varphi_2$.
- $\pi \models \neg\varphi$ if, and only if, $\pi \not\models \varphi$.

- $\pi \models X\varphi$ if, and only if, $s_1 s_2 ... \models \varphi$.
- $\pi \models \varphi_1 \text{ U } \varphi_2$ if, and only if:

$$\exists i \geq 0 \text{ s.t. } (s_i s_{i+1} ... \models \varphi_2) \wedge (\forall j < i : s_j s_{j+1} ... \models \varphi_1) \tag{2.10}$$

The operator F (eventually) can also be defined as:

$$F\varphi = \texttt{true} \text{ U } \varphi \tag{2.11}$$

■

It can be noticed that, as a result of the linear time consideration in LTL, all the formulas are taken over paths. This is not the case in PCTL which can have both path and state formulae, as shown below.

Definition 7. Syntax and Semantics of PCTL

A *PCTL state formulae* Φ over a set of atomic propositions AP can formed according to the following grammar:

$$\Phi ::= \texttt{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathbb{P}_{\triangleright p}(\varphi) \tag{2.12}$$

where $a \in AP$, Φ_i are state formulas, φ is a path formula, $p \in [0,1]$, $\triangleright \in \{>, <, \geq, \leq\}$ is a probability bound, and $\mathbb{P}_{\triangleright p}$ is the probabilistic operator as defined below for Markov chains and decision processes.

For a given state $s \in S$ in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $s \models \texttt{true}$ always.
- $s \models a$ if, and only if, $a \in L(s)$.
- $s \models \Phi_1 \wedge \Phi_2$ if, and only if, $s \models \Phi_1 \wedge s \models \Phi_2$.
- $s \models \neg\Phi$ if, and only if, $s \not\models \Phi$.

- $s \models \mathbb{P}_{\triangleright p}(\varphi)$ if, and only if, $P_s(\pi \in Paths(s) : \pi \models \varphi) \triangleright p$ (that is, the probability, from state s, that φ is true for an outgoing path satisfies $\triangleright p$).

A *PCTL path formulae* φ over a set of atomic propositions AP can formed according to the following grammar:

$$\varphi ::= \mathrm{X}\Phi \ \mid \ \Phi_1 \ \mathrm{U}^{\leq k} \, \Phi_2 \ \mid \ \Phi_1 \ \mathrm{U} \ \Phi_2 \tag{2.13}$$

where $a \in AP$, $k \in \mathbb{N}$, Φ_i are state formulas, X is the next operator, U is the until operator, and $\mathrm{U}^{\leq k}$ is the bounded-until operator as defined below for Markov chains and decision processes.

For a given path $\pi = s_0 s_1 s_2 ...$ in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $\pi \models \mathrm{X}\Phi$ if, and only if, $s_1 \models \Phi$.
- $\pi \models \Phi_1 \ \mathrm{U}^{\leq k} \, \Phi_2$ if, and only if:

$$\exists i \leq k \text{ s.t. } (s_i \models \Phi_2) \wedge (\forall j < i : s_j \models \Phi_1) \tag{2.14}$$

- $\pi \models \Phi_1 \ \mathrm{U} \ \Phi_2$ if, and only if:

$$\exists i \geq 0 \text{ s.t. } (s_i \models \Phi_2) \wedge (\forall j < i : s_j \models \Phi_1) \tag{2.15}$$

Similarly to CTL, it is possible to define the operators F (eventually) and G (globally) as:

$$\begin{aligned} \mathrm{F}\Phi &= \texttt{true} \ \mathrm{U} \ \Phi \\ \mathrm{G}\Phi &= \neg\mathrm{E}(\neg\Phi) \end{aligned} \tag{2.16}$$

■

While PCTL path formulas are defined above, it should be mentioned that all PCTL formulas are state formulas, and that path formulas should only appear inside the probabilistic operator.

In relation to the probabilistic operator in PCTL, the generalisation from the Markov chain to the Markov decision process can be done using the concept of adversary (also denoted strategy or scheduler), initially introduced in Section 2.3.1. In the case of an MDP, the operator $\mathbb{P}_{\triangleright p}$ should be interpreted under the following semantics:

$$s \models \mathbb{P}_{\triangleright p}(\varphi) \text{ if, and only if, } \forall \sigma \in Adv : Pr_s^{\sigma}(\pi \in Paths^{\sigma}(s) : \pi \models \varphi) \triangleright p \tag{2.17}$$

that is, the probability, from state s, that φ is true for an outgoing path satisfies $\triangleright p$ for all adversaries $\sigma \in Adv$ (with Adv constituting the set of all adversaries that the MDP admits).

2.3.3 PRISM Language

PRISM is a probabilistic model checking tool developed in cooperation between the Universities of Oxford and Birmingham [39]. It was one of the first tools within stochastic model checking to be released and its development is currently headed by David Parker. In order to use PRISM (and other alternative model checking tools), it is essential to be able to specify the model. To achieve this, the team behind PRISM developed the PRISM Language, initially described in [40].

A model described in PRISM Language (typically a `.pm`, `.nm` or `.prism` text file) should contain a keyword that indicates the kind of model to expect. For example, the keyword `dtmc` indicates the model is a DTMC (discrete-time Markov chain) and `mdp` indicates a MDP (Markov decision process).

PRISM Language uses two different basic components to define a model: *modules* and *variables*. A module is composed of a set of local variables whose behaviour is described by *commands*. A general command is of the form:

```
[] guard -> prob_1: update_1 + ... + prob_n: update_n;
```

where *guard* is a boolean expression over the variables, *prob_i* is the probability that *update_i* will be taken (with $\sum_i \text{prob\_i} = 1$) and *update_i* is the new value the local and/or global variables take if the guard is true and a transition is taken to this configuration.

A state of the module is defined as a configuration of its local variables, and a state of the model is a configuration of the states of all modules which compose it and the global variables.

In Listing 2.1, an example of a simple MDP model described in PRISM Language is shown. This example is comprised of two modules, `M1` and `M2`, each with their own local variables, and no global variables are declared. It should be noted that the variables in PRISM Language should either be integers (`int`) or booleans (`bool`) and can be initialised by specifying `init` after the initial declaration. Due to the fact that we are dealing with finite models, the integers must be in a finite range.

```
// Example 1 (modified)
// Two process mutual exclusion

mdp

module M1

    x : [0..2] init 0;

    [] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
    [] x=1 & y!=2 -> (x'=2);
    [] x=2 -> 1:(x'=2);
    [] x=2 -> 1:(x'=0);

endmodule

module M2

    y : [0..2] init 0;

    [] y=0 -> 0.8:(y'=0) + 0.2:(y'=1);
    [] y=1 & x!=2 -> (y'=2);
    [] y=2 -> 1:(y'=2);
    [] y=2 -> 1:(y'=0);

endmodule
```

Listing 2.1 Example of a simple MDP model from [41] (modified)

Models in the PRISM Language can also include *formulas*, which are expressions that avoid repetition of code, and *labels*, which are boolean expressions that identify sets of states of interest.

It is also possible to define rewards structures using the following convention:

```
rewards "name"
    guard: state_reward;
    ...
    [action] guard: transition_reward;
    ...
endrewards
```

where the *guards* are similarly defined as before, but the distinction between state and transition rewards is defined by the *action* (which may be empty).

2.3.4 Model Checking and Strategy Synthesis in DTMCs and MDPs

After the description of the model using a language such as PRISM Language, it is possible to verify properties on the said model. These properties can be of different types according to the model and the model checking tool chosen.

In this dissertation, the models built will be represented through discrete-time Markov chains and Markov decision processes, so the properties of interest tend to be best expressed in the temporal logics LTL or PCTL. The PRISM model checker allows a user to import a text file with the probabilistic model, build it and verify LTL and PCTL properties on it [39]. Another tool used for this purpose is the Storm model checker [42]. Each of these tools has it strengths and weaknesses. PRISM is an extremely robust tool, with lots of options in terms of engines to perform the required calculations, and a graphical interface. Storm is a relatively recent tool which builds on top of some of PRISM's features and introduces new ones, such as an optimised Sparse engine (a great advantage when dealing with models with a small state space) and conditional properties in DTMCs [42]. However, it does not provide a graphical interface and is limited in terms of capabilities and I/O. Due to the fact that Storm builds on top of PRISM, it allows models described in PRISM Language as input and the properties are specified using the same convention.

In order to illustrate properties, consider the example of Listing 2.1. A PCTL property in this case could be defined as:

```
P>=0.5 [F x = 2]
```

This property can be interpreted as *"with probability greater or equal to 0.5, eventually the model reaches a state where* `x = 2`*"*. Building the model and verifying the property yields the value `false`, which means that, according to this model, there exists at least one adversary under which this property is not satisfied. Both PRISM and Storm allow for quantitative probabilistic properties as well. These properties take different formats in DTMCs and MDPs, as presented below.

```
// DTMC
P=? [F x = 2]

// MDP
Pmin=? [F x = 2]
Pmax=? [F x = 2]
```

Instead of returning a boolean value of satisfaction, these return the actual values of the probabilities of the properties being satisfied. They are, therefore, extremely useful metrics for model comparison. For the case of the DTMC, due to the fact that there is no nondeterminism, the probability of a path is uniquely defined (as seen in Section 2.3.2), and thus can be expressed simply using the operator `P=?`. For MDPs, this is not the case. In order to verify the probabilistic operator $\mathbb{P}_{\trianglerighteq p}$ on the path formula φ, a model checker needs to calculate the following quantities:

$$\begin{aligned} p_{\max,s}(\varphi) &= \sup_{\sigma \in Adv} P_s^\sigma(\varphi) \\ p_{\min,s}(\varphi) &= \inf_{\sigma \in Adv} P_s^\sigma(\varphi) \end{aligned} \tag{2.18}$$

where $P_s^\sigma(\varphi) = Pr_s^\sigma(\pi \in Paths^\sigma(s) : \pi \models \varphi)$.

Therefore, PRISM and Storm allow users to access these values by writing properties using `Pmax` and `Pmin` (boolean or quantitative).

Storm allows for conditional probability properties of the type:

```
P=? [ F x=2 || F y=2 ]
```

This property can be interpreted as *"what is the probability that the model eventually* *reaches the state* `x = 2`*, given that it eventually reaches the state* `y = 2`*"*. These properties, in DTMCs, correspond simply to the application of Bayes' theorem of conditional probability, which states that:

$$P_s(\varphi_1 \ || \ \varphi_2) = \frac{P_s(\varphi_1 \wedge \varphi_2)}{P_s(\varphi_2)} \tag{2.19}$$

For $P_s(\varphi_2) \neq 0$. It must be emphasised that this is **not** the case for MDPs, where the existence of multiple adversaries complicates the calculations of conditional probabilities.

Note: while uncommon in mathematical notation, the event *"A conditioned on B"* will be represented throughout the dissertation by $A \ || \ B$, so as to be distinguishable from the boolean disjunction operator which is represented as $A \mid B$ (reads *"A or B"*).

This does not mean that it is not possible to reason over conditional probabilities in MDPs. For some specific scenarios, it is possible to obtain meaningful bounds which allow comparisons between the ones obtained in DTMCs and MDPs. One of those scenarios is defined in the proposition below, with the proof following it.

Proposition 2.3.1. Consider a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ which admits adversaries $\sigma \in Adv$. Let φ_1, φ_2 be two PCTL path formulas, such that, for any path $\pi \in Paths(s)$: $\pi \models \varphi_1 \Rightarrow \pi \models \varphi_2$. Assume as well that:

$$\sigma' \in \arg \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_1 \ || \ \varphi_2) \tag{2.20}$$

And that:

$$P_s^{\sigma'}(\varphi_1) = p_{\max,s}(\varphi_1) \tag{2.21}$$

That is, an adversary which maximises the probability of $\varphi_1 \mid \varphi_2$ also maximises the probability of φ_1. In this case it holds that:

$$p_{\max,s}(\varphi_1 \ || \ \varphi_2) \geq \frac{p_{\max,s}(\varphi_1)}{p_{\max,s}(\varphi_2)} \tag{2.22}$$

with:

$$p_{\max,s}(\varphi_1 \ || \ \varphi_2) = \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_1 \ || \ \varphi_2) \tag{2.23}$$

Proof. Consider $\sigma_2 \in Adv$ such that:

$$\sigma_2 \in \arg \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_2) \tag{2.24}$$

Assume, for the sake of contradiction, that:

$$p_{\max,s}(\varphi_1 \ || \ \varphi_2) < \frac{p_{\max,s}(\varphi_1)}{p_{\max,s}(\varphi_2)} \tag{2.25}$$

Under adversary σ', it can be written that, through Bayes' theorem of conditional probability:

$$p_{\max,s}(\varphi_1 \ || \ \varphi_2) = P_s^{\sigma'}(\varphi_1 \ || \ \varphi_2) = \frac{P_s^{\sigma'}(\varphi_1 \wedge \varphi_2)}{P_s^{\sigma'}(\varphi_2)} = \frac{P_s^{\sigma'}(\varphi_1)}{P_s^{\sigma'}(\varphi_2)} \tag{2.26}$$

Therefore, it follows that:

$$\frac{P_s^{\sigma'}(\varphi_1)}{P_s^{\sigma'}(\varphi_2)} < \frac{p_{\max,s}(\varphi_1)}{p_{\max,s}(\varphi_2)} \tag{2.27}$$

For this to be true, it must be that $P_s^{\sigma'}(\varphi_2) > p_{\max,s}(\varphi_2)$, since $P_s^{\sigma'}(\varphi_1) = p_{\max,s}(\varphi_1)$. This constitutes a contradiction, since in that case σ_2 would not maximise the probability of φ_2 ($p_{\max,s}(\varphi_2) = P_s^{\sigma_2}(\varphi_2)$, by definition), a violation of the assumption described in Equation 2.24. Thus, the proposition is proven. ■

In MDPs it is also possible to perform multi-objective verification, whereas multiple properties will be tested simultaneously (both in PRISM and Storm). One example of this would be:

```
multi(P>=0 [F x=2], P>=0.2 [F y=2])
```

The value of the verification of this property corresponds to the boolean conjunction of the individual properties. It is also possible to optimise over a single property while constraining other properties, restricting the possibilities in terms of outcomes. An example of this would be:

```
multi(Pmax=? [F x=2], P>=0.2 [F y=2])
```

Finally, it is also possible to perform multi-objective optimisation over the properties by defining them as (for example):

```
multi(Pmax=? [F x=2], Pmax=? [F y=2])
```

This leads to a situation where compromises between the optimisation of either one of the properties might have to be reached. Pareto curves are the result of the verification of such properties and their formal definition is presented below (adapted from [43]).

Definition 8. Multi-objective Query, Pareto Vector and Pareto Curve (alternatively Set)

A *multi-objective query* (MQ) ϕ of n objectives is a positive boolean combination of n predicates of the form $r_i \geq v_i$, where r_i is a reward function, $v_i \in \mathbb{Q}$ is a bound. The notation $\phi[\mathbf{x}]$, $\mathbf{x} \in \mathbb{R}^n$ is used to denote ϕ in which each $r_i \geq v_i$ is replaced by $r_i \geq x_i$

Consider a multi-objective query MQ ϕ of n objectives. The vector $\mathbf{x} \in \mathbb{R}^n$ is a *Pareto vector* of ϕ if, and only if:

1. $\phi[\mathbf{x} - \varepsilon]$ is achievable for all $\varepsilon > 0$, and
2. $\phi[\mathbf{x} + \varepsilon]$ is not achievable for any $\varepsilon > 0$

A *Pareto curve* of ϕ is the set of all Pareto vectors that ϕ admits.

■

Thus, the result of the verification of a multi-objective optimisation property corresponds to a Pareto curve (this curve might contain a single vector). Both PRISM and Storm allow for the verification of properties containing up to 2 objectives each. As a result, 2D plots are the best way to represent these solutions. In Figure 2.4, the result of the verification of the property `multi(Pmax=?  [F G x=2], Pmax=?  [F G y=2])` in the model described in Listing 2.1 is shown. As it can be observed, either property can be satisfied with certainty, but not both simultaneously (mutual exclusion). The light green area under the curve corresponds to the achievable set.

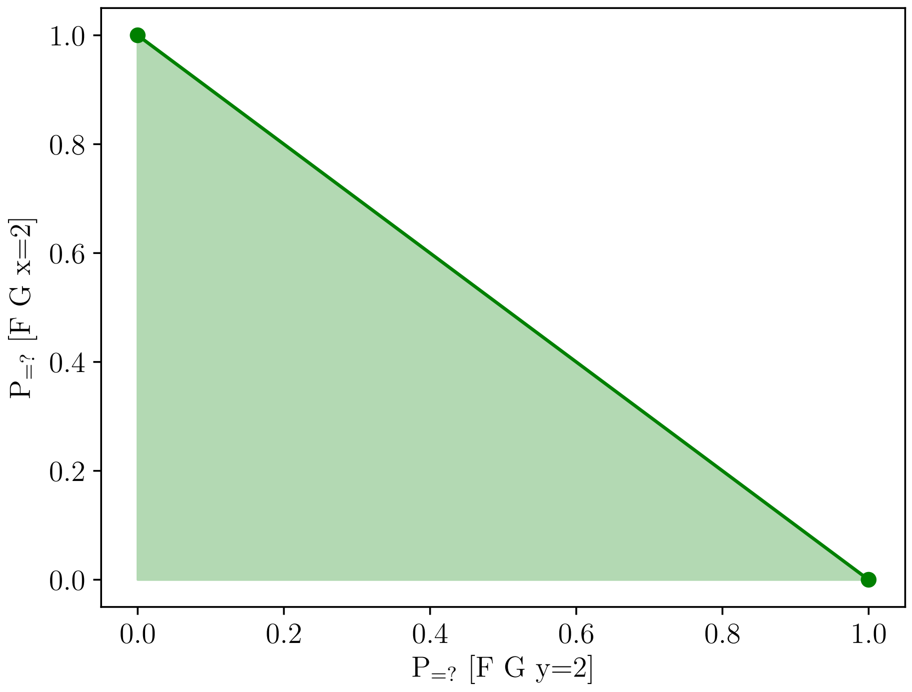


Figure 2.4 Example of a Pareto curve represented as a 2D plot.

Each of the values resulting of the verification of quantitative properties in MDPs (whether they be a single value from a single-objective optimisation, or a Pareto vector) is the result of an adversary which solves the nondeterminism and converts the MDP into a DTMC [7]. The process of obtaining the adversary which satisfies certain properties is denoted by **strategy synthesis** or **adversary generation**, and it is formally defined below.

Definition 9. Strategy Synthesis (alternatively Adversary Generation)

Consider a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$. The process of maximal (similarly minimal) *strategy synthesis* of the path formula φ is defined as a process that yields an adversary $\sigma \in Adv$ such that:

$$\sigma \in \arg\sup_{\sigma' \in Adv} P_s^{\sigma'}(\varphi) \tag{2.28}$$

(respectively $\arg\inf$ for minimal), where Adv is the set of all adversaries that $\mathcal{M}$ admits.

■

Chapter 3

Human Driver Modelling

In this chapter, the human driver model that Salvucci presents in [26] is implemented and the metrics used to verify the model are established. Initially, the model is implemented in *Matlab* based on the integrated driver model. To be able to verify it using the probabilistic model checking techniques presented in Chapter 2, the model undergoes an abstraction process, out of which a discrete-time Markov chain is obtained using several different approaches and assumptions. Metrics are then established to be able to evaluate whether the model meets the requirements and how well the human performs in a given situation. Finally, a simulator is presented in Section 3.4 in order to allow visualisation of paths in the model.

3.1 Continuous Driver Model using ACT-R

In order to obtain a continuous driver model, we interpret Salvucci's integrated driver model as presented in [26]. The scenario considered is the one Salvucci envisioned: a multilane highway with moderate (or low) traffic [26]. The model uses the three modules presented in Figure 2.2, that is, monitoring, decision making and control. All these modules rely on the constant update of the environment, as some are based on visual or low-level perception cues. While the human model is presented clearly in [26], the dynamics of the car (which is part of the environment) are left to the reader, as it is outside the scope of the paper. In that sense, some assumptions had to be made about the environment.

The main initial assumption is that the driver is perfectly aware of its surroundings, and thus it is able to obtain the positions of other vehicles (within a certain distance) and the near and far points perfectly (this assumption is challenged later in the abstraction process). Additionally, the environment is assumed to change at the same rate as a cycle of the ACT-R model.

The overall view of the system is presented in Figure 3.1. The information flow is marked using the dotted arrows, while the sequential flow of the program is marked in the filled, lighter gray ones. Both the monitoring and the decision making make use of the perception which corresponds to querying the environment for the information (e.g. near and far point or another vehicle's position). The control both queries and sends updated information (e.g. vehicle position, velocity and acceleration) to the environment. Modules are updated sequentially, following the order: control, monitoring, decision making and environment (e.g. position, velocity and acceleration of other vehicles).

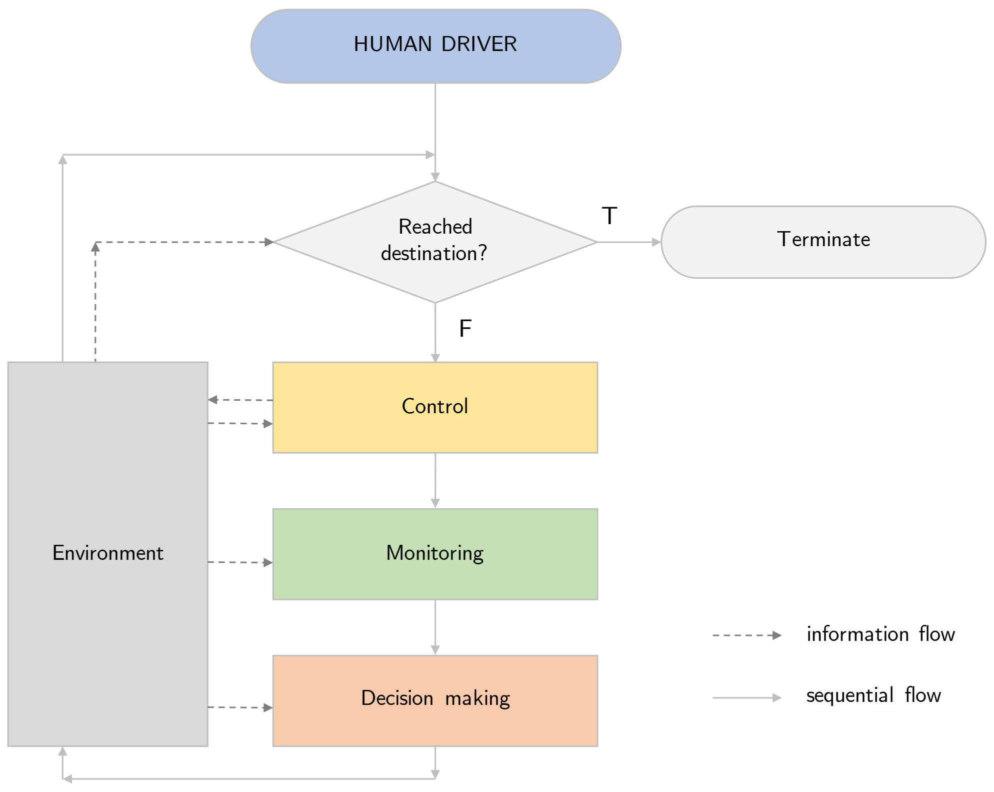


Figure 3.1 Continuous Driver Model overview.

The monitoring module is implemented according to the flowchart presented in Figure 3.2, from the description given in [26]. The threshold for monitoring is set at 0.2, as this is the value suggested by Salvucci in [26]. The output of this process corresponds to altering the global variable a corresponding to the declarative memory cell composed of k chunks. No specific value for k is given by Salvucci in [26], but it is taken to be 8 from [25]. The function `GET_DISTANCE` is not described in depth due to the fact that it is trivial under the assumption stated previously.

The overall flowchart of the decision making module is shown in Figure 3.3. The high level part is similar to the flowchart presented in Figure 2.3, with small differences related to the loading of variables and information from the different memories available. It is worth noticing that the control makes use of the function `LOOK_VEHICLE` initially presented in the monitoring module to verify the presence of a vehicle in a relative position of a lane. The flow of the function `SET_LANE_FOLLOWING` is also omitted due to the trivial nature of this function under the assumptions made.

Finally, a flowchart for the control module is presented in Figure 3.4. This module is responsible for the calculation of $\Delta\varphi(t)$ and $\Delta\psi(t)$ and the update of the position. Given these two values, the position is updated following discrete laws of motion applied to rigid objects, particularly:

$$\begin{aligned}
&\mathbf{v}(t) = \mathbf{v}(t-1) + \mathbf{a}(t)\Delta t \\
&\mathbf{x}(t) = \mathbf{x}(t-1) + \mathbf{v}(t-1)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2 \\
&t = t + \Delta t
\end{aligned} \tag{3.1}$$

where $\mathbf{x}(t) = (x(t), y(t))$, $\mathbf{v}(t) = (v_x(t), v_y(t))$, and $\mathbf{a}(t) = (a_x(t), a_y(t))$, with:

$$\begin{aligned}
a_x(t) &= \Delta\psi(t) \\
a_y(t) &= \sin(\Delta\varphi(t))
\end{aligned} \tag{3.2}$$

In this case, as Salvucci mentions in [26], $\Delta t = 0.5s$. It is also assumed that the vehicle complies with the speed limits, and, therefore, $v_x \in [15, 34]\ m/s$ (between $50km/h$ and $120km/h$ - typical limits in highway speeds).

Using these assumptions, the continuous driver model is implemented in *Matlab* as presented in Appendix A.1.

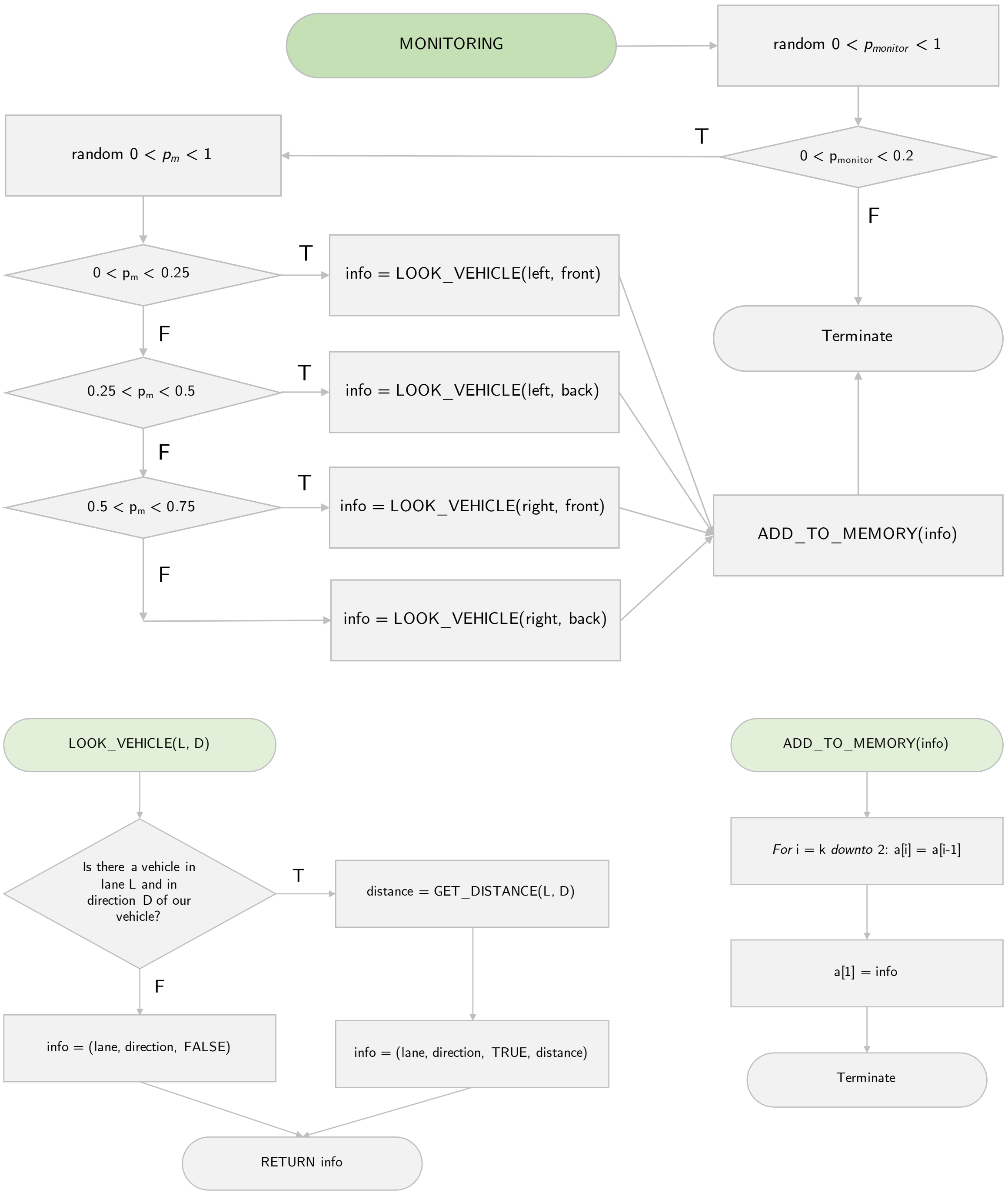
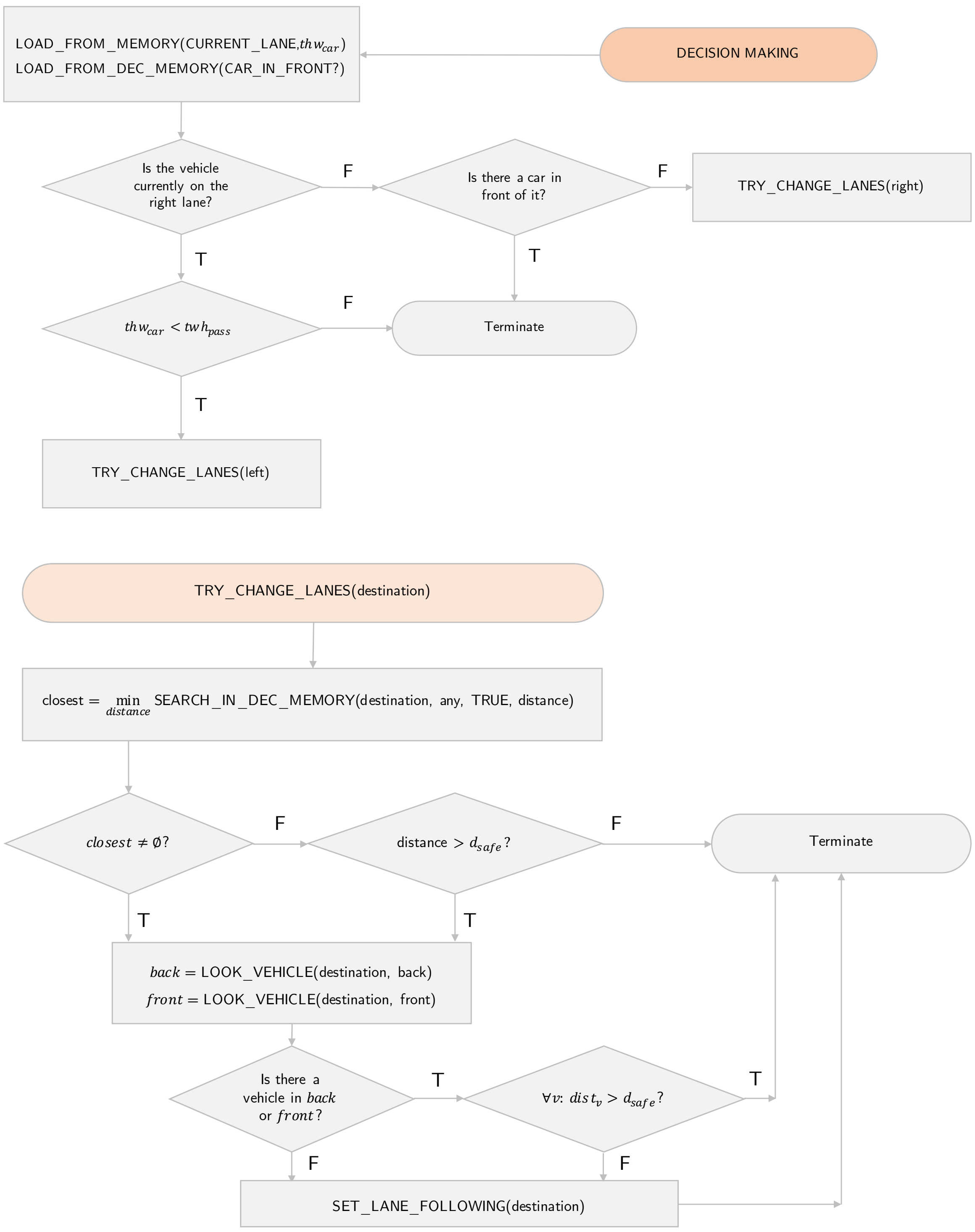
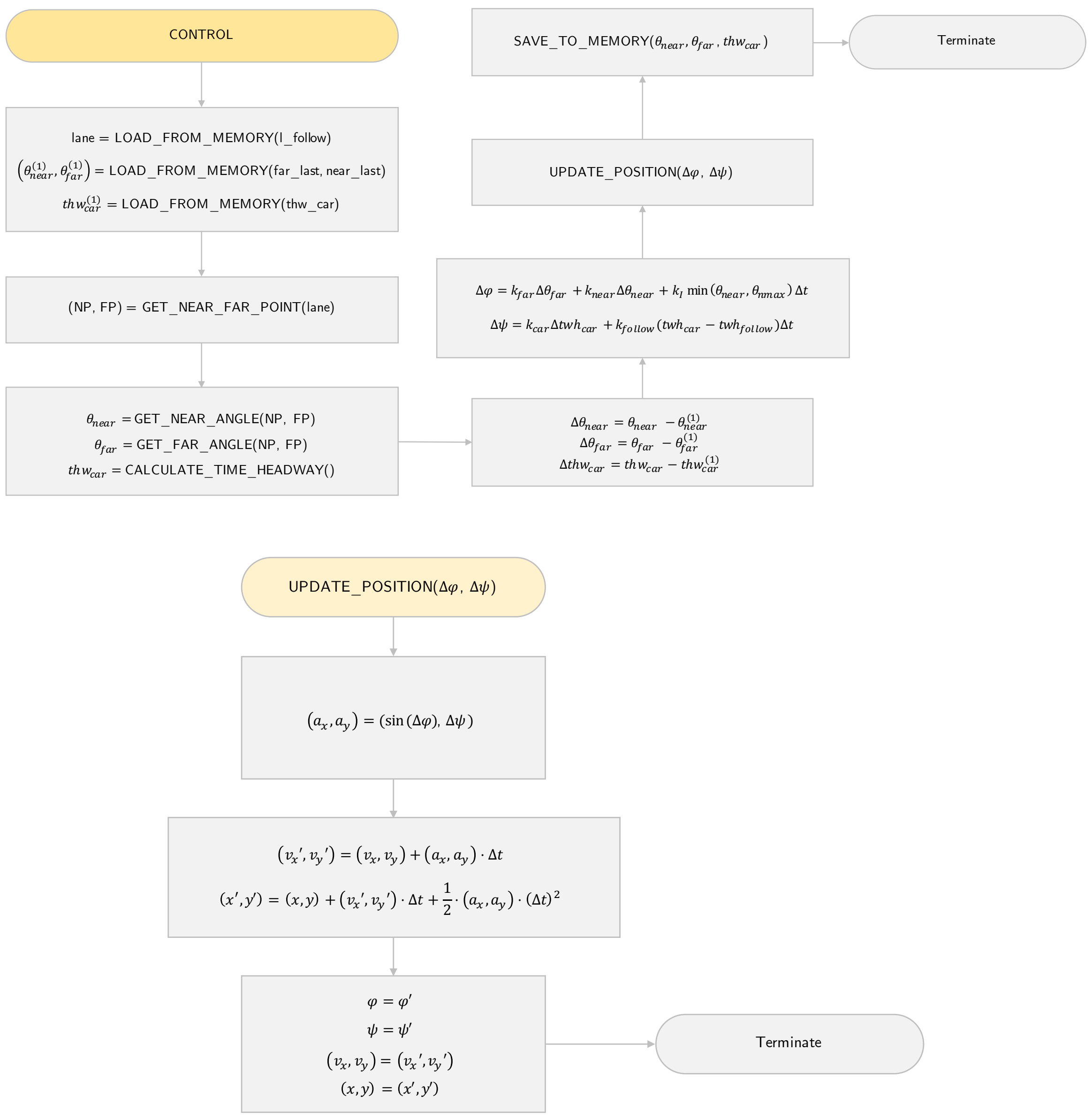

MONITORING
random $0 < p_{monitor} < 1$
$0 < p_{monitor} < 0.2$
T
F
random $0 < p_m < 1$
$0 < p_m < 0.25$
T
info = LOOK_VEHICLE(left, front)
F
$0.25 < p_m < 0.5$
T
info = LOOK_VEHICLE(left, back)
F
$0.5 < p_m < 0.75$
T
info = LOOK_VEHICLE(right, front)
F
info = LOOK_VEHICLE(right, back)
Terminate
ADD_TO_MEMORY(info)
LOOK_VEHICLE(L, D)
Is there a vehicle in lane L and in direction D of our vehicle?
T
distance = GET_DISTANCE(L, D)
F
info = (lane, direction, FALSE)
info = (lane, direction, TRUE, distance)
RETURN info
ADD_TO_MEMORY(info)
For i = k *downto* 2: a[i] = a[i-1]
a[1] = info
Terminate

Figure 3.2 Flowchart of the Monitoring module.



LOAD_FROM_MEMORY(CURRENT_LANE,thw_{car})
LOAD_FROM_DEC_MEMORY(CAR_IN_FRONT?)
DECISION MAKING
Is the vehicle currently on the right lane?
F
Is there a car in front of it?
F
TRY_CHANGE_LANES(right)
T
T
$thw_{car} < twh_{pass}$
F
Terminate
T
TRY_CHANGE_LANES(left)
TRY_CHANGE_LANES(destination)
closest = $\min_{distance}$ SEARCH_IN_DEC_MEMORY(destination, any, TRUE, distance)
$closest \neq \emptyset$?
F
distance > d_{safe}?
F
Terminate
T
T
$back$ = LOOK_VEHICLE(destination, back)
$front$ = LOOK_VEHICLE(destination, front)
Is there a vehicle in $back$ or $front$?
T
$\forall v$: $dist_v > d_{safe}$?
T
F
F
SET_LANE_FOLLOWING(destination)

Figure 3.3 Flowchart of the Decision Making module.



CONTROL
lane = LOAD_FROM_MEMORY(l_follow)
$\left(\theta_{near}^{(1)}, \theta_{far}^{(1)}\right)$ = LOAD_FROM_MEMORY(far_last, near_last)
$thw_{car}^{(1)}$ = LOAD_FROM_MEMORY(thw_car)
(NP, FP) = GET_NEAR_FAR_POINT(lane)
θ_{near} = GET_NEAR_ANGLE(NP, FP)
θ_{far} = GET_FAR_ANGLE(NP, FP)
thw_{car} = CALCULATE_TIME_HEADWAY()
$\Delta\theta_{near} = \theta_{near} - \theta_{near}^{(1)}$
$\Delta\theta_{far} = \theta_{far} - \theta_{far}^{(1)}$
$\Delta thw_{car} = thw_{car} - thw_{car}^{(1)}$
$\Delta\varphi = k_{far}\Delta\theta_{far} + k_{near}\Delta\theta_{near} + k_I \min(\theta_{near}, \theta_{nmax})\,\Delta t$
$\Delta\psi = k_{car}\Delta twh_{car} + k_{follow}(twh_{car} - twh_{follow})\Delta t$
UPDATE_POSITION($\Delta\varphi$, $\Delta\psi$)
SAVE_TO_MEMORY($\theta_{near}, \theta_{far}, thw_{car}$)
Terminate
UPDATE_POSITION($\Delta\varphi$, $\Delta\psi$)
$(a_x, a_y) = (\sin(\Delta\varphi), \Delta\psi)$
$(v_x', v_y') = (v_x, v_y) + (a_x, a_y)\cdot\Delta t$
$(x', y') = (x, y) + (v_x', v_y')\cdot\Delta t + \frac{1}{2}\cdot(a_x, a_y)\cdot(\Delta t)^2$
$\varphi = \varphi'$
$\psi = \psi'$
$(v_x, v_y) = (v_x', v_y')$
$(x, y) = (x', y')$
Terminate

Figure 3.4 Flowchart of the Control module.

3.2 Model Abstraction

The process of abstraction consists in transforming the evolution of the continuous variables that constitute the model and discretising it in order to obtain a finite discrete model. In this case, the goal is to go from the continuous model of driver behaviour presented in the previous section to a discrete-time Markov chain (DTMC) which represents the same process. This transformation is less than trivial due to the fact that, in model checking of finite state abstractions, as the number of state variables in the system increases, the size of the system state space grows exponentially [44]. This problem is known as state explosion, and there has been some research into efficient ways of dealing with it [45, 46].

In the following sections, the abstraction process for the different modules of ACT-R is presented in detail, as well as the thought process behind the choices made. Finally, the unified model is presented using the different modules designed.

3.2.1 Non-Probabilistic Control Module

The control module, as presented in the previous section and following Salvucci's approach in [26], is fully deterministic, in the sense that there is no probabilistic reasoning involved. It makes use of perception (for the near and far points) and two simple control laws which influence the position, velocity and acceleration of the vehicle in the environment.

The straightforward approach to this problem would be to represent it as a simple $N \times M$ grid, in which a tuple (x, y) with $x \in \{0, ..., N\}$ and $y \in \{0, ..., M\}$ would describe the position of the vehicle, and all other variables (i.e. v_x, v_y, a_x, a_y and t) would be integer versions of its continuous model representation. In this approach, the control laws could be applied directly over the grid and the results would be the movement of the vehicle in it. However, there is an intrinsic problem with this approach. The error associated with the use of the grid as a way to discretise space would, logically, be reduced with an increase of N and M for a representation of the same road segment (i.e. increase in resolution). For example, the error associated with a road segment of $500 \times 7m^2$ represented by $N = 500$ and $M = 7$ (each grid square corresponds to $1m^2$) would be significantly greater than if $N = 5000$ and $M = 70$ (each grid square corresponds to 10^{-2} m^2). The conclusion would be that one should increase N and M in order to obtain an appropriate resolution, but this incurs in the problem of state explosion presented in the beginning of the section: as the variable ranges increase, the number of states in the system grows exponentially. Thus, while the situation of $N = 5000$ and $M = 70$ is appealing in terms of error minimisation, it would be intractable to actually build and perform model checking on it (e.g. for $v_x, v_y \in \{15, ..., 34\}$ and $a_x, a_y \in \{-3, ..., 3\}$, the control module alone would have approximately 6.89×10^9 states, making it impossible to add decision making and monitoring on top of it).

The straightforward strategy is not, by any standards, efficient in terms of state space and does not use any of the information of the problem to simplify it. In particular, one should notice that, in this scenario, the movement in the y direction is used simply for the purpose of lane changing, a process which is purely deterministic (as described in [26]). Therefore, a more interesting approach to the abstraction of the control module could take this into account and **pre-simulate the lane changing operations**.

In this approach, the vehicle's position is represented by two discrete integer variables $x \in \{0, ..., length\}$ for a given $length$ and $lane \in \{right, left\}$ (can be extended to more than two lanes), and the acceleration and velocity of the vehicle are simply $a = a_x$ and $v = v_x$. Time is still represented in the same way, with $t \in \{0, ..., max\_time\}$. If a lane change is decided by the decision making module, then, **within one transition of the control**, the *lane* variable is updated with the final destination lane, and the variables x, v, a and t are updated to reflect the obtained values after a lane change. These values are obtained from a look-up table and depend on the distance to the other vehicle (d) and the velocities of both the vehicle in question (v) and the other vehicle (v_1) (an obstacle on the road can be modelled by setting $v_1 = 0$). Additionally, the variable *crashed* is used to represent whether the vehicle has collided in the process or not. The linear acceleration is implemented similarly using pre-simulation to determine the value of acceleration and the motion law is updated in the model itself.

The continuous model of driver behaviour obtained in the previous section in *Matlab* is used as the basis for the simulation of lane changes and linear acceleration. From this model, look-up tables are obtained which, given an origin lane (o_{lane}), a distance to the other vehicle and the velocities of both the vehicle in question and the other vehicle determines whether a crash happened or not, what is the Δx and ΔT incurred (how did the position in x changed and how long did it take), and the final velocity of the vehicle in question. The code presented in Appendix A.2 corresponds to a similar version of the control (but probabilistic, as presented later in this section). An example of the simulation of the scenario $o_{lane} = 1$, $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$ is shown in Figure 3.5.

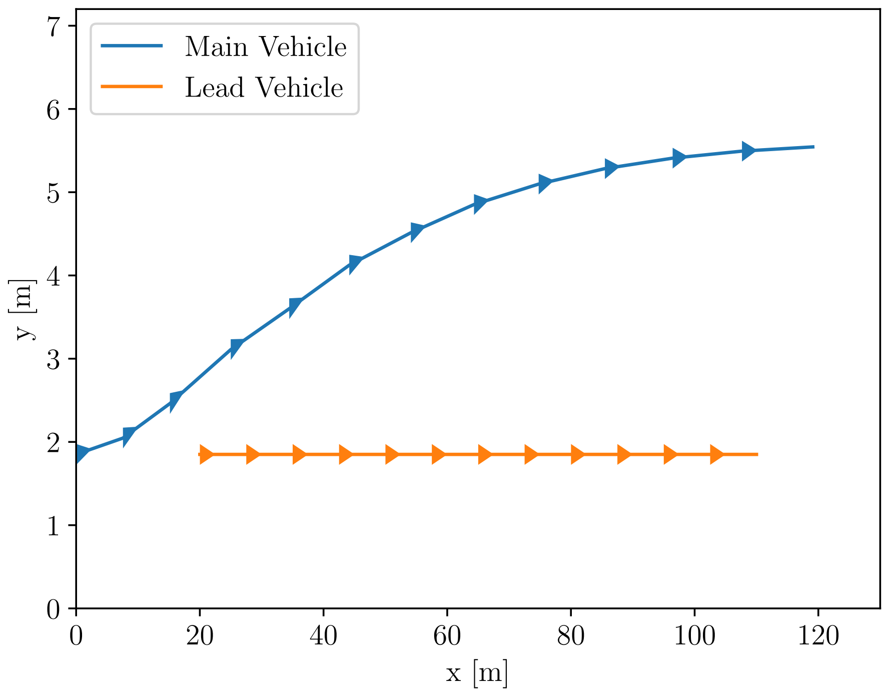


Figure 3.5 Example of the simulation of the lane change for $o_{lane} = 1$ (right lane), $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$. In this case, after the change is complete the variable assignments are $crashed = 0$ (no collision happened), $\Delta x = 119m$, $\Delta T = 6s$ and $v = 23m/s$.

This process focuses the computational effort on the pre-calculations, simplifying the final model significantly in terms of the state space, while removing the error made by discretising in the y direction of the grid (the error in the x direction is still present though). With regards to the size of the lane changing table, for the considered scenario of 2 lanes, $v, v_1 \in \{15, ..., 34\}$ and for any discrete $d \in \{1, ..., length\}$, the size of the table would be $2 \times 20^2 \times length = 800 \times length$. Given that the only value of the table that is effectively altered with a change in d is the value of $crashed$ (as the Δx, ΔT and final v depend solely on the initial velocities of both vehicles), it is possible to define $d_{\max}$ such that:

$$d_{\max} < length, \text{ s.t. } \forall v, v_1 \in \{15, ..., 34\}, d' \geq d_{\max} : crashed_{d'} = false \tag{3.3}$$

that is, $d_{\max}$ is a great enough value of d such that, regardless of the speed of the two vehicles, no crash will occur between them. With this change, it is possible to reduce the table from $800 \times length$ to $800 \times d_{\max}$ (since all other rows would be equal to the one for d_{max}). For the given ranges of v and v_1, $d_{\max}$ is determined to be $43m$, and thus the obtained look-up table contains $34,400$ rows.

The linear acceleration table depends on the thw_{car} which varies only with the distance to the other vehicle and the current velocity of the vehicle in question. Considering the distance in this case to be up to $80m$ (otherwise maximum acceleration will be applied), the look-up table generated has $1,600$ rows.

3.2.2 Decision Making and Monitoring Module

Due to the fact that, in the model presented by Salvucci in [26], the monitoring module influences the declarative memory which is then used exclusively for decision making purposes (and not for control), logically these two can be incorporated into one for abstraction purposes. Again, taking the straightforward approach of trying to include the declarative memory and the process associated with monitoring and decision making into the model directly would make it intractable.

In its simplest form, the decision making process for the right lane consists in localising the other vehicle and deciding whether or not to change lanes based on the time headway, thw_{car}, to the lead vehicle (if there is one). This value essentially corresponds to the time the vehicle in question has before it crashes into the lead vehicle, assuming that latter came to a full stop ($v = a = 0$) [47]. The lower this time headway, the more likely a driver is to perform the manoeuvre. While Salvucci presents a fully deterministic driver in [26] based on the average driver, in this dissertation it was decided to try and analyse drivers according to different profiles in order to simulate how different parts of the population of drivers might make decisions. The decision making in such a case follows from a stochastic reasoning which can be simulated and incorporated in the model using look-up tables.

In this case, for a driver in the right lane, it was decided to model the probability of the driver performing a lane change based on the time headway as an exponentially decreasing function, writing it as:

$$\mathrm{P}[lC = true]_{thw} = P_{lC}(d, v) = e^{-\alpha \cdot thw} = e^{-\alpha \cdot d/v} \tag{3.4}$$

where α is a parameter unique to each population of drivers.

The same logic can be applied for a driver in the left lane overtaking a vehicle behind it in the left lane, except in this case the opposite effect will be seen in the decision making. In such a case, the probability of changing lane can be modelled as a normalised logarithmic function over the distance of the vehicles (it doesn't make sense to define time headway in this context):

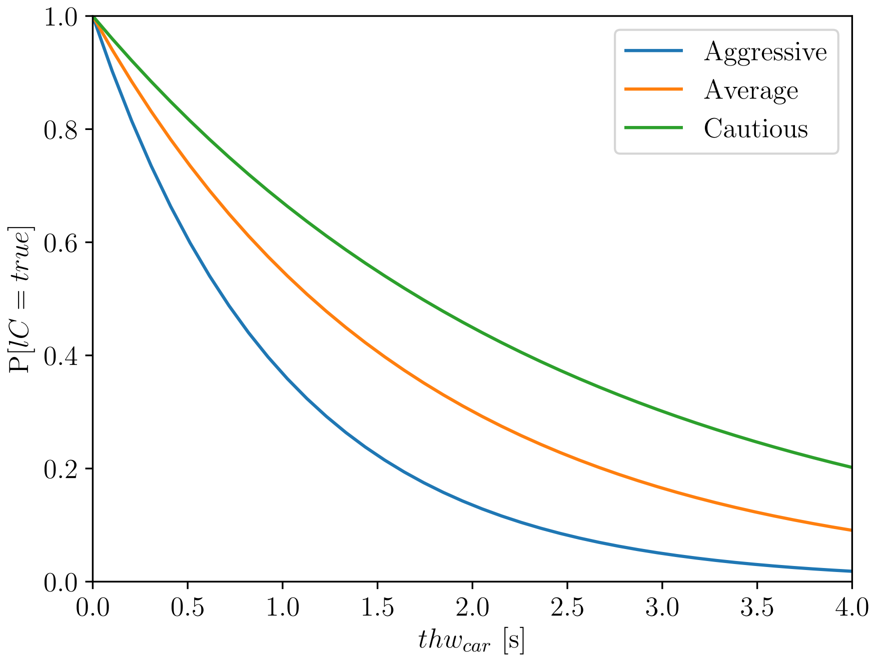
$$\mathrm{P}[lC = true]_{d} = P_{lC}(d) = \frac{1}{\log(\beta \cdot \max_d + 1)} \cdot \log(\beta \cdot d + 1) \tag{3.5}$$

where $\max_d$ is the maximum length considered and β is a parameter unique to each population of drivers.

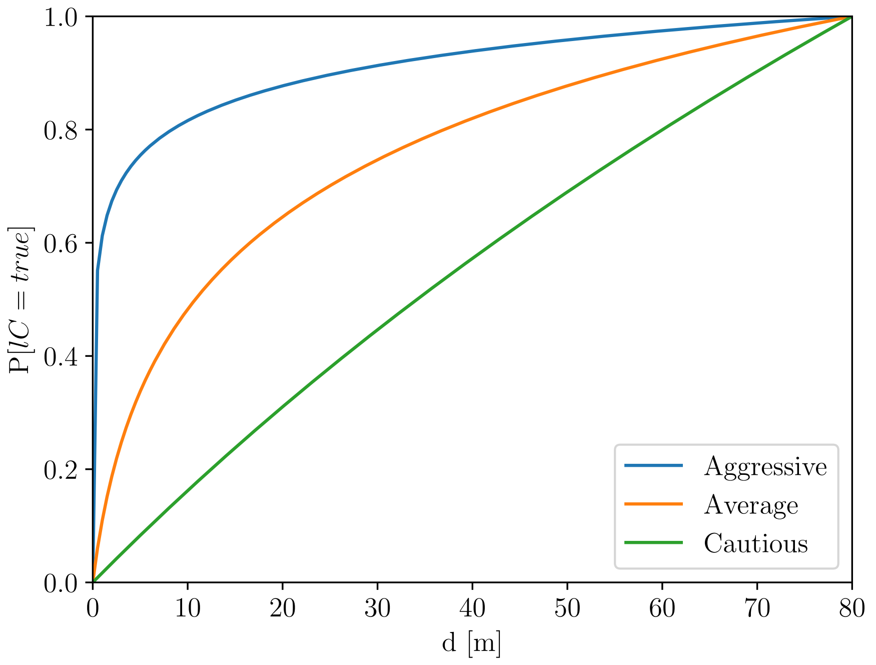
It should be noted that these functions, while logical, are an assumption of the abstraction, since the model presented by Salvucci in [26] does not make any reference to these parameterisations. It is also worth referring that, while there was no real-world data involved in this project, this function could have been learnt from real data without changing the fundamental paradigm of the abstraction.

For the purposes of analysis, three classes of drivers are considered: **Aggressive**, **Average** and **Cautious** drivers (with different values for the parameter α and β). In Figure 3.6, the functions are presented for the three profiles of drivers assumed in this project, for the lane changes originating in the right (Figure 3.6 (a)) and left lane (Figure 3.6 (b)).

So far, this version of decision making is not influenced by the monitoring module at all, and it relies on the correct calculation of the value of thw_{car} and d. This assumption is unrealistic, and in order to simulate these issues, it was decided that stochastic noise could be added to the measurement of the distance (as this is what



a. Aggressive ($\alpha = 1$), Average ($\alpha = 0.6$) and Cautious ($\alpha = 0.4$) drivers.



b. Aggressive ($\beta = 1000$), Average ($\beta = 0.5$) and Cautious ($\beta = 0.01$) drivers.

Figure 3.6 $P[lC = true]$ as a function of thw_{car} and d for vehicles originating from the right lane (a) and left lane (b).

human drivers have to instinctively measure through perception). It was chosen that this noise, n, would be normally distributed as:

$$n \sim \mathcal{N}(0, \sigma) \tag{3.6}$$

such that:

$$thw'_{car} = thw_{car} + n(thw_{car}) \tag{3.7}$$

Alternatively, this can be represented directly as (using discrete integration steps of width δ):

$$\begin{aligned} P'_{lC}(d, v) &= \sum_{i=-\infty}^{+\infty} [N(d+i+\delta/2) - N(d+i-\delta/2)] \cdot P_{lC}(d+i, v) \\ &= \sum_{i=-\max_d}^{+\max_d} [N(d+i+\delta/2) - N(d+i-\delta/2)] \cdot P_{lC}(d+i, v) \end{aligned} \tag{3.8}$$

where $\max_d$ is the maximum length considered and $N(d)$ corresponds to the cumulative probability function (CDF) defined as:

$$N(d) = \int_{-\infty}^{d} n(t)dt \tag{3.9}$$

From this, a look-up table can be generated which, for different driver types, yields the probability of lane changing for a certain distance to the lead car and velocity. For the 3 driver profiles mentioned, considering $d \in \{1, ..., 80\}$ ($\max_d = 80$) and $v \in \{15, ..., 34\}$, the table generated for the vehicle in the right lane has $4,800$ rows [1]. Under the same conditions, the look-up table for the vehicle in the left obtained has 240 rows (the difference is explained by the fact that the velocity does not influence this table). Thus, the unified decision making table has $5,040$ rows.

The code presented in Appendix A.3 corresponds to the methods used to obtain the unified look-up table used for the decision making and monitoring module.

[1]In the data obtained for the results presented in this dissertation, $\delta = 1$.

3.2.3 Probabilistic Control Module

In light of the uncertainty introduced in the decision making and monitoring module regarding the visual perception of distance, it becomes only natural that this would be extended to the control module as well. In order to cope with this, a probabilistic version of the control module was designed, where the noise n presented in the previous section is taken into consideration.

In order to obtain the equivalent look-up tables for the probabilisitic control module, the **Monte Carlo method** can be used. In this case, the noise can be simulated in repeated trials, and an average can be obtained for the crashing probability, the estimated final difference in position, the estimated difference in time and estimated final velocity of the vehicle. In the case of this dissertation, the tables generated (for linear and lane changing control) were the result of 100 trials.

The code presented in Appendix A.2 corresponds to the simulation of this probabilistic version of the control, which generates the linear and steering control tables.

3.2.4 Unified Two-Module Model

Using the three tables generated (two for the probabilisitic control - linear and steering control - and one for the decision making), the final DTMC model unifies both using a sequential variable $actr\_state \in \{1, 2\}$, where $actr\_state = 1$ corresponds to the control and $actr\_state = 2$ corresponds to decision making. In the version of the unified model built for this thesis, only two vehicles are present: the first is the one controlled by the driver and the second is a lead vehicle starting at a distance $x = x_{1,0}$ (in meters) from the main vehicle (which starts at $x = 0$) and moves with a constant speed of v_1 on a road segment of *length* meters. The movement of the lead vehicle is, therefore, completely determined by the formula:

$$x_1(t) = x_{1,0} + v_1 \cdot t \tag{3.10}$$

The distance between the two vehicles can be obtained as:

$$d(t) = |x(t) - x_1(t)| \tag{3.11}$$

And the boolean function *posDist* can be defined as:

$$posDist(t) = \begin{cases} \text{true}, & \text{if } x(t) \geq x_1(t) \\ \text{false}, & \text{otherwise.} \end{cases} \tag{3.12}$$

Due to the large size of the tables generated, the models should be generated automatically for a given tuple of initial conditions $(d_{type}, v, v_1, x_{1,0})$ (where d_{type} is the driver class according to the ones defined in the decision making and monitoring subsection - 1 is aggressive, 2 is average and 3 is cautious), following the assumptions described below:

General assumptions

1. A transition from state s to state s' is possible if, and only if, the variable *actr_state* takes different values in s and s'.
2. A transition from state s to state s' is possible if, and only if, $t_{s'} \geq t_s$ (where t_α is the value of variable t in state α).
3. The model should have no states with self transitions, as there is always a continuous evolution of the state of the vehicles.
4. A deadlock state should only be entered if, and only if, the vehicle either crashes ($crashed = true$) or reaches the end of the road ($x = length$).

Control ($actr\_state = 1$)

1. If no lane change has been decided, the model reasons over linear acceleration using the corresponding look-up table for $d(t)$ and $v(t)$, and updates the state variables t, x, v, a and *crashed* accordingly using the discrete laws of motion previously presented.
2. If a lane change has been decided, the model reasons over the lane change using the corresponding look-up table for o_{lane}, $d(t)$, $v(t)$, v_1, and updates the variables x and t incrementally using the values Δx and ΔT of the table, as well as sets the value of v and *crashed*.

Decision Making and Monitoring ($actr\_state = 2$)

1. If the vehicle is on the left lane and behind the other vehicle ($lane = left$ and $posDist = false$), the model will not attempt a lane change.
2. If the vehicle is on the left lane and in front of the other vehicle ($lane = left$ and $posDist = true$), the model will reason over lane changes using the decision making and monitoring look-up table for d_{type}.
3. If the vehicle is on the right lane and behind the other vehicle ($lane = right$ and $posDist = false$), the model will reason over lane changes using the decision making and monitoring look-up table for d_{type}.
4. If the vehicle is on the right lane and in front of the other vehicle ($lane = left$ and $posDist = true$), the model will not attempt a lane change.

From these rules and the tables obtained in the abstraction of the individual modules, it is possible to build a model generator. The code for an example of such a generator written in *Python* (used throughout the rest of the dissertation) is presented in Appendix A.4.

By running the model generator using the conditions $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 51)$, that is, an aggressive driver, starting at $30m/s$ with another vehicle going at $22m/s$ and starting at $51m$, the resulting model description is the one presented in Listing 3.1 (shortened for the sake of space saving; the full model is $13,435$ lines long).

```
//Model automatically built using model_generator.py for v1 = 22 and
    driver_type = 1 (to alter these values, run the script again).
//Generated on 31-07-2018 at 18:47.

dtmc

const int length = 500; // road length
const int driver_type = 1; // 1 = aggressive, 2 = average, 3 = cautious
    drivers - do not alter this manually!
const int max_time = 35; // maximum time of experiment

// Other vehicle
const int v1 = 22; // do not alter this manually!
const int x1_0 = 51;

```

```
// Environment variables
global t : [0..max_time] init 0; // time
global crashed : bool init false;

// Vehicle controlled
global actrState : [1..2] init 1; // active module: 1 = control (both cars
   ), 2 = decision making + monitoring
global lC : bool init false; // lane changing occuring?
global x : [0..length] init 0;
global v : [15..34] init 30;
global a : [-2..3] init 0;
global lane : [1..2] init 1;

formula x1 = x1_0 + v1*t;
formula dist = x1>x?(x1 - x):(x - x1);
formula positiveDist = (x < length)?x > x1:true;

module Decision_Making_Monitoring

  // If a crash occurs, then nothing else can happen
  //[] actrState = 2 & crashed -> 1:(crashed' = true);

  // If we are in lane 2, but behind the other vehicle, don't try to pass
  [] actrState = 2 & !crashed & lane = 2 & positiveDist = false -> 1:(
     actrState' = 1);

  // If we are in lane 1, and no vehicle is in front, don't change lanes
  [] actrState = 2 & !crashed & lane = 1 & positiveDist = true -> 1:(
     actrState' = 1);

  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = 1
      & v = 15 -> 0.8:(actrState' = 1) & (lC' = true) + 0.2:(actrState' =
      1) & (lC' = false);

  ...

  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >=
     80 -> 1:(actrState' = 1) & (lC' = true) + 0:(actrState' = 1) & (lC'
     = false);
endmodule

module Control

  // If we are in lane 1, and no lane change was decided, continue forward
      (which might result in crash)
  // The vehicle is behind the other driver (positiveDist = false, x < x1)
  [] actrState = 1 & !crashed & !lC & lane = 1 & x <= length - v & t < max
```

```
    _time & positiveDist = false & (x1 + v1 - x - v) >= 6 & v + a < 34 &
     v + a > 15 & dist = 1 & v = 15 -> 1:(x' = x + v) & (t' = t + 1) & (
    v' = v + a) & (a' = -2) & (actrState' = 2);

  ...

  [] actrState = 1 & !crashed & lC & lane = 2 & dist >= 43 & v = 34 & x >
    length - 136 & t > max_time - 6 -> 1:(crashed' = false) & (x' =
    length) & (v' = 34) & (t' = max_time) & (a' = 0) & (lane' = 1) & (
    actrState' = 2) & (lC' = false);

endmodule
```

Listing 3.1 Example of the model generated for the tuple $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 51)$ (shortened)

By loading and building the model in either PRISM or Storm, it is observable that it has 257 states and 295 transitions, a significant reduction from the possibilities presented in the straightforward control abstraction. Due to the fact that there are $3 \times 20 \times 20 \times length$ different possibilities in terms of initial conditions, it would be intractable, in the timeline of the dissertation, to obtain the number of states for all the combinations. Table 3.1 presents the number of states and transitions for some arbitrarily generated conditions (for a road of length $500m$).

d_{type}	v	v_1	$x_{1,0}$	**# States**	**# Transitions**
3	21	30	20	305	333
1	27	22	66	396	471
1	28	17	43	167	183
2	33	15	35	6	7
3	28	21	38	199	222
1	19	16	81	391	443
2	25	23	28	390	481
3	15	17	36	487	569
1	29	18	74	201	220
2	31	29	52	234	265

Table 3.1 Results of the number of states and transitions for arbitrary initial conditions.

3.3 Model Evaluation Metrics

With the model efficiently represented as a DTMC through abstraction, it becomes possible to perform model checking of certain properties on it in order to evaluate the relative performance of different profiles of drivers. In the following subsections, different categories of evaluation are presented and properties representing metrics within that category are proposed and tested in a small population of scenarios (the same test cases used to generate Table 3.1). In the rest of the dissertation it is assumed (unless explicitly mentioned otherwise) that the road segment has a length of $500m$.

3.3.1 Completeness Properties

From the rules used to build the model, it is explicit that a correct model should only enter a deadlock if it crashes or the vehicle reaches the end of road. As such, it is expected that every path from the initial state leads to states where the vehicle is in one of those two conditions. This property can be explicitly written as:

```
P>=1 [F (crashed | x=length)]
```

and it tests whether a model is complete by verifying that every possible outcome satisfies the restrictions imposed (regardless of the value of the other variables). Additionally, both PRISM and Storm allow the following property to be verified:

```
P>=1 [F "deadlock"]
```

which guarantees that a deadlock will eventually be reached and no infinite loop is generated by mistake within the DTMC.

Using PRISM or Storm, the results of the model checking of these properties in the test cases are presented in Table 3.2. As expected, for all the scenarios the properties are satisfied, confirming that these models are complete in terms of the expected outcome. While these properties do not need to be extensively analysed, they constitute a guarantee that the generated model is complete. As such, its verification is performed in all tests done in the analysis of Chapter 5.

d_{type}	v	v_1	$x_{1,0}$	`P>=1 [F (crashed | x=length)]`	`P>=1 [F "deadlock"]`
3	21	30	20	`true`	`true`
1	27	22	66	`true`	`true`
1	28	17	43	`true`	`true`
2	33	15	35	`true`	`true`
3	28	21	38	`true`	`true`
1	19	16	81	`true`	`true`
2	25	23	28	`true`	`true`
3	15	17	36	`true`	`true`
1	29	18	74	`true`	`true`
2	31	29	52	`true`	`true`

Table 3.2 Results of the verification of the completeness properties.

3.3.2 Safety Property

One essential evaluation metric of the human driver is its capability of driving safely, i.e. without crashing. To that effect, the following safety property is devised:

```
P=? [F crashed]
```

It should be noted that, for a given set of initial conditions, the lower the quantitative value of the property, the safer the driver is. From the assumptions made in the abstraction process, it is expected that Aggressive drivers will perform worse than Average ones in this metric, which in turn will perform worse than Cautious drivers. Extensive experimental results regarding this property are presented in Chapter 5. The results of the verification of the safety property in the test cases are presented in Table 3.3. It can observed that these vary quite significantly according to the situation, with the quantitative values ranging from 0.0193 (unlikely to crash) to 1 (will crash with certainty).

d_{type}	v	v_1	$x_{1,0}$	`P=? [F crashed]`
3	21	30	20	0.0232
1	27	22	66	0.3017
1	28	17	43	0.7119
2	33	15	35	1
3	28	21	38	0.1604
1	19	16	81	0.6074
2	25	23	28	0.0562
3	15	17	36	0.0276
1	29	18	74	0.5123
2	31	29	52	0.0193

Table 3.3 Results of the verification of the safety property.

3.3.3 Liveness Properties

Liveness can be defined in terms of the efficiency of the drivers, i.e. how quickly do they reach the end of the road. One of the variables of the model is the time, t, in seconds since the beginning of the execution. For a given constant T, the property:

```
P=? [F (x=length & t < T)]
```

captures the probability of reaching the end within less than T seconds. It should be noted that this constant T needs to be adjusted to different values in order to test specific properties, so this property should actually be defined as the list of properties:

```
P=? [F (x=length & t < 1)]
P=? [F (x=length & t < 2)]
...
P=? [F (x=length & t < max_time-1)]
P=? [F (x=length & t < max_time)]
```

However, the problem with these properties is based on the fact that they intrinsically rely on the safety property, since a higher probability of crashing naturally implies a lower probability of reaching the end, and, therefore, an even lower probability of reaching the end under T seconds. To mitigate this issue, the following conditional properties are introduced:

```
P=? [F (x=length & t<T) || F (x=length)]
```

which reads as *"what is the probability that the model eventually reaches a state where* `x = length` *and* `t < T` *given that it reaches one where* `x = length`*"*. This allows for direct comparisons between driver profiles and different scenarios. It should be noted that these are actually *max_time* different properties, as with the case of the unconditional ones.

Using Storm, both the unconditional and conditional properties can be verified directly. While PRISM supports the unconditional ones, conditional properties are not yet supported by the tool. However, it is possible to verify separately the properties `P=? [F (x=length & t<T)]` and `P=? [F (x=length)]`, and use Bayes' theorem to calculate:

$$\texttt{P=? [F (x=length \& t<T) || F (x=length)]} = \frac{\texttt{P=? [F (x=length \& t<T)]}}{\texttt{P=? [F (x=length)]}} \tag{3.13}$$

Table 3.4 presents the results of the verification of the liveness properties in the test cases. In the interest of brevity, only two of each (unconditional and conditional) properties verified are presented, for $T = 19$ and $T = 24$ (arbitrarily selected), and they will be identified using the following number system (for presentation purposes):

```
1. P=? [F (x=length & t<19)]
2. P=? [F (x=length & t<24)]
3. P=? [F (x=length & t<19) || F (x=length)]
4. P=? [F (x=length & t<24) || F (x=length)]
```

From Table 3.4, it is possible to observe that different scenarios, despite having distinct values of the unconditional properties, have a similar value of the conditional ones. For example, for $T < 24$, the scenario $(d_{type}, v, v_1, x_{1,0}) = (1, 28, 17, 43)$ has a a significantly lower unconditional probability than the $(d_{type}, v, v_1, x_{1,0}) = (3, 28, 21, 38)$, yet they have exactly the same conditional one. While the unconditional properties can be interpreted in these scenarios as *"the vehicle will reach the end of the road and be under 24s with probability 0.2881"* and *"the vehicle will reach*

d_{type}	v	v_1	$x_{1,0}$	`1`	`2`	`3`	`4`
3	21	30	20	0.0085	1	0.0085	1
1	27	22	66	0.5908	0.6983	0.8460	0.9999
1	28	17	43	0	0.2881	0	1
2	33	15	35	0	0	0	0
3	28	21	38	0	0.8396	0	1
1	19	16	81	0	0.3926	0	0.9999
2	25	23	28	0.1555	0.9438	0.1648	0.9999
3	15	17	36	0	0.5849	0	0.6015
1	29	18	74	0.4662	0.4877	0.9559	1
2	31	29	52	0.9992	0.9992	1	1

Table 3.4 Results of the verification of the liveness properties.

the end of the road and be under 24s with probability 0.8396", in both these cases the conditional probability should be interpreted as *"if the vehicle reaches the end of the road, then it will do so before 24s with certainty"*. The conditional property can be understood as an elimination of the safety bias inherent to each scenario, leading to more meaningful inter-situational comparisons. Extensive experimental results regarding these properties are presented in Chapter 5

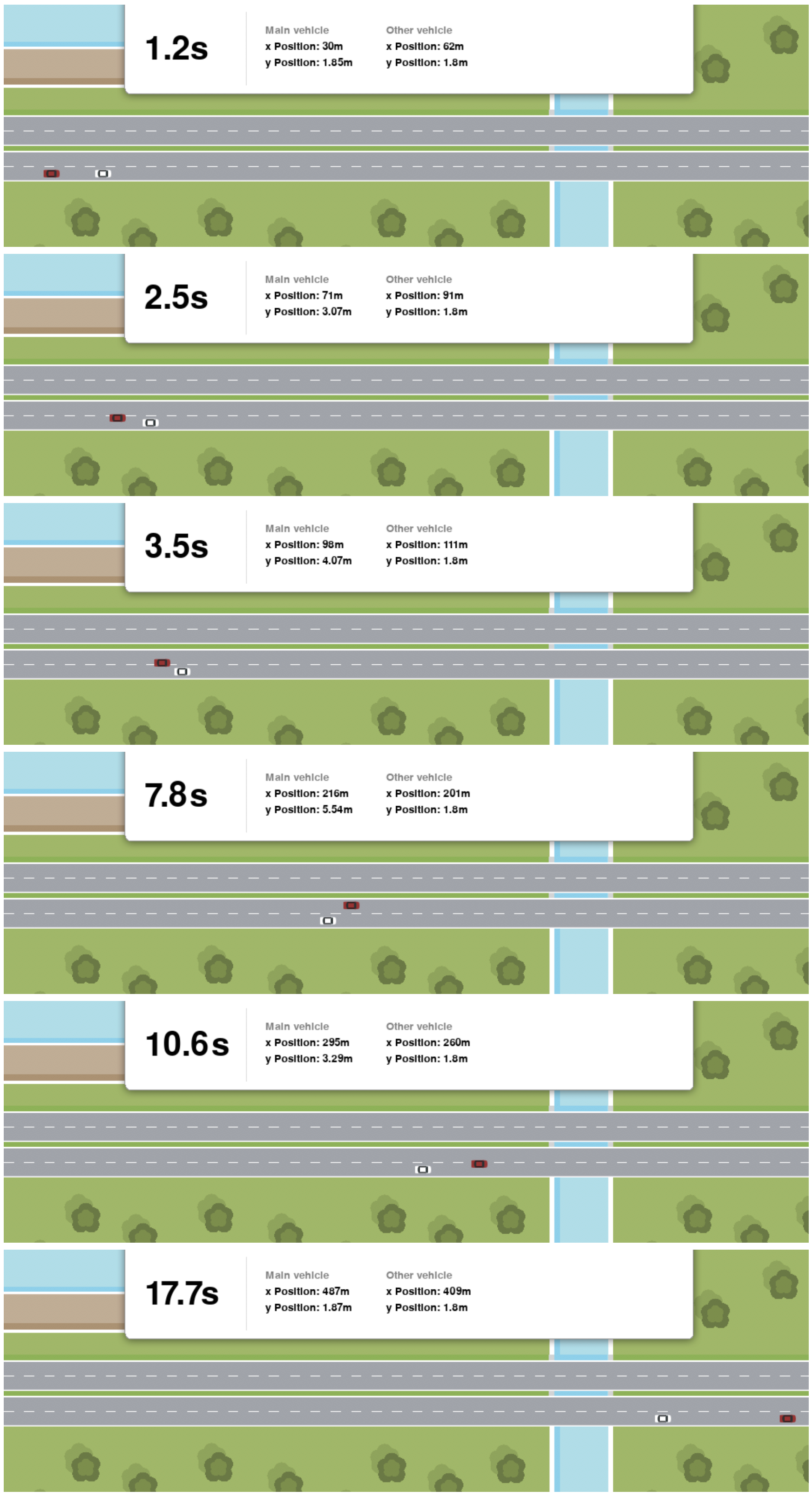
3.4 Simulation of Paths in the Model

In order to visualise possible executions of the trajectory and decisions taken in the model, a visual simulator was designed and implemented in *Python*, with the code presented in Appendix A.7.

The simulator essentially uses the option `simpath` in the PRISM command line tool which allows it to output a simulated path in the model without having to actually build it. Using this outputted path in the model, the script reads it and plays it back to the user using a GUI built in *pygame* for *Python* [48].

In order to faithfully represent the lane change operations the driver performs, an additional table is obtained using the steering control abstraction, which corresponds to the interpolation of the x and y positions of the vehicle as a function of time (i.e. $x(t)$ and $y(t)$). Given the complexity of the y movement compared to the x movement, the positions are represented by a 6^{th} and 2^{nd} degree polynomial, respectively.

Due to the size of the example, only one test case is presented of a simulation of a path in a model. In Figure 3.7, an example of several frames of the simulation of a path in the scenario $(d_{type}, v, v_1, x_{1,0}) = (3, 28, 21, 38)$ is presented. In this case, the driver initiates a lane change to perform an overtake at the 2s mark and returns to the original lane at the 8s mark, with no crashing occurring. In the simulation, the vehicle took $18.3s$ to reach the end of the road. It should be noted that, as verified by the conditional liveness property, the vehicle took less than $24s$ to reach the end of the road segment. While it would appear the vehicle took less than $19s$ to reach the end (and thus would violate the probability of 0 obtained for this liveness property), the model considers steps of $1s$, so in the model, the vehicle actually took $19s$ to finish instead of $18.3s$.


1.2s

Main vehicle
x Position: 30m
y Position: 1.85m

Other vehicle
x Position: 62m
y Position: 1.8m

2.5s

Main vehicle
x Position: 71m
y Position: 3.07m

Other vehicle
x Position: 91m
y Position: 1.8m

3.5s

Main vehicle
x Position: 98m
y Position: 4.07m

Other vehicle
x Position: 111m
y Position: 1.8m

7.8s

Main vehicle
x Position: 216m
y Position: 5.54m

Other vehicle
x Position: 201m
y Position: 1.8m

10.6s

Main vehicle
x Position: 295m
y Position: 3.29m

Other vehicle
x Position: 260m
y Position: 1.8m

17.7s

Main vehicle
x Position: 487m
y Position: 1.87m

Other vehicle
x Position: 409m
y Position: 1.8m

Figure 3.7 Snapshots of the simulator for one of the paths in the scenario $(d_{type}, v, v_1, x_{1,0}) = (3, 28, 21, 38)$.

Chapter 4

Advanced Driver Assistance Systems

This chapter explores and compares several ideas for assistance systems which can be implemented on top of the human driver model generated in the previous one. It starts by tackling the design of the system in terms of the placement in the existing module based model and the capabilities within it, following an incremental approach (considering the limitations of the vehicle and drivers). It then performs a comparison of the different approaches using two distinct test cases and it concludes which ADAS performs the best (within reasonable assumptions). This solution is then evaluated and compared to the human driver in Chapter 5.

4.1 Driver Assistance System Design

One of the aims of this dissertation is to obtain correct-by-construction driver assistance systems. As such, the design of the assistance system in this context corresponds mostly to determining which actions are available to the system at each state (the model becomes an MDP), and then performing synthesis using adequate properties. It should be noted that these actions must be realistic in nature, otherwise the obtained assistance system would prove to be useless in a real-world scenario - and usefulness is the end goal in terms of deployment. Figure 4.1 presents the underlying assumptions of where the system would lie in the environment, which influences the possibilities in terms of the design.

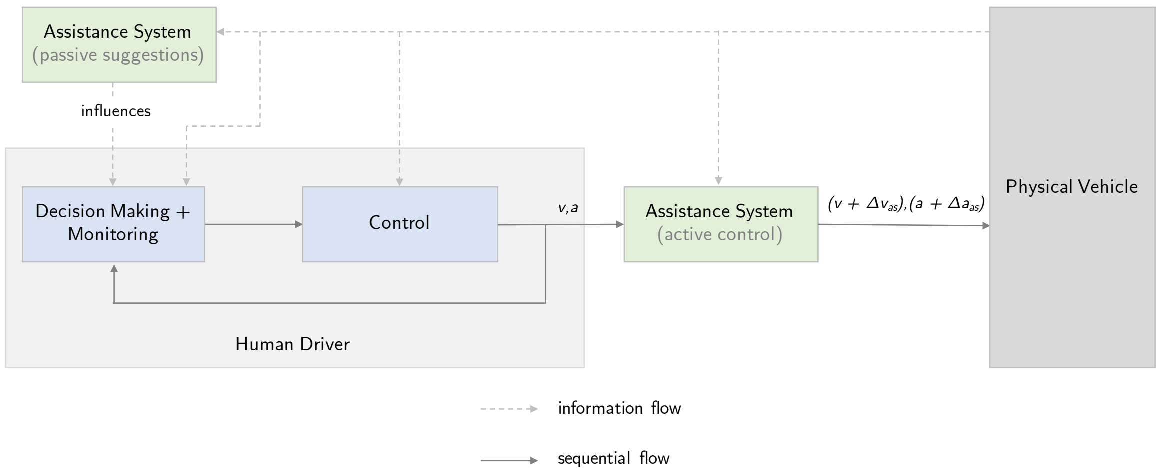


Figure 4.1 Overview of the system with the possibilities of the ADAS intervention.

It is assumed that the assistance system can not change the decision making (as it is a human cognitive process), but it can influence it to a certain degree through suggestion. This would be the ideal way of implementing the ADAS, as it would not require any intervention in the physical systems of the vehicle. Therefore, the first possibilities explored are based on this option. However, in the interest of safety and efficiency, incremental control based options are also developed, both at the level of linear acceleration (influencing a through Δa_{as}), as well as at the level of steering control (by influencing v and a through Δv_{as} and Δa_{as}, respectively).

4.1.1 Decision Making-based ADAS

4.1.1.1 Decision Making with Fully Compliant Drivers

At the decision making level, the human driver model considered has two options: it either changes lanes, or it does not. These options can be influenced using suggestions (e.g. through visual or auditive cues) which can lead to either one or the other being taken. Salvucci in [26] does not consider accelerating and decelerating as part of the decision making because he argues they happen instinctively and therefore they should be exclusively part of the control. However, if a driver can be influenced to make a conscious decision to decelerate, for example, through the driver assistance system, then there is an argument for including this action in the decision making. Thus, a 3-option ADAS was designed, where an action α is defined as:

$$\alpha \in A := \{(lc) \vee (\neg lc \wedge a = a_p) \vee (\neg lc \wedge a = a_d)\} \tag{4.1}$$

where lc corresponds to the decision to perform a lane change, a is the acceleration in the next state, a_p is the acceleration the vehicle is currently holding and a_d is a constant value for deceleration that a driver applies when suggested to decelerate. In this dissertation, it was assumed that $a_d = -1$ (minimum decelerating value).

Considering that the drivers are fully compliant with the suggestions given by the system, the decision making at each step can be replaced by the all the possible actions, obtaining an MDP with three choices at this level. The code presented in Appendix B.1.1 corresponds to the implementation of a generator for this ADAS.

4.1.1.2 Decision Making with Partially Compliant Drivers

In the case of the ADAS previously designed, it is assumed that humans will not only follow all the suggestions, but they will comply with them fully and immediately after they are received (within the same ACT-R cycle in the decision making). However, this is not a realistic assumption by any means. In fact, people are more prone to following suggestions when these align with their original intent, than otherwise. With this in mind, a new solution is presented which more accurately represents the suggestive decision making considered in this section.

In the human driver model, a single action was available for a set of conditions, such that at any decision making state ($actr\_state = 2$) s there would be a value p such that the next state s' could be written as:

$$s' = p : (lC) + (1 - p) : (\neg lC \,\&\, a = a_p) \tag{4.2}$$

Consider a factor $\gamma \in [0, 1]$ which corresponds to how responsive a driver is to the suggestions given. The following rule can be written for each of the actions $\alpha_i \in A$ of the MDP according to the states s'_i which they lead to:

$$s'_i = \gamma : \alpha_i + (1 - \gamma) \cdot p : (lC) + (1 - \gamma) \cdot (1 - p) : (\neg lC \,\&\, a = a_p) \tag{4.3}$$

Given that $\gamma, p \in [0, 1]$, the transitions are guaranteed to sum up to one for every case. The decision making with fully compliant drivers corresponds to the case where $\gamma = 1$. The code presented in Appendix B.1.2 corresponds to the implementation of a generator for this particular ADAS.

4.1.2 Control-based ADAS

While the decision making under the assumption of fully compliant drivers appears to be a powerful option in terms of safety and liveness, the weakening of the assumption to partially compliant drivers is expected to reduce performance substantially, particularly for lower values of γ. As such, control based assistance is added on top of the decision making assistance considered for partially compliant drivers, so as to improve performance. It should be noted that this type of assistance does not require human intervention, as per the assumptions noted in Figure 4.1.

4.1.2.1 Active Linear Acceleration Control

Considering the assumptions previously described, active linear acceleration control consists in an incremental addition to the acceleration value proposed by the human in the control module. Assume the acceleration of the vehicle imposed by the human is given in the model by $a \in \{a^{\min}, ..., a^{\max}\}$. In this module, a value Δa_{as} is considered such that:

$$\Delta a_{as} \in \{\Delta a_{as}^{\min}, ..., \Delta a_{as}^{\min}\} : \Delta a_{as}^{\min} > a^{\min} \wedge \Delta a_{as}^{\max} < a^{\max} \tag{4.4}$$

and the final acceleration applied to the vehicle becomes (considering as well that $a' \in \{a^{\min}, ..., a^{\max}\}$):

$$a'(t) = \max(\min(a(t) + \Delta a_{as}, a^{\max}), a^{\min}) \tag{4.5}$$

The restriction to the values of Δa_{as} presented in Equation 4.4 allows the system to be incremental (i.e. corrective) instead of enforcing the specific values chosen by the control assistance system. This is important to avoid strategies for the control assistance system which sharply contrast in terms of the values chosen, e.g. a strategy which at a certain time chooses an acceleration of 3 and in the next time step chooses one of -2 (not allowed for a small enough range of Δa_{as} and according to the linear control abstraction obtained in Chapter 3). In this dissertation, it is assumed that $\Delta a_{as} \in \{-1, 0, 1\}$.

The implementation of such system consists in replacing the existing linear control at each step of the control module by the resulting accelerations of applying all the possible values of Δa_{as} to the acceleration decided by the human control module, obtaining an MDP with at most three actions at the linear acceleration control level (and at least two). The code presented in Appendix B.1.3 corresponds to the implementation of a generator for this ADAS (with the decision making assistance for partially compliant drivers).

4.1.2.2 Active Steering Control

While linear acceleration control assistance improves safety and liveness, steering assistance can also be improved using incremental velocity and acceleration.

In [26], Salvucci introduces a control law for the steering angle φ, as presented in Section 2.2, for a given k_{far}, k_{near} and k_I. By changing the values of these constants, different control laws are obtained, which introduce different accelerations and velocities at each time step, mimicking the behaviour of the incremental control previously assumed (i.e. within such a strategy, the difference in acceleration and velocity introduced is the difference between these values for the two control laws at each time step). The value of $\theta_{\max}$ (as defined in Section 2.2), guarantees the feasibility of the movement in terms of the acceleration induced. Thus, actions in this assistance system at the model level correspond to different sets of (k_{far}, k_{near}, k_I) available to the ADAS.

In this dissertation, three distinct sets of parameters were considered as possible actions, $(k_{far}, k_{near}, k_I) \in \{(15, 3, 5), (17, 3, 6), (14.5, 3, 7)\}$. An example of the simulation for the situation where $o_{lane} = 1$, $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$ (with the non-probabilistic control) is presented in Figure 4.2.

Using these values, new look-up tables were obtained with the probability of crashing, the Δx and ΔT incurred and the final velocity of the vehicle in question (following the probabilistic control module presented in Section 3.2.3) for each origin lane, distance

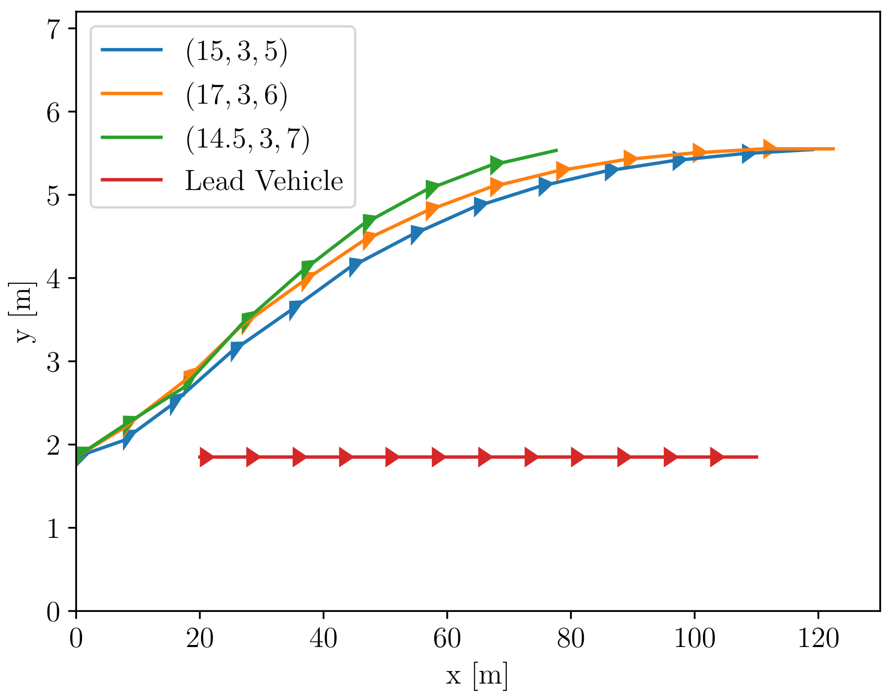


Figure 4.2 Example of the simulation of the lane change for $o_{lane} = 1$ (right lane), $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$ (the legend of each path corresponds to the situation with parameters k_{far}, k_{near}, k_I, respectively).

to the other vehicle, velocities of both the vehicle in question and the other vehicle and the parameters of the control law (out of the three possibilities presented). Following the same calculations as shown in Section 3.2.1, the obtained look-up table contains $103, 200$ rows.

In terms of implementation, the MDP is obtained through simply reading the three possible actions for lane changing directly from the look-up table, similarly to the human driver model (the difference being the latter only has one option). The code presented in Appendix B.1.4 corresponds to the implementation of a generator for this ADAS (with the decision making assistance for partially compliant drivers and active linear acceleration control).

4.2 Design Evaluation

While it would appear to be the case that an ADAS with both decision making and control assistance at the linear acceleration and steering level would be the optimal choice for the driver assistance system, there are no guarantees that this is true. As such, an evaluation and comparison of all the possible designs must be performed in order to determine the best option which can then be compared to the human driver model in Chapter 5. To do so, it is necessary to establish multi-objective metrics and some meaningful test cases (as testing all the options would be infeasible).

4.2.1 Multi-Objective Metrics

As established in Section 3.3, safety and liveness are two important metrics which will be used in Chapter 5 to evaluate the human driver model. Therefore, it makes sense that such properties should be the focus of the synthesis for the driver assistance system. The goal is to minimise the probability of crashing and to maximise the liveness property. This can be obtained using a multi-objective query such as the one below:

```
multi(Pmin=? [F crashed], Pmax=? [F (x=length) & (t<T)])
```

for a given constant T. Due to the trade-offs between both properties, the model checking of the multi-objective property of the format of the above generates a Pareto curve, where each point on the curve corresponds to optimal achievable values in terms of `P=? [F crashed]` and `P=? [F (x=500) & (t<T)]` under a given strategy. As such, it is possible to compare strategies through the comparison of the Pareto curves obtained for the different ADAS for some test cases.

However, the problem of the safety bias discussed in Section 3.3.3 persists through this comparison, and the ideal metric would take into account the conditional probability instead of the unconditional one. Since PRISM does not allow for the model checking of conditional properties, the Pareto curve using the conditional properties needs to be obtained in a different way. At this point, it is important to notice the following equality for a given strategy σ in the MDP:

$$P_s^\sigma(\texttt{F crashed}) = 1 - P_s^\sigma(\texttt{F x=length}) \tag{4.6}$$

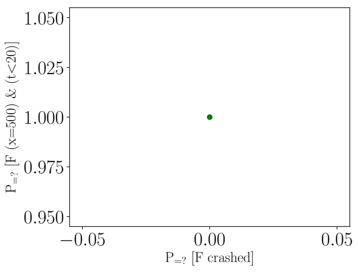
as long as $p_{\max,s}(\texttt{F (crashed | x = length)}) = p_{\min,s}(\texttt{F (crashed | x = length)}) = 1$ (completeness property for the MDP). This is true from the fact that the events crashing and reaching the end are mutually exclusive. Thus, it is possible to write:

$$\begin{aligned} P_s^\sigma(\texttt{F (x=length) \& (t<T)} \mid \texttt{F x=length}) &= \frac{P_s^\sigma(\texttt{F (x=length) \& (t<T))}}{P_s^\sigma(\texttt{F x=length})} \\ &= \frac{P_s^\sigma(\texttt{F (x=length) \& (t<T))}}{1 - P_s^\sigma(\texttt{F crashed})} \end{aligned} \tag{4.7}$$

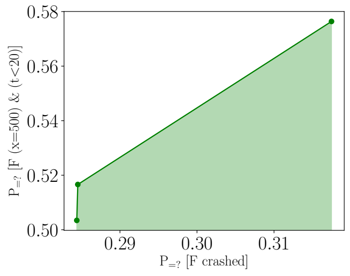
It is therefore possible to obtain the Pareto curve with the conditional properties by using Equation 4.7 on each of the points in the unconditional curve (generating the achievable set through this).

4.2.2 Test Cases

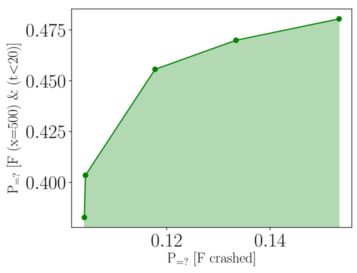
Due to the infeasibility of generating all the possible combinations of test cases, two test cases were considered when comparing the four different ADAS designed in the previous section: the first test case in the scenario $(v, v_1, x_{1,0}) = (21, 19, 70)$ and the second in the scenario $(v, v_1, x_{1,0}) = (30, 22, 50)$. For each of these test cases, the unconditional and conditional Pareto curves for each driver type are presented for a T representative of the scenario and which allows for comparison between the ADAS considered, with Figures 4.3 and 4.4 referring to the first test case, and Figures 4.5 and 4.6 referring to the second test case. In terms of parameters of the model, the same values were used as in the human driver model part of the implementation, and γ was set to 0.1.



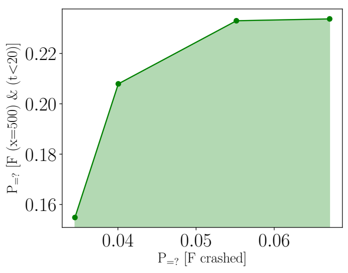
a. (1)



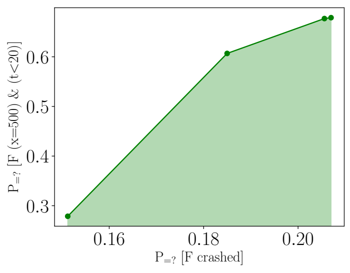
b. (2) - Aggressive drivers



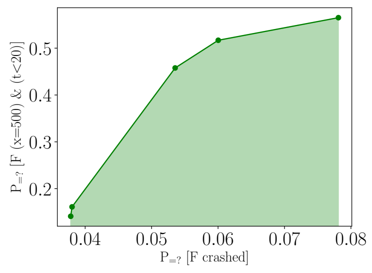
c. (2) - Average drivers



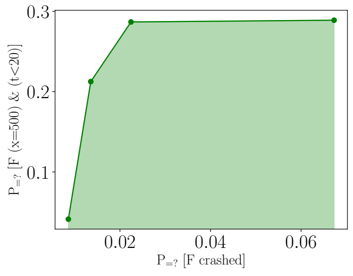
d. (2) - Cautious drivers



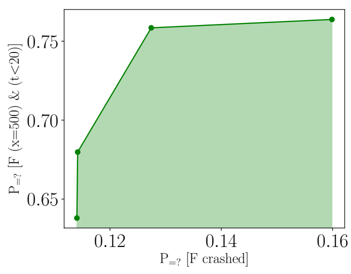
e. (3) - Aggressive drivers



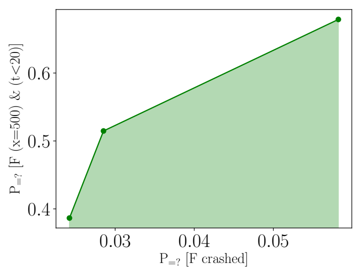
f. (3) - Average drivers



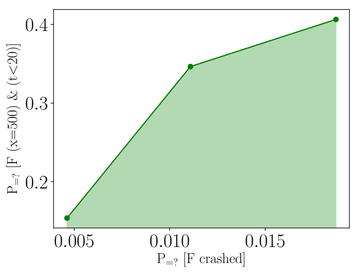
g. (3) - Cautious drivers



h. (4) - Aggressive drivers

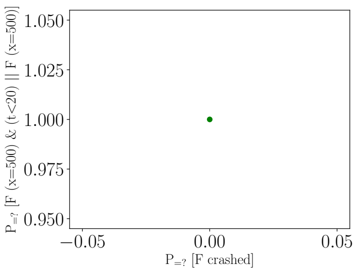


i. (4) - Average drivers

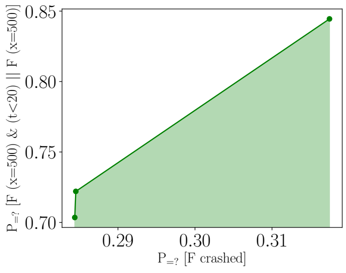


j. (4) - Cautious drivers

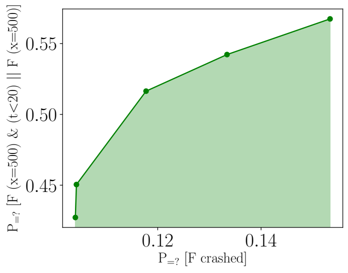
Figure 4.3 Pareto curves of the unconditional properties for the scenario $v = 21m/s$, $v_1 = 19m/s$, $x_{1,0} = 70m$ and for $T = 20$. The ADAS are represented using the following numbering: (1) decision making-based with fully compliant drivers, (2) decision making-based with partially compliant drivers, (3) decision making with partially compliant drivers + linear acceleration control, and (4) decision making with partially compliant drivers + linear acceleration control + steering control.



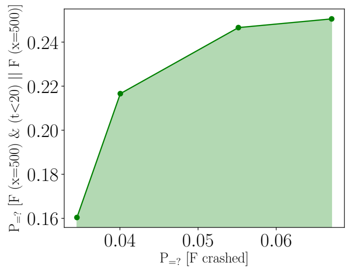
a. (1)



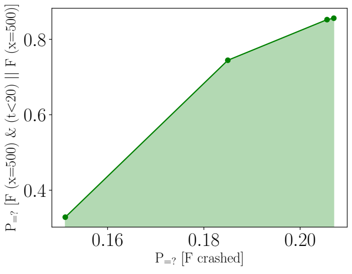
b. (2) - Aggressive drivers



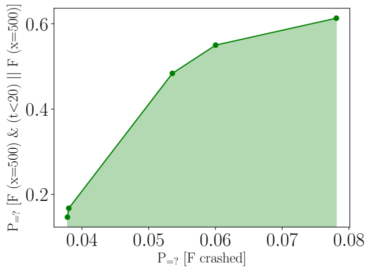
c. (2) - Average drivers



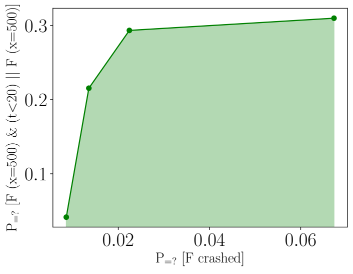
d. (2) - Cautious drivers



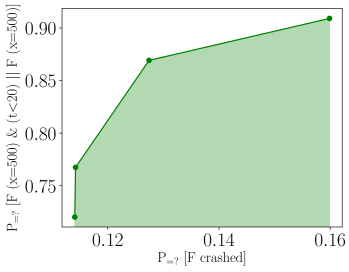
e. (3) - Aggressive drivers



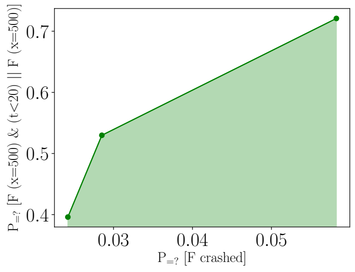
f. (3) - Average drivers



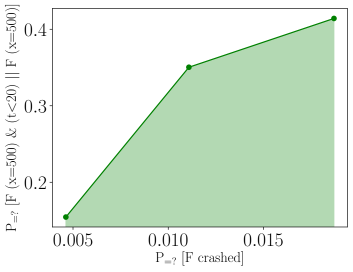
g. (3) - Cautious drivers



h. (4) - Aggressive drivers

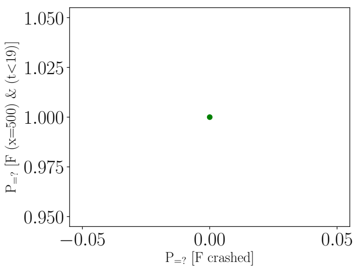


i. (4) - Average drivers

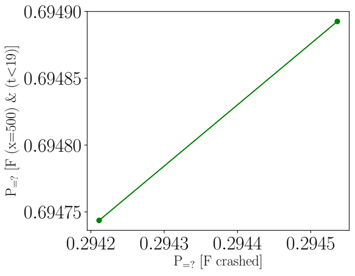


j. (4) - Cautious drivers

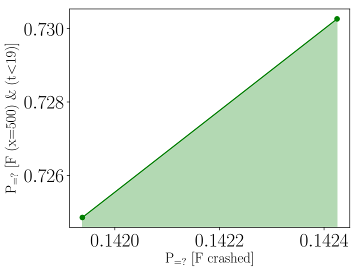
Figure 4.4 Pareto curves of the conditional properties for the scenario $v = 21m/s$, $v_1 = 19m/s$, $x_{1,0} = 70m$ and for $T = 20$. The ADAS are represented using the following numbering: (1) decision making-based with fully compliant drivers, (2) decision making-based with partially compliant drivers, (3) decision making with partially compliant drivers + linear acceleration control, and (4) decision making with partially compliant drivers + linear acceleration control + steering control.



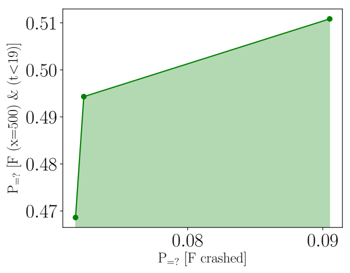
a. (1)



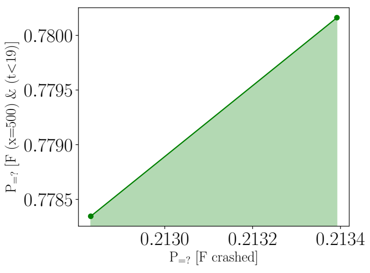
b. (2) - Aggressive drivers



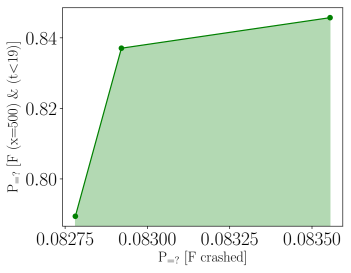
c. (2) - Average drivers



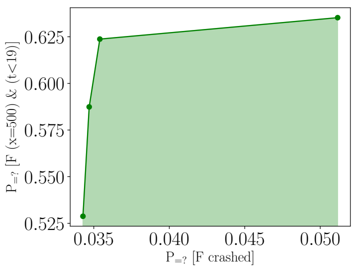
d. (2) - Cautious drivers



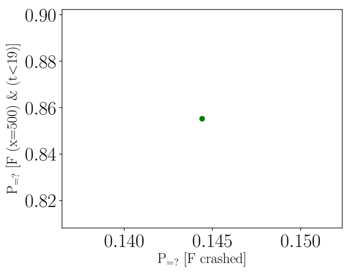
e. (3) - Aggressive drivers



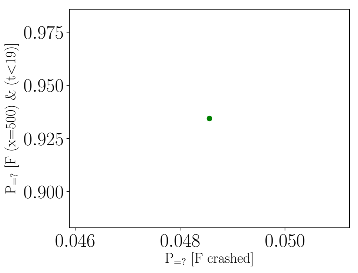
f. (3) - Average drivers



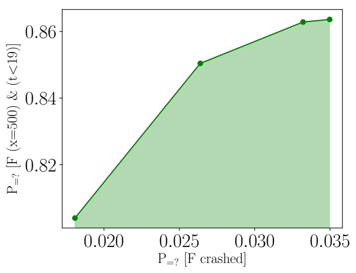
g. (3) - Cautious drivers



h. (4) - Aggressive drivers

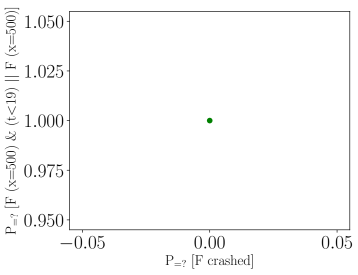


i. (4) - Average drivers

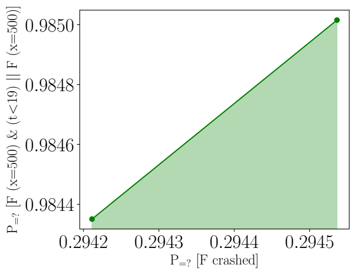


j. (4) - Cautious drivers

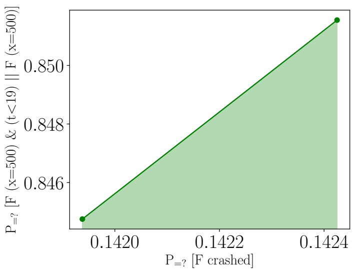
Figure 4.5 Pareto curves of the unconditional properties for the scenario $v = 30m/s$, $v_1 = 22m/s$, $x_{1,0} = 50m$ and for $T = 19$. The ADAS are represented using the following numbering: (1) decision making-based with fully compliant drivers, (2) decision making-based with partially compliant drivers, (3) decision making with partially compliant drivers + linear acceleration control, and (4) decision making with partially compliant drivers + linear acceleration control + steering control.



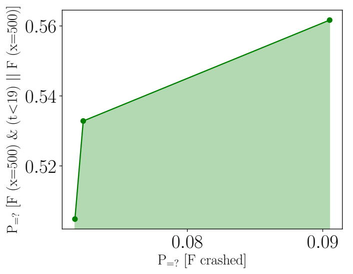
a. (1)



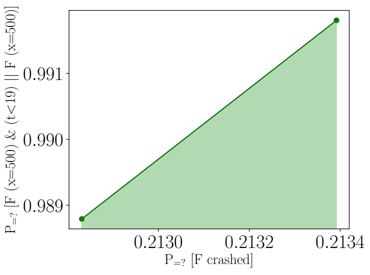
b. (2) - Aggressive drivers



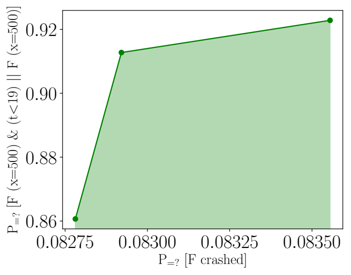
c. (2) - Average drivers



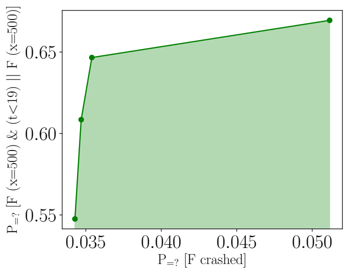
d. (2) - Cautious drivers



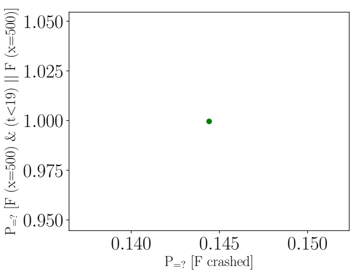
e. (3) - Aggressive drivers



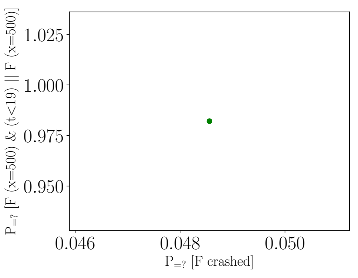
f. (3) - Average drivers



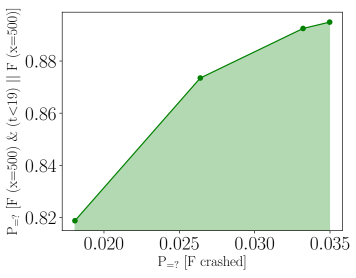
g. (3) - Cautious drivers



h. (4) - Aggressive drivers



i. (4) - Average drivers



j. (4) - Cautious drivers

Figure 4.6 Pareto curves of the conditional properties for the scenario $v = 30m/s$, $v_1 = 22m/s$, $x_{1,0} = 50m$ and for $T = 19$. The ADAS are represented using the following numbering: (1) decision making-based with fully compliant drivers, (2) decision making-based with partially compliant drivers, (3) decision making with partially compliant drivers + linear acceleration control, and (4) decision making with partially compliant drivers + linear acceleration control + steering control.

4.2.3 Overall Comparison

From the Pareto curves presented, it is noticeable that the best performing ADAS is the first presented, that is, a decision making assistance system with the assumption of fully compliant drivers. This assistance system allows maximum safety (`P=? [F crashed]` $= 0$) and efficiency (`P=? [F (x=500) & (t<T)]` $= 1$) for both test cases and respective values of T considered. However, as discussed before, this situation is unrealistic in nature. By adding the partial compliance assumption, the performance drops drastically, leading to fairly high probabilities of crashing in aggressive drivers (minimum of 0.285 for the first test case and 0.294 for the second one) and lower liveness performance as well. By introducing linear acceleration control, safety increases significantly (the probability of crashing is lowered), but the increase of the probability of the liveness property being satisfied is almost negligible in some situations (and it decreases in some cases; e.g. the aggressive drivers in the first test case). In all the cases tested, the introduction of steering control improves, in both cases and for all driver classes, both safety and liveness. Therefore, the assistance system number 4, that is, decision making with partially compliant drivers, active linear acceleration control and active steering control is the best performing system in the two test cases presented (other than number 1, which is based on unrealistic assumptions).

This analysis does not provide a statistical guarantee that this is the best performing ADAS out of the ones tested in the majority of the situations (as this would be too time consuming for the time frame of this dissertation). However, the systems considered are incremental in nature, in the sense that they were built through iteration and by adding more actions to the one immediately before. As such, number 4 was expected for perform better than numbers 2 and 3, simply due to the fact that there were more choices available to it. The two test cases presented corroborate this idea. Therefore, it can be concluded that the assistance system number 4 would be the most complete and better candidate for deployment, and, as such, the in-depth experimental results and discussion presented in Chapter 5 use this ADAS.

Chapter 5

Experimental Results

In this chapter, the performance of the driver assistance system is evaluated by comparing the value of quantitative properties in the human driver model obtained in Chapter 3 to similar properties in the model of the human driver with the ADAS designed in Chapter 4 (decision making with partially compliant drivers and with active linear acceleration and steering control). This is achieved using the various metrics defined throughout the dissertation, through random test cases of initial conditions and further comparison using a randomly generated sample population. After a discussion of the main results, some demonstrations of possible extensions regarding different properties outside the scope of the performance evaluation and scenarios with more vehicles are presented.

5.1 Performance Evaluation of ADAS

The first step towards the comparison between the human driver model and the full system (human driver with the ADAS designed) lies in the building of the model and, as such, it is only natural that the evaluation should start by comparing the state space and building time of each of the two models.

5.1.1 State Space and Building Time Results

A scenario, in both cases, can be uniquely defined as a tuple $(d_{type}, v, v_1, x_{1,0})$, where $d_{type} \in \{1, 2, 3\}$ is the driver class considered (aggressive, average or cautious, respectively). In practice, this means that, for a road segment of $500m$ and considering $v, v_1 \in \{15, ..., 34\}$ and $x_{1,0} \in \{1, ..., 500\}$, there are $20 \times 20 \times 500 \times 3 = 600,000$ different scenarios to consider. Assuming that the building time of each of those scenarios for both the human driver model and the full system is 5 minutes in total, this task of building all the possible scenarios would take approximately $2,083$ days to complete. Given the time frame of this dissertation, this would be infeasible. As such, a sample population of 10 random scenarios was generated and the results in terms of the number of states (#S), the number of transitions (#T) of the built model and time, in seconds, (t [s]) it took to build the model[1] are presented in Table 5.1.

d_{type}	v	v_1	$x_{1,0}$	**Human**			**Human and ADAS**		
				#S	**#T**	**t [s]**	**#S**	**#T**	**t [s]**
1	34	18	78	106	119	2.479	12,213	26,743	309.949
1	31	31	52	227	251	4.064	103,531	360,505	269.900
2	22	18	64	358	399	5.327	401,632	1,231,288	282.307
3	28	25	58	334	399	4.695	209,143	710,168	277.272
1	24	24	84	278	316	4.264	320,088	1,055,570	281.231
3	21	19	68	364	426	5.262	448,145	1,414,788	290.160
2	24	17	52	215	238	3.623	225,095	650,096	297.895
3	29	29	66	212	237	4.025	153,464	530,137	279.170
2	25	17	44	153	168	3.276	63,403	156,201	302.744
2	31	20	40	85	94	2.559	22,135	50,746	267.165

Table 5.1 Result of the state space and build time for 10 test cases.

As expected, the state space is considerably larger in the full system than in the human driver model alone. However, the time for model construction in both cases is larger than expected for models of such dimensions, particularly when compared to other similar results in [25]. For a similarly obtained randomly generated sample of 100 test cases, the average build time for the human model was 3.861s and in the case of the full system this value was 283.256s.

[1]The models were built using a Macbook Pro with a 2,6 GHz Intel Core i7 (quad core). The human model results were obtained using Storm's sparse engine, while the full system results were obtained using PRISM's hybrid engine.

5.1.2 Safety Results

As initially described in Chapter 3, the safety property for the human driver model can be written as:

```
P=? [F crashed]
```

A similar safety property is presented in Chapter 4, adapted for the fact that the model of the human driver and driver assistance system is an MDP instead of a DTMC:

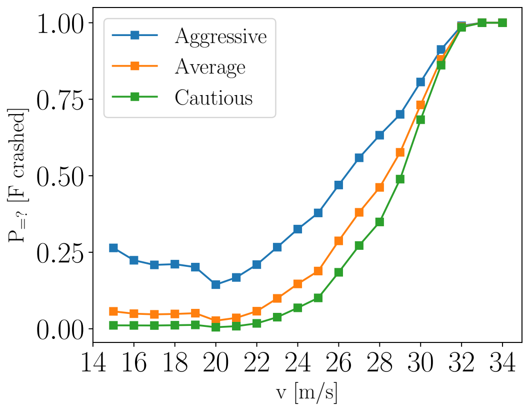
```
Pmin=? [F crashed]
```

These properties are directly comparable, as they represent the same quantitative value (this is not the case with the liveness properties). It should be noted that the lower the value of this property, the safer the system is. This means that these properties can be used not only for comparisons between the human driver alone and the full system (human and driver assistance system) for a given scenario, but also for inter-situational comparisons (i.e. with different initial conditions).

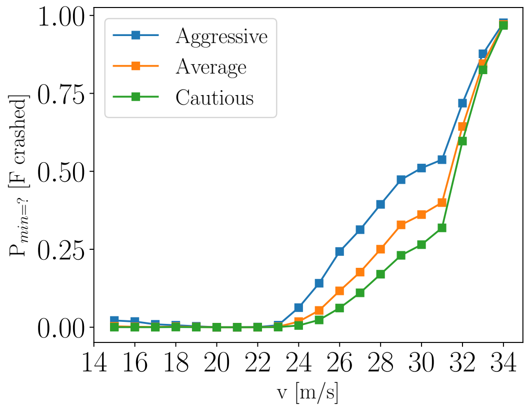
As explained in Section 5.1.1, it would be infeasible to obtain the results of model checking the safety property in all the possible scenarios. As such, three other methods where designed as a way to compare both systems:

- **Main vehicle initial velocity test cases**: for two randomly selected tuples $(v_1, x_{1,0})$, the variation of the value of the safety properties for each driver profile with the variation of v were obtained. This directly compares 20 different scenarios for each driver class. The two tuples were randomly generated as $(v_1, x_{1,0}) = (20, 35)$ and $(v_1, x_{1,0}) = (22, 40)$, and the results are presented in Figures 5.1 and 5.2, respectively.
- **Bivariant initial velocity test cases**: for a randomly selected $x_{1,0}$, the variation of the values of the safety properties for each driver profile with the change of v and v_1 (within certain ranges; $v \in \{20, ..., 30\}$ and $v_1 \in \{15, ..., 25\}$) were generated. This directly compares 120 different scenarios for each driver class. The value $x_{1,0} = 50$ was randomly generated for this test case, and the results are presented in Figure 5.3.

- **Box plots**: for a randomly generated sample of 100 different initial conditions of $(v, v_1, x_{1,0})$, a box plot which represents the distribution of the values of the safety properties for this sample and for each driver profile was obtained. The results are shown in Figure 5.4.

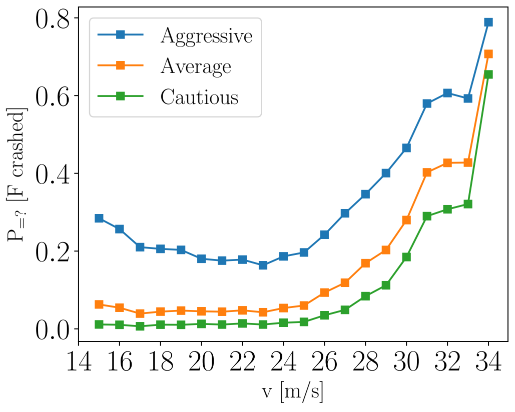


a. Human driver model

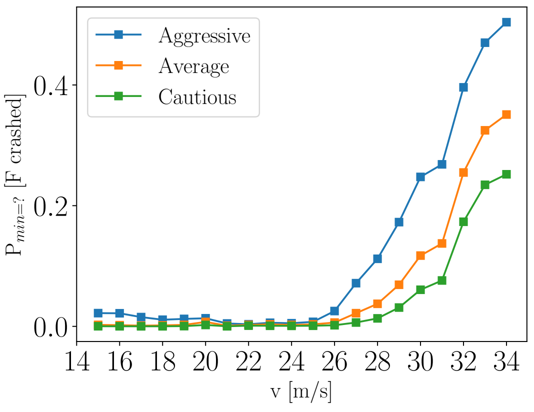


b. ADAS

Figure 5.1 Plots of the variation of the value of the safety property with the initial velocity of the main vehicle for the conditions $v_1 = 20m/s$ and $x_{1,0} = 35m$.

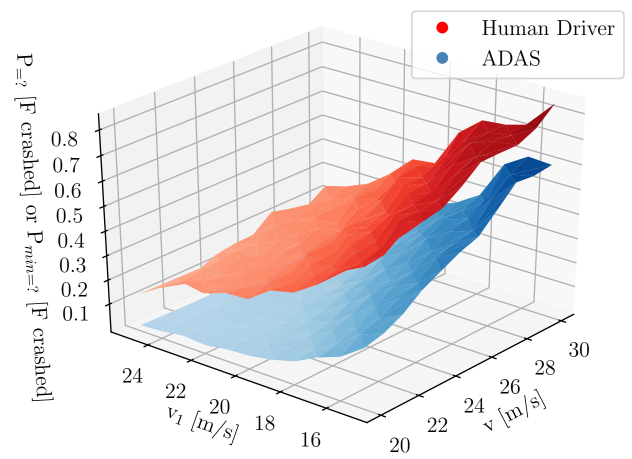


a. Human driver model

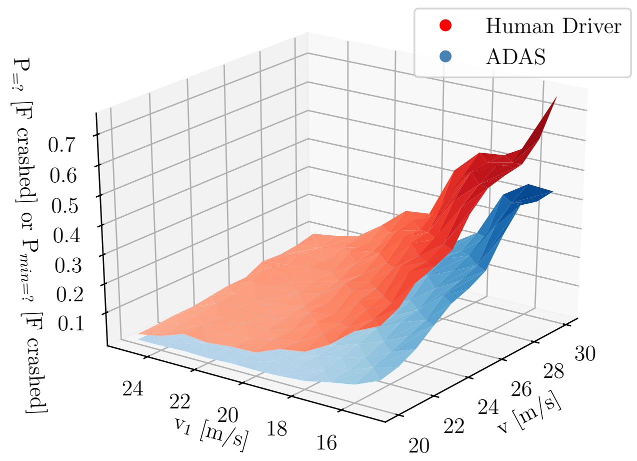


b. ADAS

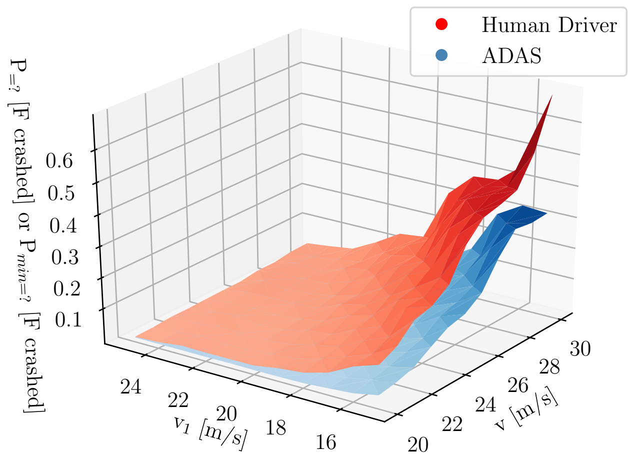
Figure 5.2 Plots of the variation of the value of the safety property with the initial velocity of the main vehicle for the conditions $v_1 = 22m/s$ and $x_{1,0} = 40m$.



a. Aggressive drivers

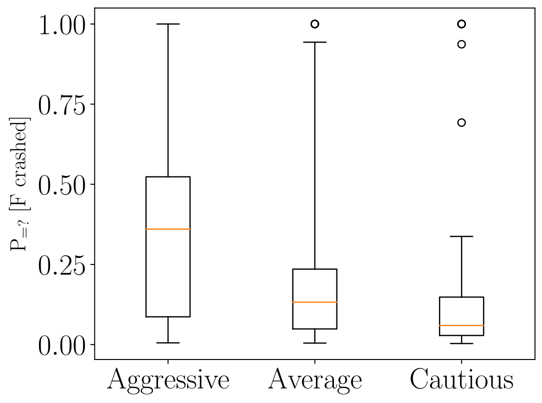


b. Average drivers

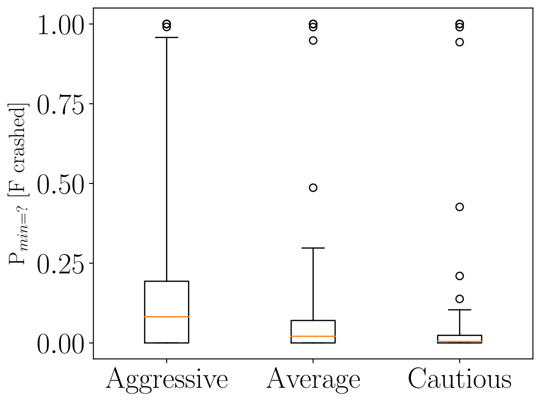


c. Cautious drivers

Figure 5.3 Variation of the value of the safety properties with the initial velocities of both vehicles for $x_{1,0} = 50m$.



a. Human driver model



b. ADAS

Figure 5.4 Box plot of the value of the safety properties for a randomly sampled population of 100 different initial conditions (equal in both cases).

5.1.3 Liveness Results

As introduced in Chapter 3, there are multiple liveness properties that can be considered when evaluating the human driver model. However, the most appropriate ones for inter-situational comparisons are the conditional properties of the type:

```
P=? [F (x=length) & (t<T) || F (x=length)]
```

for a given constant T.

In the case of the full system (human driver and ADAS), the model becomes an MDP and, as mentioned in Chapter 2, conditional properties are not so easily calculated in MDPs due to the existence of multiple adversaries. Despite this, it is possible to reason over lower bounds of these types of properties under the conditions of Proposition 2.3.1. For the PCTL formulas $\varphi_1 =$ `F (x=length) & (t<T)` and $\varphi_2 =$ `F (x=length)`, it is trivial that for any path $\pi \in \mathit{Paths}(s)$: $\pi \models \varphi_1 \Rightarrow \pi \models \varphi_2$. Furthermore, it can also be assumed that for:

$$\sigma' \in \arg \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_1 \mid \varphi_2) \tag{5.1}$$

it is true that:

$$P_s^{\sigma'}(\varphi_1) = p_{\max,s}(\varphi_1) \tag{5.2}$$

since intuitively the interest is in adversaries which maximise the conditional property through the maximisation of the probability of eventually reaching `(x=length) & (t<T)` (as this is the comparable case in the human driver model). Thus, under the conditions of Proposition 2.3.1, it is possible to write:

$$p_{\max,s}(\texttt{F (x=length) \& (t<T)} \mid \texttt{F (x=length)}) \geq \frac{p_{\max,s}(\texttt{F (x=length) \& (t<T)})}{p_{\max,s}(\texttt{F (x=length)})} \tag{5.3}$$

Therefore, a lower bound on the conditional probability, $\zeta(T)$, can be obtained using the quantitative properties:

$$\zeta(T) = \frac{\texttt{Pmax=? [F (x=length) \& (t<T)]}}{\texttt{Pmax=? [F (x=length)]}} \tag{5.4}$$

At this point it should also be noted that (as described in Section 4.2.1):

$$p_{\max,s}(\texttt{F x=length}) = 1 - p_{\min,s}(\texttt{F crashed}) \tag{5.5}$$

so long as $p_{\max,s}(\texttt{F (crashed | x = length)}) = p_{\min,s}(\texttt{F (crashed | x = length)}) = 1$ (completeness property for the MDP). Therefore, ζ can be re-written as:

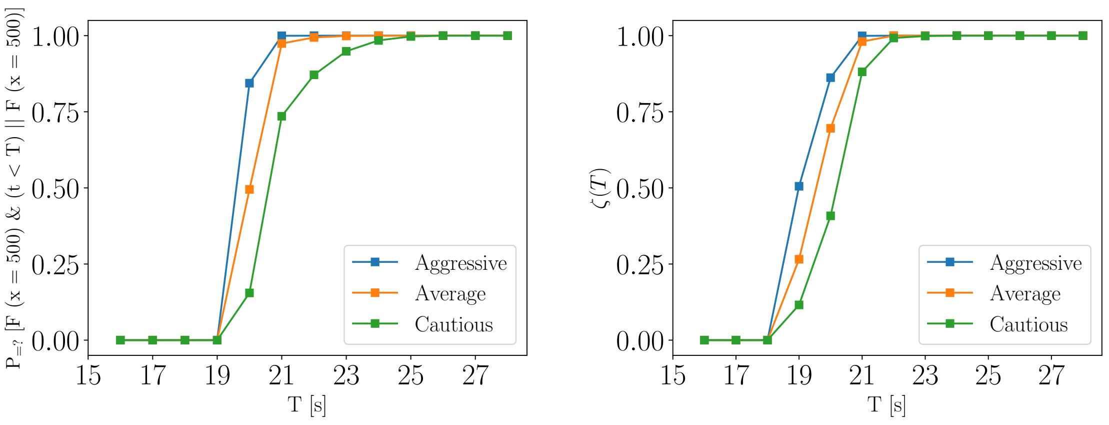
$$\zeta(T) = \frac{\texttt{Pmax=? [F (x=length) \& (t<T)]}}{1 - \texttt{Pmin=? [F crashed]}} \tag{5.6}$$

Once more, model checking the entire space of possible initial conditions would be infeasible given the time frame of this dissertation. Similarly to the safety properties, two methods were designed to compare both systems at the liveness level:

- **Temporal variation of liveness**: for two randomly selected tuples $(v, v_1, x_{1,0})$, the value of all the pertinent liveness properties ($T \in \{16, ..., 28\}$) was obtained. The two tuples were randomly generated as $(v, v_1, x_{1,0}) = (21, 19, 70)$

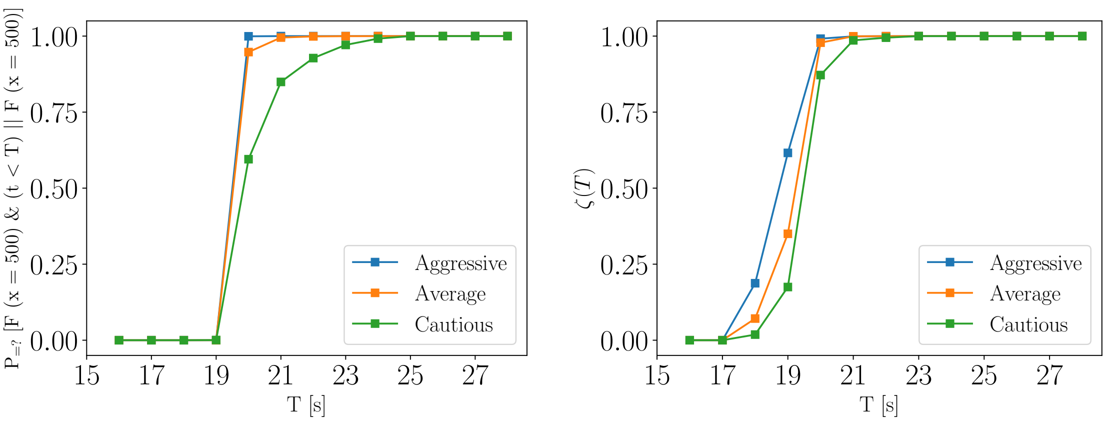
and $(v, v_1, x_{1,0}) = (26, 22, 46)$, and the results are presented in Figures 5.5 and 5.6, respectively.

- **Box plots**: for a randomly generated sample of 100 different initial conditions of $(v, v_1, x_{1,0})$, obtain box plot which represents the distribution of the values of two liveness properties using appropriate values of T (given the sample) for this sample and for each driver profile. After the sample was generated, it was determined that $T = 21$ and $T = 22$ were representative values (in the sense that they are great enough to avoid a significant portion of the values of both properties being 0, and not so large that a significant portion of the values for all scenarios are 1; these situations would make it impossible to compare the human and the full system). The results are presented in Figures 5.7 and 5.8 for $T = 21$ and $T = 22$, respectively.



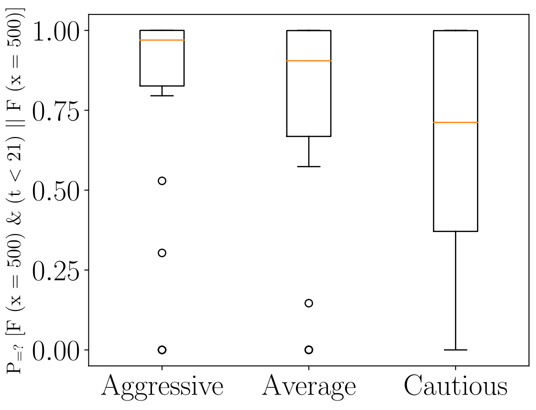
a. Human driver model **b.** ADAS

Figure 5.5 Plots of the variation of the value of the liveness properties with the value of T (in s) for the initial conditions $v = 21m/s$, $v_1 = 19m/s$ and $x_{1,0} = 70m$.

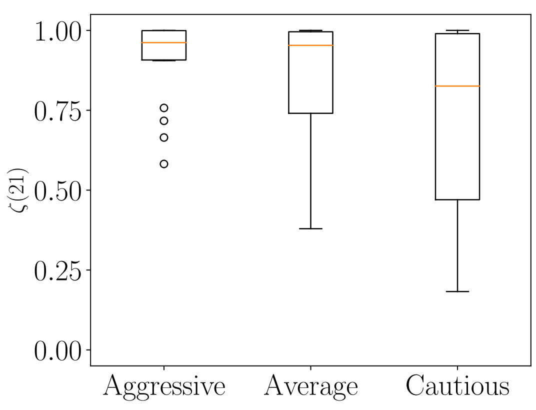


a. Human driver model **b.** ADAS

Figure 5.6 Plots of the variation of the value of the liveness properties with the value of T (in s) for the initial conditions $v = 26m/s$, $v_1 = 22m/s$ and $x_{1,0} = 46m$.

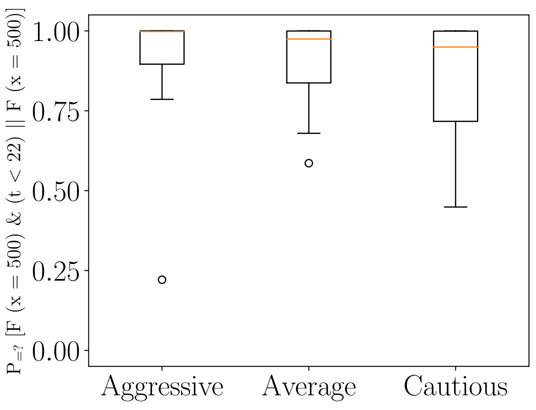


a. Human driver model

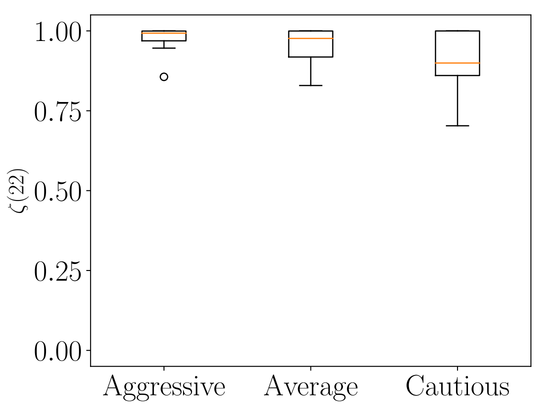


b. ADAS

Figure 5.7 Box plot of the value of the liveness properties for $T = 21s$ for a randomly sampled population of 100 different initial conditions (equal in both cases).



a. Human driver model



b. ADAS

Figure 5.8 Box plot of the value of the liveness properties for $T = 22s$ for a randomly sampled population of 100 different initial conditions (equal in both cases).

5.1.4 Discussion

The discussion of the results presented in Sections 5.1.2 and 5.1.3 is threefold in nature. Firstly, the comparison between the different driver profiles within each of the individual systems (i.e. human driver model and the human and ADAS system) is presented. Secondly, the results of the experiments of the human driver alone and the full system in both safety and liveness are discussed and compared (within the same driver class). Finally, an overall discussion is presented on the extent of the validity of such comparisons through the drawbacks of the metrics used.

With respect to the individual driver classes presented in both the human driver model and the human and ADAS system, the results overwhelmingly support the initial idea that aggressive drivers perform the worse in terms of the safety metrics, followed by average drivers and finally cautious ones. While this is observable in the individual test cases (Figures 5.1 and 5.2), Figure 5.4 firmly supports this conclusion within both systems. In terms of liveness, the reverse is true, with aggressive drivers outperforming average drivers who in turn outperform cautious drivers. Once more, while individual test cases support this result (Figures 5.5 and 5.6), Figures 5.7 and 5.8 emphatically reinforce it. Intuitively, this is what was expected from the construction of the human driver model, and it is still noticeable even using the ADAS designed.

In terms of the comparison between the human driver model and the driver with the ADAS system, it becomes necessary to interpret the plots while considering the differences in the metrics used.

In the safety evaluation, as mentioned in Section 5.1.2, the results are directly comparable as the metrics are the equivalent of one another for the representation of the models of both systems (i.e. a DTMC for the human driver model and an MDP for the human and ADAS system). The most useful comparison in terms of the test cases in this metric is presented in Figure 5.3, as the 3D plots are divided into the three categories of drivers considered. It is observable that the introduction of the ADAS increases safety for all situations considered in this test case (i.e. reduces the value of the safety property). From the more general overview seen in Figures 5.7 and 5.8, the same conclusion is drawn for each driver class, with the 25%, 50% (median) and 75% quartiles being lower for the system with the ADAS than those for the human driver alone.

With respect to liveness, the comparison needs to be carefully drawn, as the conditional property in the human driver model alone (DTMC) is not the equivalent of the metric chosen for liveness in the human and ADAS system (MDP). However, ζ is proven to be the lower bound of the conditional property in the human and ADAS system under the conditions presented in Section 5.1.3. From Figures 5.5 and 5.6, it can be observed that, for each T, the values of ζ are always greater or equal than those of the conditional property of the human driver model. The same result can be seen in Figures 5.7 and 5.8 for $T = 21$ and $T = 22$, respectively, for the 25% and 75% quartiles. Given that ζ is the lower bound of the maximal conditional property, it can be concluded that the system with the ADAS outperforms the human driver alone in this metric.

Despite the results pointing to the improvements in terms of the safety and liveness properties the ADAS designed brings, they should be taken with care, as there are drawbacks in the metrics used.

The first drawback has to do with the fact that the comparison in the full system (i.e. human driver model and ADAS system) deals with minimal and maximal properties due to the existence of adversaries in MDPs. Thus, the fact that the safety property is lower in this case than in the human driver model and the liveness properties are higher, does not mean that there exists a feasible strategy where both safety and liveness are optimal and outperform the human driver (there might be a trade-off). This is not the case with the human driver model, where it is known that both quantitative properties are satisfied simultaneously. In order to evaluate the feasibility properly, Pareto curves should be generated for each individual scenario and it should be seen whether there exists a Pareto point where both values outperform the human driver. This analysis is time consuming with the tools currently available (i.e. PRISM and Storm) and was therefore infeasible in the time frame of this dissertation.

An additional drawback in terms of the liveness properties in the full system case is that these are defined as lower bounds which are not proven to be tight. In fact, in Figure 5.8, while the 25% and 75% quartiles appear to be higher in the case of the full system than the human driver model, this is not the case with the median, which is at similar values (or even lower for the class of cautious drivers). From the rest of the data, it would appear that this value might be the result of the fact that the lower bound is not tight and might be underestimating the value of the maximal conditional property for the sample population used. However, this is not certain, and it might also be that the full system is outperformed, in terms of the liveness properties, by the human driver in certain conditions. The use of the unconditional liveness properties here would be meaningless, as it would incur in the safety bias first presented in Section 3.3.3. A solution to this problem would be to develop the tools to model check conditional properties in MDPs, however, this was outside the scope of this dissertation and would be highly time consuming.

5.2 Demonstration of Possible Extensions

While outside the scope of the main goal of the dissertation of evaluating safety and liveness, extensions of the work presented can be obtained fairly easily. They can be achieved through additional properties that can be used to synthesise strategies for the ADAS which enforce other types of behaviours, or through minor modifications of the model to encompass different assumptions.

5.2.1 Left Lane Penalising

When a driver attempts an overtake, it should do so by performing two lane changes, one from the current lane to the one immediately to the left, and another one to return to the original lane after it has passed the lead vehicle. The manoeuvre should be performed safely, yet as quickly as possible [49]. With this in mind, properties can be specified to assert such a regulation to the ADAS to guarantee compliance.

In particular, the property should penalise the usage of the left lane ($lane = 2$). For this purpose, the following reward structure was designed:

```
rewards
  [] lane = 1: 0;
  [] lane = 2: 1;
endrewards
```

And the property:

```
Rmin=? [C]
```

minimises the cumulative value of this reward, and, therefore, the time spent in the left lane. Thus, the following multi-objective property is introduced:

```
multi(Pmin=? [F crashed], Rmin=? [C])
```

To exemplify the use of this reward structure and property, the scenario $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 50)$ (an aggressive driver with an initial speed of $30m/s$ and a lead vehicle at a distance of $50m$ going at $22m/s$) is considered. It should also be noted that the length of the road segment considered in this case was $400m$. By verifying this property, the Pareto curve obtained is presented in Figure 5.9

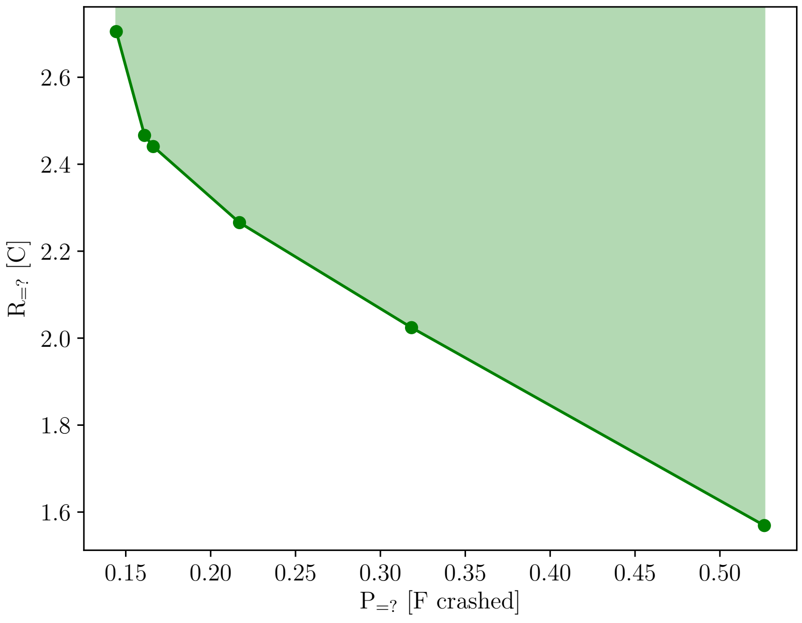
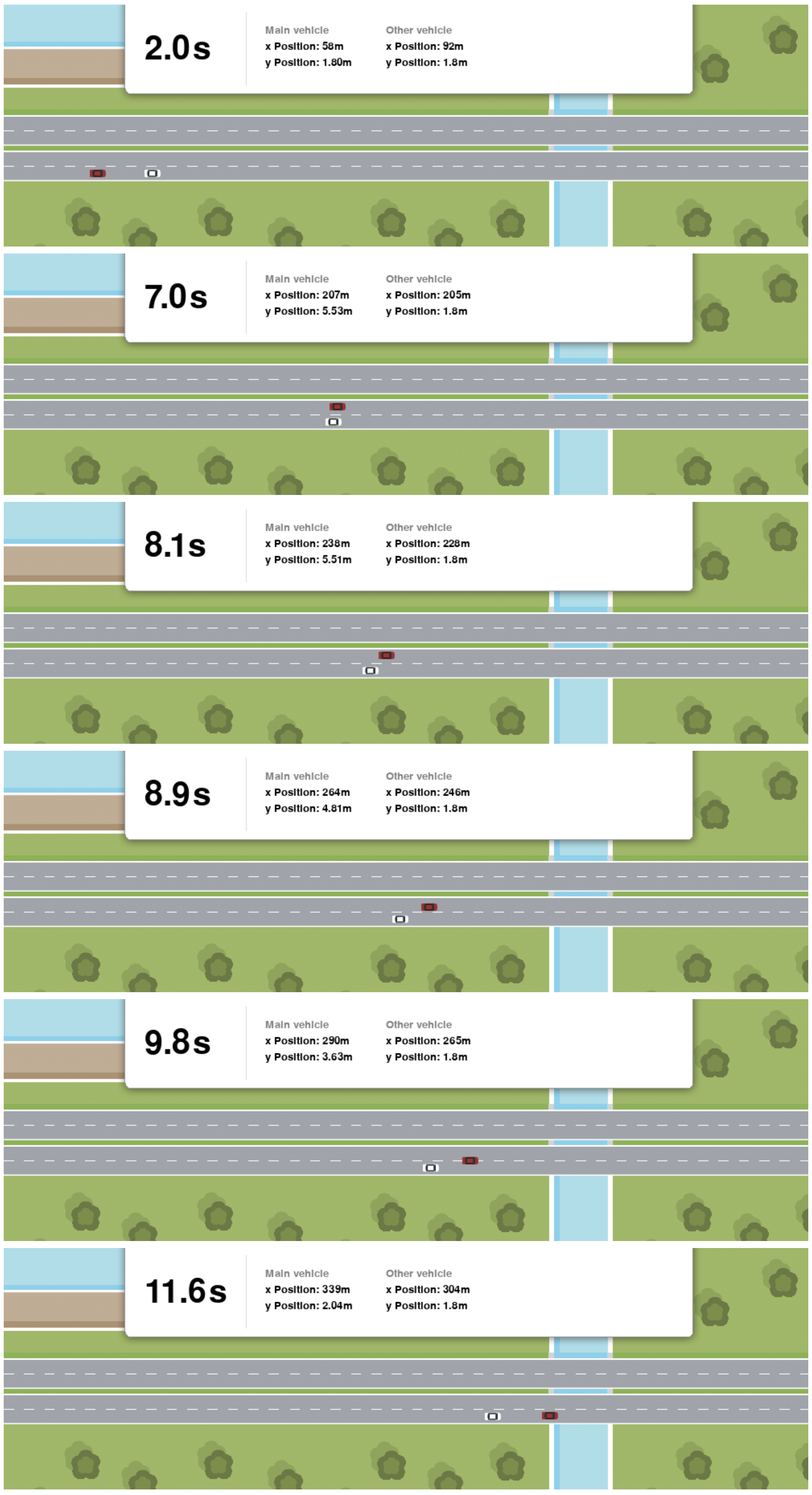


Figure 5.9 Pareto curve for the model checking of the lane penalising property.

Observing this curve, it is possible to conclude that it is feasible to have `R<=1.8 [C]`, and thus the following property:

```
multi(Pmin=? [F crashed], R<=1.8 [C])
```

yields an adversary which can be used to generate a sample path in the model. Similarly to what is presented in Section 3.4, a simulator was created to allow for the visualisation of a path in the model (the code for which is shown in Appendix B.3). The results of the simulation are presented in Figure 5.10.



2.0s

Main vehicle
x Position: 58m
y Position: 1.80m

Other vehicle
x Position: 92m
y Position: 1.8m

7.0s

Main vehicle
x Position: 207m
y Position: 5.53m

Other vehicle
x Position: 205m
y Position: 1.8m

8.1s

Main vehicle
x Position: 238m
y Position: 5.51m

Other vehicle
x Position: 228m
y Position: 1.8m

8.9s

Main vehicle
x Position: 264m
y Position: 4.81m

Other vehicle
x Position: 246m
y Position: 1.8m

9.8s

Main vehicle
x Position: 290m
y Position: 3.63m

Other vehicle
x Position: 265m
y Position: 1.8m

11.6s

Main vehicle
x Position: 339m
y Position: 2.04m

Other vehicle
x Position: 304m
y Position: 1.8m

Figure 5.10 Snapshots of the simulator for one of the paths of the model for $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 50)$, obtained using the left lane penalising property.

As observed, minimal time is spent in the left lane, with the main vehicle returning to the right lane as soon as it passes the lead vehicle.

5.2.2 Unsafe by Construction

Until this point, the assumption has been that the goal of the assistance system was to enforce safety and liveness. However, assume a bad actor has access to the model and the deployment process used in the vehicle. In such a case, it becomes important to evaluate the extent of the damage that such an actor could do. Take the same scenario as above, $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 50)$. Let's also assume that the bad actor wants the accident to happen after the the driver has driven between 100 and $200m$ (for a road segment of $400m$). The following property can be considered:

```
multi(Pmax=? [F crashed], Pmax=? [x <= 100 U (x>100 & x<=200)])
```

which, when model checked, generates a Pareto curve of the trade-off between maximising the probability of crashing and the probability of reaching at least $100m$ and at most $200m$. The Pareto curve generated is presented in Figure 5.11.

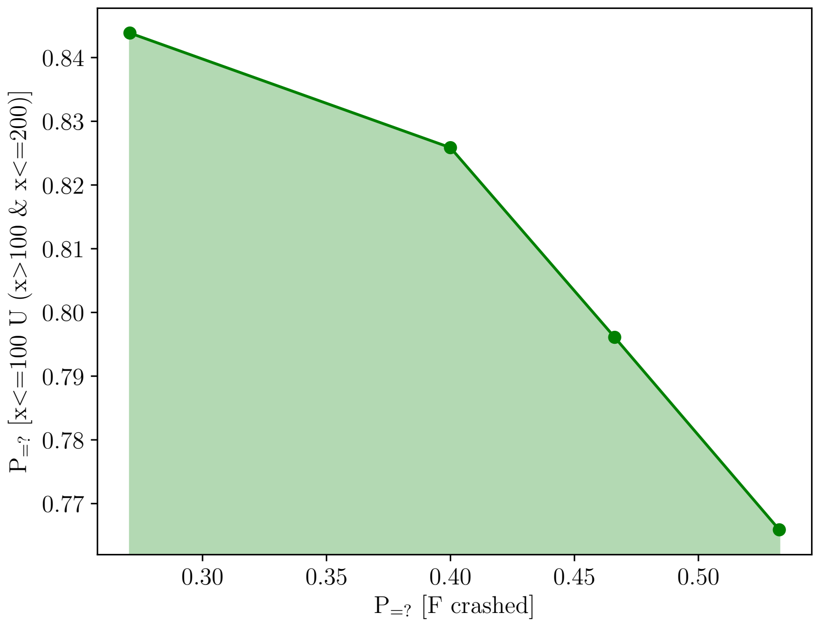


Figure 5.11 Pareto curve for the model checking of the unsafe driving property.

By observing the Pareto curve, the following property can be devised:

```
multi(Pmax=? [F crashed], P>=0.5 [x <= 100 U (x>100 & x<=200)])
```

This property maximises the probability of crashing, while requiring the vehicle to drive for at least $100m$ and at most $200m$ with probability at least 0.5 before the crash happens. By model checking this property, the maximum value of `P=? [F crashed]` comes out to be 0.5326, and the adversary obtained permits the generation of paths such as the one presented in the simulator frames in Figure 5.12. It should be noted that the value of `P=? [F crashed]` in the human driver model is 0.3935, and thus the bad actor has successfully increased the probability of crashing.

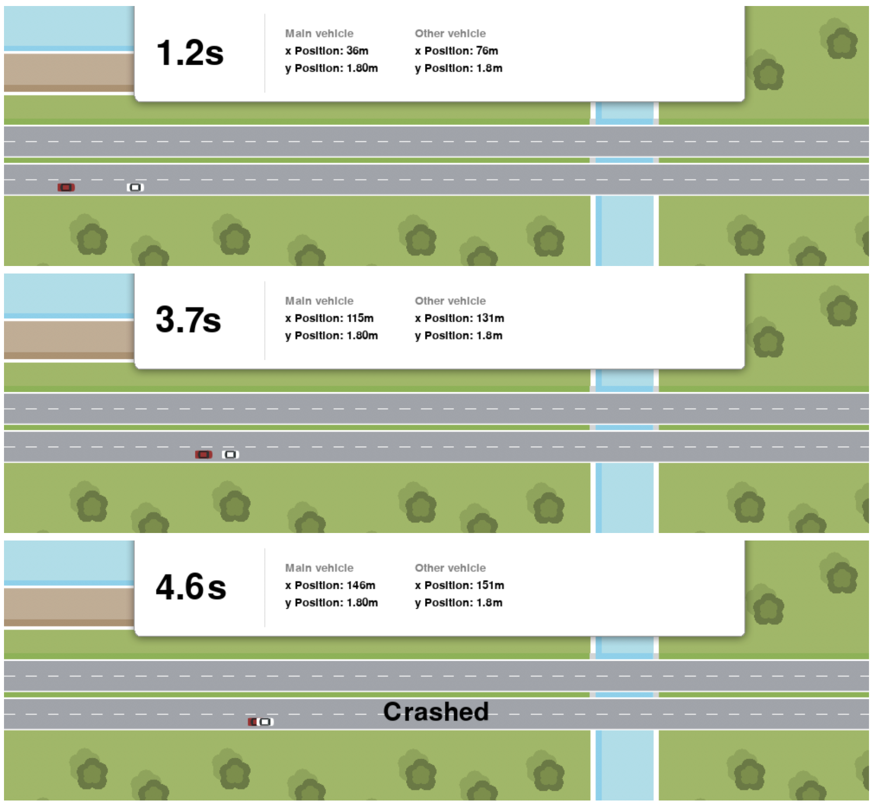


Figure 5.12 Snapshots of the simulator for one of the paths of the model for $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 50)$ obtained using the unsafe property.

5.2.3 Three Vehicle Highway Scenario

So far the scenarios studied have assumed there are only two vehicles on the road: the main vehicle considered and another lead vehicle. It would be interesting to extend the current model to a simple three vehicle scenario.

In this case, it was considered that two other vehicles were on the road, each starting at a certain distance from the main vehicle, $x_{1,0}$ and $x_{2,0}$, respectively, and going at a constant velocity of v_1 and v_2, respectively, with $x_{1,0} < x_{2,0}$ and $v_1 \leq v_2$. A scenario in this case is thus represented by a tuple $(d_{type}, v, v_1, x_{1,0}, v_2, x_{2,0})$. The model is obtained by adding a variable, $focusVehicle \in \{1, 2\}$ which defines which vehicle is the current focus of attention of the driver. The constant movement of the two vehicles are simply defined as:

$$\begin{aligned} x_1(t) &= x_{1,0} + v_1 \cdot t \\ x_2(t) &= x_{2,0} + v_2 \cdot t \end{aligned} \tag{5.7}$$

Considering $i = focusVehicle$ (for conciseness sake), the distance between the two vehicles can be then obtained as:

$$d(t) = |x(t) - x_i(t)| \tag{5.8}$$

And the boolean function *posDist* can be defined as:

$$posDist(t) = \begin{cases} \text{true}, & \text{if } x(t) \geq x_i(t) \\ \text{false}, & \text{otherwise.} \end{cases} \tag{5.9}$$

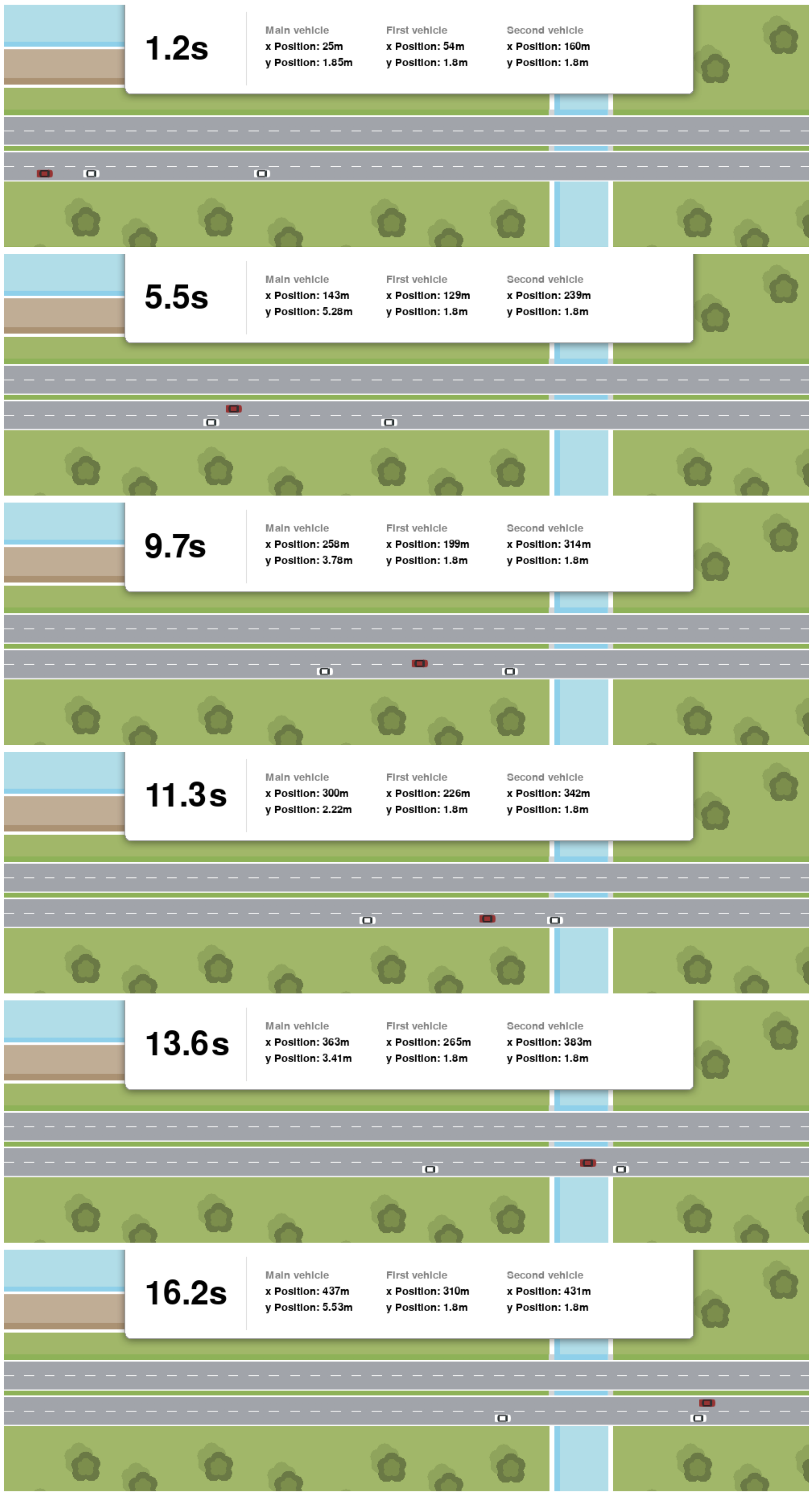
As soon as the main vehicle overtakes the first vehicle successfully, $focusVehicle$ becomes 2 and the formulas switch from the first to the second vehicle.

This constitutes an efficient way of introducing an additional vehicle in the model, as it is essentially achieved through the use of a single variable and the modifying of the required formulas. It should be noted that, despite this, the reachable states in the model will increase as well as the size of the model definition (and, consequently, the building time of the model) from the need to include more of the lane changing look-up table (for v_1 and v_2).

Take, as an example, the case $(d_{type}, v, v_1, x_{1,0}, v_2, x_{2,0}) = (3, 24, 17, 35, 18, 140)$, that is, a cautious driver going at $24m/s$ and two lead vehicles, one starting $35m$ in front of the main and going at $17m/s$, and the other one starting at $140m$ and going at $18m/s$. In this scenario, for simulation purposes, it is considered that the length of the road segment is $400m$. Building the model, an MDP with $52,059$ states is obtained. By verifying the property `Pmin=? [F crashed]`, the value of this property comes out to be 0.1580 and `Pmax=? [F (x=length) & (t<16)]` is the property with the lowest T with a non-zero value, with a value of 0.1193. As such, the following multi-objective property:

```
multi(Pmax=? [F x=400 & t < 16], P<=0.158 [F crashed])
```

generates an adversary which maximises this liveness property while guaranteeing the minimum possible value for the safety property. The adversary obtained permits the generation of paths such as the one presented in the simulator frames in Figure 5.13 (for a slightly modified simulator which includes information about the two other vehicles). It should be noted that, while the simulation continues after $400m$, the model considered has a road segment of length $400m$ (it continues past this point in this case because the main vehicle is performing a lane change).


1.2s
Main vehicle
x Position: 25m
y Position: 1.85m
First vehicle
x Position: 54m
y Position: 1.8m
Second vehicle
x Position: 160m
y Position: 1.8m

5.5s
Main vehicle
x Position: 143m
y Position: 5.28m
First vehicle
x Position: 129m
y Position: 1.8m
Second vehicle
x Position: 239m
y Position: 1.8m

9.7s
Main vehicle
x Position: 258m
y Position: 3.78m
First vehicle
x Position: 199m
y Position: 1.8m
Second vehicle
x Position: 314m
y Position: 1.8m

11.3s
Main vehicle
x Position: 300m
y Position: 2.22m
First vehicle
x Position: 226m
y Position: 1.8m
Second vehicle
x Position: 342m
y Position: 1.8m

13.6s
Main vehicle
x Position: 363m
y Position: 3.41m
First vehicle
x Position: 265m
y Position: 1.8m
Second vehicle
x Position: 383m
y Position: 1.8m

16.2s
Main vehicle
x Position: 437m
y Position: 5.53m
First vehicle
x Position: 310m
y Position: 1.8m
Second vehicle
x Position: 431m
y Position: 1.8m

Figure 5.13 Snapshots of the simulator for one of the paths of the three vehicle highway model for $(d_{type}, v, v_1, x_{1,0}, v_2, x_{2,0}) = (3, 24, 17, 35, 18, 140)$.

Chapter 6

Conclusions and Future Work

This dissertation attempted to demonstrate the benefits of using model checking and synthesis to generate correct-by-construction driver assistance systems through an application using a cognitive architecture. This required a model of a human driver in a cognitive architecture to be implemented in PRISM's modelling language. Since Salvucci established his driver model in ACT-R in [26], this dissertation focused on the abstraction process to convert the integrated continuous model into a discrete-time Markov chain. This was done using pre-simulation and Monte Carlo simulation techniques, which allowed the generated models to be relatively small in terms of state space and transitions. Afterwards, several designs were considered in terms of the capabilities and placement of the driver assistance system within the model. These options ranged from suggestions at the decision making level to correcting linear or steering acceleration. They were evaluated carefully, and the best performing one was then more extensively compared to the human driver model previously obtained. The results showed a significant performance increase (in terms of the metrics used) by the introduction of the driver assistance system.

Most of the expected contributions were achieved. The modelling of a complex scenario (a 2-lane highway and the interactions that arise with various profiles of drivers e.g. follow, crash or overtake another vehicle) through the abstraction of Salvucci's human driver model presented in [26] was achieved through the use of efficient pre-simulation and Monte Carlo simulation techniques and automation of the model writing which lead to compact models (e.g. in [25], Lam considers a one way road with at most $270m$, and [20] considers similarly simple scenarios through its model assumptions). For a road length of $500m$, none of the initial conditions considered had a model with more than 700 states, an achievement given the original complexity of the scenario and the constraints of the ACT-R architecture. The establishment of insightful metrics in terms of safety and liveness and their application to multi-objective synthesis of possible ADAS was another central achievement of the dissertation (e.g. in [25], only single-objective safety centric metrics are considered). Furthermore, it should be noted that, as shown in Chapter 5, the final models (human driver and ADAS) obtained were also manageable in terms of size - with none exceeding 1 million states - an accomplishment as it shows the methodology's flexibility towards even more complex models. These two main achievements combined pave the way for the establishment of a general framework which can be used to tackle similar driving situations (e.g. intersections, multilane highways with more vehicles or urban driving), another aim of the dissertation.

Despite the achievements and the contributions of this work, there are several limitations to the methodology introduced. The first limitation is the building time of the models. While the models built have a relatively small state space, the building time is quite high for both the human driver model and the full system (human and ADAS), as shown in Section 5.1.1. This is mostly due to the simulation and the tables which result out of this process. While these reduce the state space of the final model, they make the model description quite big (particularly compared to other similarly sized models found in [25, 20]). The model checking tool, in turn, then takes quite some time to process the description files, leading to high building times for the models. Despite this, the time can be reduced by the selection of the right tool and engine (e.g. the sparse engine in Storm outperforms all the engines PRISM supports for the human driver model, but PRISM's hybrid engine outperforms all of Storm's engines in the full model).

A second limitation has to do with the tools available. As the reader might have noticed, the models presented in Section 5.2 were built considering a road segment of $400m$ instead of the $500m$ assumed throughout the rest of the dissertation. This came from the fact that the adversary generation options in PRISM (`-adv` or `-advmdp`) make use of the LP engine which, as of right now, is not capable of solving a problem as big as this would have been for the length of $500m$ (despite the MDP having under $500,000$ states). Another limitation of the methodology has to do with the lack of support for the model checking of conditional properties in MDPs in both PRISM and Storm (despite the latter being able to verify them in DTMCs). While PRISM does not support the generation of Pareto curves for more than two objectives, PRISM-games - an extension of PRISM to stochastic multi-player games [50] - does so. However, this tool does not use a symbolic representation of the model and, as such, is quite limited in terms of the size of the models it supports.

The final limitation presented is related to the underlying assumptions made and the usefulness of the driver assistance systems generated. There were quite some assumptions made throughout the dissertation which are not necessarily true. While the control laws that Salvucci presents in [26] are the result of corrections made to his original laws presented in [31] through the use of real data he collected, most of the assumptions used in this dissertations (e.g. driver profiles) are not. As such, while it is still possible to compare between situations and discuss the advantages of the driver assistance system, it is questionable to what degree the ADAS synthesised in the course of this dissertation are actually accurate and useful.

In the future, the model description might be reduced through pre-computation and elimination of the guards that are never satisfied within the modules, which make up a significant portion of the file. Furthermore, a data driven approach should be followed to validate the results obtained and correct the assumptions made about the drivers. Once the models are accurate according to the real world data, it is possible to deploy the solutions obtained.

Extensions of this work might include implementations of similar driving situations using the ideas presented in terms of abstraction and driver assistance in order to validate the approach. A framework to formalise and generalise the process of obtaining the driver assistance systems can also be the focus of future research in this area.

Appendix A

Code for Human Driver Modelling

A.1 ACT-R Implementation (Matlab)

A.1.1 Main Loop

File A.1 act_r.m

```
% ACT_R — Matlab implementation of Salvucci's human driver model using the
% cognitive architecture ACT—R.

% Author: Francisco Girbal Eiras, MSc Computer Science
% University of Oxford, Department of Computer Science
% Email: francisco.eiras@cs.ox.ac.uk
% 07—Mar—2018; Last revision: 24—Apr—2018

%————————————— BEGIN CODE ——————————————

figure('pos',[300 300 800 350])
g = animatedline('Color','g','LineWidth',3);
h = animatedline('Color','r','LineWidth',3);

% Draw the separating line

global delta_t length width mem_size pMonitor;
length = 10000; % 10km
width = 7.2;
delta_t = 1;
mem_size = 8;
pMonitor = 0.8;

hold on
plot([0, length], [width/2, width/2], '——k');
plot([0, length], [width/4, width/4], '——b');
plot([0, length], [3*width/4, 3*width/4], '——g');

% Draw the car in the other line

```

```
axis([0 length 0 width])

% Vehicle info
x = 0;
y = width/4;
lane = 1;
vx = 15;
vy = 0;

% Obstacles info
x1 = 50;
y1 = width/4;
vx1 = 34;
vy1 = 0;

x2 = 5500;
y2 = 3*width/4;

vehicles_pos = [x1,y1; x2,y2];

% Memory preparation
old_theta_near = 0;
old_theta_far = 0;
old_thw_car = (x1 - x)/vx;

% Graph preparation
ll = plot(x2, y2, 'sk', 'MarkerSize', 14, 'MarkerFaceColor','black');
hold off
lh = legend([g h ll], sprintf('vx = %.2d, vy = %.2f', vx, vy), sprintf('vx = %.2d, vy = %.2f',
     vx1, vy1), 'Time: ');

changing = false;
t = 0;
monitoring_memory = zeros(mem_size,3);

while x < length

    [x,y,vx,vy, old_theta_near, old_theta_far ,old_thw_car] = control(x,y,vx,vy,lane,
     old_theta_near, old_theta_far ,old_thw_car, vehicles_pos);

    monitoring_memory = monitor(monitoring_memory,x,vehicles_pos)

    [lane, changing] = decision_making(lane, old_thw_car, x, y, monitoring_memory, changing,
     vehicles_pos);

    % Other car
    vehicles_pos(1,1) = vehicles_pos(1,1) + vx1*delta_t;
    vehicles_pos(1,2) = vehicles_pos(1,2) + vy1*delta_t;

    % Graphical visualisation
    addpoints(g,x,y);
    addpoints(h, vehicles_pos(1,1), vehicles_pos(1,2));

    set(lh, 'string', {sprintf('vx = %.2f, vy = %.2f', vx, vy), sprintf('vx = %.2f, vy = %.2f'
    , vx1, vy1), sprintf('Time: %.0f s', t)});

    drawnow

    t = t + delta_t;
```

```
end

%-------------- END OF CODE ---------------
```

A.1.2 Control Module

File A.2 control.m

```
function [x,y,vx,vy, old_theta_near , old_theta_far ,old_thw_car] = control( x, y, vx, vy, lane, old_theta_near , old_theta_far , old_thw_car , vehicles_pos )
%CONTROL ACT-R control module

global delta_t length width;

% Constants
k_far = 3.0;%15;
k_near = 0.6;%3;
k_i = 0.3;%5;
theta_nmax = 0.07;
k_car = 0.03;%4;
k_follow = 0.01;
max_speed = 34; % approximately 120 km/h

if lane == 1
    y_follow = width/4;
else
    y_follow = 3*width/4;
end

near_point = [x + 4, y_follow ];
far_point = [x + 10, y_follow ];

theta_near = tan((near_point(2) - y)/(near_point(1) - x));
theta_far = tan(( far_point (2) - y)/( far_point (1) - x));

Delta_theta_near = theta_near - old_theta_near ;
Delta_theta_far = theta_far - old_theta_far ;

Delta_varphi = k_far* Delta_theta_far + k_near*Delta_theta_near + k_i*min(theta_near,theta_nmax )*delta_t ;

% To determine timeheadway, check for vehicles/accidents between the
% car position and far point; the timeheadway corresponds to the
% timeheadway to the closest vehicle (or the end of the road)

min_timeheadway = (length + 1000 - x)/vx;
for i = 1:size( vehicles_pos ,1)
    if vehicles_pos (i ,1) > x && abs(vehicles_pos(i,2) - y) < 0.2
        th = (vehicles_pos(i,1) - x)/vx;
        min_timeheadway = min(min_timeheadway, th);
    end
end
time_headway = min_timeheadway;

Delta_time_headway = time_headway - old_thw_car;

Delta_psi = k_car*Delta_time_headway + k_follow*(time_headway - 2.5)*delta_t;
```

```

ax = Delta_psi;
 if  Delta_varphi  ~= 0
     ay = sin( Delta_varphi );
 else
     ay = 0;
end

% Update variables

 if  vx + ax*delta_t  < max_speed
     vx = vx + ax*delta_t;
end
vy = vy + ay*delta_t;

x = x + vx*delta_t + 0.5*ax*delta_t^2;
y = y + vy*delta_t + 0.5*ay*delta_t^2;

 old_theta_near  = theta_near;
  old_theta_far  = theta_far;
old_thw_car  = time_headway;

end
```

A.1.3 Decision Making Module

File A.3 decision_making.m

```
function [ lane, changing ] = decision_making( lane, old_thw_car, x, y, dec_memory, changing,
     vehicles_pos )
%DECISION_MAKING ACT-R decision making module

global width;

% Constant
thw_pass = 20;

% Check whether a lane change is happening or not
 if  changing == true
     if  lane  == 1 && abs(y - width/4) < 0.1
         changing = false;
      elseif  lane  == 2 && abs(y - 3*width/4) < 0.1
         changing = false;
     end

     return
end

 if  old_thw_car  < thw_pass && lane == 1
     % attempt to change lanes
     [lane,changing]  = try_change_lanes(2,  dec_memory,  vehicles_pos,  x);
     if  changing == 1
         abs(x - vehicles_pos(1,1))
         pause
     end
 elseif  old_thw_car  < thw_pass && lane == 2
     % attempt to change lanes
```

```
        [lane,changing] = try_change_lanes(1, dec_memory, vehicles_pos, x);
        if changing == 1
            abs(x - vehicles_pos(2,1))
            pause
        end
end

end
```

A.1.4 Monitoring Module

File A.4 monitor.m

```
function [ monitoring_memory ] = monitor( monitoring_memory,x,vehicles_pos )
%MONITOR ACT-R monitor module

global pMonitor width length;

if rand > pMonitor
    return
end

% Look vehicle
pm = rand;
if pm < 0.25
    lane_dir = 1; % lane 1, back

    min_dist = length+1;
    for i = 1:size(vehicles_pos,1)
        if vehicles_pos(i,1) < x && abs(vehicles_pos(i,2) - width/4) < 0.2
            d = abs(vehicles_pos(i,1) - x);
            min_dist = min(min_dist, d);
        end
    end
    if min_dist == length+1
        exists = false;
    else
        exists = true;
    end
    new_info = [lane_dir, exists, min_dist];

elseif pm < 0.5
    lane_dir = 2; % lane 1, front

    min_dist = length+1;
    for i = 1:size(vehicles_pos,1)
        if vehicles_pos(i,1) > x && abs(vehicles_pos(i,2) - width/4) < 0.2
            d = abs(vehicles_pos(i,1) - x);
            min_dist = min(min_dist, d);
        end
    end
    if min_dist == length+1
        exists = false;
    else
        exists = true;
    end
    new_info = [lane_dir, exists, min_dist];
```

```

 elseif pm < 0.75
      lane_dir = 3; % lane 2, back

      min_dist = length+1;
      for i = 1:size(vehicles_pos,1)
           if vehicles_pos(i,1) < x && abs(vehicles_pos(i,2) - 3*width/4) < 0.2
                d = abs(vehicles_pos(i,1) - x);
                min_dist = min(min_dist, d);
           end
      end
      if min_dist == length+1
           exists = false;
      else
           exists = true;
      end
      new_info = [lane_dir, exists, min_dist];

 else
      lane_dir = 4; % lane 2, front

      min_dist = length+1;
      for i = 1:size(vehicles_pos,1)
           if vehicles_pos(i,1) > x && abs(vehicles_pos(i,2) - 3*width/4) < 0.2
                d = abs(vehicles_pos(i,1) - x);
                min_dist = min(min_dist, d);
           end
      end
      if min_dist == length+1
           exists = false;
      else
           exists = true;
      end
      new_info = [lane_dir, exists, min_dist];
end

% Add to memory
monitoring_memory = [monitoring_memory(2:size(monitoring_memory,1),:); new_info];

end
```

A.2 Control Abstraction (Matlab)

File A.5 lane_changing_generator.m

```
% LANE_CHANING_GENERATOR - File that generates the table of a passing and returning to
% the initial lane.

% Author: Francisco Girbal Eiras, MSc Computer Science
% University of Oxford, Department of Computer Science
% Email: francisco.eiras@cs.ox.ac.uk
% 16-Apr-2018; Last revision: 12-May-2018

%------------- BEGIN CODE --------------

clc
warning('off','all')
```

```matlab

% Car size for collision purposes (both cars are assumed to be of equal dimensions)
h = 1.9;
w = 4.8;

% Possible initial distances between the vehicles
ds = linspace(1,43,43);
% ds = 25;

% Possible vehicles initial velocity
vi1s = linspace(15,34,20);
vi2s = linspace(15,34,20);
% vi1s = 30;
% vi2s = 20;

% Road width/length + time intervals for action
global width len delta_t
width = 7.4;
len = 500;
delta_t = 0.5;
n_iter = 100;

% Code starts

generated_table = zeros(2*length(ds)*length(vi1s)*length(vi2s), 9);
display(sprintf('Generating table of %d entries', size(generated_table,1)))
t1 = cputime;

other_table = zeros(2*length(ds)*length(vi1s)*length(vi2s), 24);

textprogressbar('Progress: ')

% Simulate all possible combinations
for lane = 2:-1:1
for vi2_i = 1:length(vi2s)
    for vi1_i = 1:length(vi1s)
        for d_i = 1:length(ds)

            d = ds(d_i);
            vi1 = vi1s(vi1_i);
            vi2 = vi2s(vi2_i);

            sum_x = 0;
            sum_vx = 0;
            sum_t = 0;
            sum_col = 0;

            x_y_t = zeros(13*n_iter,3);

            f = figure(1);
            hold on;

            n_write = 1;
            bad_run_example = zeros(13,3);
            temp = zeros(13,3);

            for j=1:n_iter

                if lane == 1
                x = d;
            else
                x = 0;
            end
            y = (-2*lane + 5)*width/4;
            vx = vi1;
            vy = 0;

            if lane == 1
                x1 = 0;
            else
                x1 = d;
            end
            y1 = width/4;
            vx1 = vi2;
            vy1 = 0;

            old_theta_near = 0;
            old_theta_far = 0;
            old_thw_car = (x1 - x)/vx;

            vehicles_pos = [x1,y1];

            col = false;
            t = 0;
            ay = 100;

            temp(1,:) = [0,x,y];
            idx_sub = 2;

            while x < len && ~(abs(y - (2*lane-1)*width/4) < 0.02)
                col = check_collision ([x,y], vehicles_pos, h, w) || col;

                [x,y,vx,vy,ay, old_theta_near, old_theta_far ,old_thw_car] = control(x,y,vx,
    vy,lane, old_theta_near, old_theta_far ,old_thw_car, vehicles_pos);

                vehicles_pos (1,1) = vehicles_pos (1,1) + vx1*delta_t;
                vehicles_pos (1,2) = vehicles_pos (1,2) + vy1*delta_t;

                t = t + delta_t;

                temp(idx_sub,:) = [t, x, y];
                idx_sub = idx_sub + 1;

                plot( vehicles_pos (1,1), vehicles_pos (1,2), 'or')
            end

            while floor(t) ~= t
                x = x + vx*delta_t;
                vehicles_pos (1,1) = vehicles_pos (1,1) + vx1*delta_t;
                t = t + delta_t;

                temp(idx_sub,:) = [t, x, y];
                idx_sub = idx_sub + 1;
                plot( vehicles_pos (1,1), vehicles_pos (1,2), 'or')
            end

%               sum_x = sum_x + x;
            sum_vx = sum_vx + vx;
```

```
                sum_t = sum_t + t;
                sum_col = sum_col + col;

                if col == 0
                    x_y_t(13*(n_write - 1) + 1:13*n_write,:) = temp;
                    n_write = n_write + 1;
                else
                    bad_run_example = temp;
                end
            end

            if lane == 1
                sub_mat = d*[zeros(size(x_y_t,1), 1), ones(size(x_y_t,1),1), zeros(size(x_y_t
    ,1), 1)];
                x_y_t = x_y_t - sub_mat;
                sub_mat = d*[zeros(size(bad_run_example,1), 1), ones(size(bad_run_example,1)
    ,1), zeros(size(bad_run_example,1), 1)];
                bad_run_example = bad_run_example - sub_mat;
            end

            if n_write ~= 1
                x_y_t = x_y_t(1:13*(n_write-1),:);
            else
                x_y_t = [0,0,0];
            end

            if sum_col/n_iter == 0
                bad_run_example = [0,0,0];
            end

            p_x = polyfit(x_y_t(:,1), x_y_t(:,2),2);
            p_y = polyfit(x_y_t(:,1), x_y_t(:,3),6);

            bad_p_x = polyfit(bad_run_example(:,1),bad_run_example(:,2),2);
            bad_p_y = polyfit(bad_run_example(:,1),bad_run_example(:,3),6);

            plot(x_y_t(:,2), x_y_t(:,3),'ob')
            plot(bad_run_example(:,2), bad_run_example(:,3),'*r')
            p_xy = polyfit(x_y_t(:,2),x_y_t(:,3),5);
            bad_p_xy = polyfit(bad_run_example(:,2),bad_run_example(:,3),5);
            x1 = linspace(0,max(x_y_t(:,2)));
            y1 = polyval(p_xy,x1);
            plot(x1,y1,'b')
            x1 = linspace(0,max(bad_run_example(:,2)));
            y1 = polyval(bad_p_xy,x1);
            plot(x1,y1,'r')

            pause

%           est_x1 = round(sum_x/n_iter)
            est_vx = max(round(sum_vx/n_iter), vi1);
            est_t = round(sum_t/n_iter);
            est_x = round(polyval(p_x, est_t));
            est_col = sum_col/n_iter;

            idx = (2-lane)*length(ds)*length(vi1s)*length(vi2s) + (vi2_i - 1)*length(vi1s)*
    length(ds) + (vi1_i - 1)*length(ds) + d_i;

%           generated_table(idx,:) = [3-lane,d,vi1,vi2,col,round(x - (d)*(2-lane)),round(vx)
```

```
        ,round( vehicles_pos (1,1)−(d)∗(lane−1)),t];
                generated_table (idx ,:)  = [3−lane,d,vi1 , vi2 , est_col ,round(est_x  − (d)∗(2−lane)),
        est_vx ,round(vi2∗ est_t −(d)∗(lane−1)),est_t ];
                other_table ( idx ,:)  = [3−lane,d,vi1 , vi2 , p_x , p_y , bad_p_x, bad_p_y];

                textprogressbar (100∗idx/(2∗length(ds)∗length( vi1s )∗length( vi2s )))
            end
        end
end
end

 textprogressbar ( 'Done')

 display ( sprintf ( 'Generated in  %.3f seconds' ,  cputime − t1))

% Display the  table
header = {'o_lane' , 'd' , ' vi1 ' , ' vi2 ' , 'Acc?' , ' delta_x1 ' , ' vf1 ' , ' delta_x2 ' , ' delta_t ' };
% xForDisplay = [header;  num2cell( generated_table ) ];
% disp(xForDisplay)

header_1  = {'o_lane' , 'd' , ' vi1 ' , ' vi2 ' , 'p_x(1)' , 'p_x(2)' , 'p_x(3)' , 'p_y(1)' , 'p_y(2)' , 'p_y(3)' , '
        p_y(4)' , 'p_y(5)' , 'p_y(6)' , 'p_y(7)' , 'bad_p_x(1)' , 'bad_p_x(2)' , 'bad_p_x(3)' , 'bad_p_y(1)' , '
        bad_p_y(2)' , 'bad_p_y(3)' , 'bad_p_y(4)' , 'bad_p_y(5)' , 'bad_p_y(6)' , 'bad_p_y(7)' };
% xForDisplay = [header_1;  num2cell( other_table ) ];
% disp(xForDisplay)

% Save the table  generated  to  a  CSV  file  with  a  header
cHeader = header;
commaHeader = [cHeader;repmat({','},1,numel(cHeader))]; %insert commaas
commaHeader = commaHeader(:)';
textHeader  = cell2mat(commaHeader); %cHeader in text with commas
textHeader  = textHeader(1:end−1);

%write header to   file
 fid   = fopen('data/new/control_table.csv' , 'w');
 fprintf ( fid , '%s\n',textHeader);
 fclose ( fid );

%write data to  end of   file
dlmwrite( 'data/new/control_table.csv' , generated_table , '−append');

% Save the other  table  as  well
cHeader = header_1;
commaHeader = [cHeader;repmat({','},1,numel(cHeader))]; %insert commaas
commaHeader = commaHeader(:)';
textHeader  = cell2mat(commaHeader); %cHeader in text with commas
textHeader  = textHeader(1:end−1);

%write header to   file
 fid   = fopen('data/new/other_table.csv' , 'w');
 fprintf ( fid , '%s\n',textHeader);
 fclose ( fid );

%write data to  end of   file
dlmwrite( 'data/new/other_table.csv' , other_table , '−append');

%−−−−−−−−−−−−−− END OF CODE −−−−−−−−−−−−−−−
```

File A.6 linear_control_generator.m

```
% LINEAR_ACCELARATION_GENERATOR - File that generates the accelaration
% table based on the time headway of the car.

% Author: Francisco Girbal Eiras, MSc Computer Science
% University of Oxford, Department of Computer Science
% Email: francisco.eiras@cs.ox.ac.uk
% 24-Apr-2018; Last revision: 6-Jun-2018

%------------- BEGIN CODE --------------

clc

% Possible initial distances between the vehicles
ds = linspace(1,80,80);

% Possible vehicle initial velocity
vs = linspace(15,34,20);

generated_table = zeros(length(ds)*length(vs), 4);
display(sprintf('Generating table of %d entries', size(generated_table,1)))
t1 = cputime;

for d_i = 1:length(ds)
    for v_i = 1:length(vs)

        d = ds(d_i);
        v = vs(v_i);

        delta_crash = d/v;

%         if delta_crash < 0.3
%             a = -1;
%         elseif delta_crash <= 1
%             a = 0;
%         else
%             a = 1;
%         end

        if delta_crash <= 0.2
            a = -2;
        elseif delta_crash <= 0.4
            a = -1;
        elseif delta_crash <= 1
            a = 0;
        elseif delta_crash <= 1.75
            a = 1;
        elseif delta_crash <= 2.5
            a = 2;
        else
            a = 3;
        end

        idx = (d_i - 1)*length(vs) + v_i;

        generated_table(idx,:) = [d,v,round(delta_crash,2),a];
    end
end
```

```
display ( sprintf ('Generated in %.3f seconds', cputime − t1))

% Display the table
header = {'d','v','delta_crash','a'};
xForDisplay = [header; num2cell(generated_table)];
disp(xForDisplay)

% Save the table generated to a CSV file with a header
cHeader = header;
commaHeader = [cHeader;repmat({','},1,numel(cHeader))]; %insert commaas
commaHeader = commaHeader(:)';
textHeader = cell2mat(commaHeader); %cHeader in text with commas
textHeader = textHeader(1:end−1);

%write header to file
fid = fopen('data/mod_acceleration_table.csv','w');
fprintf (fid,'%s\n',textHeader);
fclose (fid);

%write data to end of file
dlmwrite('data/mod_acceleration_table.csv', generated_table,'−append');

%−−−−−−−−−−−−−−−−−−−−−−−−− END OF CODE −−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

File A.7 control.m

```
function [x,y,vx,vy,ay,old_theta_near,old_theta_far,old_thw_car] = control( x, y, vx, vy, lane
    , old_theta_near, old_theta_far, old_thw_car, vehicles_pos )
%CONTROL ACT−R control module

global delta_t width;

% Constants
k_far = 15;%15;
k_near = 3;%3;
k_i = 5;%5;
theta_nmax = 1.57;
k_car = 3;%4;
k_follow = 1;
max_speed = 34; % approximately 120 km/h

std_dev = 2;

if lane == 1
    y_follow = width/4;
else
    y_follow = 3*width/4;
end

near_point = [x + 4, y_follow];
far_point = [x + 10, y_follow];

theta_near = tan((near_point(2) − y)/(near_point(1) − x));
theta_far = tan((far_point(2) − y)/(far_point(1) − x));

Delta_theta_near = theta_near − old_theta_near;
Delta_theta_far = theta_far − old_theta_far;

```

```
Delta_varphi  = k_far* Delta_theta_far  + k_near*Delta_theta_near + k_i*min(theta_near,theta_nmax
    )*delta_t;

% To determine timeheadway, check for vehicles/accidents between the
% car position and far point; the timeheadway corresponds to the
% timeheadway to the closest vehicle (or the end of the road)

min_timeheadway = 3 + normrnd(0,std_dev); % the maximum timeheadway
for i = 1:size(vehicles_pos,1)
    if vehicles_pos(i,1) > x && abs(vehicles_pos(i,2) - y) < 0.2
        th = (vehicles_pos(i,1) - x + normrnd(0,std_dev))/vx;
        min_timeheadway = min(min_timeheadway, th);
    end
end
time_headway = min_timeheadway;

Delta_time_headway = time_headway - old_thw_car;

Delta_psi  = k_car*Delta_time_headway + k_follow*(time_headway - 1)*delta_t;

ax = Delta_psi;
if Delta_varphi ~= 0
    ay = sin(Delta_varphi);
else
    ay = 0;
end

% Update variables

if vx + ax*delta_t < max_speed
    vx = vx + ax*delta_t;
end
vy = vy + ay*delta_t;

x = x + vx*delta_t + 0.5*ax*delta_t^2;
y = y + vy*delta_t + 0.5*ay*delta_t^2;

old_theta_near  = theta_near;
old_theta_far  = theta_far;
old_thw_car  = time_headway;

end
```

File A.8 check_collision.m

```
function happened = check_collision( vehicle_1, vehicle_2, h, w )
%CHECK_COLLISION Verify the occurence of a collision

rect_1 = [vehicle_1(1) - w/2, vehicle_1(2) + h/2, w, h];
rect_2 = [vehicle_2(1) - w/2, vehicle_2(2) + h/2, w, h];

area = rectint(rect_1, rect_2);

happened = (area ~= 0);

end
```

A.3 Decision Making and Monitoring Abstraction (Matlab)

File A.9 decision_making_generator.m

```matlab
% DECISION_MAKING_GENERATOR - File that generates the decision making table
% in terms of probability of changing lanes at certain points

% Author: Francisco Girbal Eiras, MSc Computer Science
% University of Oxford, Department of Computer Science
% Email: francisco.eiras@cs.ox.ac.uk
% 24-Apr-2018; Last revision: 24-Apr-2018

%------------- BEGIN CODE --------------

clc

% Possible initial distances between the vehicles
ds = linspace(1,80,80);

% Possible vehicle initial velocity
vs = linspace(15,34,20);

generated_table = zeros(3*length(ds)*length(vs) + 3*length(ds), 7);
display(sprintf('Generating table of %d entries', size(generated_table,1)))
t1 = cputime;

mean_dt = [0,2,4];
std_dt = [0.6,0.3,1];

exp_param_dt = [1,0.6,0.4];
std_dev = 2;
l = 20;

% the model will have a parameter 'driver_type' which is equal to
% 1 if aggressive, 2 if average or 3 if conservative
for driver_i = 1:3
    for d_i = 1:length(ds)
        for v_i = 1:length(vs)

            d = ds(d_i);
            v = vs(v_i);

            delta_crash = d/v;

%           x = 0:0.1:4;
%           val1 = exp(-exp_param_dt(1)*x);
%           val2 = exp(-exp_param_dt(2)*x);
%           val3 = exp(-exp_param_dt(3)*x);
%
%           figure;
%           hold on;
%           plot(x,val1);
%           plot(x,val2);
%           plot(x,val3);
%
%           pause
```

```
%
%                 return

                x = 1:1:80;
                 dist_small_0  = normcdf(0, d, std_dev);
                 dist_greater_80  = 1 - normcdf(80,d,std_dev);
                P = normcdf(x + 0.5, d, std_dev) - normcdf(x - 0.5, d, std_dev);
                P_IC = exp(-exp_param_dt(driver_i)*x/v);

                pIC = P*P_IC' + (dist_small_0 + dist_greater_80 )*exp(-exp_param_dt(driver_i)*d/v);

                idx = ( driver_i  - 1)*length(ds)*length(vs) + (d_i - 1)*length(vs) + v_i;

                generated_table(idx,:)  = [1, driver_i ,d,v,round(delta_crash,2),round(pIC,2),1-round
    (pIC,2)];
          end
     end
end

mean_dd = [10,40,70];
std_dd = [20,7,20];

log_param_dd = [1000,0.5,0.01];

for  driver_i  = 1:3
     for  d_i  = 1:length(ds)

          d = ds(d_i);
%
          x = 0:0.5:80;
          val1  = 1/log(80*1000+1)*log(1000*x+1);
          val2  = 1/log(80*0.5+1)*log(0.5*x+1);
          val3  = 1/log(80*0.01+1)*log(0.01*x+1);

          figure;
          hold on;
          plot(x,val1);
          plot(x,val2);
          plot(x,val3);

          pause

          pIC = 1/log(80*log_param_dd(driver_i)+1)*log(log_param_dd(driver_i)*d+1);

          idx = 3*length(ds)*length(vs) + ( driver_i  - 1)*length(ds) + d_i;

          generated_table(idx,:)  = [2, driver_i ,d,-1,0,round(pIC,2),1-round(pIC,2)];
     end
end

display ( sprintf ('Generated in %.3f seconds', cputime - t1))

% Display the table
header = {'lane','type','d','v','delta_crash','P_IC','P_nIC'};
xForDisplay  = [header; num2cell(generated_table)];
disp(xForDisplay)

% Save the table generated to a CSV file with a header
cHeader = header;
```

```
commaHeader = [cHeader;repmat({','},1,numel(cHeader))]; %insert commaas
commaHeader = commaHeader(:)';
textHeader = cell2mat(commaHeader); %cHeader in text with commas
textHeader = textHeader(1:end-1);

%write header to  file
 fid  = fopen('data/dm_table.csv' , 'w');
 fprintf ( fid , '%s\n',textHeader);
 fclose ( fid );

%write data to end of  file
dlmwrite( 'data/dm_table.csv' , generated_table , '-append');

%------------- END OF CODE --------------
```

A.4 Probabilistic Two Module Model Generator (Python)

File A.10 model_generator.py

```
# MODEL_GENERATOR - Tranform the generated tables using
# generator.m and decision_making.m into a PRISM file
# with the modules.

# Author: Francisco  Girbal  Eiras , MSc Computer Science
# University  of Oxford, Department of Computer Science
# Email: francisco . eiras@cs . ox . ac . uk
# 25-Apr-2018; Last revision: 9-Jun-2018

import sys , csv , argparse , datetime

 parser=argparse.ArgumentParser(
     description ='''Tranform the generated tables using generator . m and decision_making.m into
     a PRISM file with the two modules. ''' )
 parser . add_argument('lane_change_table' , type=str, help='Table for the lane change part of the
       control module.')
 parser . add_argument('acc_table' , type=str, help='Table for the linear accelaration part of the
       control module.')
 parser . add_argument('dm_table', type=str, help='Table for the decision making module.')
 parser . add_argument('[ driver_type ] ' , type=int, default =2, help='1 = aggressive, 2 = average, 3
       = cautious' )
 parser . add_argument('[v] ' , type=int, default =29, help=' Initial velocity of the vehicle . ' )
 parser . add_argument('[v1] ' , type=int, default =30, help=' Initial velocity of the other vehicle .
       ' )
 parser . add_argument('[x1_0] ' , type=int, default =15, help=' Initial position of the other
       vehicle . ' )
 parser . add_argument('--filename [NAME]', type=str, help='Output name for the file generated. ' )
 args=parser. parse_args ()

 if len(sys . argv) == 8:
   f = open("two_component_model.pm" , "w")
 else :
   f = open("%s.pm"%sys.argv[9], "w")

 driver_type  = sys. argv [4]
v = sys. argv [5]
```

```
v1 = sys.argv[6]
x1_0 = sys.argv[7]
if not (int(v) >= 15 and int(v) <= 34) or not (int(v1) >= 15 and int(v1) <= 34) or not (int(
    x1_0) >= 1 and int(x1_0) <= 500) or int(driver_type) not in [1,2,3]:
  raise ValueError("Input out of range.")

max_control_dist = "43"
max_dm_dist = "80"
crash_dist = "6"

now = datetime.datetime.now()

print('----------------------------------------\nNon-deterministic
    control version of model_generator.py.\n
    ----------------------------------------')

# Write the beginning of the file
f.write("//Model automatically built using model_generator.py for v1 = %s and driver_type = %
    s (to alter these values, run the script again).\n"%(v1,driver_type))
f.write("//Generated on %s.\n\n"%(now.strftime("%d-%m-%Y at %H:%M")))
f.write("dtmc\n\n")
f.write("const int length = 500; // road length\n")
f.write("const int driver_type = 1; // 1 = aggressive, 2 = average, 3 = cautious drivers - do
    not alter this manually!\n")
f.write("const int max_time = 35; // maximum time of experiment\n\n")
f.write("// Other vehicle\n")
f.write("const int v1 = %s; // do not alter this manually!\n"%v1)
f.write("const int x1_0 = %s;\n\n"%x1_0)
f.write("// Environment variables\n")
f.write("global t : [0..max_time] init 0; // time \n")
f.write("global crashed : bool init false; \n\n")
f.write("// Vehicle controlled\n")
f.write("global actrState : [1..2] init 1; // active module: 1 = control (both cars), 2 =
    decision making + monitoring\n")
f.write("global IC : bool init false; // lane changing occuring? \n")
f.write("global x : [0..length] init 0;\n")
f.write("global v : [15..34] init %s;\n"%v)
f.write("global a : [-3..3] init 0;\n")
f.write("global lane : [1..2] init 1;\n\n")
f.write("formula x1 = x1_0 + v1*t;\n")
f.write("formula dist = x1>x?(x1 - x):(x - x1);\n")
f.write("formula positiveDist = (x < length)?x > x1:true;\n\n")

# Decision making + monitoring module
f.write("module Decision_Making_Monitoring\n\n")
f.write("   // If a crash occurs, then nothing else can happen\n")
f.write("  //[] actrState = 2 & crashed -> 1:(crashed' = true);\n\n")

f.write("   // If we are in lane 2, but behind the other vehicle, don't try to pass\n")
f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = false -> 1:(actrState' = 1)
    ;\n\n")

f.write(" // If we are in lane 1, and no vehicle is in front, don't change lanes\n")
f.write(" [] actrState = 2 & !crashed & lane = 1 & positiveDist = true -> 1:(actrState' = 1);\
    n\n")

with open(sys.argv[3]) as csvfile:
  reader = csv.DictReader(csvfile)
  for row in reader:
    # should we change from lane 1 to lane 2? it's based on delta_crash!
    if row["type"] == driver_type and row["lane"] == "1" and row["d"] != max_dm_dist:
      line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = %s & v
    = %s -> %s:(actrState' = 1) & (lC' = true) + %s:(actrState' = 1) & (lC' = false);\n" % (
    row["d"],row["v"],row["P_lC"],row["P_nlC"])
      f.write(line)
    elif row["type"] == driver_type and row["lane"] == "1" and row["d"] == max_dm_dist:
      line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist >= %s &
    v = %s -> %s:(actrState' = 1) & (lC' = true) + %s:(actrState' = 1) & (lC' = false);\n" % (
    row["d"],row["v"],row["P_lC"],row["P_nlC"])
      f.write(line)

    # should we go back to lane 1 from lane 2? it's based on the distance we are at!
    if row["type"] == driver_type and row["lane"] == "2" and row["d"] != max_dm_dist:
      line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist = %s ->
    %s:(actrState' = 1) & (lC' = true) + %s:(actrState' = 1) & (lC' = false);\n" % (row["d"],row
    ["P_lC"],row["P_nlC"])
      f.write(line)
    elif row["type"] == driver_type and row["lane"] == "2" and row["d"] == max_dm_dist:
      line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >= %s
    -> %s:(actrState' = 1) & (lC' = true) + %s:(actrState' = 1) & (lC' = false);\n" % (row["d"
    ],row["P_lC"],row["P_nlC"])
      f.write(line)

f.write("endmodule\n\n")

# Control module
f.write("module Control\n\n")

f.write("  // If we are in lane 1, and no lane change was decided, continue forward (which
    might result in crash)\n")
f.write("  // The vehicle is behind the other driver (positiveDist = false, x < x1)\n")
with open(sys.argv[2]) as csvfile:
  reader = csv.DictReader(csvfile)
  for row in reader:
    if row["d"] != max_dm_dist:
      line = "  [] actrState = 1 & !crashed & !lC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 &
    dist = %s & v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s) & (
    actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
      line = "  [] actrState = 1 & !crashed & !lC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist = %s &
    v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s) & (actrState' = 2);\n" %
    (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
      line = "  [] actrState = 1 & !crashed & !lC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist = %s &
    v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s) & (actrState' = 2);\n" %
    (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
    elif row["d"] == max_dm_dist:
      line = "  [] actrState = 1 & !crashed & !lC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 &
    dist >= %s & v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s) & (
    actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
      line = "  [] actrState = 1 & !crashed & !lC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist >= %s
```

```
        & v = %s −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s) & (actrState' = 2);\n"
        % (crash_dist,row["d"],row["v"],row["a"])
            f.write(line)
            line = "   [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t <
        max_time & positiveDist = false & (x1 + v1 − x − v) >= %s & v + a <= 15 & dist >= %s
        & v = %s −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s) & (actrState' = 2);\n"
        % (crash_dist,row["d"],row["v"],row["a"])
            f.write(line)

    f.write("\n [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) >= %s −> 1:(x' = length) & (t' = t + 1) & (
        actrState' = 2);\n"%crash_dist)
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s −> 1:(x' = length) & (t' = t + 1) & (crashed'
         = true) & (actrState' = 2);\n\n"%crash_dist)

    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s & v + a < 34 & v + a > 15 −> 1:(x' = x + v
        ) & (t' = t + 1) & (v' = v + a) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s & v + a >= 34 −> 1:(x' = x + v) & (t' = t
        + 1) & (v' = 34) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s & v + a <= 15 −> 1:(x' = x + v) & (t' = t
        + 1) & (v' = 15) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)

    f.write("   // The vehicle is in front of the other driver ( positiveDist = true, x > x1)\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = true & v + a < 34 & v + a > 15 −> 1:(x' = x + v) & (t' = t + 1) & (v' = v
        + a) & (a' = 0) & (actrState' = 2);\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = true & v + a >= 34 −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (
        actrState' = 2);\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = true & v + a <= 15 −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (
        actrState' = 2);\n\n")

    f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
        positiveDist = true −> 1:(x' = length) & (t' = t + 1) & (actrState' = 2);\n\n")

    f.write("   // If we are in lane 2, and no lane change was decided, continue forward\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a < 34 & v + a > 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
        a) & (a' = 0) & (actrState' = 2);\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a >= 34 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0) &
         (actrState' = 2);\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a <= 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0) &
         (actrState' = 2);\n\n")

    f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a < 34 & v + a > 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
         a) & (a' = 0) & (actrState' = 2);\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a >= 34 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0)
        & (actrState' = 2);\n")
    f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a <= 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0)
        & (actrState' = 2);\n\n")

f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a < 34 & v + a > 15 -> 1:(x' = length) & (t' = t + 1) & (v' = v + a) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a >= 34 -> 1:(x' = length) & (t' = t + 1) & (v' = 34) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a <= 15 -> 1:(x' = length) & (t' = t + 1) & (v' = 15) & (actrState' = 2);\n\n")

with open(sys.argv[1]) as csvfile:
  reader = csv.DictReader( csvfile )
  for row in reader:
    probCrash = float(row["Acc?"])

    if probCrash != 0 and probCrash != 1:
      if row["d"] != max_control_dist and row["vi2"] == v1:
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write( line )
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write( line )
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
        f.write( line )
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
        f.write( line )
      elif row["d"] == max_control_dist and row["vi2"] == v1:
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write( line )
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
```

```
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], probCrash,1-probCrash,row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"], row["delta_t"], probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
            f.write(line)
    elif probCrash == 0:
        if row["d"] != max_control_dist and row["vi2"] == v1:
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], row["delta_t"], str(3 - int(row["o_lane"])))
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], row["vf1"], row["delta_t"], str(3 - int(row["o_lane"])))
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"], row["vf1"], str(3 - int(row["o_lane"])))
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], row["vf1"], str(3 - int(row["o_lane"])))
            f.write(line)
        elif row["d"] == max_control_dist and row["vi2"] == v1:
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], row["delta_t"], str(3 - int(row["o_lane"])))
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
            f.write(line)
```

```python
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], row["vf1"], str(3 - int(row["o_lane"])))
                f.write(line)
        else:
            if row["d"] != max_control_dist and row["vi2"] == v1:
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)
            elif row["d"] == max_control_dist and row["vi2"] == v1:
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
                f.write(line)


f.write("\nendmodule")

f.close()
```

A.5 Prism Automatic Model Checker (Python)

File A.11 prism_automatic_model_checker.py

```python
# AUTOMATIC_MODEL_CHECKER - Executes the script model_generator.py
# for some given parameters in order to build the model and perform
# model checking subquentially.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 26-Apr-2018; Last revision: 3-May-2018

import sys, os, subprocess, csv, argparse

parser=argparse.ArgumentParser(
    description='''Executes the script model_generator.py for some given parameters in order
    to build the model and perform model checking subquentially. ''' )
parser.add_argument('properties_file', type=str, help='File of the properties to be checked (
    PCTL or LTL).')
parser.add_argument('[v]', type=int, default=29, help='Initial velocity of the vehicle.')
parser.add_argument('[v1]', type=int, default=30, help='Initial velocity of the other vehicle.
    ')
parser.add_argument('[x1_0]', type=int, default=15, help='Initial position of the other
     vehicle.')
parser.add_argument('--clean [VALUE]', type=str, help='If [VALUE] = False, then generated
     files (model and individual results) will be maintained (default = True).')
args=parser.parse_args()

types = ['aggressive','average','cautious']
res = {}

source_path = ""
path = "results/"
 properties_file = sys.argv[1]
v = sys.argv[2]
v1 = sys.argv[3]
x1_0 = sys.argv[4]

if len(sys.argv) > 5 and sys.argv[6] == "False":
   cleaning_up = False
else:
   cleaning_up = True

num_properties = sum(1 for line in open( properties_file ))
with open( properties_file ) as f1:
      props = f1.readlines()
props = [x.strip() for x in props]

for driver_type in range(1,4):
   print('------ %s driver ------'%types[driver_type-1])

   filename = "gen_model_%s_%s_%s_%s"%(driver_type,v,v1,x1_0)
   r_filename = "results_%s_%s_%s_%s_%s"%(driver_type,properties_file,v,v1,x1_0)

   # Construct the file
   print('Generating the model...')
   os.system('python3 %smodel_generator.py %smodel_tables/control_table.csv %smodel_tables/
      acc_table.csv %smodel_tables/dm_table.csv %s %s %s %s --filename %s%s > /dev/null'%(
      source_path,source_path,source_path,source_path,driver_type,v,v1,x1_0,path,filename))

   print('Building the model and performing model checking...')
   subprocess.run("prism %s%s.pm properties.pctl -exportresults %s%s.txt &> /dev/null"%(path,
      filename,path,r_filename), shell=True)

   print('Obtaining the results ...')

   f = open("%s%s.txt"%(path,r_filename), "r")

```

```
  if num_properties == 1:
    f.readline()
    probability = f.readline()
    f.close()

    probability = float(probability[:-1])
    res[driver_type] = [props[0], probability]
  else:
    f.readline()
    f.readline()
    probability = f.readline()

    probability = float(probability[:-1])
    res[driver_type] = [[props[0], probability]]

    for i in range(1,num_properties):
      f.readline()
      f.readline()
      f.readline()
      probability = f.readline()

      probability = float(probability[:-1])
      res[driver_type].append([props[i], probability])

  f.close()

  if cleaning_up == True:
    print('Cleaning up...')
    os.system('rm %s%s.pm'%(path,filename))
    os.system('rm %s%s.txt'%(path,r_filename))

with open('%s%s_%s_%s_%s.csv'%(path,properties_file,v,v1,x1_0), 'w') as csvfile:
    fieldnames = ['type_driver', 'property', 'probability']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    for key,val in res.items():
      for propty in val:
        writer.writerow({'type_driver': key, 'property': propty[0], 'probability': propty[1]})

print('Done.')
```

A.6 Storm Automatic Model Checker (Python)

File A.12 storm_model_checker.py

```
# STORM_MODEL_CHECKER - Executes the script model_generator.py
# for some given parameters in order to build the model and perform
# model checking subquentially in storm (changed to use the path source/).

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 3-Jun-2018; Last revision: 3-Jun-2018

import sys, os, subprocess, csv, argparse
from datetime import datetime
```

```
startTime = datetime.now()

parser=argparse.ArgumentParser(
    description='''Executes the script model_generator.py for some given parameters in order
    to build the model and perform model checking subquentially in storm (changed to use the
    path source/). ''' )
parser.add_argument(' properties_file ', type=str, help='File of the properties to be checked (
    PCTL or LTL).')
parser.add_argument('[v]', type=int, default=29, help=' Initial velocity of the vehicle.')
parser.add_argument('[v1]', type=int, default=30, help=' Initial velocity of the other vehicle.
    ')
parser.add_argument('[x1_0]', type=int, default=15, help=' Initial position of the other
    vehicle.')
parser.add_argument('--clean [VALUE]', type=str, help='If [VALUE] = False, then generated
    files (model and individual results) will be maintained (default = True).')
parser.add_argument('--path [PATH]', type=str, help='Generated file will be saved in PATH.')
args=parser.parse_args()

types = ['aggressive','average','cautious']
res = {}

 properties_file = sys.argv[1]
v = sys.argv[2]
v1 = sys.argv[3]
x1_0 = sys.argv[4]
path = "gen_files"

progress = True

if len(sys.argv) > 5 and sys.argv[5] == "--clean" and sys.argv[6] == "False":
  cleaning_up = False
else:
  cleaning_up = True

if len(sys.argv) > 5 and sys.argv[5] == "--path":
  path = sys.argv[6]

if progress == True:
  toolbar_width = 9
  sys.stdout.write("Progress: [%s]" % (" " * toolbar_width))
  sys.stdout.flush()
  sys.stdout.write("\b" * (toolbar_width+1))

for driver_type in range(1,4):
  if progress == False:
    print('------- %s driver -------'%types[driver_type-1])

  filename = "gen_model_%s_%s_%s_%s"%(driver_type,v,v1,x1_0)
  r_filename = "results_%s_%s_%s_%s_%s"%(driver_type,properties_file,v,v1,x1_0)

  # Construct the file
  if progress == False:
    print('Generating the model...')
  os.system('python3 source/model_generator.py source/model_tables/control_table.csv source/
    model_tables/acc_table.csv source/model_tables/dm_table.csv %s %s %s %s --filename
    source/%s > /dev/null'%(driver_type,v,v1,x1_0,filename))

  if progress == True:
    sys.stdout.write("=")
```

```
        sys.stdout.flush()

    if progress == False:
        print('Building the model and performing model checking...')
    proc = subprocess.Popen('storm --prism source/%s.pm --prop source/%s'%(filename,
        properties_file), stdout=subprocess.PIPE, stderr=subprocess.STDOUT, shell=True)
    output = str(proc.stdout.read())

    if progress == True:
        sys.stdout.write("=")
        sys.stdout.flush()

    idx = output.find('\\n
        ------------------------------------------------------------------
        \\n', 500)
    output = output[idx + 69:]
    idx3 = 0

    while idx3 != -1:
        idx1 = output.find('\\n')
        first_line = output[0:idx1]
        idx2 = output.find('\\n', idx1+1)
        second_line = output[idx1+2:idx2+2]
        idx3 = output.find('\\n\\n', idx2+1)

        prop = first_line[24: first_line.find(' ... ')-1]
        val = second_line[29: second_line.find('\\n')]

        try:
            probability = float(val)
        except ValueError:
            print('------------------------------------\n\n')
            print(output)
            os.system('rm source/%s.pm'%filename)
            exit()

        if not driver_type in res.keys():
            res[driver_type] = [[prop, probability]]
        else:
            res[driver_type].append([prop, probability])
        if progress == False:
            print(prop + ' : ' + str(probability))
        output = output[idx3+4:]

    if cleaning_up == True:
        if progress == False:
            print('Cleaning up...')
        os.system('rm source/%s.pm'%filename)

    if progress == True:
        sys.stdout.write("=")
        sys.stdout.flush()

  if progress == True:
    sys.stdout.write("\n")

with open('%s/r_%s_%s_%s.csv'%(path,v,v1,x1_0), 'w') as csvfile:
    fieldnames = ['type_driver', 'property', 'probability']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    for key,val in res.items():
      for propty in val:
        writer.writerow({'type_driver': key, 'property': propty[0], 'probability': propty[1]})

if progress == False:
  print('Done.')
  print(datetime.now() - startTime)
```

A.7 Simulator (Python)

File A.13 simulate.py

```
# SIMULATE - Given a PRISM model, obtain a sample
# trace using get_trace.py and simulate it.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 12-May-2018; Last revision: 2-Aug-2018

import pygame
import sys, os, csv, argparse, subprocess

parser=argparse.ArgumentParser(
    description='''Given a PRISM model, obtain a sample trace using get_trace.py and simulate
    it.''')
parser.add_argument('model', type=str, help='The model from which to obtain the sample
    execution (built using model_generator.py).')
parser.add_argument('-x', '--times', type=float, default=1, help='Execution is X times faster.
    ')
parser.add_argument('-rt', '--read_trace', action="store_true", help='Read an existing trace.'
    )
args=parser.parse_args()

model = args.model
speed = args.times

# Helper functions
_image_library = {}
def get_image(path):
  global _image_library
  image = _image_library.get(path)
  if image == None:
      canonicalized_path = path.replace('/', os.sep).replace('\\', os.sep)
      image = pygame.image.load('graphics/' + canonicalized_path)
      if path == "background.png":
        image = pygame.transform.scale(image, (1000,400))
      if path == "car_red.png" or path == "car_grey.png":
        image = pygame.transform.scale(image, (20,11))
      _image_library[path] = image
  return image

def render_centered(screen, text, color):
  label = crashed_font.render(text, 1, color)
  size = crashed_font.size(text)
  screen.blit(label, (500 - size[0]/2.0, 200 - size[1]/2.0))

def detect_crash(x1, y1, x2, y2, w, h):
  x = max(x1 - w/2, x2 - w/2)
  y = max(y1 - h/2, y2 - h/2)
  w_new = min(x1 + w/2, x2 + w/2) - x
  h_new = min(y1 + h/2, y2 + h/2) - y
  if w_new <= 0 or h_new <= 0:
    return False
  return True

# Main code
v1 = -1
x1_0 = -1

f = open(model, 'r')

for line in f:
  if "const int v1" in line:
    nums = line.split()
    v1 = int(nums[4][:-1])
  elif "const int x1_0" in line:
    nums = line.split()
    x1_0 = int(nums[4][:-1])
    break

if v1 == -1 or x1_0 == -1:
  print("Error: model not generated using the model_generator.py (problems reading v1 and x1_0
    )")
  quit()

f.close()

# Generate the trace
if not args.read_trace:
  subprocess.run("python3 get_trace.py %s %d %d"%(model, v1, x1_0), shell=True)

pygame.init()
pygame.display.set_caption('Sample Path Simulation')
screen = pygame.display.set_mode((1000,400))
myfont = pygame.font.SysFont("monospaced", 20)
crashed_font = pygame.font.SysFont("monospaced", 45)
time_font = pygame.font.SysFont("monospaced", 60)

csvfile = open('gen_trace.csv')
reader = csv.DictReader(csvfile)
comm = next(reader)

# Read the first command
t_init = 0
x_init = 0
y_init = 1.8
t_end = int(comm['t_end'])
type_comm = int(comm['type'])
curr_v = int(comm['v'])
crashed = bool(int(comm['crashed']))
x_coeffs = [float(comm['x_t_1']), float(comm['x_t_2']), float(comm['x_t_3'])]
y_coeffs = [float(comm['y_t_1']), float(comm['y_t_2']), float(comm['y_t_3']), float(comm['
    y_t_4']), float(comm['y_t_5']), float(comm['y_t_6']), float(comm['y_t_7'])]
```

```

# Setup of the other variables
x0 = x1_0
v0 = v1

t = 0.0
deltaT = 0.05
scaleX = 2
scaleY = 5
c_height = 9
c_width = 20

x = x_init
y = y_init

update_action = False
permanent = False
force_crash = False

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
      update_action = not update_action

  screen.fill((255,255,255))
  screen.blit(get_image('background.png'), (0, 0))

  screen.blit(get_image('car_red.png'), (x*scaleX - c_width/2, 218 - y*scaleY - c_height/2))

  # Other vehicle
  screen.blit(get_image('car_grey.png'), ((x0 + t*v0)*scaleX - c_width/2, 218 - 1.8*scaleY -
    c_height/2))

  # Display info
  time_label = time_font.render("%2.1fs"%t, 1, (0,0,0))
  screen.blit(time_label, (175, 35))

  main_vahicle_label = myfont.render("Main vehicle", 1, (128,128,128))
  screen.blit(main_vahicle_label, (325, 25))

  x_label = myfont.render("x Position: %dm"%x, 1, (0,0,0))
  screen.blit(x_label, (325, 45))

  y_label = myfont.render("y Position: %.2fm"%y, 1, (0,0,0))
  screen.blit(y_label, (325, 65))

  other_vahicle_label = myfont.render("Other vehicle", 1, (128,128,128))
  screen.blit(other_vahicle_label, (475, 25))

  x_label = myfont.render("x Position: %dm"%(x0 + t*v0), 1, (0,0,0))
  screen.blit(x_label, (475, 45))

  y_label = myfont.render("y Position: 1.8m", 1, (0,0,0))
  screen.blit(y_label, (475, 65))

  if detect_crash(x,y,(x0 + t*v0),1.8,4.8,1.9) == True or force_crash == True:
        render_centered(screen, "Crashed", (0,0,0))
        permanent = True

    # Main vehicle
    if update_action == True and permanent == False:
        if t < t_end:
            if type_comm == 1:
                x = x + curr_v*deltaT
            elif type_comm == 2:
                curr_t = t - t_init
                x = x_init + (x_coeffs[0]* curr_t**2 + x_coeffs[1]* curr_t + x_coeffs[2])
                y = (y_coeffs[0]* curr_t**6 + y_coeffs[1]* curr_t**5 + y_coeffs[2]* curr_t**4 + y_coeffs[3]* curr_t**3 + y_coeffs[4]* curr_t**2 + y_coeffs[5]* curr_t + y_coeffs[6])

        if t >= t_end:
            if crashed == True:
                permanent = True
                force_crash = True
            else:
                try:
                    comm = next(reader)
                except:
                    print('Done')
                    break

                # Read the next command
                t_init = t
                x_init = x
                y_init = y
                t_end = int(comm['t_end'])
                type_comm = int(comm['type'])
                curr_v = int(comm['v'])
                crashed = bool(int(comm['crashed']))
                x_coeffs = [float(comm['x_t_1']), float(comm['x_t_2']), float(comm['x_t_3'])]
                y_coeffs = [float(comm['y_t_1']), float(comm['y_t_2']), float(comm['y_t_3']), float(comm['y_t_4']), float(comm['y_t_5']), float(comm['y_t_6']), float(comm['y_t_7'])]

        if x >= 500 or t >= 30:
            update_action = False
            permanent = True

        t = t + deltaT

    elif crashed == False and update_action == False and permanent == False:
        if x == 0:
            render_centered(screen, "Press SPACE to start", (0,0,0))
        else:
            render_centered(screen, "Press SPACE to continue", (0,0,0))
    elif crashed == False and permanent == True:
        render_centered(screen, "Done", (0,0,0))

    pygame.display.flip()
    pygame.time.wait(int(1000*deltaT/speed))

pygame.quit()
```

File A.14 get_trace.py

```
# GET_TRACE - Given a PRISM model, obtain a sample
# trace and modify it .

# Author: Francisco Girbal Eiras , MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco . eiras@cs .ox.ac.uk
# 14−May−2018; Last revision: 14−May−2018

import sys, os, subprocess, csv, argparse

parser=argparse.ArgumentParser(
    description='''Given a PRISM model, obtain a sample trace and modify it . ''' )
parser .add_argument('model.pm', type=str, help='The model from which to obtain the sample
    execution ( built using model_generator.py) . ')
parser .add_argument('[v1] ' , type=int, help=' Initial velocity of the other vehicle . ')
parser .add_argument('[x1_0] ' , type=int, help=' Initial position of the other vehicle . ')
args=parser. parse_args ()

cleaning_up = False

v1 = int(sys .argv [2])
x1_0 = int(sys .argv [3])

# Construct the file
print ( 'Obtaining the sample path ... ' )
subprocess .run(”prism %s −simpath 'deadlock,sep=comma' path1.txt &> /dev/null” %sys.argv[1],
    shell=True)

# Read the generated text file
print ( 'Read the generated file and modify the trace file to become readable in simulation ... ' )

 csvfile = open('gen_trace.csv' , 'w')
fieldnames = ['t_end' , 'type' , 'v' , 'crashed' , 'lane' , 'x_t_1 ' , 'x_t_2 ' , 'x_t_3 ' , 'y_t_1 ' , 'y_t_2 ' , '
    y_t_3 ' , 'y_t_4 ' , 'y_t_5 ' , 'y_t_6 ' , 'y_t_7 ']
writer = csv.DictWriter( csvfile , fieldnames=fieldnames)

writer . writeheader ()

next_change_lanes = False
curr_v = 0
curr_t = 0
curr_x = 0

with open('path1.txt ') as csvfile :
  reader = csv.DictReader( csvfile )
  # Skip the first line
  next( reader )
  for row in reader :
    if row[' action ' ] == ”Decision_Making_Monitoring” and row[”IC”] == ”true”:
      next_change_lanes = True
      curr_v = int(row['v' ])
      curr_t = int(row[' t ' ])
      curr_x = int(row['x' ])

    if row[' action ' ] == ”Control” and next_change_lanes == False:
      if row['crashed' ] == 'false' :
        writer .writerow({'t_end': row['t' ], 'type': '1', 'v': row['v' ], 'crashed': '0', 'lane'
    : row[' lane' ], 'x_t_1 ': '0', 'x_t_2 ': '0', 'x_t_3 ': '0', 'y_t_1 ': '0', 'y_t_2 ': '0', '
    y_t_3 ': '0', 'y_t_4 ': '0', 'y_t_5 ': '0', 'y_t_6 ': '0', 'y_t_7 ': '0'})
      else :
```

```
                writer.writerow({'t_end': row['t'], 'type': '1', 'v': row['v'], 'crashed': '1', 'lane': row['lane'], 'x_t_1': '0', 'x_t_2': '0', 'x_t_3': '0', 'y_t_1': '0', 'y_t_2': '0', 'y_t_3': '0', 'y_t_4': '0', 'y_t_5': '0', 'y_t_6': '0', 'y_t_7': '0'})

        if row['action'] == "Control" and next_change_lanes == True:
            lane = int(row['lane'])
            o_lane = 3 - lane
            idx_line = (2-lane)*20*20*43 + (v1 - 15)*20*43 + (curr_v - 15)*43 + min(abs(x1_0 + v1*curr_t - curr_x), 43)

            infile = open("data/other_table.csv")
            r = csv.DictReader( infile )
            for i in range( idx_line -1):
                next(r)
            this_row = next(r)

            if row['crashed'] == 'false':
                writer.writerow({'t_end': row['t'], 'type': '2', 'v': row['v'], 'crashed': '0', 'lane': o_lane, 'x_t_1': this_row['p_x(1)'], 'x_t_2': this_row['p_x(2)'], 'x_t_3': this_row['p_x(3)'], 'y_t_1': this_row['p_y(1)'], 'y_t_2': this_row['p_y(2)'], 'y_t_3': this_row['p_y(3)'], 'y_t_4': this_row['p_y(4)'], 'y_t_5': this_row['p_y(5)'], 'y_t_6': this_row['p_y(6)'], 'y_t_7': this_row['p_y(7)']})
            else:
                writer.writerow({'t_end': row['t'], 'type': '2', 'v': row['v'], 'crashed': '1', 'lane': o_lane, 'x_t_1': this_row['p_x(1)'], 'x_t_2': this_row['p_x(2)'], 'x_t_3': this_row['p_x(3)'], 'y_t_1': this_row['p_y(1)'], 'y_t_2': this_row['p_y(2)'], 'y_t_3': this_row['p_y(3)'], 'y_t_4': this_row['p_y(4)'], 'y_t_5': this_row['p_y(5)'], 'y_t_6': this_row['p_y(6)'], 'y_t_7': this_row['p_y(7)']})

            next_change_lanes = False


if cleaning_up == True:
    print('Cleaning up...')
    os.system('rm path1.txt')
```

Appendix B

Code for the Advanced Driver Assistance Systems

B.1 MDP Model Generators for Different ADAS (Python)

B.1.1 Fully Compliant Drivers in Decision Making

File B.1 model_generator.py

```
# MDP_GENERATOR - Transform the tables into the MDP
# in order to perform multi-objective synthesis.
#
# VERSION:
# - decision making: lane change, deccelarate or do nothing
#
# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 26-Jun-2018; Last revision: 26-Jun-2018

import sys, csv, argparse, datetime

parser=argparse.ArgumentParser(
    description='''Transform the tables into the MDP in order to perform multi-objective
    synthesis. ''')
parser.add_argument('lane_change_table', type=str, help='Table for the lane change part of the
     control module.')
parser.add_argument('acc_table', type=str, help='Table for the linear accelaration part of the
     control module.')
parser.add_argument('[v]', type=int, default=29, help='Initial velocity of the vehicle.')
parser.add_argument('[v1]', type=int, default=30, help='Initial velocity of the other vehicle.
    ')
parser.add_argument('[x1_0]', type=int, default=15, help='Initial position of the other
    vehicle.')
```

```
parser.add_argument('--filename [NAME]', type=str, help='Output name for the file generated.')
args=parser.parse_args()

if len(sys.argv) == 6:
  f = open("mdp_model.pm", "w")
else:
  f = open("%s.pm"%sys.argv[8], "w")

v = sys.argv[3]
v1 = sys.argv[4]
x1_0 = sys.argv[5]
if not (int(v) >= 15 and int(v) <= 34) or not (int(v1) >= 15 and int(v1) <= 34) or not (int(
    x1_0) >= 1 and int(x1_0) <= 500):
  raise ValueError("Input out of range.")

max_control_dist = "43"
max_dm_dist = "80"
crash_dist = "6"

now = datetime.datetime.now()

# Write the beginning of the file
f.write("//MDP automatically built using mdp_generator.py for v1 = %s (to alter this value,
    run the script again).\n"%v1)
f.write("//Generated on %s.\n\n"%(now.strftime("%d-%m-%Y at %H:%M")))
f.write("//Version: decision making: lane change, deccelarate or do nothing\n\n")

f.write("mdp\n\n")
f.write("const int length = 500; // road length\n")
f.write("const int max_time = 30; // maximum time of experiment\n\n")
f.write("// Other vehicle\n")
f.write("const int v1 = %s; // do not alter this manually!\n"%v1)
f.write("const int x1_0 = %s;\n\n"%x1_0)
f.write("// Environment variables\n")
f.write("global t : [0..max_time] init 0; // time \n")
f.write("global crashed : bool init false; \n\n")
f.write("// Vehicle controlled\n")
f.write("global actrState : [1..2] init 1; // active module: 1 = control (both cars), 2 =
    decision making + monitoring\n")
f.write("global lC : bool init false; // lane changing occuring? \n")
f.write("global x : [0..length] init 0;\n")
f.write("global v : [15..34] init %s;\n"%v)
f.write("global a : [-2..3] init 0;\n")
f.write("global lane : [1..2] init 1;\n\n")
f.write("formula x1 = x1_0 + v1*t;\n")
f.write("formula dist = x1>x?(x1 - x):(x - x1);\n")
f.write("formula positiveDist = (x < length)?x > x1:true;\n\n")

# Decision making + monitoring module
f.write("module Decision_Making_Monitoring\n\n")
f.write("   // If a crash occurs, then nothing else can happen\n")
f.write(" //[] actrState = 2 & crashed -> 1:(crashed' = true);\n\n")

f.write("   // If we are in lane 2, but behind the other vehicle, don't try to pass\n")
f.write(" [] actrState = 2 & !crashed & lane = 2 & positiveDist = false & x < length -> 1:(
    actrState' = 1);\n\n")

f.write(" // If we are in lane 1, and no vehicle is in front, don't change lanes\n")
f.write(" [] actrState = 2 & !crashed & lane = 1 & positiveDist = true & x < length -> 1:(
    actrState' = 1);\n\n")
f.write(" [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(IC'
     = true) & (actrState' = 1);\n")
f.write(" [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(IC'
     = false) & (a' = -1) & (actrState' = 1);\n")
f.write(" [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(
    actrState' = 1);\n\n")

f.write(" [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(IC'
    = true) & (actrState' = 1);\n")
f.write(" [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(IC'
    = false) & (a' = -1) & (actrState' = 1);\n")
f.write(" [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(
    actrState' = 1);\n\n")

f.write("endmodule\n\n")

# Control module
f.write("module Control\n\n")

f.write("   // If we are in lane 1, and no lane change was decided, continue forward (which
    might result in crash)\n")
f.write("   // The vehicle is behind the other driver (positiveDist = false, x < x1)\n")
with open(sys.argv[2]) as csvfile:
  reader = csv.DictReader(csvfile)
  for row in reader:
    if row["d"] != max_dm_dist:
      line = "   [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 &
    dist = %s & v = %s  -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s) & (
    actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
      line = "   [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist = %s &
    v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s) & (actrState' = 2);\n" %
    (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
      line = "   [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist = %s &
    v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s) & (actrState' = 2);\n" %
    (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
    elif row["d"] == max_dm_dist:
      line = "   [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 &
    dist >= %s & v = %s  -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s) & (
    actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
      line = "   [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist >= %s
    & v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s) & (actrState' = 2);\n"
    % (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)
      line = "   [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
    max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist >= %s
    & v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s) & (actrState' = 2);\n"
    % (crash_dist,row["d"],row["v"],row["a"])
      f.write(line)

```

```
f.write("\n [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
      positiveDist = false & (x1 + v1 − x − v) >= %s −> 1:(x' = length) & (t' = t + 1) & (
      actrState' = 2);\n"%crash_dist)
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
      positiveDist = false & (x1 + v1 − x − v) < %s −> 1:(x' = length) & (t' = t + 1) & (crashed'
       = true) & (actrState' = 2);\n\n"%crash_dist)

f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
      positiveDist = false & (x1 + v1 − x − v) < %s & v + a < 34 & v + a > 15 −> 1:(x' = x + v
      ) & (t' = t + 1) & (v' = v + a) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
      positiveDist = false & (x1 + v1 − x − v) < %s & v + a >= 34 −> 1:(x' = x + v) & (t' = t
      + 1) & (v' = 34) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
      positiveDist = false & (x1 + v1 − x − v) < %s & v + a <= 15 −> 1:(x' = x + v) & (t' = t
      + 1) & (v' = 15) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)

f.write("    // The vehicle is in front of the other driver ( positiveDist = true, x > x1)\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
      positiveDist = true & v + a < 34 & v + a > 15 −> 1:(x' = x + v) & (t' = t + 1) & (v' = v
      + a) & (a' = 0) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
      positiveDist = true & v + a >= 34 −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (
      actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
      positiveDist = true & v + a <= 15 −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (
      actrState' = 2);\n\n")

f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
      positiveDist = true −> 1:(x' = length) & (t' = t + 1) & (actrState' = 2);\n\n")

f.write("    // If we are in lane 2, and no lane change was decided, continue forward\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a < 34 & v + a > 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
      a) & (a' = 0) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a >= 34 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0) &
       (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a <= 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0) &
       (actrState' = 2);\n\n")

f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a < 34 & v + a > 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
       a) & (a' = 0) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a >= 34 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0)
      & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a <= 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0)
      & (actrState' = 2);\n\n")

f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length − v & t < max_time & v +
       a < 34 & v + a > 15 −> 1:(x' = length) & (t' = t + 1) & (v' = v + a) & (actrState' = 2);\
      n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length − v & t < max_time & v +
       a >= 34 −> 1:(x' = length) & (t' = t + 1) & (v' = 34) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length − v & t < max_time & v +
       a <= 15 −> 1:(x' = length) & (t' = t + 1) & (v' = 15) & (actrState' = 2);\n\n")
```

```

with open(sys.argv[1]) as csvfile:
    reader = csv.DictReader( csvfile )
    for row in reader:
        probCrash = float(row["Acc?"])

        if probCrash != 0 and probCrash != 1:
            if row["d"] != max_control_dist and row["vi2"] == v1:
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
                f.write(line)
            elif row["d"] == max_control_dist and row["vi2"] == v1:
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
```

```
length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
        f.write(line)
    elif probCrash == 0:
      if row["d"] != max_control_dist and row["vi2"] == v1:
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], str(3 - int(row["o_lane"])))
        f.write(line)
      elif row["d"] == max_control_dist and row["vi2"] == v1:
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], str(3 - int(row["o_lane"])))
        f.write(line)
    else:
      if row["d"] != max_control_dist and row["vi2"] == v1:
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
        length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
        false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
        length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false
        );\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
        length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false)
        ;\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
        length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false)
        ;\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
          elif row["d"] == max_control_dist and row["vi2"] == v1:
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x
        <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
        false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
        length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
        false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x
        <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
        false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
        length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false)
        ;\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)


f.write("\nendmodule")

f.close()
```

B.1.2 Partially Compliant Drivers in Decision Making

File B.2 model_generator.py

```
# MDP_GENERATOR - Transform the tables into the MDP
# in order to perform multi-objective synthesis.
#
# VERSION:
# - imperfect decision making (gamma)
#
# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 29-Jun-2018; Last revision: 29-Jun-2018

import sys, csv, argparse, datetime

parser=argparse.ArgumentParser(
    description='''Transform the tables into the MDP in order to perform multi-objective
    synthesis . ''' )
parser.add_argument('lane_change_table', type=str, help='Table for the lane change part of the
      control module.')
parser.add_argument('acc_table', type=str, help='Table for the linear accelaration part of the
      control module.')
parser.add_argument('dm_table', type=str, help='Table for the decision making module.')
parser.add_argument('driver_type', type=int, default=2, help='1 = aggressive, 2 = average, 3 =
      cautious')
parser.add_argument('v', type=str, help='Initial velocity of the vehicle.')
parser.add_argument('v1', type=str, help='Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=str, help='Initial position of the other vehicle.')
parser.add_argument('--filename', '-f', type=str, default="mdp_model", help='Output name for
      the file generated.')
args=parser.parse_args()

f = open("%s.pm"%args.filename, "w")

v = args.v
v1 = args.v1
x1_0 = args.x1_0
driver_type = args.driver_type

if not (int(v) >= 15 and int(v) <= 34) or not (int(v1) >= 15 and int(v1) <= 34) or not (int(
    x1_0) >= 1 and int(x1_0) <= 500) or not (driver_type >= 1 and driver_type <= 3):
  raise ValueError("Input out of range.")

driver_type = str(driver_type)
max_control_dist = "43"
max_dm_dist = "80"
crash_dist = "6"
gamma = 0.10

now = datetime.datetime.now()

# Write the beginning of the file
f.write("//MDP automatically built using mdp_generator.py for v1 = %s (to alter this value,
    run the script again).\n"%v1)
f.write("//Generated on %s.\n\n"%(now.strftime("%d-%m-%Y at %H:%M")))
f.write("//Version: imperfect decision making, gamma = %.2f\n\n"%gamma)

f.write("mdp\n\n")
f.write("const int length = 500; // road length\n")
f.write("const int max_time = 35; // maximum time of experiment\n")
f.write("const double gamma = %.2f; // gamma value\n\n"%gamma)
f.write("// Other vehicle\n")
f.write("const int v1 = %s; // do not alter this manually!\n"%v1)
f.write("const int x1_0 = %s;\n\n"%x1_0)
f.write("// Environment variables\n")
f.write("global t : [0..max_time] init 0; // time \n")
f.write("global crashed : bool init false; \n\n")
f.write("// Vehicle controlled\n")
f.write("global actrState : [1..2] init 1; // active module: 1 = control (both cars), 2 =
    decision making + monitoring\n")
f.write("global lC : bool init false; // lane changing occuring? \n")
f.write("global x : [0..length] init 0;\n")
f.write("global v : [15..34] init %s;\n"%v)
f.write("global a : [-3..3] init 0;\n")
f.write("global lane : [1..2] init 1;\n\n")
f.write("formula x1 = x1_0 + v1*t;\n")
```

```
f.write("formula  dist  = x1>x?(x1 − x):(x − x1);\n")
f.write("formula  positiveDist  = (x < length)?x > x1:true;\n\n")

# Decision making + monitoring module
f.write("module Decision_Making_Monitoring\n\n")
f.write("    // If  a crash  occurs, then nothing  else  can happen\n")
f.write("  //[]  actrState  = 2 & crashed  −> 1:(crashed' = true);\n\n")

f.write("    // If  we are  in  lane  2, but behind the other  vehicle , don't  try  to  pass\n")
f.write("  []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = false  & x < length  −> 1:(
    actrState' = 1);\n\n")

f.write("  //  If  we are  in  lane  1, and no vehicle  is  in  front , don't  change lanes\n")
f.write("  []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = true & x < length  −> 1:(
    actrState' = 1);\n\n")

with open(args.dm_table) as  csvfile :
  reader  = csv.DictReader( csvfile )
  for  row  in  reader:
    # should we change from lane 1 to  lane  2?  it 's  based on  delta_crash !  (and the  ADAS)
    if  row["type"]  == driver_type and row["lane"]  == "1"  and row["d"] != max_dm_dist:
      line  = "   []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & dist  = %s & v
     = %s & x < length  −> gamma:(IC' = true) & (actrState' = 1) + (1−gamma)∗%s:(actrState'
     = 1) & (IC' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n"  % (row["d"],row["
    v"],row["P_IC"],row["P_nIC"])
      f.write( line )
      line  = "   []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & dist  = %s & v
     = %s & x < length  −> gamma:(IC' = false) & (a' = −1) & (actrState' = 1) + (1−gamma)
    ∗%s:(actrState' = 1) & (IC' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n"  %
    (row["d"],row["v"],row["P_IC"],row["P_nIC"])
      f.write( line )
      line  = "   []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & dist  = %s & v
     = %s & x < length  −> gamma:(actrState' = 1) + (1−gamma)∗%s:(actrState' = 1) & (IC' =
     true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n"  % (row["d"],row["v"],row["P_IC"
    ],row["P_nIC"])
      f.write( line )
    elif  row["type"]  == driver_type and row["lane"]  == "1"  and row["d"] == max_dm_dist:
      line  = "   []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & dist  >= %s &
     v = %s & x < length  −> gamma:(IC' = true) & (actrState' = 1) + (1−gamma)∗%s:(
    actrState' = 1) & (IC' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n"  % (row[
    "d"],row["v"],row["P_IC"],row["P_nIC"])
      f.write( line )
      line  = "   []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & dist  >= %s &
     v = %s & x < length  −> gamma:(IC' = false) & (a' = −1) & (actrState' = 1) + (1−gamma)
    ∗%s:(actrState' = 1) & (IC' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n"  %
    (row["d"],row["v"],row["P_IC"],row["P_nIC"])
      f.write( line )
      line  = "   []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & dist  >= %s &
     v = %s & x < length  −> gamma:(actrState' = 1) + (1−gamma)∗%s:(actrState' = 1) & (IC'
    = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n"  % (row["d"],row["v"],row["
    P_IC"],row["P_nIC"])
      f.write( line )

    # should we go back to lane  1  from  lane  2?  it 's  based on the  distance  we are  at!  (and the
    ADAS)
    if  row["type"]  == driver_type and row["lane"]  == "2"  and row["d"] != max_dm_dist:
      line  = "   []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & dist  = %s & x
    < length  −> gamma:(IC' = true) & (actrState' = 1) + (1−gamma)∗%s:(actrState' = 1) & (IC
    ' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n"  % (row["d"],row["P_IC"],row[
            "P_nIC"])
          f.write( line )
          line  = "   []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & dist  = %s & x
            < length —> gamma:(IC' = false) & (a' = −1) & (actrState' = 1) + (1−gamma)∗%s:(
            actrState' = 1) & (IC' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n" % (row[
            "d"],row["P_IC"],row["P_nIC"])
          f.write( line )
          line  = "   []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & dist  = %s & x
            < length —> gamma:(actrState' = 1) + (1−gamma)∗%s:(actrState' = 1) & (IC' = true) +
            (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
          f.write( line )
        elif  row["type"] == driver_type and row["lane"] == "2" and row["d"] == max_dm_dist:
          line  = "   []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & dist  >= %s &
            x < length —> gamma:(IC' = true) & (actrState' = 1) + (1−gamma)∗%s:(actrState' = 1) & (
            IC' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],
            row["P_nIC"])
          f.write( line )
          line  = "   []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & dist  >= %s &
            x < length —> gamma:(IC' = false) & (a' = −1) & (actrState' = 1) + (1−gamma)∗%s:(
            actrState' = 1) & (IC' = true) + (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n" % (row[
            "d"],row["P_IC"],row["P_nIC"])
          f.write( line )
          line  = "   []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & dist  >= %s &
            x < length —> gamma:(actrState' = 1) + (1−gamma)∗%s:(actrState' = 1) & (IC' = true) +
            (1−gamma)∗%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
          f.write( line )

# f.write("  []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & x < length —> 1:(
    IC' = true) & (actrState'  = 1);\n")
# f.write("  []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & x < length —> 1:(
    IC' = false)  & (a'  = −1) & (actrState' = 1);\n")
# f.write("  []  actrState  = 2 & !crashed & lane = 1 & positiveDist  = false  & x < length —> 1:(
    actrState'  = 1);\n\n")

# f.write("  []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & x < length —> 1:(
    IC' = true) & (actrState'  = 1);\n")
# f.write("  []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & x < length —> 1:(
    IC' = false)  & (a'  = −1) & (actrState' = 1);\n")
# f.write("  []  actrState  = 2 & !crashed & lane = 2 & positiveDist  = true & x < length —> 1:(
    actrState'  = 1);\n\n")

f.write("endmodule\n\n")

# Control module
f.write("module Control\n\n")

f.write("    // If  we are  in  lane  1,  and no  lane  change was decided,  continue  forward  (which
    might  result  in  crash)\n")
f.write("    // The vehicle  is  behind the  other  driver  ( positiveDist  = false ,  x  < x1)\n")
with open(args.acc_table )  as  csvfile :
  reader  = csv.DictReader( csvfile )
  for  row  in  reader :
    if  row["d"] != max_dm_dist:
      line  = "   []  actrState  = 1 & !crashed & !IC & lane = 1 & x <= length − v & t <
        max_time & positiveDist = false & (x1 + v1 − x − v) >= %s & v + a < 34 & v + a > 15 &
        dist = %s & v = %s  —> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s) & (
        actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"])
      f.write( line )
      line  = "   []  actrState  = 1 & !crashed & !IC & lane = 1 & x <= length − v & t <
```

```
max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist = %s &
v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s) & (actrState' = 2);\n" %
(crash_dist,row["d"],row["v"],row["a"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist = %s &
v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s) & (actrState' = 2);\n" %
(crash_dist,row["d"],row["v"],row["a"])
        f.write(line)
      elif row["d"] == max_dm_dist:
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 &
dist >= %s & v = %s  -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s) & (
actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist >= %s
& v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s) & (actrState' = 2);\n"
% (crash_dist,row["d"],row["v"],row["a"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t <
max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist >= %s
& v = %s -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s) & (actrState' = 2);\n"
% (crash_dist,row["d"],row["v"],row["a"])
        f.write(line)

  f.write("\n [] actrState = 1 & !crashed & !IC & lane = 1 & x > length - v & t < max_time &
      positiveDist = false & (x1 + v1 - x - v) >= %s -> 1:(x' = length) & (t' = t + 1) & (
      actrState' = 2);\n"%crash_dist)
  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x > length - v & t < max_time &
      positiveDist = false & (x1 + v1 - x - v) < %s -> 1:(x' = length) & (t' = t + 1) & (crashed'
       = true) & (actrState' = 2);\n\n"%crash_dist)

  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time &
      positiveDist = false & (x1 + v1 - x - v) < %s & v + a < 34 & v + a > 15 -> 1:(x' = x + v
      ) & (t' = t + 1) & (v' = v + a) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time &
      positiveDist = false & (x1 + v1 - x - v) < %s & v + a >= 34 -> 1:(x' = x + v) & (t' = t
      + 1) & (v' = 34) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time &
      positiveDist = false & (x1 + v1 - x - v) < %s & v + a <= 15 -> 1:(x' = x + v) & (t' = t
      + 1) & (v' = 15) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)

  f.write("   // The vehicle is in front of the other driver (positiveDist = true, x > x1)\n")
  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time &
      positiveDist = true & v + a < 34 & v + a > 15 -> 1:(x' = x + v) & (t' = t + 1) & (v' = v
      + a) & (a' = 0) & (actrState' = 2);\n")
  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time &
      positiveDist = true & v + a >= 34 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (
      actrState' = 2);\n")
  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time &
      positiveDist = true & v + a <= 15 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (
      actrState' = 2);\n\n")

  f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x > length - v & t < max_time &
      positiveDist = true -> 1:(x' = length) & (t' = t + 1) & (actrState' = 2);\n\n")

  f.write("   // If we are in lane 2, and no lane change was decided, continue forward\n")
  f.write("  [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length - v & t < max_time & v
      + a < 34 & v + a > 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
      a) & (a' = 0) & (actrState' = 2);\n")
  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a >= 34 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0) &
       (actrState' = 2);\n")
  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a <= 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0) &
       (actrState' = 2);\n\n")

  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a < 34 & v + a > 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
       a) & (a' = 0) & (actrState' = 2);\n")
  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a >= 34 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0)
      & (actrState' = 2);\n")
  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
      + a <= 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0)
      & (actrState' = 2);\n\n")

  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x > length − v & t < max_time & v +
       a < 34 & v + a > 15 −> 1:(x' = length) & (t' = t + 1) & (v' = v + a) & (actrState' = 2);\
      n")
  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x > length − v & t < max_time & v +
       a >= 34 −> 1:(x' = length) & (t' = t + 1) & (v' = 34) & (actrState' = 2);\n")
  f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x > length − v & t < max_time & v +
       a <= 15 −> 1:(x' = length) & (t' = t + 1) & (v' = 15) & (actrState' = 2);\n\n")

  with open(args.lane_change_table) as  csvfile :
    reader  = csv.DictReader( csvfile )
    for  row  in  reader:
      probCrash = float(row["Acc?"])

      if  probCrash != 0 and probCrash != 1:
        if  row["d"]  != max_control_dist  and row["vi2"]  == v1:
          line  = "   []  actrState  = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
       length − %s & t <= max_time − %s −> %.2f:(crashed' = true) & (actrState' = 2) & (IC' =
      false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (
      lane' = %s) & (actrState' = 2) & (IC' = false);\n"  % (row["o_lane"], row["d"],row["vi1"],row["
      delta_x1"],row["delta_t"],probCrash,1−probCrash,row["delta_x1"],row["vf1"],row["delta_t"],
       str(3 − int(row["o_lane"])))
          f.write(line)
          line  = "  []  actrState  = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
      length − %s & t <= max_time − %s −> %.2f:(crashed' = true) & (actrState' = 2) & (IC' =
      false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (
      lane' = %s) & (actrState' = 2) & (IC' = false);\n"  % (row["o_lane"], row["d"],row["vi1"],row[
      "delta_x1"],row["delta_t"],probCrash,1−probCrash,row["vf1"],row["delta_t"], str(3 − int(row
      ["o_lane"])))
          f.write(line)
          line  = "   []  actrState  = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
       length − %s & t > max_time − %s −> %.2f:(crashed' = true) & (actrState' = 2) & (IC' =
      false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0)
      & (lane' = %s) & (actrState' = 2) & (IC' = false);\n"  % (row["o_lane"], row["d"],row["vi1"],
      row["delta_x1"],row["delta_t"],probCrash,1−probCrash,row["delta_x1"],row["vf1"], str(3 − int
      (row["o_lane"])))
          f.write(line)
          line  = "  []  actrState  = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
      length − %s & t > max_time − %s −> %.2f:(crashed' = true) & (actrState' = 2) & (IC' =
      false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) &
      (lane' = %s) & (actrState' = 2) & (IC' = false);\n"  % (row["o_lane"], row["d"],row["vi1"],
  row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
          f.write(line)
        elif row["d"] == max_control_dist and row["vi2"] == v1:
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
          f.write(line)
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
          f.write(line)
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
          f.write(line)
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
          f.write(line)
      elif probCrash == 0:
        if row["d"] != max_control_dist and row["vi2"] == v1:
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
          f.write(line)
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
          f.write(line)
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
          f.write(line)
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], str(3 - int(row["o_lane"])))
          f.write(line)
        elif row["d"] == max_control_dist and row["vi2"] == v1:
          line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x
```

```
<= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], row["delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], row["vf1"], str(3 - int(row["o_lane"])))
        f.write(line)
    else:
      if row["d"] != max_control_dist and row["vi2"] == v1:
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)
      elif row["d"] == max_control_dist and row["vi2"] == v1:
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)
        line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
        f.write(line)

```

```

f.write("\nendmodule")

f.close()
```

B.1.3 Additive Linear Control

File B.3 model_generator.py

```
# MDP_GENERATOR − Transform the tables into the MDP
# in order to perform multi−objective synthesis.
#
# VERSION:
# − imperfect decision making (gamma)
# − linear acceleration assistance
#
# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 1−Jul−2018; Last revision: 11−Jul−2018

import sys, csv, argparse, datetime

parser=argparse.ArgumentParser(
    description='''Transform the tables into the MDP in order to perform multi−objective
    synthesis. ''')
parser.add_argument('lane_change_table', type=str, help='Table for the lane change part of the
    control module.')
parser.add_argument('acc_table', type=str, help='Table for the linear accelaration part of the
    control module.')
parser.add_argument('dm_table', type=str, help='Table for the decision making module.')
parser.add_argument('driver_type', type=int, default=2, help='1 = aggressive, 2 = average, 3 =
    cautious')
parser.add_argument('v', type=str, help=' Initial velocity of the vehicle.')
parser.add_argument('v1', type=str, help=' Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=str, help=' Initial position of the other vehicle.')
parser.add_argument('−−filename', '−f', type=str, default="mdp_model", help='Output name for
    the file generated.')
args=parser.parse_args()

f = open("%s.pm"%args.filename, "w")

v = args.v
v1 = args.v1
x1_0 = args.x1_0
driver_type = args.driver_type

if not (int(v) >= 15 and int(v) <= 34) or not (int(v1) >= 15 and int(v1) <= 34) or not (int(
    x1_0) >= 1 and int(x1_0) <= 500) or not (driver_type >= 1 and driver_type <= 3):
  raise ValueError("Input out of range.")

driver_type = str(driver_type)
max_control_dist = "43"
max_dm_dist = "80"
crash_dist = "6"
gamma = 0.10

```

```
now = datetime.datetime.now()

# Write the beginning of the file
f.write("//MDP automatically built using mdp_generator.py for v1 = %s (to alter this value, run the script again).\n"%v1)
f.write("//Generated on %s.\n\n"%(now.strftime("%d-%m-%Y at %H:%M")))
f.write("//Version: imperfect decision making, gamma = %.2f; linear acceleration assistance\n\n"%gamma)

f.write("mdp\n\n")
f.write("const int length = 500; // road length\n")
f.write("const int max_time = 35; // maximum time of experiment\n")
f.write("const double gamma = %.2f; // gamma value\n\n"%gamma)
f.write("// Other vehicle\n")
f.write("const int v1 = %s; // do not alter this manually!\n"%v1)
f.write("const int x1_0 = %s;\n\n"%x1_0)
f.write("// Environment variables\n")
f.write("global t : [0..max_time] init 0; // time \n")
f.write("global crashed : bool init false; \n\n")
f.write("// Vehicle controlled\n")
f.write("global actrState : [1..2] init 1; // active module: 1 = control (both cars), 2 = decision making + monitoring\n")
f.write("global lC : bool init false; // lane changing occuring? \n")
f.write("global x : [0..length] init 0;\n")
f.write("global v : [15..34] init %s;\n"%v)
f.write("global a : [-3..3] init 0;\n")
f.write("global lane : [1..2] init 1;\n\n")
f.write("formula x1 = x1_0 + v1*t;\n")
f.write("formula dist = x1>x?(x1 - x):(x - x1);\n")
f.write("formula positiveDist = (x < length)?x > x1:true;\n\n")

# Decision making + monitoring module
f.write("module Decision_Making_Monitoring\n\n")
f.write("   // If a crash occurs, then nothing else can happen\n")
f.write("  //[] actrState = 2 & crashed -> 1:(crashed' = true);\n\n")

f.write("   // If we are in lane 2, but behind the other vehicle, don't try to pass\n")
f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = false & x < length -> 1:(actrState' = 1);\n\n")

f.write("  // If we are in lane 1, and no vehicle is in front, don't change lanes\n")
f.write("  [] actrState = 2 & !crashed & lane = 1 & positiveDist = true & x < length -> 1:(actrState' = 1);\n\n")

with open(args.dm_table) as csvfile:
  reader = csv.DictReader( csvfile )
  for row in reader:
    # should we change from lane 1 to lane 2? it's based on delta_crash! (and the ADAS)
    if row["type"] == driver_type and row["lane"] == "1" and row["d"] != max_dm_dist:
      line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = %s & v = %s & x < length -> gamma:(lC' = true) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (lC' = true) + (1-gamma)*%s:(actrState' = 1) & (lC' = false);\n" % (row["d"],row["v"],row["P_lC"],row["P_nlC"])
      f.write(line)
      line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = %s & v = %s & x < length -> gamma:(lC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (lC' = true) + (1-gamma)*%s:(actrState' = 1) & (lC' = false);\n" % (row["d"],row["v"],row["P_lC"],row["P_nlC"])
      f.write(line)
```

```
            line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = %s & v = %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["v"],row["P_IC"],row["P_nIC"])
            f.write(line)
        elif row["type"] == driver_type and row["lane"] == "1" and row["d"] == max_dm_dist:
            line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist >= %s & v = %s & x < length -> gamma:(IC' = true) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["v"],row["P_IC"],row["P_nIC"])
            f.write(line)
            line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist >= %s & v = %s & x < length -> gamma:(IC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["v"],row["P_IC"],row["P_nIC"])
            f.write(line)
            line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist >= %s & v = %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["v"],row["P_IC"],row["P_nIC"])
            f.write(line)

        # should we go back to lane 1 from lane 2? it's based on the distance we are at! (and the ADAS)
        if row["type"] == driver_type and row["lane"] == "2" and row["d"] != max_dm_dist:
            line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist = %s & x < length -> gamma:(IC' = true) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
            f.write(line)
            line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist = %s & x < length -> gamma:(IC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
            f.write(line)
            line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist = %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
            f.write(line)
        elif row["type"] == driver_type and row["lane"] == "2" and row["d"] == max_dm_dist:
            line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >= %s & x < length -> gamma:(IC' = true) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
            f.write(line)
            line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >= %s & x < length -> gamma:(IC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
            f.write(line)
            line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >= %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
            f.write(line)

    # f.write("  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(IC' = true) & (actrState' = 1);\n")
    # f.write("  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(IC' = false) & (a' = -1) & (actrState' = 1);\n")
```

```
# f.write("  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(actrState' = 1);\n\n")

# f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(IC' = true) & (actrState' = 1);\n")
# f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(IC' = false) & (a' = -1) & (actrState' = 1);\n")
# f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(actrState' = 1);\n\n")

f.write("endmodule\n\n")

# Control module
f.write("module Control\n\n")

a_vals = [-1, 0, 1]

f.write("   // If we are in lane 1, and no lane change was decided, continue forward (which might result in crash)\n")
f.write("   // The vehicle is behind the other driver (positiveDist = false, x < x1)\n")
with open(args.acc_table) as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        if row["d"] != max_dm_dist:
            for delta_a in a_vals:
                line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 & dist = %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist = %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist = %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
                f.write(line)
        elif row["d"] == max_dm_dist:
            for delta_a in a_vals:
                line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 & dist >= %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist >= %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist >= %s
        & v = %s & %s + %d <= 3 & %s + %d >= −3 −> 1:(x' = x + v) & (t' = t + 1) & (v' =
        15) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a
        ,row["a"],delta_a,row["a"],delta_a)
            f.write(line)


f.write("\n [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) >= %s −> 1:(x' = length) & (t' = t + 1) & (
        actrState' = 2);\n"%crash_dist)
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s −> 1:(x' = length) & (t' = t + 1) & (crashed'
         = true) & (actrState' = 2);\n\n"%crash_dist)

f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s & v + a < 34 & v + a > 15 −> 1:(x' = x + v
        ) & (t' = t + 1) & (v' = v + a) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s & v + a >= 34 −> 1:(x' = x + v) & (t' = t
        + 1) & (v' = 34) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = false & (x1 + v1 − x − v) < %s & v + a <= 15 −> 1:(x' = x + v) & (t' = t
        + 1) & (v' = 15) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)

f.write("  // The vehicle is in front of the other driver (positiveDist = true, x > x1)\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = true & v + a < 34 & v + a > 15 −> 1:(x' = x + v) & (t' = t + 1) & (v' = v
        + a) & (a' = 0) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = true & v + a >= 34 −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (
        actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length − v & t < max_time &
        positiveDist = true & v + a <= 15 −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (
        actrState' = 2);\n\n")

f.write(" [] actrState = 1 & !crashed & !IC & lane = 1 & x > length − v & t < max_time &
        positiveDist = true −> 1:(x' = length) & (t' = t + 1) & (actrState' = 2);\n\n")

f.write("  // If we are in lane 2, and no lane change was decided, continue forward\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a < 34 & v + a > 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
        a) & (a' = 0) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a >= 34 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0) &
         (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a <= 15 & positiveDist = true −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0) &
         (actrState' = 2);\n\n")

f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a < 34 & v + a > 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = v +
         a) & (a' = 0) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a >= 34 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0)
        & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x <= length − v & t < max_time & v
        + a <= 15 & positiveDist = false −> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0)
        & (actrState' = 2);\n\n")

f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length − v & t < max_time & v +
    a < 34 & v + a > 15 -> 1:(x' = length) & (t' = t + 1) & (v' = v + a) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a >= 34 -> 1:(x' = length) & (t' = t + 1) & (v' = 34) & (actrState' = 2);\n")
f.write(" [] actrState = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a <= 15 -> 1:(x' = length) & (t' = t + 1) & (v' = 15) & (actrState' = 2);\n\n")

with open(args.lane_change_table) as csvfile:
  reader = csv.DictReader(csvfile)
  for row in reader:
    probCrash = float(row["Acc?"])

    if probCrash != 0 and probCrash != 1:
      if row["d"] != max_control_dist and row["vi2"] == v1:
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"],str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],row["delta_t"],str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
        f.write(line)
      elif row["d"] == max_control_dist and row["vi2"] == v1:
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"],str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],row["delta_t"],str(3 - int(row["o_lane"])))
        f.write(line)
        line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC'
= false) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' =
0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"
],row["delta_x1"],row["delta_t"], probCrash,1-probCrash,row["delta_x1"],row["vf1"], str(3 -
int(row["o_lane"])))
        f.write(line)
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
 length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' =
false) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) &
(lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["o_lane"], row["d"],row["vi1"],row[
"delta_x1"],row["delta_t"],probCrash,1-probCrash,row["vf1"],str(3 - int(row["o_lane"])))
        f.write(line)
    elif probCrash == 0:
      if row["d"] != max_control_dist and row["vi2"] == v1:
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
 length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s)
& (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["
o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["
 delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) &
(t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["
o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"],row["delta_t"], str(3 -
 int(row["o_lane"])))
        f.write(line)
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
 length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) &
(t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["
o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], str(3 -
 int(row["o_lane"])))
        f.write(line)
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t'
 = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["
o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], str(3 - int(row["
o_lane"])))
        f.write(line)
      elif row["d"] == max_control_dist and row["vi2"] == v1:
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x
<= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %
s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row[
"o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["
 delta_t"], str(3 - int(row["o_lane"])))
        f.write(line)
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
 length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) &
(t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["
o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"],row["delta_t"], str(3 -
int(row["o_lane"])))
        f.write(line)
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x
<= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s)
 & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row[
"o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"], str(3
- int(row["o_lane"])))
        f.write(line)
        line = "   [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
 length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t
' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false);\n" % (row["
```

```
        o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], row["vf1"], str(3 - int(row["
        o_lane"])))
            f.write(line)
        else:
          if row["d"] != max_control_dist and row["vi2"] == v1:
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
         length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
        false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
        length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false
        );\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <=
         length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false)
        ;\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x >
        length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false)
        ;\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
          elif row["d"] == max_control_dist and row["vi2"] == v1:
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x
        <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
         false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
         length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
        false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x
        <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' =
        false);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x >
         length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false)
        ;\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"])
            f.write(line)


f.write("\nendmodule")

f.close()
```

B.1.4 Additive Steering Control

File B.4 model_generator.py

```
# MDP_GENERATOR - Transform the tables into the MDP
# in order to perform multi-objective synthesis.
#
# VERSION:
# - imperfect decision making (gamma)
# - linear acceleration assistance
# - steering control assistance
#
# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 1-Jul-2018; Last revision: 18-Jul-2018

import sys, csv, argparse, datetime

parser=argparse.ArgumentParser(
    description='''Transform the tables into the MDP in order to perform multi-objective
     synthesis. ''' )
parser.add_argument('lane_change_table', type=str, help='Table for the lane change part of the
     control module.')
parser.add_argument('acc_table', type=str, help='Table for the linear accelaration part of the
     control module.')
parser.add_argument('dm_table', type=str, help='Table for the decision making module.')
parser.add_argument('driver_type', type=int, default=2, help='1 = aggressive, 2 = average, 3 =
     cautious')
parser.add_argument('v', type=str, help='Initial velocity of the vehicle.')
parser.add_argument('v1', type=str, help='Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=str, help='Initial position of the other vehicle.')
parser.add_argument('--filename', '-f', type=str, default="mdp_model", help='Output name for
     the file generated.')
args=parser.parse_args()

f = open("%s.pm"%args.filename, "w")

v = args.v
v1 = args.v1
x1_0 = args.x1_0
driver_type = args.driver_type

if not (int(v) >= 15 and int(v) <= 34) or not (int(v1) >= 15 and int(v1) <= 34) or not (int(
    x1_0) >= 1 and int(x1_0) <= 500) or not (driver_type >= 1 and driver_type <= 3):
    raise ValueError("Input out of range.")

driver_type = str(driver_type)
max_control_dist = "43"
max_dm_dist = "80"
crash_dist = "6"
gamma = 0.10
length = 500

now = datetime.datetime.now()

# Write the beginning of the file
f.write("//MDP automatically built using mdp_generator.py for v1 = %s (to alter this value,
    run the script again).\n"%v1)
f.write("//Generated on %s.\n\n"%(now.strftime("%d-%m-%Y at %H:%M")))
f.write("//Version: imperfect decision making, gamma = %.2f; linear acceleration assistance;
    lane changing assistance; \n\n"%gamma)

f.write("mdp\n\n")
f.write("const int length = %d; // road length\n"%length)
f.write("const int max_time = 35; // maximum time of experiment\n")
f.write("const double gamma = %.2f; // gamma value\n\n"%gamma)
f.write("// Other vehicle\n")
f.write("const int v1 = %s; // do not alter this manually!\n"%v1)
f.write("const int x1_0 = %s;\n\n"%x1_0)
f.write("// Environment variables\n")
f.write("global t : [0..max_time] init 0; // time \n")
```

```
f.write("global crashed : bool init false; \n\n")
f.write("// Vehicle controlled\n")
f.write("global actrState : [1..2] init 1; // active module: 1 = control (both cars), 2 =
    decision making + monitoring\n")
f.write("global lC : bool init false; // lane changing occuring? \n")
f.write("global x : [0..length] init 0;\n")
f.write("global v : [15..34] init %s;\n"%v)
f.write("global a : [-3..3] init 0;\n")
f.write("global k_chosen : [1..3] init 1;\n")
f.write("global lane : [1..2] init 1;\n\n")

f.write("formula x1 = x1_0 + v1*t;\n")
f.write("formula dist = x1>x?(x1 - x):(x - x1);\n")
f.write("formula positiveDist = (x < length)?x > x1:true;\n\n")

# Decision making + monitoring module
f.write("module Decision_Making_Monitoring\n\n")
f.write("   // If a crash occurs, then nothing else can happen\n")
f.write(" //[] actrState = 2 & crashed -> 1:(crashed' = true);\n\n")

f.write("   // If we are in lane 2, but behind the other vehicle, don't try to pass\n")
f.write(" [] actrState = 2 & !crashed & lane = 2 & positiveDist = false & x < length -> 1:(
    actrState' = 1);\n\n")

f.write(" // If we are in lane 1, and no vehicle is in front, don't change lanes\n")
f.write(" [] actrState = 2 & !crashed & lane = 1 & positiveDist = true & x < length -> 1:(
    actrState' = 1);\n\n")

with open(args.dm_table) as csvfile:
  reader = csv.DictReader( csvfile )
  for row in reader:
    # should we change from lane 1 to lane 2? it's based on delta_crash! (and the ADAS)
    if row["type"] == driver_type and row["lane"] == "1" and row["d"] != max_dm_dist:
      line = " [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = %s & v
       = %s & x < length -> gamma:(lC' = true) & (actrState' = 1) + (1-gamma)*%s:(actrState'
       = 1) & (lC' = true) + (1-gamma)*%s:(actrState' = 1) & (lC' = false);\n" % (row["d"],row["
      v"],row["P_lC"],row["P_nlC"])
      f.write(line)
      line = " [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = %s & v
       = %s & x < length -> gamma:(lC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)
      *%s:(actrState' = 1) & (lC' = true) + (1-gamma)*%s:(actrState' = 1) & (lC' = false);\n" %
      (row["d"],row["v"],row["P_lC"],row["P_nlC"])
      f.write(line)
      line = " [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = %s & v
       = %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (lC' =
       true) + (1-gamma)*%s:(actrState' = 1) & (lC' = false);\n" % (row["d"],row["v"],row["P_lC"
      ],row["P_nlC"])
      f.write(line)
    elif row["type"] == driver_type and row["lane"] == "1" and row["d"] == max_dm_dist:
      line = " [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist >= %s &
       v = %s & x < length -> gamma:(lC' = true) & (actrState' = 1) + (1-gamma)*%s:(
      actrState' = 1) & (lC' = true) + (1-gamma)*%s:(actrState' = 1) & (lC' = false);\n" % (row[
      "d"],row["v"],row["P_lC"],row["P_nlC"])
      f.write(line)
      line = " [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist >= %s &
       v = %s & x < length -> gamma:(lC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)
      *%s:(actrState' = 1) & (lC' = true) + (1-gamma)*%s:(actrState' = 1) & (lC' = false);\n" %
      (row["d"],row["v"],row["P_lC"],row["P_nlC"])
      f.write(line)
```

```
        line = "  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist >= %s & v = %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["v"],row["P_IC"],row["P_nIC"])
        f.write(line)

    # should we go back to lane 1 from lane 2? it's based on the distance we are at! (and the ADAS)
    if row["type"] == driver_type and row["lane"] == "2" and row["d"] != max_dm_dist:
        line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist = %s & x < length -> gamma:(IC' = true) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
        f.write(line)
        line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist = %s & x < length -> gamma:(IC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
        f.write(line)
        line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist = %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
        f.write(line)
    elif row["type"] == driver_type and row["lane"] == "2" and row["d"] == max_dm_dist:
        line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >= %s & x < length -> gamma:(IC' = true) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
        f.write(line)
        line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >= %s & x < length -> gamma:(IC' = false) & (a' = -1) & (actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
        f.write(line)
        line = "  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >= %s & x < length -> gamma:(actrState' = 1) + (1-gamma)*%s:(actrState' = 1) & (IC' = true) + (1-gamma)*%s:(actrState' = 1) & (IC' = false);\n" % (row["d"],row["P_IC"],row["P_nIC"])
        f.write(line)

# f.write("  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(IC' = true) & (actrState' = 1);\n")
# f.write("  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(IC' = false) & (a' = -1) & (actrState' = 1);\n")
# f.write("  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & x < length -> 1:(actrState' = 1);\n\n")

# f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(IC' = true) & (actrState' = 1);\n")
# f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(IC' = false) & (a' = -1) & (actrState' = 1);\n")
# f.write("  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & x < length -> 1:(actrState' = 1);\n\n")

f.write("endmodule\n\n")

# Control module
f.write("module Control\n\n")

a_vals = [-1, 0, 1]
```

```
f.write("    // If we are in lane 1, and no lane change was decided, continue forward (which might result in crash)\n")
f.write("    // The vehicle is behind the other driver ( positiveDist = false, x < x1)\n")
with open(args.acc_table) as csvfile:
  reader = csv.DictReader( csvfile )
  for row in reader:
    if row["d"] != max_dm_dist:
      for delta_a in a_vals:
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 & dist = %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
        f.write( line )
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist = %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
        f.write( line )
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist = %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
        f.write( line )
    elif row["d"] == max_dm_dist:
      for delta_a in a_vals:
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a < 34 & v + a > 15 & dist >= %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
        f.write( line )
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a >= 34 & dist >= %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
        f.write( line )
        line = "  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s & v + a <= 15 & dist >= %s & v = %s & %s + %d <= 3 & %s + %d >= -3 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = %s + %d) & (actrState' = 2);\n" % (crash_dist,row["d"],row["v"],row["a"],delta_a,row["a"],delta_a,row["a"],delta_a)
        f.write( line )


f.write("\n  [] actrState = 1 & !crashed & !IC & lane = 1 & x > length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) >= %s -> 1:(x' = length) & (t' = t + 1) & (actrState' = 2);\n"%crash_dist)
f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x > length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) < %s -> 1:(x' = length) & (t' = t + 1) & (crashed' = true) & (actrState' = 2);\n\n"%crash_dist)

f.write("  [] actrState = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) < %s & v + a < 34 & v + a > 15 -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
```

```
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) < %s & v + a >= 34 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = false & (x1 + v1 - x - v) < %s & v + a <= 15 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (crashed' = true) & (actrState' = 2);\n"%crash_dist)

f.write("    // The vehicle  is  in  front  of the other  driver  ( positiveDist  = true, x > x1)\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = true & v + a < 34 & v + a > 15 -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = 0) & (actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = true & v + a >= 34 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & ( actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 1 & x <= length - v & t < max_time & positiveDist = true & v + a <= 15 -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & ( actrState' = 2);\n\n")

f.write("  []  actrState  = 1 & !crashed & !IC & lane = 1 & x > length - v & t < max_time & positiveDist = true -> 1:(x' = length) & (t' = t + 1) & (actrState'  = 2);\n\n")

f.write("    // If  we are  in  lane  2,  and no  lane  change was decided,  continue  forward\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length - v & t < max_time & v + a < 34 & v + a > 15 & positiveDist = true -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = 0) & (actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length - v & t < max_time & v + a >= 34 & positiveDist = true -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0) & (actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length - v & t < max_time & v + a <= 15 & positiveDist = true -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0) & (actrState' = 2);\n\n")

f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length - v & t < max_time & v + a < 34 & v + a > 15 & positiveDist = false -> 1:(x' = x + v) & (t' = t + 1) & (v' = v + a) & (a' = 0) & (actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length - v & t < max_time & v + a >= 34 & positiveDist = false -> 1:(x' = x + v) & (t' = t + 1) & (v' = 34) & (a' = 0) & (actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x <= length - v & t < max_time & v + a <= 15 & positiveDist = false -> 1:(x' = x + v) & (t' = t + 1) & (v' = 15) & (a' = 0) & (actrState' = 2);\n\n")

f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a < 34 & v + a > 15 -> 1:(x' = length) & (t' = t + 1) & (v' = v + a) & (actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a >= 34 -> 1:(x' = length) & (t' = t + 1) & (v' = 34) & (actrState' = 2);\n")
f.write("  []  actrState  = 1 & !crashed & !IC & lane = 2 & x > length - v & t < max_time & v + a <= 15 -> 1:(x' = length) & (t' = t + 1) & (v' = 15) & (actrState' = 2);\n\n")

keys = ['o_lane', 'd', 'vi1', 'vi2']

with open(args.lane_change_table) as csvfile:
  reader = csv.DictReader( csvfile )
  prev_row = ['1','1','15','15']
  k = 0
  for row in reader:
    new_row = [row['o_lane'], row['d'], row['vi1'], row['vi2']]
    if not new_row == prev_row:
            prev_row = new_row
            k = 1
        else:
            k = k + 1

        probCrash = float(row["Acc?"])

        if probCrash != 0 and probCrash != 1:
            if row["d"] != max_control_dist and row["vi2"] == v1:
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,k,1-probCrash, row["delta_x1"],row["vf1"], row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,k,1-probCrash, row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,k,1-probCrash,row["delta_x1"], row["vf1"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"], row["delta_t"], probCrash,k,1-probCrash,row["vf1"], str(3 - int(row["o_lane"])),k)
                f.write(line)
            elif row["d"] == max_control_dist and row["vi2"] == v1:
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,k,1-probCrash,row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], probCrash,k,1-probCrash, row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = " [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],probCrash,k,1-probCrash,row["delta_x1"],row["vf1"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> %.2f:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d) + %.2f:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], probCrash,k,1-probCrash,row["vf1"], str(3 - int(row["o_lane"])),k)
                f.write(line)
        elif probCrash == 0:
            if row["d"] != max_control_dist and row["vi2"] == v1:
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"], row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"], row["vf1"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"], row["vf1"], str(3 - int(row["o_lane"])),k)
                f.write(line)
            elif row["d"] == max_control_dist and row["vi2"] == v1:
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = t + %s) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"],row["delta_t"], str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = x + %s) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["delta_x1"],row["vf1"],str(3 - int(row["o_lane"])),k)
                f.write(line)
                line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = false) & (x' = length) & (v' = %s) & (t' = max_time) & (a' = 0) & (lane' = %s) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],row["vf1"], str(3 - int(row["o_lane"])),k)
                f.write(line)
        else :
```

```
        if row["d"] != max_control_dist and row["vi2"] == v1:
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist = %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)
        elif row["d"] == max_control_dist and row["vi2"] == v1:
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t <= max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x <= length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)
            line = "  [] actrState = 1 & !crashed & IC & lane = %s & dist >= %s & v = %s & x > length - %s & t > max_time - %s -> 1:(crashed' = true) & (actrState' = 2) & (IC' = false) & (k_chosen' = %d);\n" % (row["o_lane"], row["d"],row["vi1"],row["delta_x1"],row["delta_t"],k)
            f.write(line)


f.write("\nendmodule")

f.close()
```

B.2 Pareto Curve Generator (Python)

File B.5 pareto_curve.py

```
# PARETO_CURVE - Generate the model, perform verification to obtain
# the appropriate multi-objective synthesis problem and
# perform synthesis to obtain the pareto curve desired (changed to
# use PRISM instead of storm).

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 29-Jun-2018; Last revision: 10-Jul-2018

import sys, os, subprocess, csv, argparse
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection
from matplotlib import cm

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size=13)

parser=argparse.ArgumentParser(
    description='''Generate the model, perform verification to obtain the appropriate multi-
    objective synthesis problem and perform synthesis to obtain the pareto curve desired.''')
parser.add_argument('driver_type', type=int, default=2, help='1 = aggressive, 2 = average, 3 =
     cautious.')
parser.add_argument('v', type=int, default=29, help='Initial velocity of the vehicle.')
parser.add_argument('v1', type=int, default=30, help='Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=int, default=15, help='Initial position of the other vehicle.
    ')
parser.add_argument('--cond', '-c', action="store_true", help='If set, conditional
     probabilities will be displayed.')
parser.add_argument('--output', '-o', type=str, default="paretopoints", help='Name of the
    generated file')
parser.add_argument('--query', '-q', type=str, default="", help='Query to build the Pareto
    curve on.')
parser.add_argument('--path', '-p', type=str, default="results", help='Generated file will be
    saved in PATH.')
args=parser.parse_args()

driver_type = args.driver_type
v = args.v
v1 = args.v1
x1_0 = args.x1_0
path = args.path
output = args.output
query = args.query
cond = args.cond


def build_model(ex_path):
  os.system("mkdir %s > /dev/null"%ex_path)

  # Construct the file
  print('Generating the model...')
  os.system('python3 ../model/mdp_generator.py ../model/model_tables/control_table.csv ../
    model/model_tables/acc_table.csv ../model/model_tables/dm_table.csv %s %s %s %s > /dev/
    null'%(driver_type,v,v1,x1_0))

  # ------------- Verification -------------
  print('Building the model and performing verification (could take awhile - no longer than 10
     minutes)...')
  os.system('prism mdp_model.pm properties/verification_mod.pctl -exportmodel %s/out.all -
    exportresults %s/res.txt -javamaxmem 4g'%(ex_path, ex_path))

    f = open("%s/res.txt"%ex_path, "r")

    Tmin = 30

    while True:
        prop = f.readline()[:-2]
        if prop == "":
            break
        f.readline()
        probability = f.readline()[:-1]

        print(prop + ' : ' + probability)

        if 'Pmax=? [ F (x=500&t<' in prop and float(probability) > 0.001:
            T = int(prop[20:22])
            Tmin = min(T, Tmin)

        f.readline()

    f.close()

    f = open("%s/time.txt"%ex_path, "w")
    f.write(str(Tmin))
    f.close()


def synthesis(ex_path, query, output):
    # -------------- Synthesis -------------
    if query == "":
        f = open("%s/time.txt"%ex_path, "r")
        Tmin = int(f.readline())
        f.close()
        multi_obj_query = "multi(Pmin=? [F crashed], Pmax=? [F (x=500) & (t<%d)])"%Tmin
    else:
        multi_obj_query = query

    print('Synthesis using the query "%s"...'%multi_obj_query)

    os.system('prism -importmodel %s/out.all -pctl "%s" -exportpareto %s/%s.txt'%(ex_path,
        multi_obj_query,ex_path,output))
    # output_1 = str(proc.stdout.read())

    print('Outputing to %s.txt and %s_query.txt'%(output,output))

    f = open("%s/%s_query.txt"%(ex_path,output), "w")
    f.write(multi_obj_query)
    f.close()


def draw_curve(ex_path, input_file, cond):
    x = []
    y = []

    f = open("%s/%s_query.txt"%(ex_path,input_file), "r")
    query = f.readline()
    f.close()

    # LaTeX display handlers
    query = query.replace("min=?", "$_{=?}$")
    query = query.replace("max=?", "$_{=?}$")
    query = query.replace("&", "\&")
    query = query.replace("<", "$<$")
    query = query.replace(">", "$>$")

    xlabel = query.split(',')[0][6:]
    ylabel = query.split(',')[1][:-1]

    f = open("%s/%s.txt"%(ex_path,input_file), "r")
    arr_val = f.readline()[2:-1]
    new_arr = arr_val.split(', (')
    for tup in new_arr:
      text_tup = tup[:-1]
      x.append(float(text_tup.split(', ')[0]))
      y.append(float(text_tup.split(', ')[1]))
    f.close()

    new_x, new_y = zip(*sorted(zip(x, y)))
    new_x = [xs for xs in new_x]
    new_y = [ys for ys in new_y]

    if cond:
      for i in range(len(new_y)):
        new_y[i] = min(new_y[i]/(1-new_x[i]),1)

      ylabel = "%s $||$ F (x=500)]"%ylabel[:-1]

    fig, ax = plt.subplots()

    plt.plot(new_x, new_y, marker = 'o', color='g')

    if len(new_x) > 1:
      new_x.append(max(x))
      new_y.append(0)

      new_x.append(min(x))
      new_y.append(0)

      xy = np.vstack((new_x, new_y)).T

      polygon = [Polygon(xy, True)]
      p = PatchCollection(polygon, alpha=0.3)
      p.set_color("g")

      ax.add_collection(p)

    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    # plt.title(query)

    plt.show()

  def_path = "%s/r_%s_%s_%s_%s"%(path,driver_type,v,v1,x1_0)

  if not os.path.exists("%s/res.txt"%def_path):
    build_model(def_path)

  if not os.path.exists("%s/%s.txt"%(def_path,output)):
    synthesis(def_path, query, output)

  draw_curve(def_path, output, cond)
  print('Done.')
```

B.3 Strategy Synthesis and Simulator (Python)

File B.6 synthesis_and_simulate.py

```
# SYNTHESIS_AND_SIMULATE - Given some initial conditions,
# obtain a sample trace and simulate it.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 1-Jul-2018; Last revision: 1-Jul-2018

import sys, os, csv, argparse, subprocess

parser=argparse.ArgumentParser(
    description='''Given some initial conditions, obtain a sample trace and simulate it.''')
parser.add_argument('driver_type', type=int, default=2, help='1 = aggressive, 2 = average, 3 = cautious.')
parser.add_argument('v', type=int, default=29, help='Initial velocity of the vehicle.')
parser.add_argument('v1', type=int, default=30, help='Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=int, default=15, help='Initial position of the other vehicle.')
parser.add_argument('-p', '--path', type=str, help='Path where the all the files will be generated.')
parser.add_argument('-q', '--query', type=str, help='Use the multi-objective query given.')
parser.add_argument('-o', '--output', type=str, default="adv", help='Name of the output adversary generated.')
parser.add_argument('-x', '--times', type=float, default=1, help='Execution is X times faster.')
parser.add_argument('-rt', '--read_trace', action="store_true", help='Read an existing trace.')
parser.add_argument('-ra', '--read_adv', action="store_true", help='Read an adversary and generate a new trace only.')
parser.add_argument('-a', '--adv', type=str, default="adv", help='Read an adversary and generate a new trace only.')
args=parser.parse_args()

driver_type = args.driver_type
v = args.v
v1 = args.v1
x1_0 = args.x1_0
output = args.output

speed = args.times
adv = args.adv

if args.path:
  p = args.path
else:
  p = "built_models/r_%d_%d_%d_%d"%(driver_type,v,v1,x1_0)

if not args.read_trace:

  if not args.read_adv:
    # Synthesis
    if not args.query:
      os.system("python3 helpers/synthesis.py %d %d %d %d -p %s -o %s"%(driver_type, v, v1, x1_0, p, output))
    else:
      os.system('python3 helpers/synthesis.py %d %d %d %d -p %s -q "%s" -o %s'%(driver_type, v, v1, x1_0, p, args.query, output))
  else:
    print("Skip synthesis; read adversary from %s.tra and generate new trace to simulate."%adv)

  states_file = "out.sta"
  labels_file = "out.lab"
  new_states_file = "%s_new_states.sta"%adv
  adv_file = "%s.tra"%adv

  # Multi-objective synthesis
  os.system("python3 helpers/multi_gen_trace.py %s %s %s %s %d %d -p %s"%(states_file, labels_file, new_states_file, adv_file, v1, x1_0, p))

else:
  print("Reading existing trace...")

os.system("python3 helpers/simulate.py %d %d %d %s/gen_trace.csv -x %f"%(v, v1, x1_0, p, speed))
```

File B.7 helpers/synthesis.py

```
# SYNTHESIS - Given a set of initial conditions, build a model,
# verify it and obtain the correct multi-objective property for
# synthesis in order to obtain an adversary.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 1-Jul-2018; Last revision: 1-Jul-2018

import sys, os, subprocess, csv, argparse
import random

parser=argparse.ArgumentParser(
    description='''Given a set of initial conditions, build a model, verify it and obtain the correct multi-objective property for synthesis in order to obtain an adversary.''')
parser.add_argument('driver_type', type=int, default=2, help='1 = aggressive, 2 = average, 3 = cautious.')
parser.add_argument('v', type=int, default=29, help='Initial velocity of the vehicle.')
parser.add_argument('v1', type=int, default=30, help='Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=int, default=15, help='Initial position of the other vehicle.')
parser.add_argument('--query', '-q', type=str, default="", help='Query to generate an adversary on.')
parser.add_argument('--output', '-o', type=str, default="adv", help='Name of the adversary file generated for the query.')
parser.add_argument('--path', '-p', type=str, default="", help='Generated file will be saved in PATH.')
args=parser.parse_args()

```

```
driver_type = args.driver_type
v = args.v
v1 = args.v1
x1_0 = args.x1_0
path = args.path
query = args.query
output = args.output

def build_model(ex_path):
  os.system("mkdir %s > /dev/null"%ex_path)

  # Construct the file
  print('Generating the model...')
  os.system('python3 model/mdp_generator.py model/model_tables/control_table.csv model/
     model_tables/acc_table.csv model/model_tables/dm_table.csv %s %s %s %s > /dev/null'%(
     driver_type,v,v1,x1_0))

  # -------------- Verification --------------
  print('Building the model and performing verification (could take awhile - no longer than 10
      minutes)...')
  os.system('prism mdp_model.pm helpers/properties/verification_mod.pctl -exportmodel %s/out.
     all -exportresults %s/res.txt -javamaxmem 4g -cuddmaxmem 2g'%(ex_path, ex_path))

  f = open("%s/res.txt"%ex_path, "r")

  Tmin = 30

  while True:
    prop = f.readline()[:-2]
    if prop == "":
      break
    f.readline()
    probability = f.readline()[:-1]

    print(prop + ' : ' + probability)

    if 'Pmax=? [ F (x=length&t<' in prop and float(probability) > 0.001:
      T = int(prop[23:25])
      Tmin = min(T, Tmin)

    f.readline()

  f.close()

  f = open("%s/time.txt"%ex_path, "w")
  f.write(str(Tmin))
  f.close()


def synthesis(ex_path, query, output):
  # -------------- Synthesis --------------
  if query == "":
    f = open("%s/time.txt"%ex_path, "r")
    Tmin = int(f.readline())
    f.close()
    multi_obj_query = "multi(Pmax=? [F x=400 & t < %d], P<=0.18 [F crashed])"%Tmin
  else:
    multi_obj_query = query

    print ('Synthesis using the query "%s"...'%multi_obj_query)

    os.system('prism -importmodel %s/out.all -lp -pctl "%s" -exportadv %s/%s.tra -exportprodstates %s/%s_new_states.sta -javamaxmem 4g -cuddmaxmem 2g'%(ex_path,multi_obj_query,ex_path,output,ex_path,output))

    print ('Outputing to %s.tra, %s_new_states.sta and %s_query.txt'%(output,output,output))

    f = open("%s/%s_query.txt"%(ex_path,output), "w")
    f.write(multi_obj_query)
    f.close()


# def_path = "%s/r_%s_%s_%s_%s"%(path,driver_type,v,v1,x1_0)
def_path = path

if not os.path.exists("%s/res.txt"%def_path):
    build_model(def_path)

if not os.path.exists("%s/%s.txt"%(def_path,output)):
    synthesis(def_path, query, output)
```

File B.8 helpers/multi_gen_trace.py

```
# MULTI_GEN_TRACE - Given the states and labels of an MDP,
# the adversary generated and the states of the product of
# the MDP by the DRA, obtain a trace in the original MDP

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 13-Jul-2018; Last revision: 18-Jul-2018

import sys, os, subprocess, csv, argparse
import random

parser=argparse.ArgumentParser(
    description='''Given the states and labels of an MDP, the adversary generated and the states of the product of the MDP by the DRA, obtain a trace in the original MDP''')
parser.add_argument('mdp_states', type=str, help='States of the original MDP.')
parser.add_argument('mdp_labels', type=str, help='Labels of the original MDP.')
parser.add_argument('new_states', type=str, help='States of the product MDP (of the original and the DRA of the property).')
parser.add_argument('adversary', type=str, help='Transitions in the new state space.')
parser.add_argument('v1', type=int, help='Initial speed of the other vehicle.')
parser.add_argument('x1_0', type=int, help='Initial position of the other vehicle.')
parser.add_argument('-o', '--output', type=str, default="gen_trace.csv", help='Name of the generated .csv file (path excluded).')
parser.add_argument('-p', '--path', type=str, default="", help='Path where the the input files are stored, and text and csv files will be generated.')
parser.add_argument('-c', '--clear', action="store_true", help='Clear the generated trace text file.')
args=parser.parse_args()

states_file = "%s/%s"%(args.path,args.mdp_states)
labels_file = "%s/%s"%(args.path,args.mdp_labels)
new_states_file = "%s/%s"%(args.path,args.new_states)
adv_file = "%s/%s"%(args.path,args.adversary)

```

```
def multi_transform_label_file(old_states, old_labels, new_states):
    new_init_states = []
    new_deadlocks = []

    sta = {}
    init_states = []
    deadlocks = []

    # import internal state representation
    f = open(old_states, "r")
    header = f.readline()[:-1]
    while True:
        s = f.readline()[:-1]
        if s == "":
            break

        s = s.split(':')
        sta[s[0]] = s[1][1:-1]

    f.close()

    # import init and deadlock states
    f = open(old_labels, "r")
    f.readline()
    while True:
        s = f.readline()[:-1]
        if s == "":
            break

        s = s.split(':')
        if s[1] == " 0":
            init_states.append(sta[s[0]])
        elif s[1] == " 1":
            deadlocks.append(sta[s[0]])

    f.close()

    # import internal state representation
    f = open(new_states, "r")
    header = f.readline()[1:-1]
    header_sp = header.split(',')
    t_index = header_sp.index("t")
    lane_index = header_sp.index("lane")

    while True:
        s = f.readline()[:-2]
        if s == "":
            break

        s = s.split(':')
        s1_spl = s[1].split(',')
        new_s1 = ','.join(s1_spl[t_index:lane_index+1])
        if t_index == 0:
            new_s1 = new_s1[1:]

        if new_s1 in init_states:
            new_init_states.append(s[0])
        elif new_s1 in deadlocks:
            new_deadlocks.append(s[0])
```

```python

    f.close()

    new_labels = {'init': new_init_states, 'deadlocks': new_deadlocks}
    return new_labels

def multi_generate_sample_path(new_states_f, adv_f, new_labels, result_file):
    print('Generate a sample path in the synthesised model...')

    sta = {}
    tra = {}
    init_states = new_labels['init']
    deadlocks = new_labels['deadlocks']
    step = 0

    # import internal state representation
    f = open(new_states_f, "r")
    header = f.readline()[:-1]
    while True:
        s = f.readline()[:-1]
        if s == "":
            break

        s = s.split(':')
        sta[s[0]] = s[1]

    f.close()

    # import state transitions
    f = open(adv_f, "r")
    f.readline()
    while True:
        s = f.readline()[:-1]
        if s == "":
            break

        s = s.split()
        if s[0] in tra.keys():
            if '0' in tra[s[0]].keys():
                tra[s[0]]['0'].append([s[1], s[2]])
            else:
                tra[s[0]]['0'] = [[s[1], s[2]]]
        else:
            tra[s[0]] = {}

            if '0' in tra[s[0]].keys():
                tra[s[0]]['0'].append([s[1], s[2]])
            else:
                tra[s[0]]['0'] = [[s[1], s[2]]]

    # print("\n".join("{}\t{}".format(k, v) for k, v in tra.items()))

    f.close()

    # start simulation
    f = open(result_file, "w")
    f.write("step," + header[1:-1] + "\n")
    curr_state = random.choice(init_states)
    f.write(str(step) + "," + sta[curr_state][1:-1] + "\n")

    while (not curr_state in deadlocks) and (curr_state in tra.keys()):
        possible_trans = tra[curr_state]
        action_taken = random.choice(list(possible_trans.keys()))
        diff_trans = possible_trans[action_taken]

        if len(diff_trans) == 1:
            curr_state = diff_trans[0][0]
        else:
            possible_states = []
            cum_probabilities = []

            for elem in diff_trans:
                possible_states.append(elem[0])
                if len(cum_probabilities) == 0:
                    cum_probabilities.append(float(elem[1]))
                else:
                    cum_probabilities.append(cum_probabilities[len(cum_probabilities) - 1] + float(elem[1]))

            random_prob = random.uniform(0, 1)
            i = 0
            while True:
                if random_prob <= cum_probabilities[i]:
                    break
                i = i+1

            curr_state = possible_states[i]

        step = step + 1
        f.write(str(step) + "," + sta[curr_state][1:-1] + "\n")

    if not curr_state in tra.keys():
        print("----------------------------------------")
        print("ERROR in adversary file: end state not in deadlock, but no transition found.")
        print("----------------------------------------")

    f.close()

def multi_generate_trace_from_file(file, v1, x1_0, out):
    # Read the generated text file
    print('Read the generated file and modify the trace file to become readable in simulation ...')

    csvfile = open(out, 'w')
    fieldnames = ['t_end','type','v','crashed','lane','x_t_1','x_t_2','x_t_3','y_t_1','y_t_2','y_t_3','y_t_4','y_t_5','y_t_6','y_t_7']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()

    next_change_lanes = False
    curr_v = 0
    curr_t = 0
    curr_x = 0

    with open(file) as csvfile:
        reader = csv.DictReader(csvfile)
        # Skip the first line
# next(reader)
for row in reader:

  if row['actrState'] == "1" and row["IC"] == "false":
    curr_v = int(row['v'])

  if row['actrState'] == "1" and row["IC"] == "true":
    next_change_lanes = True
    curr_v = int(row['v'])
    curr_t = int(row['t'])
    curr_x = int(row['x'])

  if row['actrState'] == "2" and next_change_lanes == False:
    if row['crashed'] == 'false':
      writer.writerow({'t_end': row['t'], 'type': '1', 'v': curr_v, 'crashed': '0', 'lane': row['lane'], 'x_t_1': '0', 'x_t_2': '0', 'x_t_3': '0', 'y_t_1': '0', 'y_t_2': '0', 'y_t_3': '0', 'y_t_4': '0', 'y_t_5': '0', 'y_t_6': '0', 'y_t_7': '0'})
    else:
      writer.writerow({'t_end': row['t'], 'type': '1', 'v': curr_v, 'crashed': '1', 'lane': row['lane'], 'x_t_1': '0', 'x_t_2': '0', 'x_t_3': '0', 'y_t_1': '0', 'y_t_2': '0', 'y_t_3': '0', 'y_t_4': '0', 'y_t_5': '0', 'y_t_6': '0', 'y_t_7': '0'})

  if row['actrState'] == "2" and next_change_lanes == True:
    lane = int(row['lane'])
    o_lane = 3 - lane
    if row['actrState'] == "2" and row['crashed'] == 'true':
      lane = o_lane
      o_lane = 3 - lane

    # print(str(lane) + ", " + str(min(abs(x1_0 + v1*curr_t - curr_x), 43)) + ", " + str(curr_v) + ", " + str(v1))

    idx_line = (2-lane)*20*20*43*3 + (v1 - 15)*20*43*3 + (curr_v - 15)*43*3 + (min(abs(x1_0 + v1*curr_t - curr_x), 43) - 1)*3 + int(row['k_chosen'])

    infile = open("helpers/data/other_table.csv")
    r = csv.DictReader( infile )
    for i in range( idx_line -1):
      next(r)
    this_row = next(r)

    print(this_row)

    if row['crashed'] == 'false':
      writer.writerow({'t_end': row['t'], 'type': '2', 'v': row['v'], 'crashed': '0', 'lane': o_lane, 'x_t_1': this_row['p_x(1)'], 'x_t_2': this_row['p_x(2)'], 'x_t_3': this_row['p_x(3)'], 'y_t_1': this_row['p_y(1)'], 'y_t_2': this_row['p_y(2)'], 'y_t_3': this_row['p_y(3)'], 'y_t_4': this_row['p_y(4)'], 'y_t_5': this_row['p_y(5)'], 'y_t_6': this_row['p_y(6)'], 'y_t_7': this_row['p_y(7)']})
    else:
      writer.writerow({'t_end': str(int(row['t']) + 6), 'type': '2', 'v': row['v'], 'crashed': '1', 'lane': o_lane, 'x_t_1': this_row['bad_p_x(1)'], 'x_t_2': this_row['bad_p_x(2)'], 'x_t_3': this_row['bad_p_x(3)'], 'y_t_1': this_row['bad_p_y(1)'], 'y_t_2': this_row['bad_p_y(2)'], 'y_t_3': this_row['bad_p_y(3)'], 'y_t_4': this_row['bad_p_y(4)'], 'y_t_5': this_row['bad_p_y(5)'], 'y_t_6': this_row['bad_p_y(6)'], 'y_t_7': this_row['bad_p_y(7)']})

    next_change_lanes = False

```

```

new_labels = multi_transform_label_file ( states_file , labels_file , new_states_file )
multi_generate_sample_path( new_states_file , adv_file , new_labels, "%s/trace.txt"%args.path)
 multi_generate_trace_from_file ("%s/trace.txt"%args.path, args.v1, args.x1_0, "%s/%s"%(args.
     path,args.output))
 if args.clear:
   os.system("rm -f %s/trace.txt"%args.path)
```

File B.9 helpers/simulate.py

```
# SIMULATE - Given a trace, simulate it.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 23-Jun-2018; Last revision: 23-Jun-2018

import pygame
import sys, os, csv, argparse, subprocess
import numpy as np

parser=argparse.ArgumentParser(
    description='''Given some initial conditions, obtain a sample trace and simulate it.''')
parser.add_argument('v', type=int, default=29, help='Initial velocity of the vehicle.')
parser.add_argument('v1', type=int, default=30, help='Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=int, default=15, help='Initial position of the other vehicle.
    ')
parser.add_argument('trace', type=str, help='Trace file to be read.')
parser.add_argument('-x', '--times', type=float, default=1, help='Execution is X times faster.
    ')
args=parser.parse_args()

v = args.v
v1 = args.v1
x1_0 = args.x1_0
trace = args.trace
speed = args.times

# Helper functions
_image_library = {}
def get_image(path):
   global _image_library
   image = _image_library.get(path)
   if image == None:
         canonicalized_path = path.replace('/', os.sep).replace('\\', os.sep)
         image = pygame.image.load('helpers/graphics/' + canonicalized_path)
         if path == "background.png":
           image = pygame.transform.scale(image, (1000,400))
         if path == "car_red.png" or path == "car_grey.png":
           image = pygame.transform.scale(image, (20,11))
         _image_library[path] = image
   return image

def render_centered(screen, text, crashed_font, color):
   label = crashed_font.render(text, 1, color)
   size = crashed_font.size(text)
   screen.blit(label, (500 - size[0]/2.0, 200 - size[1]/2.0))

def detect_crash(x1, y1, x2, y2, w, h):
  top1 = [x1 - w/2, y1 - h/2]
  top2 = [x2 - w/2, y2 - h/2]
  return not (top1[0] + w < top2[0] or top1[1] + h < top2[1] or top1[0] > top2[0] + w or top1
    [1] > top2[1] + h)

# Main

pygame.init()
pygame.display.set_caption('Sample Path Simulation')
screen = pygame.display.set_mode((1000,400))
myfont = pygame.font.SysFont("monospaced", 20)
crashed_font = pygame.font.SysFont("monospaced", 45)
time_font = pygame.font.SysFont("monospaced", 60)

csvfile = open(trace)
reader = csv.DictReader( csvfile )
comm = next(reader)

# Read the first command
t_init = 0
x_init = 0
y_init = 1.8
t_end = int(comm['t_end'])
type_comm = int(comm['type'])
curr_v = int(comm['v'])
crashed = bool(int(comm['crashed']))
x_coeffs = [float(comm['x_t_1']), float(comm['x_t_2']), float(comm['x_t_3'])]
y_coeffs = [float(comm['y_t_1']), float(comm['y_t_2']), float(comm['y_t_3']), float(comm['
    y_t_4']), float(comm['y_t_5']), float(comm['y_t_6']), float(comm['y_t_7'])]

# Setup of the other variables
x0 = x1_0
v0 = v1

t = 0.0
deltaT = 0.05
scaleX = 2
scaleY = 5
c_height = 9
c_width = 20
actual_c_width = 1.1*4.8
actual_c_height = 1.1*1.9

x = x_init
y = y_init

update_action = False
permanent = False
force_crash = False

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      quit()
    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
      update_action = not update_action

  screen.fill((255,255,255))
  screen.blit(get_image('background.png'), (0, 0))

  screen.blit(get_image('car_red.png'), (x*scaleX - c_width/2, 218 - y*scaleY - c_height/2))

  # Other vehicle
  screen.blit(get_image('car_grey.png'), ((x0 + t*v0)*scaleX - c_width/2, 218 - 1.8*scaleY -
    c_height/2))

  # Display info
  time_label = time_font.render("%2.1fs"%t, 1, (0,0,0))
  screen.blit(time_label, (175, 35))

  main_vahicle_label = myfont.render("Main vehicle", 1, (128,128,128))
  screen.blit(main_vahicle_label, (325, 25))

  x_label = myfont.render("x Position: %dm"%x, 1, (0,0,0))
  screen.blit(x_label, (325, 45))

  y_label = myfont.render("y Position: %.2fm"%y, 1, (0,0,0))
  screen.blit(y_label, (325, 65))

  other_vahicle_label = myfont.render("Other vehicle", 1, (128,128,128))
  screen.blit(other_vahicle_label, (475, 25))

  x_label = myfont.render("x Position: %dm"%(x0 + t*v0), 1, (0,0,0))
  screen.blit(x_label, (475, 45))

  y_label = myfont.render("y Position: 1.8m", 1, (0,0,0))
  screen.blit(y_label, (475, 65))

  if detect_crash(x,y,(x0 + t*v0),1.8, actual_c_width, actual_c_height) == True or force_crash
    == True:
    render_centered(screen, "Crashed", crashed_font, (0,0,0))
    update_action == False
    permanent = True

  # Main vehicle
  if update_action == True and permanent == False:
    if int(np.floor(round(t,2))) == t_end:
      if crashed == True:
        permanent = True
        force_crash = True
      else:
        try:
          comm = next(reader)
        except:
          update_action = False
          permanent = True
          continue

        # Read the next command
        t_init = t
        x_init = x
        y_init = y
        t_end = int(comm['t_end'])
        type_comm = int(comm['type'])
        curr_v = int(comm['v'])
        crashed = bool(int(comm['crashed']))
        x_coeffs = [float(comm['x_t_1']), float(comm['x_t_2']), float(comm['x_t_3'])]
```

```
          y_coeffs  = [float(comm['y_t_1']), float(comm['y_t_2']), float(comm['y_t_3']), float(
        comm['y_t_4']), float(comm['y_t_5']), float(comm['y_t_6']), float(comm['y_t_7'])]

        if int(np.floor(round(t,2))) < t_end:
          if type_comm == 1:
            x = x + curr_v*deltaT
          elif type_comm == 2:
            curr_t = t - t_init
            x = x_init + (x_coeffs[0]* curr_t **2 + x_coeffs[1]* curr_t + x_coeffs[2])
            y = (y_coeffs[0]* curr_t **6 + y_coeffs[1]* curr_t **5 + y_coeffs[2]* curr_t **4 + y_coeffs
        [3]* curr_t **3 + y_coeffs[4]* curr_t **2 + y_coeffs[5]* curr_t + y_coeffs[6])

        if x >= 500 or t >= 30:
          update_action = False
          permanent = True

        t = t + deltaT

      elif crashed == False and update_action == False and permanent == False:
        if x == 0:
          render_centered(screen, "Press SPACE to start", crashed_font, (0,0,0))
        else:
          render_centered(screen, "Press SPACE to continue", crashed_font, (0,0,0))
      elif crashed == False and permanent == True:
        render_centered(screen, "Done", crashed_font, (0,0,0))

    pygame.display.flip()
    pygame.time.wait(int(1000*deltaT/speed))

pygame.quit()
```

B.4 Prism Automatic Model Checker (Python)

File B.10 automatic_model_checker.py

```
# AUTOMATIC_MODEL_CHECKER - Executes the script model_generator.py
# for some given parameters in order to build the model and perform
# model checking subquentially.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 13-Jul-2018; Last revision: 13-Jul-2018

import sys, os, subprocess, csv, argparse
from datetime import datetime
startTime = datetime.now()

parser=argparse.ArgumentParser(
    description='''Executes the script model_generator.py for some given parameters in order
    to build the model and perform model checking subquentially. ''')
parser.add_argument('properties_file', type=str, help='File of the properties to be checked (
    PCTL or LTL).')
parser.add_argument('v', type=int, default=29, help='Initial velocity of the vehicle.')
parser.add_argument('v1', type=int, default=30, help='Initial velocity of the other vehicle.')
parser.add_argument('x1_0', type=int, default=15, help='Initial position of the other vehicle.
    ')
```

```
parser.add_argument('--path', '-p', type=str, default='gen_files', help='Path where the generated files will be saved.')
parser.add_argument('--clean', '-c', action="store_true", help='If set, then generated files (model and individual results) will be maintained.')
args=parser.parse_args()

types = ['aggressive','average','cautious']
res = {}

 properties_file = args.properties_file
v = args.v
v1 = args.v1
x1_0 = args.x1_0
cleaning_up = not args.clean
path = args.path

num_properties = sum(1 for line in open("%s"%properties_file))
with open("%s"%properties_file) as f1:
    props = f1.readlines()
props = [x.strip() for x in props]

for driver_type in range(1,4):
  print('------- %s driver -------'%types[driver_type-1])

  filename = "gen_model_%s_%s_%s_%s"%(driver_type,v,v1,x1_0)
  r_filename = "results_%s_%s_%s_%s"%(driver_type,v,v1,x1_0)

  # Construct the file
  print('Generating the model...')
  os.system('python3 ../model/mdp_generator.py ../model/model_tables/control_table.csv ../model/model_tables/acc_table.csv ../model/model_tables/dm_table.csv %s %s %s %s --filename %s > /dev/null'%(driver_type,v,v1,x1_0,filename))

  print('Building the model and performing model checking...')
  subprocess.run("prism %s.pm %s -exportresults %s/%s.txt -javamaxmem 4g -cuddmaxmem 2g"%(filename, properties_file, path, r_filename), shell=True)

  print('Obtaining the results ...')

  f = open("%s/%s.txt"%(path, r_filename), "r")

  if num_properties == 1:
    f.readline()
    probability = f.readline()
    f.close()

    probability = float(probability[:-1])
    res[driver_type] = [props[0], probability]
  else:
    f.readline()
    f.readline()
    probability = f.readline()

    probability = float(probability[:-1])
    res[driver_type] = [[props[0], probability]]

    for i in range(1,num_properties):
      f.readline()
      f.readline()
```

```
            f.readline()
            probability = f.readline()

            probability = float(probability[:-1])
            res[driver_type].append([props[i], probability])

    f.close()

    if cleaning_up == True:
        print('Cleaning up...')
        os.system('rm %s.pm'%filename)
        # os.system('rm %s/%s.txt'%(path,r_filename))

with open('%s/r_%s_%s_%s.csv'%(path,v,v1,x1_0), 'w') as csvfile:
    fieldnames = ['type_driver', 'property', 'probability']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    for key,val in res.items():
        for propty in val:
            writer.writerow({'type_driver': key, 'property': propty[0], 'probability': propty[1]})

# for driver_type in range(1,4):
#   r_filename = "results_%s_%s_%s_%s"%(driver_type,v,v1,x1_0)
#   os.system('rm %s/%s.txt'%(path,r_filename))

print('Done.')
print(datetime.now() - startTime)
```

Appendix C

Code for the Experimental Results and Evaluation

C.1 Plot Generator for experiments with Human Driver Model (Python)

File C.1 plots.py

```python
# PLOTS - Generate some relevant plots that relate to the performance
# of different drivers.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 5-Jun-2018; Last revision: 17-Jul-2018

import sys, os, random, glob, csv, subprocess, itertools
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib as mpl

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size=24)

# Generate samples for some of the graphs
# seq = True iff two out of (v, v1, x1_0) are set
#
def generate_samples(N, *args, **kwargs):
  v_set = kwargs.get('v', None)
  v1_set = kwargs.get('v1', None)
  x1_0_set = kwargs.get('x1_0', None)
  path = kwargs.get('path', None)
  seq = bool(kwargs.get('seq', False))

  v_count = 15
  v1_count = 15
  x1_0_count = 10

  for i in range(1,N+1):
    while 1:
      if v1_set == None and seq == False:
        v1 = random.randint(15,34)
      elif v1_set == None and seq == True:
        v1 = v1_count
        v1_count = v1_count + 1
      else:
        v1 = int(v1_set)

      if v_set == None and seq == False:
        v = random.randint(15,34)
      elif v_set == None and seq == True:
        v = v_count
        v_count = v_count + 1
      else:
        v = int(v_set)

      if x1_0_set == None and seq == False:
        x1_0 = random.randint(10,80)
      elif x1_0_set == None and seq == True:
        x1_0 = x1_0_count
        x1_0_count = x1_0_count + 1
      else:
        x1_0 = int(x1_0_set)

      if v_count > 35 or v1_count > 35 or x1_0_count > 81:
        return

      if path == None:
        if not os.path.exists("%s/r_%s_%s_%s.csv"%(path,v,v1,x1_0)):
          break
      else:
        if not os.path.exists("%s/r_%s_%s_%s.csv"%(path,v,v1,x1_0)):
          break

    print('[%d/%d]: Evaluating drivers for v = %d, v1 = %d, x1_0 = %d...'%(i,N,v,v1,x1_0))
    if path == None:
      proc = subprocess.Popen('python3 storm_model_checker.py properties.pctl %d %d %d'%(v,v1,x1_0), stderr=subprocess.PIPE, shell=True)
    else:
      proc = subprocess.Popen('python3 storm_model_checker.py properties.pctl %d %d %d --path %s'%(v,v1,x1_0,path), stderr=subprocess.PIPE, shell=True)
    output = str(proc.stderr.read())


def generate_combination_samples(vs, v1s, x1_0s, p):
  sz = len(vs)*len(v1s)*len(x1_0s)
  print('Evaluating %d combinations...'%sz)

  for v in vs:
    for v1 in v1s:
      for x1_0 in x1_0s:
        print('v = %d, v1 = %d, x1_0 = %d:'%(v,v1,x1_0))

```

```
            if os.path.exists("%s/r_%d_%d_%d.csv"%(p,v,v1,x1_0)):
              continue

            proc = subprocess.Popen('python3 storm_model_checker.py properties.pctl %d %d %d --
        path %s'%(v,v1,x1_0,p), stderr=subprocess.PIPE, shell=True)
            output = str(proc.stderr.read())


def read_files_to_dict(path):
    props_dict = [{},{},{}]

    os.chdir("%s/"%path)
    for file in glob.glob("*.csv"):
      with open(file) as csvfile :
        reader = csv.DictReader(csvfile)
        for row in reader:
          if row["property"] in props_dict[int(row["type_driver"])-1].keys():
            props_dict[int(row["type_driver"])-1][row["property"]].append(float(row["probability
    "]))
          else:
            props_dict[int(row["type_driver"])-1][row["property"]] = [float(row["probability"])]

    return props_dict


def safety_plots(p, v1_in, x1_0_in):
    # generate_samples(20, v1=v1_in, x1_0=x1_0_in, path=p, seq=True)

    x = [];
    y = [[],[],[]];

    os.chdir("%s/"%p)
    for file in glob.glob("*.csv"):
      x.append(int(str(file).split('_')[1]))

      with open(file) as csvfile :
        reader = csv.DictReader(csvfile)
        for row in reader:
          if row["property"] == 'P=? [F crashed]':
            y[int(row["type_driver"])-1].append(float(row["probability"]))

    labels = ["Aggressive", "Average", "Cautious"]

    for i in range(0,3):
      new_x, new_y = zip(*sorted(zip(x, y[i])))

      line = plt.plot(new_x, new_y, label=labels[i], marker="s")

    plt.legend(loc='upper left', fontsize=18)

    plt.ylabel('P$_{=?}$ [F crashed]', fontsize=18)
    plt.xlabel('v [m/s]', fontsize=18)
    # plt.title('Safety property for $v_1 = %d$, $x_{1,0} = %d$'%(v1_in, x1_0_in))
    plt.xticks(np.arange(min(new_x)-1, max(new_x)+1, 2))

    plt.subplots_adjust(right=0.95, top=0.95, left=0.19, bottom=0.16)
    plt.show()


```

```
def safety_3D_plots (p):
  # generate_combination_samples(np.linspace(20, 30, 11), np.linspace(15, 25, 11), [50], p)

  x = [];
  y = [];
  z = [[],[],[]];

  os.chdir("%s/"%p)
  for file in glob.glob("*.csv"):
    x.append(int(str(file).split('_')[1]))
    y.append(int(str(file).split('_')[2]))

    with open(file) as csvfile:
      reader = csv.DictReader(csvfile)
      for row in reader:
        if row["property"] == 'P=? [F crashed]':
          z[int(row["type_driver"])-1].append(float(row["probability"]))

  fig = plt.figure()
  ax = fig.add_subplot(111, projection='3d')

  line = ax.plot_trisurf(x, y, z[0], label="Aggressive", linewidth=0.2, antialiased=True,
     cmap=cm.viridis)
  line = ax.plot_trisurf(x, y, z[1], label="Average", linewidth=0.2, antialiased=True, cmap=
     cm.plasma)
  line = ax.plot_trisurf(x, y, z[2], label="Cautious", linewidth=0.2, antialiased=True, cmap
     =cm.inferno)

  ax.xaxis._axinfo['label']['space_factor'] = 3.0
  ax.yaxis._axinfo['label']['space_factor'] = 3.0
  ax.zaxis._axinfo['label']['space_factor'] = 3.0

  ax.set_xlabel('v [m/s]')
  ax.set_ylabel('v$_1$ [m/s]')
  ax.set_zlabel('P$_{=?}$ [F crashed]')

  fake2Dline1 = mpl.lines.Line2D([0],[0], linestyle="none", c='b', marker = 'o')
  fake2Dline2 = mpl.lines.Line2D([0],[0], linestyle="none", c='b', marker = 'o')
  fake2Dline3 = mpl.lines.Line2D([0],[0], linestyle="none", c='b', marker = 'o')
  ax.legend([fake2Dline1, fake2Dline2, fake2Dline3], ['Aggressive', 'Average', 'Cautious'],
     numpoints = 1)

  # ax.set_zticks(np.arange(np.min(z)+0.1, np.max(z), 0.1))
  plt.show()


def liveness_2D_plot (p, v_in, v1_in, x1_0_in):
  # generate_samples(1, v=v_in, v1=v1_in, x1_0=x1_0_in, path=p)

  x = [[],[],[]];
  y = [[],[],[]];

  with open("%s/r_%s_%s_%s.csv"%(p,v_in,v1_in,x1_0_in)) as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
      if 'P=? [F ((x = 500) & (t <' in row["property"]:
        x_val = int(row["property"][25:27])
        x[int(row["type_driver"])-1].append(x_val)
        y[int(row["type_driver"])-1].append(float(row["probability"]))

  labels = ["Aggressive", "Average", "Cautious"]

  for i in range(0,3):
    new_x, new_y = zip(*sorted(zip(x[i], y[i])))

    line = plt.plot(new_x, new_y, label=labels[i], marker="s")

  plt.legend(loc='lower right', fontsize=18)

  plt.ylabel('P$_{=?}$ [F (x = 500) \& (t $<$ T) $||$ F (x = 500)]', fontsize=16)
  plt.xlabel('T [s]', fontsize=18)
  # plt.title('Liveness property for $v = %d$, $v_1 = %d$, $x_{1,0} = %d$'%(v_in, v1_in, x1_0_in))
  plt.xticks(np.arange(min(new_x)-1, max(new_x)+1, 2))

  plt.subplots_adjust(right=0.95, top=0.95, left=0.17, bottom=0.16)
  plt.show()


def safety_box_plot(p):
  props_dict = read_files_to_dict(p)

  vals = [[],[],[]]
  vals[0] = props_dict[0]['P=? [F crashed]']
  vals[1] = props_dict[1]['P=? [F crashed]']
  vals[2] = props_dict[2]['P=? [F crashed]']

  plt.boxplot(vals, labels=["Aggressive", "Average", "Cautious"], whis=4)
  plt.ylabel('P$_{=?}$ [F crashed]', fontsize=18)
  # plt.title('Safety property')

  plt.subplots_adjust(right=0.95, top=0.95, left=0.17, bottom=0.12)
  plt.show()


def analysis(p):
  data = {}

  os.chdir("%s/"%p)
  for file in glob.glob("*.csv"):
    v = int(str(file).split('_')[1])

    min_T = 34
    with open(file) as csvfile:
      reader = csv.DictReader(csvfile)
      for row in reader:
        if 'P=? [F ((x = 500) & (t <' in row["property"] and float(row["probability"]) > 0:
          T = int(row["property"][25:27])
          min_T = min(min_T, T)

    if v in data.keys():
      data[v].append(min_T)
    else:
      data[v] = [min_T]

  x = []
  y = []
  ret_d = {}

  for k in data.keys():
    x.append(k)
    y.append(np.floor(np.mean(data[k])))
    ret_d[k] = np.floor(np.mean(data[k]))

  return ret_d


def liveness_box_plot(T, p):
  decision_dict = analysis(p)

  props_dict = [{},{},{}]

  for file in glob.glob("*.csv"):
    v = int(str(file).split('_')[1])
    if not (T >= decision_dict[v] and T <= decision_dict[v] + 3):
      continue

    with open(file) as csvfile:
      reader = csv.DictReader(csvfile)
      for row in reader:
        if row["probability"] != "inf" and 'P=? [F ((x = 500) & (t <' in row["property"] and row["property"] in props_dict[int(row["type_driver"])-1].keys():
          props_dict[int(row["type_driver"])-1][row["property"]].append(float(row["probability"]))
        elif row["probability"] != "inf" and 'P=? [F ((x = 500) & (t <' in row["property"]:
          props_dict[int(row["type_driver"])-1][row["property"]] = [float(row["probability"])]

  ts = [[],[],[]]
  time_val = T
  # Conditional properties
  ts[0] = props_dict[0]['P=? [F ((x = 500) & (t < %d)) || F (x = 500)]'%time_val]
  ts[1] = props_dict[1]['P=? [F ((x = 500) & (t < %d)) || F (x = 500)]'%time_val]
  ts[2] = props_dict[2]['P=? [F ((x = 500) & (t < %d)) || F (x = 500)]'%time_val]

  # Unconditional properties
  # ts[0] = props_dict[0]['P=? [F ((x = 500) & (t < %d))]'%time_val]
  # ts[1] = props_dict[1]['P=? [F ((x = 500) & (t < %d))]'%time_val]
  # ts[2] = props_dict[2]['P=? [F ((x = 500) & (t < %d))]'%time_val]

  plt.boxplot(ts, labels=["Aggressive", "Average", "Cautious"], whis=1.5)
  plt.ylabel('P$_{=?}$ [F (x = 500) \& (t $<$ %d) $||$ F (x = 500)]'%time_val, fontsize=16)
  # plt.ylabel('P$_{=?}$ [F ((x = 500) \& (t $<$ %d))]'%time_val)
  # plt.title('Liveness property')
  plt.ylim(-0.05,1.05)

  plt.subplots_adjust(right=0.95, top=0.95, left=0.17, bottom=0.12)
  plt.show()


safety_plots('plot1', 20, 35)
# safety_plots('plot2', 22, 40)
# safety_3D_plots('plot3')
# liveness_2D_plot('plot4', 21, 19, 70)
# liveness_2D_plot('plot4', 26, 22, 45)
# generate_samples(250, path="box_plots")
# safety_box_plot("box_plots")
# liveness_box_plot(21, "box_plots")
# liveness_box_plot (22, "box_plots")
```

C.2 Plot Generator for experiments with the Decision Making and Full Control ADAS (Python)

File C.2 plots.py

```
# PLOTS - Generate some relevant plots that relate to the performance
# of the driver + the ADAS.

# Author: Francisco Girbal Eiras, MSc Computer Science
# University of Oxford, Department of Computer Science
# Email: francisco.eiras@cs.ox.ac.uk
# 13-Jul-2018; Last revision: 17-Jul-2018

import sys, os, random, glob, csv, subprocess, itertools
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib import rcParams
import matplotlib as mpl

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size=24)

# Generate samples for some of the graphs
# seq = True iff two out of (v, v1, x1_0) are set
#
def generate_samples(N, *args, **kwargs):
  v_set = kwargs.get('v', None)
  v1_set = kwargs.get('v1', None)
  x1_0_set = kwargs.get('x1_0', None)
  path = kwargs.get('path', None)
  seq = bool(kwargs.get('seq', False))

  v_count = 15
  v1_count = 15
  x1_0_count = 10

  for i in range(1,N+1):
    while 1:
      if v1_set == None and seq == False:
        v1 = random.randint(15,34)
      elif v1_set == None and seq == True:
        v1 = v1_count
        v1_count = v1_count + 1
      else:
        v1 = int(v1_set)

      if v_set == None and seq == False:
        v = random.randint(15,34)
      elif v_set == None and seq == True:
        v = v_count
        v_count = v_count + 1
```

```python
            else:
                v = int(v_set)

            if x1_0_set == None and seq == False:
                x1_0 = random.randint(35,80)
            elif x1_0_set == None and seq == True:
                x1_0 = x1_0_count
                x1_0_count = x1_0_count + 1
            else:
                x1_0 = int(x1_0_set)

            if v_count > 35 or v1_count > 35 or x1_0_count > 81:
                return

            if path == None:
                if not os.path.exists("%s/r_%s_%s_%s.csv"%(path,v,v1,x1_0)):
                    break
            else:
                if not os.path.exists("%s/r_%s_%s_%s.csv"%(path,v,v1,x1_0)):
                    break

        print('[%d/%d]: Evaluating drivers for v = %d, v1 = %d, x1_0 = %d...'%(i,N,v,v1,x1_0))
        if path == None:
            os.system('python3 automatic_model_checker.py properties/verification.pctl %d %d %d'%(v,v1,x1_0))
        else:
            os.system('python3 automatic_model_checker.py properties/verification.pctl %d %d %d --path %s'%(v,v1,x1_0,path))
        # output = str(proc.stderr.read())


def generate_combination_samples(vs, v1s, x1_0s, p):
    sz = len(vs)*len(v1s)*len(x1_0s)
    print('Evaluating %d combinations...'%sz)

    for v in vs:
        for v1 in v1s:
            for x1_0 in x1_0s:
                print('v = %d, v1 = %d, x1_0 = %d:'%(v,v1,x1_0))

                if os.path.exists("%s/r_%d_%d_%d.csv"%(p,v,v1,x1_0)):
                    continue

                os.system('python3 automatic_model_checker.py properties/safety.pctl %d %d %d --path %s'%(v,v1,x1_0,p))


def generate_samples_box_plots(N):
    n = 1
    while n < N + 1:
        l = random.randint(0,249)

        f = open('values.txt')
        line = f.readline()
        i = 0
        while i < l:
            line = f.readline()
            i += 1

    if  os.path. exists ('box_plots/%s'%line):
      continue

    v = line. split ('_') [1]
    v1 = line. split ('_') [2]
    x1_0 = line. split ('_') [3][0:2]

    print ( str (n) + ": v = " + v + ", v1 = " + v1 + ", x1_0 = " + x1_0)

    if  os.system('python3 automatic_model_checker.py  properties / verification  . pctl  %s %s %s
    −−path box_plots'%(v,v1,x1_0)) != 0:
      exit ()

    n = n + 1


def   read_files_to_dict  (path):
  props_dict  = [{},{},{}]

  os. chdir ("%s/"%path)
  for  file  in  glob.glob("∗.csv"):
    with  open( file )  as  csvfile :
      reader  = csv.DictReader( csvfile )
      for  row  in  reader:
        if  row["property"]  in  props_dict [ int (row[" type_driver "]) −1].keys():
          props_dict [ int (row[" type_driver "]) −1][row["property" ]]. append(float (row["  probability
  "]))
        else :
          props_dict [ int (row[" type_driver "]) −1][row["property" ]]  = [ float (row["  probability "])]

  return  props_dict


def  safety_plots (p,  v1_in ,  x1_0_in ):
  # generate_samples(20, v1=v1_in, x1_0=x1_0_in, path=p, seq=True)

  x = [];
  y =   [[],[],[]];

  os. chdir ("%s/"%p)
  for  file  in  glob.glob("∗.csv"):
    x.append(int( str ( file ). split ('_') [1]) )

    with  open( file )  as  csvfile :
      reader  = csv.DictReader( csvfile )
      for  row  in  reader:
        if  row["property"]  == 'Pmin=? [ F crashed ]':
          y[ int (row[" type_driver "]) −1].append(float(row["  probability "]) )

  labels  = ["Aggressive" , "Average", "Cautious"]

  for  i  in  range(0,3):
    new_x, new_y = zip(∗sorted( zip (x,  y[ i ]) ) )

    line  = plt. plot (new_x,  new_y,  label =labels [ i ],  marker="s")

  plt .legend( loc='upper  left ',   fontsize =18)

  plt . ylabel ('P$_{min=?}$ [F crashed]',  fontsize =18)
```

```
    plt.xlabel('v [m/s]', fontsize=18)
    # plt.title('Safety property for $v_1 = %d$, $x_{1,0} = %d$'%(v1_in, x1_0_in))
    plt.xticks(np.arange(min(new_x)-1, max(new_x)+1, 2))

    plt.subplots_adjust(right=0.95, top=0.95, left=0.19, bottom=0.16)
    plt.show()


def safety_3D_plots(p):
    # generate_combination_samples(np.linspace(20, 30, 11), np.linspace(15, 25, 11), [50], p)

    x = [];
    y = [];
    z = [[],[],[]];

    os.chdir("%s/"%p)
    for file in glob.glob("*.csv"):
        x.append(int(str(file).split('_')[1]))
        y.append(int(str(file).split('_')[2]))

        with open(file) as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                if row["property"] == 'Pmin=? [ F crashed ]':
                    z[int(row["type_driver"])-1].append(float(row["probability"]))

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    line = ax.plot_trisurf(x, y, z[0], label="Aggressive", linewidth=0.2, antialiased=True,
        cmap=cm.viridis)
    line = ax.plot_trisurf(x, y, z[1], label="Average", linewidth=0.2, antialiased=True, cmap=
        cm.plasma)
    line = ax.plot_trisurf(x, y, z[2], label="Cautious", linewidth=0.2, antialiased=True, cmap
        =cm.inferno)

    ax.xaxis._axinfo['label']['space_factor'] = 3.0
    ax.yaxis._axinfo['label']['space_factor'] = 3.0
    ax.zaxis._axinfo['label']['space_factor'] = 3.0

    ax.set_xlabel('v [m/s]')
    ax.set_ylabel('v$_1$ [m/s]')
    ax.set_zlabel('P$_{min=?}$ [F crashed]')

    fake2Dline1 = mpl.lines.Line2D([0],[0], linestyle="none", c='b', marker = 'o')
    fake2Dline2 = mpl.lines.Line2D([0],[0], linestyle="none", c='b', marker = 'o')
    fake2Dline3 = mpl.lines.Line2D([0],[0], linestyle="none", c='b', marker = 'o')
    ax.legend([fake2Dline1, fake2Dline2, fake2Dline3], ['Aggressive', 'Average', 'Cautious'],
        numpoints = 1)

    # ax.set_zticks(np.arange(np.min(z)+0.1, np.max(z), 0.1))
    plt.show()


def liveness_2D_plot(p, v_in, v1_in, x1_0_in):
    # generate_samples(1, v=v_in, v1=v1_in, x1_0=x1_0_in, path=p)

    x = [[],[],[]];
    y = [[],[],[]];
    pmin_crashed = [0,0,0];

    with open("%s/r_%s_%s_%s.csv"%(p,v_in,v1_in,x1_0_in)) as csvfile :
        reader = csv.DictReader( csvfile )
        for row in reader:
            if 'Pmin=? [ F crashed ]' in row["property"]:
                pmin_crashed[int(row["type_driver"])-1] = float(row["probability"])

            if 'Pmax=? [ F (x=500 & t < ' in row["property"]:
                x_val = int(row["property"][24:26])
                x[int(row["type_driver"])-1].append(x_val)
                y[int(row["type_driver"])-1].append(float(row["probability"]))

    labels = ["Aggressive", "Average", "Cautious"]

    for i in range(0,3):
        for l in range(0, len(x[i])):
            y[i][l] = min(y[i][l]/(1-pmin_crashed[i]) ,1)

    for i in range(0,3):
        x[i].append(27)
        y[i].append(1)

        x[i].append(28)
        y[i].append(1)

        new_x, new_y = zip(*sorted(zip(x[i], y[i])))

        line = plt.plot(new_x, new_y, label=labels[i], marker="s")

    plt.legend(loc='lower right', fontsize=18)

    # plt.ylabel('P$_{max=?}$ [F (x = 500) \& (t $<$ T)]/P$_{max=?}$ [F (x = 500)]', fontsize
        =15)
    plt.ylabel('$\zeta(T)$', fontsize=18)
    plt.xlabel('T [s]', fontsize=18)
    # plt.title('Liveness property for $v = %d$, $v_1 = %d$, $x_{1,0} = %d$'%(v_in, v1_in,
        x1_0_in))
    plt.xticks(np.arange(min(new_x)-1, max(new_x)+1, 2))

    plt.subplots_adjust(right=0.95, top=0.95, left=0.19, bottom=0.16)
    plt.show()


def safety_box_plot(p):
    props_dict = read_files_to_dict (p)

    vals = [[],[],[]]
    vals[0] = props_dict[0]['Pmin=? [ F crashed ]']
    vals[1] = props_dict[1]['Pmin=? [ F crashed ]']
    vals[2] = props_dict[2]['Pmin=? [ F crashed ]']

    plt.boxplot(vals, labels=["Aggressive", "Average", "Cautious"], whis=4)
    plt.ylabel('P$_{min=?}$ [F crashed]', fontsize=18)
    # plt.title('Safety property')

    plt.subplots_adjust(right=0.95, top=0.95, left=0.17, bottom=0.12)
    plt.show()

```

```

def analysis(p):
  data = {}

  os.chdir("%s/"%p)
  for file in glob.glob("*.csv"):
    v = int(str(file).split('_')[1])

    min_T = 34
    with open(file) as csvfile:
      reader = csv.DictReader(csvfile)
      for row in reader:
        if 'Pmax=? [ F (x=500 & t < ' in row["property"] and float(row["probability"]) > 0:
          T = int(row["property"][24:26])
          min_T = min(min_T, T)

    if v in data.keys():
      data[v].append(min_T)
    else:
      data[v] = [min_T]

  x = []
  y = []
  ret_d = {}

  for k in data.keys():
    x.append(k)
    y.append(np.floor(np.mean(data[k])))
    ret_d[k] = np.floor(np.mean(data[k]))

  return ret_d


def liveness_box_plot(T, p):
   decision_dict = analysis(p)

   props_dict = [{},{},{}]

  for file in glob.glob("*.csv"):
    v = int(str(file).split('_')[1])
    if not (T >= decision_dict[v] and T <= decision_dict[v] + 3):
      continue

    with open(file) as csvfile:
      reader = csv.DictReader(csvfile)

      pmin = [0,0,0]

      for row in reader:
        if 'Pmin=? [ F crashed ]' in row["property"]:
          pmin[int(row["type_driver"])-1] = float(row["probability"])

        if pmin[int(row["type_driver"])-1] == 1:
          continue

        if row["probability"] != "inf" and 'Pmax=? [ F (x=500 & t < ' in row["property"] and row["property"] in props_dict[int(row["type_driver"])-1].keys():
          props_dict[int(row["type_driver"])-1][row["property"]].append(float(row["probability"])/(1-pmin[int(row["type_driver"])-1]))
```

```
            elif  row["probability"] != "inf" and 'Pmax=? [ F (x=500 & t < ' in row["property"]:
                props_dict[int(row["type_driver"])-1][row["property"]] = [float(row["probability"])
    /(1-pmin[int(row["type_driver"])-1])]

    ts =  [[],[],[]]
    time_val = T
    # Conditional properties
    # ts[0] = props_dict [0]['P=? [F ((x = 500) & (t < %d)) || F (x = 500)]'%time_val]
    # ts[1] = props_dict [1]['P=? [F ((x = 500) & (t < %d)) || F (x = 500)]'%time_val]
    # ts[2] = props_dict [2]['P=? [F ((x = 500) & (t < %d)) || F (x = 500)]'%time_val]

    # Unconditional properties
    ts[0] = props_dict[0]['Pmax=? [ F (x=500 & t < %d) ]'%time_val]
    ts[1] = props_dict[1]['Pmax=? [ F (x=500 & t < %d) ]'%time_val]
    ts[2] = props_dict[2]['Pmax=? [ F (x=500 & t < %d) ]'%time_val]

    plt.boxplot(ts, labels=["Aggressive", "Average", "Cautious"], whis=1.5)
    # plt.ylabel('P$_{=?}$ [F ((x = 500) \& (t $<$ %d)) $||$ F (x = 500)]'%time_val)
    plt.ylabel('$\zeta(%d)$'%time_val, fontsize=18)
    # plt.title('Liveness property')
    plt.ylim(-0.05,1.05)

    plt.subplots_adjust(right=0.95, top=0.95, left=0.17, bottom=0.12)
    plt.show()


safety_plots('plot1', 20, 35)
# safety_plots('plot2', 22, 40)
# safety_3D_plots('plot3')
# liveness_2D_plot('plot4', 21, 19, 70)
# liveness_2D_plot('plot4', 26, 22, 45)
# generate_samples(40, path="box_plots")
# generate_samples_box_plots(35)
# safety_box_plot("box_plots")
# liveness_box_plot(21, "box_plots")
# liveness_box_plot(22, "box_plots")
```

Bibliography

[1] S. Singh, “Critical reasons for crashes investigated in the national motor vehicle crash causation survey,” tech. rep., 2015.

[2] M. Burgess, “When does a car become truly autonomous? Levels of self-driving technology explained.” `https://www.wired.co.uk/article/autonomous-car-levels-sae-ranking`, Mar 2018 (accessed August 18, 2018).

[3] “Autopilot.” `http://www.tesla.com/en_GB/autopilot`, (accessed August 18, 2018).

[4] A. J. Hawkins, “Ford’s new driver-assist system isn’t Autopilot, but it’s a step in the right direction.” `http://www.theverge.com/2018/3/15/17126162/ford-driver-assist-technology-copilot-360`, Mar 2018 (accessed August 18, 2018).

[5] R. Lawler, “Tesla: Autopilot was engaged in fatal Model X crash.” `http://www.engadget.com/2018/03/30/tesla-autopilot-model-x-crash-mountain-view/`, Mar 2018 (accessed August 18, 2018).

[6] “HWY18FH011 Preliminary Accident Report.” `https://www.ntsb.gov/investigations/AccidentReports/Pages/HWY18FH011-preliminary.aspx`, Jun 2018 (accessed August 18, 2018).

[7] C. Baier and J.-P. Katoen, *Principles of model checking.* MIT press, 2008.

[8] J. R. Anderson, *The Architecture of Cognition.* Psychology Press, 2013.

[9] N. Kalra and S. M. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?,” *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[10] D. Sportillo, A. Paljic, M. Boukhris, P. Fuchs, L. Ojeda, and V. Roussarie, "An immersive virtual reality system for semi-autonomous driving simulation: a comparison between realistic and 6-dof controller-based interaction," in *Proceedings of the 9th International Conference on Computer and Automation Engineering*, pp. 6–10, ACM, 2017.

[11] S. Baltodano, S. Sibi, N. Martelaro, N. Gowda, and W. Ju, "The rrads platform: a real road autonomous driving simulator," in *Proceedings of the 7th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pp. 281–288, ACM, 2015.

[12] M. Zhou, X. Qu, and S. Jin, "On the impact of cooperative autonomous vehicles in improving freeway merging: a modified intelligent driver model-based approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1422–1428, 2017.

[13] A. Gruber, M. Gadringer, H. Schreiber, D. Amschl, W. Bösch, S. Metzner, and H. Pflügl, "Highly scalable radar target simulator for autonomous driving test beds," in *Radar Conference (EURAD), 2017 European*, pp. 147–150, IEEE, 2017.

[14] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[15] J. Somers, "The coming software apocalypse." `http://www.theatlantic.com/technology/archive/2017/09/saving-the-world-from-code/540393/`, Sep 2017 (accessed August 18, 2018).

[16] M. Z. Kwiatkowska, "Model checking and strategy synthesis for stochastic games: From theory to practice (invited talk)," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 55, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[17] B. Abbott, T. Bapty, C. Biegl, G. Karsai, and J. Sztipanovits, "Model-based software synthesis," *IEEE Software*, no. 3, pp. 42–52, 1993.

[18] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: Formal models, validation, and synthesis," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 366–390, 1997.

[19] M. Mazo, A. Davitian, and P. Tabuada, “Pessoa: A tool for embedded controller synthesis,” in *International Conference on Computer Aided Verification*, pp. 566–569, Springer, 2010.

[20] T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche, “Synthesis for multi-objective stochastic games: An application to autonomous urban driving,” in *International Conference on Quantitative Evaluation of Systems*, pp. 322–337, Springer, 2013.

[21] D. Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia, “Data-driven probabilistic modeling and verification of human driver behavior,” 2014.

[22] N. Li, D. W. Oyler, M. Zhang, Y. Yildiz, I. Kolmanovsky, and A. R. Girard, “Game theoretic modeling of driver and vehicle interactions for verification and validation of autonomous vehicle control systems,” *IEEE Transactions on control systems technology*, 2017.

[23] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres, “Formal verification of autonomous vehicle platooning,” *Science of Computer Programming*, vol. 148, pp. 88–106, 2017.

[24] P. Nilsson, O. Hussien, Y. Chen, A. Balkan, M. Rungger, A. Ames, J. Grizzle, N. Ozay, H. Peng, and P. Tabuada, “Preliminary results on correct-by-construction control software synthesis for adaptive cruise control,” in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pp. 816–823, IEEE, 2014.

[25] C. Lam, “Driver Assistance Using Cognitive Modelling and Strategy Synthesis,” 2017.

[26] D. D. Salvucci, “Modeling driver behavior in a cognitive architecture,” *Human factors*, vol. 48, no. 2, pp. 362–380, 2006.

[27] P. Curzon, R. Rukšėnas, and A. Blandford, “An approach to formal verification of human–computer interaction,” *Formal Aspects of Computing*, vol. 19, no. 4, pp. 513–550, 2007.

[28] E. R. Boer and M. Hoedemaeker, “Modeling driver behavior with different degrees of automation: A hierarchical decision framework of interacting mental

models," in *Proceedings of the 17th European annual conference on human decision making and manual control*, pp. 63–72, 1998.

[29] A. Liu and D. Salvucci, "Modeling and prediction of human driver behavior," in *Intl. Conference on HCI*, 2001.

[30] D. D. Salvucci and A. Liu, "The time course of a lane change: Driver control and eye-movement behavior," *Transportation research part F: traffic psychology and behaviour*, vol. 5, no. 2, pp. 123–132, 2002.

[31] D. Salvucci, E. Boer, and A. Liu, "Toward an integrated model of driver behavior in cognitive architecture," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1779, pp. 9–16, 2001.

[32] I. Kotseruba and J. K. Tsotsos, "A review of 40 years of cognitive architecture research: Core cognitive abilities and practical applications," *arXiv preprint arXiv:1610.08602*, 2016.

[33] R. Sun, "The importance of cognitive architectures: An analysis based on CLARION," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 19, no. 2, pp. 159–193, 2007.

[34] J. R. Anderson, M. Matessa, and C. Lebiere, "ACT-R: A theory of higher level cognition and its relation to visual attention," *Human-Computer Interaction*, vol. 12, no. 4, pp. 439–462, 1997.

[35] T. Balke and N. Gilbert, "How do agents make decisions? A survey," *Journal of Artificial Societies and Social Simulation*, vol. 17, no. 4, p. 13, 2014.

[36] E. R. Boer and M. Hoedemaeker, "Modeling driver behavior with different degrees of automation: A hierarchical decision framework of interacting mental models," in *Proceedings of the 17th European annual conference on human decision making and manual control*, pp. 63–72, 1998.

[37] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, "Automated verification techniques for probabilistic systems," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pp. 53–113, Springer, 2011.

[38] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pp. 220–270, Springer, 2007.

[39] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *International conference on computer aided verification*, pp. 585–591, Springer, 2011.

[40] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pp. 200–204, Springer, 2002.

[41] "PRISM Manual | The PRISM Language | Example 1." `http://www.prismmodelchecker.org/manual/ThePRISMLanguage/Example1`, Dec 2010 (accessed August 18, 2018).

[42] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *International Conference on Computer Aided Verification*, pp. 592–600, Springer, 2017.

[43] T. Chen, V. Forejt, M. Kwiatkowska, A. Simaitis, and C. Wiltsche, "On stochastic games with multiple objectives," in *International Symposium on Mathematical Foundations of Computer Science*, pp. 266–277, Springer, 2013.

[44] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *Tools for Practical Software Verification*, pp. 1–30, Springer, 2012.

[45] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for MDPs.," in *ISAIM*, 2006.

[46] A. Mohr, "A survey of state abstraction techniques for markov decision processes,"

[47] H. C. Manual, "Highway capacity manual," *Washington, DC*, vol. 11, 2000.

[48] P. Shinners, "pygame." `https://github.com/pygame/pygame`, 2011.

[49] J. Carr, "State "keep right" laws." `http://www.mit.edu/~jfc/right.html`, (accessed August 18, 2018).

[50] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis, "PRISM-games: A model checker for stochastic multi-player games," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 185–191, Springer, 2013.