Modelling Fault Tolerance using Deontic Logic: a case study

Modelling Fault Tolerance using Deontic Logic: a case study

By

Jamil Ahmed Khan, B.Sc. Engg.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements for the Degree of

Master of Science
Department of Computing and Software
McMaster University

Master of Science (2012)
(Computing and Software)

McMaster University
Hamilton, Ontario, Canada

TITLE:	Modelling Fault Tolerance using Deontic Logic: a case study
AUTHOR:	Jamil Ahmed Khan B.Sc. Engg. (Computer Science and Information technology) Islamic University of Technology, Gazipur, Bangladesh
SUPERVISOR:	Dr. T. S. E. Maibaum
NUMBER OF PAGES:	xii, 115

To my mom and dad

Abstract

Many computer systems in our daily life require highly available applications (such as medical equipment) and some others run on difficult to access places (such as satellites). These systems are subject to a variety of potential failures that may degrade their performance. Therefore, being able to reason about faults and their impact on systems is gaining considerable attention. Existing work on fault tolerance is mostly focused on addressing faults at the programming language level. In the recent past, significant efforts have been made to use formal methods to specify and verify fault tolerant systems to provide more reliable software. Related with this, some researchers have pointed out that Deontic Logic is useful for reasoning about fault tolerant systems due to its expressive nature in relation to defining norms, used to describe expected behaviour and prescribing what happens when these norms are violated.

In this thesis, we demonstrate how Deontic Logic can be used to model an existing real world problem concerning fault tolerance mechanisms. We consider different situations that a vehicle faces on the road and the consequent reactions of the driver or vehicle based on good and bad behaviour. We got the idea and motivation for this case study from the **SASPENCE** sub-project, conducted under the *European Integrated Project* **PReVENT**. This sub-project focuses on a vehicle's behaviour in maintaining safe speed and safe distance on the road. As our first modelling attempt, we use a Propositional Deontic Logic approach, to justify to what extent we can apply this Logical approach to model a real world problem. Subsequently, we use a First Order Deontic Logic approach, as it can incorporate the use of parameters and quantification over them, which is more useful to model real world scenarios.

We state and prove some interesting expected properties of the models using a First Order proof system. Based on these modelling exercises, we acquired different engineering ideas and lessons, and present them in this thesis in order to aid modelling of future fault tolerant systems.

Acknowledgements

This thesis would not have been possible without the support of many people. First of all, many thanks to my supervisor, Dr. Thomas Maibaum, who has always been very flexible to me and have guided me throughout the whole research with his patience and knowledge. Without him this thesis would not have been completed or written. Special thanks to Pablo Castro for his valuable advice and guidance during this thesis. I cannot help thanking Dr. Alan Wassyng and Dr. Mark Lawford who offered valuable advice and comments during our weekly group meeting. I also want to thank my colleagues, Ramiro Demasi and Valentin Cassano for their help during my research work. Finally, thanks to my parents and numerous friends who endured this long process with me, always offering support and love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Existing systems are highly complex, have lots of components interacting with each other and the environment and are highly susceptible to faults. These faults disrupt the functionality and availability of software systems and sometime generate serious consequences costing time, money and even human lives. As a result, fault tolerance is, and has been, a field of active research in the software industry. Fault tolerant systems can continue running even in the presence of faults and have the capability to recover from any unexpected behaviour during their execution. Most existing software systems implement fault tolerance in low the level programming phase through code replication, voting algorithms and exception mechanisms. But it has been demonstrated that modelling and mathematical reasoning at the design level is very significant and valuable for developing fault tolerant critical systems, generally saving on costs as compared to the approach that introduces fault tolerance at a later development stage. In this thesis we aim to undertake a case study: high level modelling of an existing real world problem where fault tolerance is a major concern.

1.2 Why Deontic Logic

In recent years, several researchers have proposed to use formal methods with fault tolerance techniques to ensure reliability of software systems. Extensive work on formal specification and model checking has been done in this area and demonstrated these methods are very useful to develop high quality software. But mostly these methods are designed to deal with fault mitigation or fault elimination in software systems and are not good enough to structure fault tolerant application due to lack of modular reasoning over violations. Consequently some researchers ([CJ96, WM93, MT84, KQM91, Kho98]) have pointed out that deontic logic, a variation of logic designed for the study of norms, is useful for reasoning abstractly about fault-tolerant systems. The notion of permission and obligation is naturally embedded within this logic and widely used by philosophers and lawyers to investigate reasoning about norms. This logic is expressive enough to characterize faulty scenarios and distinguish good and bad behaviour. Violations and recovery mechanisms can be effectively defined within this logical formalism. In this thesis, we focus on using this logical framework to specify a real world example, try to formulate different behaviours, model violation scenarios and recovery mechanisms to overcome from these bad behaviours. Based on this work, we also try to derive some engineering lessons about the obstacles to be overcome and the ideas required to characterize and solve high level fault tolerance problems.

1.3 Overview of the Problem

In this thesis our case study focus is to model the behaviour of moving vehicles. We have studied some real world examples related with automotive applications and found that the scenarios related with a moving vehicle exhibit very interesting problems to formalize as there are lots of violation scenarios with which deontic logic is very much capable to deal with. We got the idea and motivation for this problem from [MAH05], [FTO05]: **SASPENCE** sub-project which ran under the *European Integrated Project* **PReVENT** . PReVENT is supporting the European Commission actions that promote the development, deployment and use of Intelligent Vehicle Safety Systems in Europe. In PReVENT, a number of sub-projects are proposed in different function fields: Safe Speed and Safe Following, Lateral Support and Driver Monitoring, Intersection Safety, Vulnerable Road Users and Collision Mitigation [MAH05].

The PReVENT sub-project SASPENCE, following the common objectives of the functional field named **Safe Speed and Safe Distance**, aims at developing and evaluating an innovative system able to implement the Safe Speed and Safe Distance concept. The main motivation is to aid the driver in avoiding accident situations related to excessive speed or too short ahead way. This project develops an expert system with radars and sensors that help and assist drivers in maintaining safe speed and distance. Thus, the system is supposed to cooperate seamlessly with the driver, suggesting the proper speed for the given conditions (such as: dangerous curve ahead, frontal obstacles, etc.) in order to prevent risky and dangerous situations (due to incorrect and inappropriate distance or excessive speed) and, lastly, to avoid a collision [MAH05].

Being motivated by this sub-project, we want to apply the Deontic Logic formalism to model the safe speed and safe distance concept of a moving vehicle. This SASPENCE sub-project gave us ideas related to the safe speed and safe distance feature in automotive system. This project defined **safe speed** based on the road structure, weather conditions and road conditions [FTO05]. Similarly we also see roads and highways have some speed limitations like **maximum speed limitation** and **minimum speed limitation** and a vehicle must follow or is bound to maintain those limitations. These limitations often also depend on road conditions and road structure. Here in this thesis we are not focusing on how to determine this speed limitations, rather our focus is on how a vehicle adheres to these limitations, what may happen if they violate these limitations and how they can recover from the violation. On the other hand, the concept of **safe distance** relates to the distance that every vehicle must maintain from its obstacles [FTO05] i.e., the immediate front and rear vehicles. Maintaining safe distance ensures that a vehicle will not crash with its front vehicle even if the front vehicle suddenly stops [FTO05]. The SASPENCE project presents a speed and safe distance relationship in

[MAH05]. It defines the safety distance based on the current speed of a vehicle, i.e., the safety distance of a vehicle when it is travelling at high speed and at low speed differs. It can also happen that opponent (i.e. immediate front and rear) vehicle's speed may play an important role in defining the safe distance. Like the safe speed concept, we will also use the safe distance concept in our formulation and will see how vehicles maintain distance limitations with respect to other vehicles and the consequences when the limitations are not maintained.

Cut-in vehicle is another important concept which has a very close relationship with safety distance. A vehicle that joins at the host vehicle's (a particular vehicle of our concern) lane from the adjacent lane and, takes immediate front or rear position with respect to the host vehicle, is considered as cut-in vehicle. The SASPENCE project implements a safety indication system for cut-in vehicle and suggests the host vehicle about required actions based on the cut-in vehicle's speed and distance. In this work we will also try to see how a cut-in vehicle changes the current condition of our focused (host) vehicle.

The SASPENCE sub-project mostly implements the safety indication system based on the current speed and distance with respect to other vehicles. On the other hand, our focus in this work is to apply Deontic Logic to see how a vehicle maintains the safe speed and safe distance restrictions, when they violate these limitations, how they can recover from these violations and, lastly, the possible situations that can arise if they cannot recover from some violation.

The SASPENCE project defined some scenarios in [FTO05] related with safe speed and safe distance. We have also obtained ideas about the situations that vehicles may face on the road and how they should react from some research work at the **National Highway Traffic Safety Administration (NHTSA)**. Many scenarios are described in [JJFN07] where they present crash imminent test cases. They also analysed pre-crash scenarios in their work. We have merged some of these ideas with the SASPENCE concept and also added our general ideas on vehicle motion to formulate our scenarios.

1.4 Thesis Organization

In chapter 1 we have presented a brief introduction to our work. We mentioned the motivation of this work, the reasons for choosing deontic logic as our modelling language and an overview of the problem that we are formulating in this thesis. This problem overview also presents a brief history of how we developed the concepts about scenarios that we are modelling.

Fault tolerance is the major concern of our work. We have studied some background literature on fault tolerance mechanisms. Here we will see how fault tolerance is currently achieved during the low level implementation phase and, occasionally, at the high level design phase. In Chapter 2 we present a brief discussion of these different fault tolerance approaches.

Deontic logic is very effective in modelling fault tolerant systems and in this thesis we are using deontic logic for modelling a real world problem. In chapter 3 we discuss deontic logic and mostly our focus in this chapter is to present the propositional and first-order deontic logic approach that Pablo Castro has developed in his Ph.D. thesis and subsequent work. We will use both the propositional and first-order approach in our modelling.

In chapter 4 we describe in detail the problem that we are modelling is this thesis. Here we discuss the scenarios in which a moving vehicle engages and the features and requirements that our model should have.

Chapter 5 presents our first model using the propositional deontic logic approach. This model is somewhat simpler than the model in the first order deontic logic approach as in this model we only consider the motion of a focused vehicle. Though sometimes a vehicle's motion depends on another vehicle, here we avoided those situations. Here our consideration is that a vehicle is responsible for all the situations that it faces, but in reality it is not!

Chapter 6 presents our second model in first order deontic logic. The reason for choosing the first order deontic logic approach is that in the propositional approach we cannot model complex concepts where quantification over parameter values is necessary. In this chapter we implement a much more advanced model than the model in chapter 5. Here we consider both the motion of a focused vehicle and the possible front and rear opponent vehicles. The idea behind this consideration is that the situations a focused vehicle faces (for example the violations) depend on both the focused vehicle and the other immediate front or rear vehicles. In this chapter we also present some expected properties of the system and prove that these properties are satisfied.

During these modelling exercises we have faced different obstacles and followed different approaches to overcome them. We also found some merits and demerits of our modelling approaches. In chapter 7 we present a description of all the obstacles, ideas and lessons that we have encountered during our modelling and analysis exercises. We are hoping that this knowledges will aid others in modelling fault tolerant systems.

In chapter 8 we conclude our thesis. Here we present the contributions that we have made in this work and present some ideas for future work.

Chapter 2

Fault Tolerance Mechanisms

2.1 Fault Tolerance

Many computer systems in our daily life require high availability of applications (for example, medical equipment) and some others run on difficult-to-access places (for example, satellites). These systems are subject to a variety of potential failures that may endanger their performance. We have seen in the literature that there exist several techniques to implement fault-tolerance (e.g., code replication, voting algorithms and exception mechanisms), but most of them are applied during the implementation phase. In fact, obtaining a fault tolerant system cannot be achieved just by adding redundant modules to a system at this stage, additionally, systematic techniques need to be used to determine its correct behaviour at design time.

Existing work on fault tolerance is mostly focused on faults at the programming language level, e.g., [Tp00, XSS, Ram07]. However, in the last few decades, significant effort has been made to use formal methods such as program transformations [Gär98, PJ94, AK98], process algebra based approaches [BFS00, JL93, GLM05], and specification languages [LM94, Abr06] to specify and verify fault-tolerant systems to provide more reliable software. Some researchers [CJ96,WM93,MT84,KQM91,Kho98] have proposed that deontic logic is very useful for reasoning about fault-tolerant systems due to its strong focus on dealing with norms.

2.2 General Techniques at the concrete level

There are much work on fault tolerance techniques at the concrete level (such as [Tp00], [XSS], [Ram07]). The author in [Tp00] categorizes software fault tolerance in single-version and multi-version techniques.

Single-version fault tolerance techniques use redundancy on a single version of software for error detection and recovery. He considers error detection, exception handling, checkpoint and restart, process pairs and data diversity as single-version fault tolerance techniques.

On the other hand, multi-version fault tolerance techniques use more than one version of a software that execute either in parallel or sequentially. Recovery block, retry block, N-version programming, N-copy programming, N-self checking programming are examples of multi-version fault tolerance techniques. In [XSS] the above mentioned techniques are categorized in terms of design diversity and data diversity. Separate design and implementation is done to obtain the same service in the design diversity category. The goal is to minimize the occurrence of identical errors from different modules. Data diversity uses related sets of points in the program data space. A decision system determines the resulting output while same software uses those points in order to find the changes in execution conditions. Recovery block, N-version programming, N-self checking programming fall to the design diversity category and retry block and N-copy programming fall to the data diversity category. Now we will discuss some of these techniques.

Assertion.

An assertion is a predicate placed in a program with the intention that the predicate is always true at that place. If the assertion becomes false, then it has to be recognized that something erroneous has occurred and corresponding action has to be taken for recovery [Ram07]. There are many programming languages that support assertions (for example pre and post conditions) and the major benefit is that it can be used to detect errors immediately and directly.

Checkpoint and Restart

This is a backward recovery approach where the variables of a system, its environment, register values, and control information are saved periodically [Tp00, Ram07] in store. The system restarts from a safe state if it detects any fault. This approach is highly applicable in case of unanticipated faults and if there is any unrecoverable action in the system. Restart can be done in static and dynamic ways. A system restarts from a predetermined checkpoint in a static restart system. And in a dynamic restart mechanism checkpoints are taken dynamically based on some timing or optimization rules [Tp00].

Recovery block

Recovery block is a fault tolerance approach where more than one version (normally two) of a segment of a software is used with check points and recovery ([Tp00], [XSS], [Ram07]). A check point is taken before the system executes that specific segment. An activation module determines if the system will take primary alternate execution (most efficient execution) or secondary alternate execution (in case of error in the primary alternate block). The system restores the checkpoint state if any fault occurs in the secondary alternate block and there is no further block to execute.

N-version programming

In the N-version programming technique multiple versions of the same program are implemented satisfying the same specification ([Tp00], [XSS], [Ram07]). A task is executed in all the versions in parallel and a voting mechanism determines the result based on majority voting or some other selection rules.

N-self Checking programming

N-self Checking programming was developed by Laprie et. al. where multiple versions of the same program are used as in N-version programming, but an additional self checking mechanism is added to those versions [Tp00], [XSS]. Separate acceptance tests can be added to each version [Tp00] or a single acceptance test can check the result of two versions at the same time [XSS] (if N is even). A selection module determines the final result based on outputs gathered from all the versions.

N-copy programming

This technique is a data diverse complement of N-version programming where the programs run in parallel in several computers or run sequentially in a single computer. This technique uses a decision mechanism and forward recovery to accomplish fault tolerance ([XSS], [Ram07]).

Retry block

The retry block is a major data diversity technique which uses acceptance tests and backward recovery to accomplish fault tolerance ([Tp00], [XSS], [Ram07]) . A watchdog timer is added to the system module and triggers a backup algorithm when the original algorithm fails to produce an acceptable result in a specified time. This triggering approach ends if the system finds an acceptable result or the time expires.

Some newer fault tolerance techniques are Adaptive N-version programming, Fuzzy voting, Reconfiguration and Rejuvenation, etc. [XSS]. These are either improvements of traditional fault tolerance techniques or based on some new concepts.

2.3 Transformational Approaches

The use of transformations in fault tolerance usually built on the view that a fault-tolerant system is composed of a basic, fault-intolerant system together with a set of special fault-tolerant components [PJ94, AK98]. As transformations are often understood as change of appearance to improve properties, adding fault tolerant components to a system to improve performance or robustness can be viewed as a system transformation.

2.3.1 Program transformation for fault modeling

In Program transformation, a fault-intolerant program is transformed to a fault tolerant program by adding necessary fault detection and correction components.

Early work on transformation for fault tolerance has focused on recovery mechanisms. For instance, in [PJ94, LJ92, LJ93], they used the method of check-pointing with forward and backward recovery. For the detection of faults, they assumed that this was accomplished by hardware mechanisms using a boolean flag which is raised indicating that a fault has been detected. After the detection, a recovery action is invoked in order to handle this fault. In this case, the first transformation for a given (fault-intolarent) program A is to model the detection of faults via checkpoints and this is done by adding additional variables and a timer to the program. The second transformation is to model correction mechanisms to obtain a fault-tolerant program. Recovery actions are invoked immediately to restore the program from a checkpoint state when the boolean flag has been raised indicating the presence of a fault. The associated recovery transformations R augments the original program P with the appropriate recovery actions.

Kulkarni and Arora [AK98, AK] have defined detector and corrector components in a certain way to separate the fault-tolerant mechanisms from the underlying program. A *detector* is an abstract component which signals that some predicate P holds on the system states. Similarly, a *corrector* component is implemented using a predicate Q which is to be imposed on the system state.

Cristian [Cri85] proposed the idea of *transient fault* which is modelled by adding a state transition of the form

$$true \ \rightarrow \ state := \langle random\ state \rangle$$

causing a transition to an arbitrary state. The method has been incorporated into a transformation approach by Liu and Joseph [LJ92, LJ93]. In this aproach the boolean flag `f` does not only signal a fault, but also disables all the normal actions of the program. This results in a fault transformation F as follows: a new "error" variable f is added to the set of variables of P and a special set of fault actions is added to the set of actions of the program. The guards of all original actions are augmented with $\neg f$ as an additional conjunct. So, applying the transformation $F(A)$ resembles the program running under the fault assumptions encoded in F. The new program $A' = F(A)$ is called the fault-affected version of A.

2.3.2 Specification transformation for fault modelling

Usually specification transformations for fault modelling will weaken the specification. In [Gär99], the author defines this as enlarging the behaviour property of faults. A behaviour over V (is a set of variables) is an infinite sequence of states, $\sigma = s_0, s_1, s_2, ...$; and, a property is a set of behaviours. In general, the task of specifying a specification transformation given a failure mode is not easy and it can affect the properties defined in the original program.

Shepers describes the notion of specification transformation and calls it a *fault hypothesis*. This is defined as a reflexive relation χ on process behaviours. Whenever a pair of behaviours σ_1 σ_2 is in χ, then σ_2 is the fault affected version of σ_1. The specification transformation $F(S)$ can then be easily defined as

$$F(S) = \{\sigma_2 \mid \sigma 1 \in S \wedge (\sigma 1, \sigma 2) \in \chi\}$$

It is a well-known fact that the original specification S of a (fault-intolerant) system may not be solvable under a sufficiently hostile fault assumption. Augmenting with fault-tolerant components may result in a system which satisfies a property close or equal to S. So, a specification transformation F should reflect this, and also be able to derive the fault tolerant specification $S' = F(S)$ which is solvable under the fault assumption encoded in F.

Gärtner argues that the set of behaviours of a system can be described as follows [Gär99]:

$$Prop(A) \;=\; Prop(I) \,\cap\, Prop(\delta) \,\cap\, L$$

where I is the initial predicate, δ is the transition relation and L is the system's liveness property. Given this formalization, fault assumptions can be added as actions, as a consequence this will add new state transitions obtaining an augmented transition relation δ'. In general, the fault assumption can be considered as another system property like the conjunction between G_s a safety property and G_l a liveness property which must hold. Thus, the properties of the transformed program $F(A)$ are calculated as: :

$$Prop(F(A)) \;=\; Prop(I) \,\cap\, Prop(\delta') \,\cap\, G_s \,\cap\, G_l \,\cap\, L$$

Lamport and Abadi argue that G_l can be incorporated into the system's liveness property L. Therefore, the fault assumption will be defined as a safety property. As a result, the specification transformation F can be simplified:

$$Prop(F(A)) \;=\; Prop(I) \,\cap\, Prop(\delta') \,\cap\, G_s \,\cap\, L$$

The fault tolerance specification S' may be equal to the original correctness specification S for the fault-free case or some acceptable degraded version of it. So different versions of S' can be defined. If $S = S'$ then it is called *masking fault tolerance* due to the fact that the effects of faults are transparent at the system interface. If the case is that S' is weaker than S, then it is called *fail-softness* or *gracefull degradation.*

Finally, transformation is a general notion of change. Several works apply transformations to the original program, interfering with the "normal" behaviour of the underlying program. Also, there are many works where the original program is transformed in an elegant way via specialized components for detection and correction of faults, and finally composed with all the original components of the system.

2.4 Process Algebra

Process algebra defines a system by defining a set of processes communicating with each other. In the literature we have seen that conventional process algebra defines a fault tolerant system where faulty actions are modelled like normal actions except that they introduce unexpected behaviour. Conventional process algebra does not have any sort of specialized operators for modelling faults and explicit fault recovery mechanisms are not discussed that much in the literature. Most of the work done in this sector design fail safe systems by encode fault tolerant mechanisms within the system design.

In [BFS00] [GLM05] the authors used replicas of the original system or the possibly failing system (PFS) as a fault tolerance technique. In their approach a system is composed of multiple identical components (called replicas) connected with a voter. The idea of adding multiple replicas is to get correct output from other components when a fault occurs in one or multiple components. The voter works as a detector and corrector component in the system and gets output from all the replicas including the faulty one. It is important to mention that Bernardeschi considered only those faults that occur within the system and did not include environmental effects generating faults.

In CCS/Meije process algebra a fault tolerant redundant system with n replicas is defined as: $(\xi_1 \parallel \ldots \parallel \xi_n \parallel M) \setminus A$. Here ξ_i denotes the i-th replica of the fault tolerant system; M = (m_i , 1 $\leq$ i $\leq$ k) denotes the set of extra components added by the fault tolerance technique and A = (a_j, 1 $\leq$ j $\leq$ s), $a_j \notin \mathcal{F}$ denotes the synchronization actions other than faulty actions (denoted by $\mathcal{F}$) between ξ_i and M.

As in the other process algebra approaches, the fault tolerance mechanism is encoded within the system design, the verification of the fault tolerant system is performed based on the assumptions of fault occurrences. Bernardeschi [BFG02] captured the additional possible ways of fault occurrences by a further process called the fault hypothesis (FH) where all the faulty actions of $\mathcal{F}$ are the actions of $FH(P)$. So the fault tolerant system design under a fault hypothesis (HFTSD) of a system P is defined as : $HFTSD(P) = FTSD(P) \,||\, FH(P) \setminus \mathcal{F} \setminus \mathit{other-actions}$, where $FTSD(P)$ denotes the fault tolerance system design of a system P and $\mathit{other-actions}$ denotes the set of actions other than the faulty actions.

In [GLM05] Gnesi used the CCS process algebra for modelling a fault tolerant system. In her approach the system works as an open system and can interact with the environment. It is important to mention that she did not impose any constraint on the *fault assumption model* and decided that it would be characterized by the environment. The environment $F_{\mathcal{F}}$ works as a fault injector and it can inject any type of faults to the system via the actions from $\mathcal{F}$. So the scenario she proposed is: $(P^{\#}_{\mathcal{F}} \parallel F_{\mathcal{F}}) \setminus \mathcal{F}$ where $P^{\#}_{\mathcal{F}}$ is the fault tolerant system. Here the occurrences of faults are always introduced by the environment $F_{\mathcal{F}}$ which is totally separate from the processes within the system and this sort of separation might be helpful for model checking. Unlike Bernardeschi, Gnesi used μ-calculus formulas for model checking of the system.

2.5 Specification Language

Specification languages are formal languages that describe a system at a much higher level of abstraction than programming languages and are mostly used during system analysis, requirement analysis and system design. Several formal languages and frameworks have been used to formalize and to prove properties of specific examples of fault-tolerant systems. These do not have any special construct for modelling fault-tolerant systems in terms of differences between correct, expected or ideal behaviour and incorrect, unexpected or abnormal behaviour. Hence, these features are encoded using ad-hoc mechanisms as part of the general design.

Some of the well-known case studies formalized are the Byzantine problem and a train system controller in TLA+ [LM94] and Even-B [Abr06] respectively.

TLA+, developed by Lamport and Merz, is a complete specification language based on TLA (Temporal Logic of Actions). This was developed for specifying and verifying concurrent and distributed systems enabling the expression of liveness and safety properties. The semantics of TLA is based on states and behaviours. TLA+ has no special constructs that can be used to specify fault tolerant systems explicitly. Rather it is a specification language based on temporal formulae where fault tolerance mechanisms are encoded within the specification. In [LM94] Lamport and Merz used TLA+ to specify and verify a well known fault tolerance problem called the Byzantine Generals problem where they encoded all the fault tolerance mechanisms in specifying the system. Boolean flags and voting mechanisms are used in this work for fault detection and recovery.

Another well-known example which is the object of active research ([HG93, AB08, Abr06, ABH+10]) in the fault-tolerance community is about train systems. These are systems that control the movement of trains through a network of rail segments. Fault-tolerance is a key aspect of these systems: a fault in the system may cause a train collision and the loss of human life.

In [ABH+10], the authors present the development of a train system controller using the specification language Event-B. The goal is to have trains safely circulating in a certain network; moreover, this case study also exhibits a very interesting case where the reliability of the final product is absolutely fundamental: several trains have to be able to safely cross the network under the complete automatic guidance of the software product. Moreover, the external environment is taken to account and the train system is carefully controlled. As a consequence, the formal modelling that the authors propose contains not only a model of the software but also a detailed model of its environment.

All these works on specification languages contain more material on fault prevention than on fault tolerance. This is essentially due to the problem at hand where faults have to be avoided at all means.

2.6 Deontic Logic

Deontic logic has been proposed as an adequate logic for the study of fault-tolerance, in particular since the notion of prescription and violation are naturally embedded in these logics. This is based on the observation that normal vs. abnormal behaviours can be treated as behaviours obeying and violating the rules of correct system conduct, respectively. This leads to a straightforward application of deontic operators (operators to express permission, obligation and prohibition) for separating normal from abnormal behaviours, and thus for expressing fault tolerant systems and their properties.

In [CJ96] an extension of standard deontic logic is proposed to reason about integrity constraints in databases and to create a clear distinction between hard (necessary) and soft (deontic) constraints. The soft integrity constraints admit violations and then the notion of recovery (from violation of static or state constraints) is characterized. The authors mentioned the static (or state) integrity constraints as, the restrictions on the permitted instances or on the states of a database schema. However, in this work, the notion of transition constraint (permitted changes of database states) is not considered, i.e., only norms regarding states are investigated.

In [LS04], a deontic interpreted logic is used to formalize a bit-transmission protocol. In this approach, the authors classifies the agent states into green and red, and a deontic machinery is developed using this classification. But the investigation of how to divide the transitions into red and green is left for further research.

In [KQM91] and [Kho98], Khosla and Maibaum propose a deontic logic to specify systems, although fault-tolerance is not dealt with in this work. The authors clearly mentioned that this logic can be used for the prescription and description of systems. They defined the description of a system as: what the system does, stated with pre and post conditions and, defined the prescription of a system as: what the system should do, stated using deontic predicates.

Khosla and Maibaum argue that the difference between prescription and description of a systems is important when specifying systems, and is useful for characterizing abnormal executions.

In [SKQ93], the authors proposed a deontic logic and used it to model a library system. Through this example, the authors tried to show how this logic can be used to specify temporal constraints and error recovery. In [WM93], the authors have stated that deontic systems are useful for reasoning about fault-tolerance because of the similarity between faults in computer systems and the situation in legal systems.

In more recent work, Castro and Maibaum in [CM09], [CM07b] investigated the utilization of deontic logics to specify concepts related to fault-tolerance. The main idea is to use axiomatic theories to specify computing systems at the design level. While theories describe components or modules, translations between them express the relationships between the different modules. The standard logical operators allow us to describe the basic behaviour of the system, while they use deontic predicates on actions to express prescriptions about the system's behaviour. These prescriptions of a system are highly related to the notion of violation: a violation occurs since the system exhibits a non-desired behaviour. They proposed a logic which is expressive enough for modelling interesting examples (e.g., the Diarrheic Philosophers, the Muller C-element, a Simple Train System, Processor Coolers, etc) and it differs from the other deontic systems proposed in the literature in terms of modelling fault tolerance mechanisms.

Following the above work in [CKAA11], the authors propose a logic especially tailored for describing fault tolerance properties based on the use of deontic operators, with an emphasis on expressing intended (temporal) properties of fault tolerant systems, rather than (axiomatically) prescribing system behaviour. This logic, which they refer to as dCTL, is composed of CTL and deontic operators for distinguishing "good" (normal) from "bad" (faulty) behaviors, as other deontic approaches, but the way in which temporal and deontic operators are combined makes the logic suitable for analysis via model checking. Their proposed dCTL logic is more expressive than CTL, which as they argue makes it useful for describing common properties of interest in the context of fault tolerant systems.

2.7 Summary

In this chapter we presented different fault tolerance approaches at the concrete and abstract levels. The concrete level fault tolerance approaches are mostly built on the ideas related with code replication, checkpointing and recovery and voting mechanisms. We have also discussed the transformational approach and process algebra based approach for modelling fault tolerance. The transformational approach covers both program transformation and specification transformation. We have presented a brief discussion on how low level fault intolerant programs or high level fault intolerant specifications are transformed to deal with faults. In process algebra based approaches we have discussed some work where fail safe systems are designed through the encoding of fault tolerance mechanisms. In this chapter we also discussed some specification languages like TLA+ and Event-B and described some of the well known fault tolerance problems modelled in these languages. Most of these approaches do not have any special construct to model faulty scenarios and recovery mechanisms. Rather they encode fault tolerance mechanisms in the system design level or implementation level. On the other hand, deontic logic has the notions of permission, obligation and violation and abstract fault tolerance mechanisms can be easily built in this logical approach. In this chapter we also presented some important work on fault tolerant systems using deontic logic approach.

Chapter 3

Deontic Logic

3.1 Deontic Logic

Deontic logic is a branch of formal logic that tries to capture the notion of reasoning on moral and ethical contexts. Philosophers and lawyers created this logic many decades ago and the main intention was to express the notions of *permission* and *obligation* in relation to the statement of a property. Naturally the related concept of *prohibition* and *violation* were added to this logic. As deontic logic is capable of using logical constructions to express normative statements, predicates such as "φ is permitted " or "φ is obligatory" can be expressed easily.

We have seen that several deontic logics have been proposed in the literature and the most cited and studied is the so-called Standard Deontic Logic (or SDL). It is built over propositional logic, and is a well known member of "normal modal logic" family. The semantics of SDL is defined in terms of Kripke Structures ([Mcn06], [Mcn]) and the axioms imply that the Kripke structures are *serial*, i.e., every state has a successor state.

Deontic logics can be classified as an $ought-to-be$ logics and $ought-to-do$ logics. The notion of deontic logic is applied to reason about different situations in the former, while in the latter one deontic operators are applied to actions, characterising them in terms of norms. An example of the former one can be: *it is allowed to have a "STOP" sign at the road intersection*, where the deontic operator is applied to define a situation that satisfies a norm. And in the statement: *a driver is obligated to stop at "STOP" sign*, a deontic operator is applied to an action. Though both versions of deontic logic are very important and effective in formalizing different situations, it seems that $ought - to - do$ logic is more effective and expressive for computer science, as in this formalism we can impose norms on actions and, consequently, on behaviours.

3.2 Deontic Action Logics

In [Mey88] J. J. Meyer introduces an $ought - to - do$ logic where he reduces deontic logic to a well known dynamic logic. The justification of this reduction is that dynamic logic is used from the beginning to prove the properties of computer programs and a computer program is a structured collection of actions of a certain kind. In this work Meyer defines $\mathrm{F}(\alpha) \leftrightarrow [\alpha]V$, i.e., *an action is forbidden if and only if execution of this action introduces a violation V.* And based on this definition he proposes other (derived) deontic operators. In his approach, from evaluating the truth value of a predicate, he was trying to capture the result that an action can generate, and, gave more importance than what happens during the action execution. That is why some philosophers named his approach as *goal-oriented* approach.

In [Bro03], Broersen rejected the inter-definability of deontic operators introduced by Meyer and describes other possible formulations of dynamic deontic logic where different violation propositions are used for defining permission, prohibition and obligation. In particular, Broersen uses a version of relative complement in this logic, arguing that this approach is more appropriate for computer science. In her approach, she considered the result of an action along with what happens during the execution of an action. She named the latter norms as *process norms* where she assumed that violation can occur in any step of a process. She considers that all the steps in an action execution must be allowed to make the whole action allowed.

In [KM87] Khosla and Maibaum propose a deontic action logic to specify a system. The key feature of their approach was to define a clear distinction between the description and prescription of a system. They argues against some other $goal - oriented$ approaches where the behaviour of a system was defined implicitly in terms of defining changes, but no attempt at relating system description and prescription was made. In their approach, they modelled systems over a collection of global states called scenarios and assumed that a system specification needs three types of information. They modelled the space of a system by *static information*. *Action description* defined the information regarding the changes of a state or scenario induced by the execution of actions and were modelled by traditional pre and post condition formulae. And *action prescription* defined when an action may or should occur and were characterized by permissible and obligatory actions. They distinguished the normative scenarios from the non-normative ones, where normative scenarios are part of the desirable behaviours and non-normative scenarios are part of the undesirable behaviours. Based on this distinction, they defined an action to be permissible (or forbidden) if its execution leads the system to a normative (respectively, non-normative) scenario. They use a constant n to denote normativeness and defined permission as $\mathrm{P}(A, \alpha) \leftrightarrow [A, \alpha]n$ (similar to Meyer's violation concept) and forbidden as $\neg \mathrm{P}(A, \alpha) \leftrightarrow [A, \alpha]\neg n$. This approach is useful as the constant n can denote those system states which are free of faults.

A Modal Action Logic (called MAL) was investigated by the FOREST project in [KQM91]. An interesting feature introduced in that work is that actions are interpreted as a set of “events”, i.e., n action represents those events that it participates in during its execution. In [Cas09], Castro followed this approach throughout his work in investigating deontic logic for fault tolerant systems and assumed that considering actions in terms of events is useful for concurrency. In the FOREST project a partial proof system is presented, but the properties of the logic were not investigated in detail.

In [FM92] Fiadeiro and Maibaum used deontic predicates to define normative executions. In this work, they introduced obligations on sequences of actions, which required that each action in the sequence had to be permitted when it was due to be executed.

3.3 Deontic Temporal Logic: A new approach

Most of the logics mentioned above use the idea of dynamic logic where a system is modelled using relational or functional interpretations of actions. In his new approach [Cas09] Castro used an algebraic structure of events to interpret actions which allows him to use the structure of the generated algebra of events to define labelled transition systems corresponding to deontic logic specifications. This approach also enabled him to prove meta logical properties of the logic such as soundness and completeness. Castro also added appropriate meta theorems in his logic which allows various sorts of analysis such as model checking on the system specification.

This logic has some new operators that are not common in the literature. It defines *relative* complement on actions, i.e., complement is based on the universe of other actions available in a given state, not based on the universal transition relation. A new operator *Done* is added to this logic where $Done(\alpha)$ means that *the last action executed was* α. The operators on events that are considered in this logic are those of boolean algebras and this logic has a sound, complete and compact axiomatization [CM07a].

In his approach to defining this logic, Castro used strong and weak versions of permission and tried to interrelate them in a new and novel manner. The weak version of permission ($\mathrm{P_w}$) says, of an action to which it is applied, that there is some way of doing the permitted action from the current state and has an exitential character. The normal or strong version of permission (P) says that every possible way of doing the action from the current state. In the literature there are several definitions of the obligation operator, but Castro introduced this operator by using both versions of permissions for its definition. He defined O as $\mathrm{O}(\alpha) \Longleftrightarrow \mathrm{P}(\alpha) \wedge \neg \mathrm{P_w}(\bar{\alpha})$, i.e., *an action is obliged if and only if every way of executing that action is allowed (permitted), and any way of executing any other action is disallowed.*

Reasoning about fault tolerant systems is a the main motivation for this logic. Castro structured his logic to define obligatory, permitted and forbidden actions along with fault tolerance properties and recovery actions. He followed the approach introduced by Khosla and Maibaum [KM87], where both description and prescription of a system are major concerns. How violation occurs in a system state and how recovery actions can take the system from a violation state to a normal state can be defined in this logic. A notion of time is introduced with this new deontic logic and the semantics of the temporal extension of this logic is given in terms of *traces*.

3.4 A Propositional Deontic Logic (DPL)

Castro introduced a *modal action logic* where he defined a propositional version of deontic logic (DPL) by introducing its syntax and semantics [Cas09]. He uses boolean operators to combine action terms and represented an action as a set of events i which the action participates [KQM91]. In his approach he considered a finite (but arbitrarily large) set of atomic actions and assumed that new actions can be defined using atomic actions and their combinators. He claims that a system always performs a finite number of actions in any execution step, which makes the axiomatic system of this logic compact and strongly complete. Now we will present the key components and axiomatic system of DPL defined in [Cas09]. We are presenting the logic here because in the next couple of chapters we will use this logic to model our example.

The following definitions, axiomatization and results are from [Cas09].

Definition 1 (Deontic vocabulary): "*A deontic vocabulary is a tuple* $\langle \Phi_0, \Delta_0 \rangle$*, Where:*

- Φ_0 *is a set of atomic propositions* $\{p_1, p_2, ...\}$
- Δ_0 *is a finite set of atomic actions* $\{\alpha_1, \alpha_2, ..., \alpha_n\}$ "

Definition 2 (Action terms Δ): "*Given a vocabulary* $\langle \Phi_0, \Delta_0 \rangle$*, we define the set of action terms (called* Δ*) as follows:*

- $\Delta_0 \subseteq \Delta$
- $\emptyset$, $\boldsymbol{U} \in \Delta$
- *If* $\alpha, \beta \in \Delta$, *then* $(\alpha \sqcup \beta) \in \Delta$, $(\alpha \sqcap \beta) \in \Delta$
- *If* $\alpha \in \Delta$, *then* $\overline{\alpha} \in \Delta$
- *No other expression belongs to* Δ"

Here $\emptyset$ denotes an impossible action and **U** denotes a non-deterministic choice of any action from the set of all actions.

Definition 3 (Fomulae Φ): "*Given the vocabulary* $\langle \Phi_0, \Delta_0 \rangle$, *we define the set of well-formed formulae* (Φ) *as follows:*

- $\Phi_0 \subseteq \Phi$
- $\top, \bot \in \Phi$
- *If* $\alpha, \beta \in \Delta$, *then* $\alpha =_{act} \beta \in \Phi$
- *If* $\varphi_1, \varphi_2 \in \Phi$, *then* $\varphi_1 \rightarrow \varphi_2 \in \Phi$
- *If* $\varphi \in \Phi$, *then* $\neg\varphi \in \Phi$
- *If* $\varphi \in \Phi$ *and* $\alpha \in \Delta$ *then* $\langle\alpha\rangle\varphi \in \Phi$ *and* $[\alpha]\varphi \in \Phi$
- *If* $\alpha \in \Delta$ *then* $P(\alpha) \in \Phi$, $P_w(\alpha) \in \Phi$ *and* $O(\alpha) \in \Phi$
- *No other expression belongs to* Φ"

Here $\langle\alpha\rangle\varphi$ intuitively means that α *can be executed in some ways such that* φ *holds after its execution.* On the other hand $[\alpha]\varphi$ means φ *holds after every possible execution of* α. $\mathrm{P_w}(\alpha)$ imposes weak permission on the action α and means α *can be executed at least in some way without leading to a fault.* $\mathrm{P}(\alpha)$ defines strong permission over α and means *every way of executing* α *is allowed.*

Definition 4 (Structures): "*An L-structure is a tuple* $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ *over the vocabulary* $\langle \Phi_0, \Delta_0 \rangle$, *where* $\mathcal{W}$ *is a set of worlds or states,* $\mathcal{R}$ *is an* $\mathcal{E}$ *labeled relation between states and* $\mathcal{E}$ *is non-empty set of events. The interpretation function* $\mathcal{I}$ *returns the set of worlds (subset of* $\mathcal{W}$*) for every proposition* $p \in \Phi_0$ *(the worlds in which p holds) and the set of events (subset of* $\mathcal{E}$*) for every action* $\alpha \in \Delta_0$ *(the set of events in which alpha participates or witnesses).*

And lastly $\mathcal{P}$ represents the relation between events in $\mathcal{E}$ and states in $\mathcal{W}$ and defines which event is permitted in which state."

The reader can inform themselves of the properties of the interpretation function $\mathcal{I}$ in [Cas09] . Castro defined the notation $w \overset{e}{\to} w'$ when $(w, w', e) \in \mathcal{R}$ and used it in defining the relation between world and formulae.

Definition 5 ($\models$): "*Given the vocabulary $\langle \Phi_0, \Delta_0 \rangle$ and a L-structure M $= \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$, the relation $\models^L$ between worlds and formulae is defined as follows :*

- $w, M \models^L p \overset{def}{\iff} w \in \mathcal{I}(p)$
- $w, M \models^L \alpha =_{act} \beta \overset{def}{\iff} \mathcal{I}(\alpha) = \mathcal{I}(\beta)$
- $w, M \models^L \neg\alpha \overset{def}{\iff}$ *not* $w, M \models \varphi$
- $w, M \models^L \varphi \to \psi \overset{def}{\iff} w, M \models^L \neg\varphi$ *or* $w, M \models^L \psi$ *or both*
- $w, M \models^L \langle\alpha\rangle\varphi \overset{def}{\iff}$ *there exists some* $w' \in \mathcal{W}$ *and* $e \in \mathcal{I}(\alpha)$ *such that* $w \overset{e}{\to} w'$ *and* $w', M \models^L \varphi$
- $w, M \models^L [\alpha]\varphi \overset{def}{\iff}$ *for all* $w' \in \mathcal{W}$ *and* $e \in \mathcal{I}(\alpha)$ *if* $w \overset{e}{\to} w'$ *then* $w', M \models^L \varphi$
- $w, M \models^L \mathrm{P}(\alpha) \overset{def}{\iff}$ *for all* $e \in \mathcal{I}(\alpha), \mathcal{P}(w, e)$ *holds*
- $w, M \models^L \mathrm{P_w}(\alpha) \overset{def}{\iff}$ *there exists some* $e \in \mathcal{I}(\alpha)$ *such that* $\mathcal{P}(w, e)$ *holds*
- $w, M \models^L \mathrm{O}(\alpha) \overset{def}{\iff}$ *for all* $e \in \mathcal{I}(\alpha), \mathcal{P}(w, e)$ *holds, and for every* $e' \in \mathcal{E} - \mathcal{I}(\alpha)$*, we have* $\neg\mathcal{P}(w, e')$*."*

M is a model of φ, i.e., $M \models^L \varphi$, iff for all worlds $w \in \mathcal{W}$ we have $w, M \models^L \varphi$ and $\models^L \varphi$, if for all models M, $M \models^L \varphi$.

Now we present a *Hilbert Style* deductive system for Castro's logic. A clear description of the underlying boolean algebra for the axiomatic system and explanations about the following axioms are given in [Cas09], [CM07b] and [CM07a].

Definition 6 (DPL Axioms): "*Given the vocabulary $\langle \Phi_0, \Delta_0 \rangle$, the axiomatic system for the Logic DPL is composed of following axioms:*

1	*The set of propositional tautologies.*
2	*A set of axioms for boolean algebras for action terms, including standard axioms for equality.*
3	*The following set of axioms:*
Ax1	$[\emptyset]\ \varphi$
Ax2	$\langle\alpha\rangle\varphi \ \wedge\ [\alpha]\psi \rightarrow \langle\alpha\rangle(\varphi \wedge \psi)$
Ax3	$[\alpha \sqcup \beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$
Ax4	$[\alpha]\varphi \ \rightarrow\ [\alpha \sqcap \beta]\varphi$
Ax5	$\mathrm{P}(\emptyset)$
Ax6	$\mathrm{P}(\alpha \sqcup \beta) \ \leftrightarrow\ \mathrm{P}(\alpha) \wedge \mathrm{P}(\beta)$
Ax7	$\mathrm{P}(\alpha) \vee \mathrm{P}(\beta) \rightarrow \mathrm{P}(\alpha \sqcap \beta)$
Ax8	$\neg\ \mathrm{P}_{\mathrm{w}}(\emptyset)$
Ax9	$\mathrm{P}_{\mathrm{w}}\ (\alpha\ \sqcup\ \beta)\ \leftrightarrow\ \mathrm{P}_{\mathrm{w}}(\alpha)\ \vee\ \mathrm{P}_{\mathrm{w}}(\beta)$
Ax10	$\mathrm{P}_{\mathrm{w}}\ (\alpha\ \sqcap\ \beta)\ \leftrightarrow\ \mathrm{P}_{\mathrm{w}}(\alpha)\ \wedge\ \mathrm{P}_{\mathrm{w}}(\beta)$
Ax11	$\mathrm{P}(\alpha) \wedge (\alpha \neq_{act} \emptyset)\ \rightarrow\ \mathrm{P}_{\mathrm{w}}(\alpha)$
Ax12	$(\bigwedge_{[\alpha]BA \wedge \alpha \sqsubseteq \alpha'} (\mathrm{P}_{\mathrm{w}}(\alpha) \vee (\alpha =_{act} \emptyset))) \rightarrow \mathrm{P}(\alpha')$
Ax13	$\mathrm{O}(\alpha) \leftrightarrow \mathrm{P}(\alpha) \wedge\neg\ \mathrm{P}_{\mathrm{w}}(\overline{\alpha})$
Ax14	$\langle\alpha\rangle\varphi \leftrightarrow \neg[\alpha]\neg\varphi$
Ax15	$(a_1 \sqcup ... \sqcup a_n) =_{act} \mathrm{U}$
Ax16	$(\alpha =_{act} \alpha')\ \leftrightarrow\ [\beta](\alpha =_{act} \alpha')$
Ax17	Subs: $\varphi[\alpha] \wedge (\alpha =_{act} \alpha')\ \rightarrow\ \varphi[\alpha/\alpha']$
	Duduction Rules:
MP	*if* $\vdash \varphi$ *and* $\varphi\ \rightarrow\ \psi$, *then* $\vdash \psi$
GN	*if* $\vdash \varphi$ *then* $\vdash [\alpha]\varphi$"

At this stage we are presenting the theorems that are used to prove soundness and completeness of this logic. A full soundness and completeness proof is given in [CM07b] and [Cas09] where theorems **T1** - **T7** are used to prove soundness and theorems **T8** - **T15** are used to prove completeness of the axiomatic system.

Theorem: "Theorems of DPL:

T1	$\mathrm{P}(\alpha) \wedge \alpha' \sqsubseteq \alpha \;\rightarrow\; \mathrm{P}(\alpha')$
T2	$\mathrm{P_w}(\alpha') \wedge \alpha' \sqsubseteq \alpha \;\rightarrow\; \mathrm{P_w}(\alpha)$
T3	$[\alpha]\varphi \wedge \alpha' \sqsubseteq \alpha \;\rightarrow\; [\alpha']\varphi$
T4	$[\alpha]\varphi \wedge [\alpha']\psi \;\rightarrow\; [\alpha \sqcup \alpha'](\varphi \vee \psi)$
T5	$[\alpha]\varphi \wedge [\alpha']\psi \;\rightarrow\; [\alpha \sqcap \alpha'](\varphi \wedge \psi)$
T6	$(\bigwedge_{[\alpha]BA\in\Delta/\Phi_{BA}\wedge\alpha\sqsubseteq\alpha'} (\mathrm{P_w}(\alpha) \vee (\alpha =_{act} \emptyset))) \rightarrow \mathrm{P}(\alpha')$
T7	$(\alpha =_{act} \alpha') \;\leftrightarrow\; [\beta](\alpha =_{act} \alpha')$
T8	$\alpha =_{act} \emptyset \;\leftrightarrow\; \mathrm{P}(\alpha) \wedge \neg\, \mathrm{P_w}(\alpha)$
T9	$\mathrm{O}(\alpha)\wedge\, \mathrm{O}(\overline{\alpha}) \;\leftrightarrow\; \mathrm{U} =_{act} \emptyset$
T10	$\mathrm{O}(\alpha)\wedge\, \mathrm{O}(\beta) \;\rightarrow\; \mathrm{O}(\alpha \sqcap \beta)$
T11	$\mathrm{P}(\mathrm{U}) \;\rightarrow\; \mathrm{P}(\alpha)$ for every action α
T12	$\mathrm{P_w}(\alpha) \;\rightarrow\; \mathrm{P_w}(\mathrm{U})$ for every action α
T13	$\mathrm{O}(\mathrm{U}) \;\leftrightarrow\; \mathrm{P}(\mathrm{U})$
T14	$\mathrm{O}(\emptyset) \;\leftrightarrow\; \neg\, \mathrm{P_w}(\mathrm{U})$
T15	$\mathrm{O}(\alpha) \;\rightarrow\; \mathrm{P}(\alpha)$"

3.4.1 DPL with Time

In his work Castro incorporated some notion of time into Deontic Propositional Logic and by adding a branching time temporal logic to DPL this new *Branching Time Logic* is very close to CTL. In this approach the formulae are evaluated with respect to fixed traces (representing executions). A new predicate $Done(\alpha)$ is introduced using this new semantics, which is true if the action α is executed in the immediate past (i.e., was used to reach the present state) [Cas09]. We now present the modified definitions for this new logic.

Definition 7 (Temporal Formulae): "*Given a DPL vocabulary* $\langle\Phi_0, \Delta_0\rangle$, *the set of temporal deontic formulae* Φ_T *is defined as follows* :

- $\Phi \subseteq \Phi_T$, *i.e., the formulae defined in definition 3 are temporal formulae.*
- *if* $\alpha \in \Delta$, *then* $\mathrm{Done}(\alpha) \in \Phi_T$

- *if* $\varphi, \psi \in \Phi_T$ *and* $\alpha \in \Delta$, *then* $(\varphi \rightarrow \psi) \in \Phi_T, [\alpha]\varphi \in \Phi_T$ *and* $\neg\varphi \in \Phi_T$
- *if* $\varphi, \psi \in \Phi_T$, *then* $\mathrm{AG}\varphi \in \Phi_T$, $\mathrm{AN}\varphi \in \Phi_T$, $\mathrm{A}(\varphi\, \mathcal{U}\, \psi) \in \Phi_T$ *and* $\mathrm{E}(\varphi\, \mathcal{U}\, \psi) \in \Phi_T$ "

Some explanation about some of these formulae will be useful. The formulae $\mathrm{AN}\varphi$ and $\mathrm{AG}\varphi$ say that: in all executions *φ is true at the next moment* and *φ is always true*, respectively. $\mathrm{A}(\varphi\, \mathcal{U}\, \psi)$ means *for every possible execution φ is true until ψ becomes true* and $\mathrm{E}(\varphi\, \mathcal{U}\, \psi)$ says *there exists some execution where φ is true until ψ becomes true.*

A set of initial states is introduced in the semantics of this new temporal extension. A initial state enables us to consider all the traces that start from this state. And traces are infinite sequences of states starting from a initial state. If there are a finite number of states in a trace , i.e., the execution terminates after a finite number of steps, then a self transition is added in the last state to make the trace infinite and this is called the completion of the trace. The labelling event for this transition does not belong to the model and the motivation for this transformation is to convey that if the system cannot do any action, the environment still keeps running [CM07b].

Definition 8 (Temporal model): "*Given a language $L = \langle \Phi_0, \Delta_0 \rangle$, $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P}, w \rangle$ is called a temporal structure where*:

- *$\langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle$ is a structure defined in definition 4*
- *$w \in \mathcal{W}$ is the initial state*"

Definition 9 (Traces): "*Given a model $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P}, w \rangle$, a trace is a labelled path $s_0 \overset{e_0}{\rightarrow} s_1 \overset{e_1}{\rightarrow} s_2 \overset{e_2}{\rightarrow} \ldots$, where for every i, $s_i \overset{e_i}{\rightarrow} s_{i+1} \in \mathcal{R}$ and $s_0 = w$. The set of all traces starting from state w is $\Sigma(w)$.*"

Some notations are necessary to understand the $\models^L_{DTL}$ relation between path, structure and formulae. The subpath π^i of an infinite path $\pi = s_0 \overset{e_0}{\rightarrow} s_1 \overset{e_1}{\rightarrow} s_2 \overset{e_2}{\rightarrow} \ldots$ is denoted by $\pi^i = s_i \overset{e_i}{\rightarrow} s_{i+1} \overset{e_{i+1}}{\rightarrow} \ldots$. That is π^i denotes the subpath that starts at position i of the infinite path π. The notation $\pi_i = s_i$ denotes the $i - th$ state in the path. Castro used the notation $\pi[i..j]$ for the finite subpath $s_i \overset{e_i}{\to} .. \overset{e_j}{\to} s_{j+1}$. And if we have a finite path $\pi' = s'_0 \overset{e'_0}{\to} .. \overset{e'_n}{\to} s'_{n+1}$ and an infinite path $\pi = s_0 \overset{e_0}{\to} s_1 \overset{e_1}{\to} s_2 \overset{e_2}{\to} ...$ where $s_i = s'_i$ and $e_i = e'_i$ for $0 \leq i \leq n$, then π' is called the initial subpath of π and denoted by $\pi' \preceq \pi$. A trace is called a maximal trace if it is not an initial subpath of any other trace and the set of maximum traces is denoted by $\Sigma^*(w)$ [Cas09].

Definition 10 ($\models^L_{DTL}$) : " *Given a model $M = \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P}, w \rangle$, a maximal trace $\pi = s_0 \overset{e_0}{\to} s_1 \overset{e_1}{\to} s_2 \overset{e_2}{\to} ... \in \Sigma^*(w)$, the relation $\models^L_{DTL}$ is defined as follows:*

- $\pi, i, M \models_{DTL} \varphi \overset{def}{\Longleftrightarrow} \pi_i, \langle \mathcal{W}, \mathcal{R}, \mathcal{E}, \mathcal{I}, \mathcal{P} \rangle \models \varphi,\ if\ \varphi \in \Phi$ does not contain any temporal predicates.
- $\pi, i, M \models_{DTL} \neg\varphi \overset{def}{\Longleftrightarrow}$ *not* $\pi, i, M \models_{DTL} \varphi$
- $\pi, i, M \models_{DTL} \varphi \to \psi \overset{def}{\Longleftrightarrow}$ *either not* $\pi, i, M \models_{DTL} \varphi$ *or* $\pi, i, M \models_{DTL} \psi$
- $\pi, i, M \models_{DTL} \text{Done}(\alpha) \overset{def}{\Longleftrightarrow} i > 0$ *and* $e_{i-1} \in \mathcal{I}(\alpha)$
- $\pi, i, M \models_{DTL} [\alpha]\varphi \overset{def}{\Longleftrightarrow} \forall \pi' = s'_0 \overset{e'_0}{\to} s'_1 \overset{e'_1}{\to} ... \in \Sigma^*(w)$ such that $\pi[0, i] \preceq \pi',\ if\ e'_i \in \mathcal{I}(\alpha)\ then\ \pi', i + 1, M \models_{DTL} \varphi$
- $\pi, i, M \models_{DTL} \text{AN}\varphi \overset{def}{\Longleftrightarrow} \forall \pi' = s'_0 \overset{e'_0}{\to} s'_1 \overset{e'_1}{\to} ... \in \Sigma^*(w)$ such that $\pi[0, i] \preceq \pi',$ *we have* $\pi', i + 1, M \models_{DTL} \varphi$
- $\pi, i, M \models_{DTL} \text{AG}\varphi \overset{def}{\Longleftrightarrow} \forall \pi' = s'_0 \overset{e'_0}{\to} s'_1 \overset{e'_1}{\to} ... \in \Sigma^*(w)$ such that $\pi[0, i] \preceq \pi',$ *we have* $\forall j \geq i : \pi', j, M \models_{DTL} \varphi$
- $\pi, i, M \models_{DTL} \text{A}(\varphi\, \mathcal{U}\, \psi) \overset{def}{\Longleftrightarrow} \forall \pi' = s'_0 \overset{e'_0}{\to} s'_1 \overset{e'_1}{\to} ... \in \Sigma^*(w)$ such that $\pi[0, i] \preceq \pi',$ *we have* $\exists j \geq i : \pi', j, M \models_{DTL} \psi$ *and* $\forall i \leq k < j : \pi', k, M \models_{DTL} \varphi$
- $\pi, i, M \models_{DTL} \text{E}(\varphi\, \mathcal{U}\, \psi) \overset{def}{\Longleftrightarrow} \exists \pi' = s'_0 \overset{e'_0}{\to} s'_1 \overset{e'_1}{\to} ... \in \Sigma^*(w)$ such that $\pi[0, i] \preceq \pi',$ *we have* $\exists j \geq i : \pi', j, M \models_{DTL} \psi$ *and* $\forall i \leq k < j : \pi', k, M \models_{DTL} \varphi$"

$M \models_{DTL} \varphi$ if $\pi, i, M \models_{DTL} \varphi$ for all paths π and instants i and $\models_{DTL} \varphi$ if φ holds for all models M.

Definition 11 (DTL Axioms): "*Given a vocabulary $\langle \Phi_0, \Delta_0 \rangle$, the axiomatic system is composed of the following axioms*:

	All the axioms in definition 6.
TempAx1	$\langle \mathrm{U} \rangle \top \rightarrow (\mathrm{AN}\varphi \leftrightarrow [\mathrm{U}]\varphi)$
TempAx2	$[\mathrm{U}]\bot \rightarrow (\mathrm{AN}\varphi \leftrightarrow \varphi)$
TempAx3	$\mathrm{AG}\varphi \leftrightarrow \neg \mathrm{E}(\top \, \mathcal{U} \, \neg\varphi)$
TempAx4	$\mathrm{E}(\varphi \, \mathcal{U} \, \psi) \leftrightarrow \psi \vee (\varphi \wedge \mathrm{ENE}\, (\varphi \, \mathcal{U} \, \psi))$
TempAx5	$\mathrm{A}(\varphi \, \mathcal{U} \, \psi) \leftrightarrow \psi \vee (\varphi \wedge \mathrm{ANA}\, (\varphi \, \mathcal{U} \, \psi))$
TempAx6	$[\alpha]\mathrm{Done}(\alpha)$
TempAx7	$[\overline{\alpha}]\neg\mathrm{Done}(\alpha)$
TempAx8	$\neg\mathrm{Done}(\emptyset)$
TempAx9	$\neg\, \mathrm{Done}(\mathrm{U}) \rightarrow \neg\, \mathrm{Done}(\alpha)$
	Deduction rules including the rules of definition 6:
TempRule1	$if \vdash \neg\mathrm{Done}(\mathrm{U}) \rightarrow \varphi \; and \vdash \varphi \rightarrow \mathrm{AN}\varphi, \; then \vdash \varphi$
TempRule2	$if \vdash \varphi, \; then \vdash \mathrm{AG}\, \varphi$
TempRule3	$if \vdash \varphi \rightarrow (\neg\psi \wedge \mathrm{EN}\varphi), \; then \vdash \varphi \rightarrow \neg\mathrm{A}(\vartheta \, \mathcal{U} \, \psi)$
TempRule4	$if \vdash \varphi \rightarrow (\neg\psi \wedge \mathrm{AN}(\varphi \vee \neg\mathrm{E}(\vartheta \, \mathcal{U} \, \psi))), \; then \vdash \varphi \rightarrow \neg\mathrm{E}(\vartheta \, \mathcal{U} \, \psi)$
TempRule5	$if \vdash \neg\mathrm{Done}(\mathrm{U}) \rightarrow \mathrm{AG}\, \varphi, \; then \vdash \varphi$"

Explanations about these axioms and rules are given in [Cas09] and [CM07b]. We know that the operator $Done(-)$ talks about immediate past and $Done(\alpha)$ says that the last action executed is α. We exhibit some important properties of the operator Done(-).

"T16	$\mathrm{Done}(\alpha) \wedge \alpha \sqsubseteq \alpha' \rightarrow \mathrm{Done}(\alpha')$
T17	$\mathrm{Done}(\alpha \sqcup \beta) \rightarrow \mathrm{Done}(\alpha) \vee \mathrm{Done}(\beta)$
T18	$\mathrm{Done}(\alpha \sqcap \beta) \leftrightarrow \mathrm{Done}(\alpha) \wedge \mathrm{Done}(\beta)$
T19	$\mathrm{Done}(\alpha \sqcup \beta) \wedge \mathrm{Done}(\overline{\alpha}) \rightarrow \mathrm{Done}(\beta)$
T20	$[\alpha]\varphi \wedge [\beta]\mathrm{Done}(\alpha) \rightarrow [\beta]\varphi$"

Contrary-to-duty structures are a set of predicates where, a violation of primary obligation introduces a secondary obligation. These kinds of formulae are sometimes problematic for this logic as we can deduce falsehood from

Contrary-to-duty predicates, which is paradoxical in some sense. To overcome this problem Castro introduced a convenient collection of different permissions in this logic: $P^1, ..., P^n$ and $P^1_w, ..., P^n_w$.

3.5 First-Order Deontic action Logic

The propositional deontic logic approach may not be suitable for modelling complex systems; for example, complex data structures may be necessary where quantification might be useful. For this reason Castro defined a first order version of deontic logic. This extended version has standard quantifiers of first order logic and algebraic operators for actions similar to those of the propositional deontic logic . In this section we will present a brief discussion on this logic extension. Later on we will use this logic to model our example and prove different properties about it.

3.5.1 Syntax and Semantics

Castro defined the language of the first order extension of his deontic logic based on finite set of action symbols, a set of flexible function symbols (whose extension may change from state to state, like that of a database relation) and set of rigid function symbols (which have the same meaning in every state). He assumed an enumerable set of variables denoted by X.

The following definitions, axiomatization and results are from [CM10].

Definition 12 (Language): " *A language or vocabulary is a tuple* $\langle \Delta_0, F_0, R_0 \rangle$ *where* Δ_0 *is a finite set of action symbols,* F_0 *is an enumerable set of flexible functin symbols and* R_0 *is an enumerable set of rigid function symbols. All action symbols, flexible and rigid function symbols have an associated arity.*"

Definition 13 (Terms): " *Given a language* $L = \langle \Delta_0, F_0, R_0 \rangle$*, the set of terms* $T_L(X)$ *over the language* L *is as follows:*

- *If* $x \in X$ *then* $x \in T_L(X)$

- *If $f \in F_0$ or $f \in R_0$ with arity n and $t_1, ..., t_n \in T_L(X)$ then $f(t_1, ..., t_n) \in T_L(X)$*
- *No other element belongs to $T_L(X)$*"

Definition 14 (Formulae): " *Given a language $L = \langle \Delta_0, F_0, R_0 \rangle$, the set of formulae over this language is defined as follows:*

- *If $t_1, t_2 \in T_L(X)$, then $t_1 = t_2 \in \Phi$*
- *If $\alpha_1, \alpha_2 \in \Delta(X)$, then $\alpha_1 =_{act} \alpha_2 \in \Phi$*
- *If $\varphi, \psi \in \Phi$, then $\neg\varphi$ and $\varphi \longrightarrow \psi \in \Phi$*
- *If $\varphi \in \Phi$ and $\alpha \in \Delta(X)$, then $[\alpha]\varphi \in \Phi, P(\alpha) \in \Phi$ and $P_w(\alpha) \in \Phi$*
- *If $\varphi \in \Phi$ and $x \in X$, then $(\forall x : \varphi) \in \Phi$*"

In this logic Castro used two equalities, one for standard terms ($=$) and another for actions ($=_{act}$). The notion of free variables, bound variables and sentence are the same as in first order logic.

Definition 15 (Action Terms): " *Given a language $L = \langle \Delta_0, F_0, R_0 \rangle$, the set of action terms ($\Delta(X)$) over this language is defined as follows:*

- $\emptyset, \mathbf{U} \in \Delta(X)$
- *If $a \in \Delta_0$ with arity n and $t_1, ..., t_n \in T_L(X)$, then $a(t_1, ..., t_n) \in \Delta(X)$*
- *If $\alpha, \beta \in \Delta(X)$, then $\alpha \sqcup \beta, \alpha \sqcap \beta$ and $\overline{\alpha} \in \Delta(X)$*
- *If $\alpha \in \Delta(X), x \in X$, then $(\bigsqcup_{x} \alpha) \in \Delta(X)$*"

The advantage of using a first order version of deontic logic over the propositional version is that it allows to define actions with parameters, which is more effective for modeling computer programs. A quantifier over a variable in an action term allow for a non-deterministic choice among the values of that parameter of the action term can take. An action term $\bigsqcup_{x_i} \alpha(x_1, ..., x_n)$ is considered as action α with $x_1, ..., x_n$ as parameters where we can nondeterministically choose the value of the i-th parameter. A complete description of the action terms is given in [CM10].

Definition 16 (Structure): "*Given a language $L = \langle \Delta_0, F_0, R_0 \rangle$, an L-Structure is a tuple $\langle \mathcal{W}, \mathcal{R}, \mathcal{P}, \mathcal{I}, \mathcal{E}, \mathcal{D} \rangle$ where $\mathcal{D}$ is a domain of elements, $\mathcal{W}$ is set of worlds, $\mathcal{E}$ is a non-empty set of events, $\mathcal{R}$ is the $\mathcal{E}$-labelled relation between states, $\mathcal{I}$ is an interpretation function that evaluates any term in a given state and $\mathcal{P}$ is the relation between worlds $\mathcal{W}$ and events $\mathcal{E}$ and defines which events are permitted in each state.*".

The interpretation function $\mathcal{I}$ has to satisfy several properties. A clear description of these properties and how this interpretation function ($\mathcal{I}$) works on flexible function symbols, rigid function symbols and actions symbols for any state ($\mathcal{W}$) is given in [CM10].

Now we present the axiomatic system for the logic described above. This system also includes the axioms and deduction rules of propositional deontic logic.

	"Axioms:
-	Axioms of Propositional Deontic Logic.
A1	$\forall x : \alpha \sqsubseteq \bigsqcup_x \alpha$, for all actions α
A2	$(\forall x : \alpha \sqsubseteq \beta) \rightarrow \bigsqcup_x \alpha \sqsubseteq \beta$, for all actions α and β, where x is not free in β
A3	$(\forall x : P(\alpha)) \rightarrow P(\bigsqcup_x \alpha)$
A4	$(\exists x : P_w(\alpha)) \rightarrow P_w(\bigsqcup_x \alpha)$
A5	$(\forall x : [\alpha]\varphi) \rightarrow [\bigsqcup_x \alpha]\varphi$
A6	$\forall \overrightarrow{x}, \overrightarrow{y} : \overrightarrow{x} \neq \overrightarrow{y} \rightarrow a(\overrightarrow{x}) \sqcap a(\overrightarrow{y}) =_{act} \emptyset$, for all $a \in \Delta_0$
FOLSub.	$(\forall x : \varphi) \rightarrow \varphi[x \backslash t]$. where t is free for x in φ
Barcan.	$(\forall x : [\alpha]\varphi) \rightarrow [\alpha](\forall x : \varphi)$
	Deduction Rues:
-	MP. and GEN. Rules of Propositional Deontic Logic.
FOL-GEN	If $\vdash \varphi$, then $\vdash \forall x : \varphi$"

The notation $\overrightarrow{x}$ denotes tuple of elements $(x_1, ..., x_n) \in X^n$ and $a(\overrightarrow{x})$ defines the action term a with these elements as parameters.

3.6 Summary

In this chapter we have discussed deontic logic and described some important extensions of this logic to deal with fault tolerant systems. The main intention in this chapter is to discuss Castro's extension of propositional and first order versions of this logic. Here we have presented the logic structure, axioms and theorems of these logics as we will use them in the later part of our thesis. In some sections we avoid creating lengthy discussions and refer the reader to [Cas09] and [CM10] to learn more about these logics.

Chapter 4

Problem Description

4.1 Scenarios

Our example focuses on some scenarios of a moving vehicle. The vehicle that we are concerned with is designated the *host vehicle* and all other vehicles are *opponent vehicles*. Our main focus is to capture the scenarios in which the host vehicle participates i.e., how the vehicle changes its speed and distance to opponent vehicles, the violations that arise due to speed and distance limitations and the recovery mechanisms. Using these scenarios, in chapter 5 we present our first model in propositional deontic logic, which is relatively simple then the model in first order deontic logic presented in chapter 6. In the first model we consider relatively simple scenarios and assume that the host vehicle is only responsible for all the situations it encounters and that the opponent vehicles are moving at a constant speed. On the other hand, in the second model we will consider some complex cases where the host vehicle's situation also depends on what the opponent vehicle is doing. For example, it might happen that the host vehicle faces a violation and the opponent vehicle is actually responsible for it.

In our scenarios, the path or lane in which the host vehicle moves is called the ego-path and there can be one or more adjacent lanes or no such lane at all. Host vehicles can change speed or change lane based on different situations.

The changing speed operation of the host vehicle also changes the distance with respect to the opponent vehicle(s) and an opponent vehicle is the one immediately in front of and behind the host vehicle. The two main properties with which we are concerned are *safe speed* and *safe distance* and the objective of our model is to ensure that the host vehicle always satisfies these criteria.

We now list some sample scenarios from **SASPENCE** [FTO05] sub-project and **NHTSA** [JJFN07] research works related with safe speed and safe distance. To formulate these scenarios, we have also added our general ideas about vehicle motion.

Scenario	Description	Action
S1	-Lane: Single/Multiple -Vehicle Speed: Safe -Obstacle: NO	Increase or decrease speed or maintain current speed.
S2	-Lane: Single/Multiple -Vehicle Speed: Exceeded max limit -Obstacle: NO	Host vehicle reduces speed until it reaches a safe speed.
S3	-Lane: Single/Multiple -Vehicle Speed: Safe -Obstacle: Safe opponent vehicle on Ego path	Host vehicle can increase or decrease speed or maintain current speed.
S4	-Lane: Multiple -Vehicle Speed: Safe -Obstacle: Opponent vehicle on the adjacent lane	Increase or decrease speed or maintain current speed.

S5	-Lane: Single -Vehicle Speed: Anything -Obstacle: Opponent vehicle on ego path. -Exceeded min safety distance: YES -Relative speed with front opponent vehicle: Case 1: Positive Case 2: Negative	Case 1: Reduce speed until the host vehicle attains safe distance from opponent vehicle. Case 2: Reduce OR maintain current speed until the host vehicle keeps safe distance from opponent vehicle.
S6	-Lane: Multiple -Vehicle Speed: Anything -Obstacle: Opponent vehicle on ego path. -Exceeded min safety distance: YES -Relative speed with front opponent vehicle: Positive -Adjacent lane: Case 1. Free or has no obstacle Case 2. Has obstacle in min safety distance	Case 1: Reduce speed until the host vehicle keeps safe distance from opponent vehicle OR change lane. Case 2: Reduce speed until the host vehicle keeps safe distance from opponent vehicle
S7	-Lane: Multiple -Vehicle Speed: Anything -Obstacle: Opponent vehicle on ego path. -Exceeded min safety distance: YES -Relative speed with front opponent vehicle: Negative -Adjacent lane: Case 1. Free or has no obstacle Case 2. Has obstacle in min safety distance	Case 1: Reduce OR maintain current speed until the host vehicle keeps safe distance from opponent vehicle OR change lane. Case 2: Reduce OR maintain current speed until the host vehicle keeps safe distance from opponent vehicle

S8	-Lane: Single/Multiple -Vehicle Speed: Anything -Obstacle: Opponent vehicle on ego path. -Exceeded min collision distance: YES	Host vehicle has to stop in order to avoid collision

Table 4.1.1: Scenarios for safe speed and safe distance

A short description of the scenarios may be useful. Scenario **S1** says that if a vehicle is moving at a safe speed and if it does not have any obstacle, then it can increase or decrease speed or maintain its current speed. **S2:** if the host vehicle exceeds the maximum speed limit, then it must decrease its speed until it attains a safe speed. Scenarios **S3** and **S4** say that if the host vehicle has a safe opponent vehicle on the ego path or adjacent lane, then it can execute any action. **S5** says that if a vehicle is moving in a single lane road and if it exceeds the minimum safety distance in relation to the obstacle, then it has to reduce speed if it has positive relative speed with respect to the obstacle and it can either reduce speed or maintain current speed if it has negative relative speed. **S6** and **S7** are like **S5** except that they deal with the cases for a multi-lane road and the host vehicle has the option to change lanes if it is free. And other actions like reducing speed or maintaining current speed remain the same.

Note: In our first model in propositional deontic logic we have considered the possibility of having multiple lanes. But in the second model in the first order deontic approach, we only focus on a single lane in order to capture more in depth scenarios (violations and recovery mechanisms) in a lane. And in the scenarios above we only focused on the recovery actions of the host vehicle, but it should be noted that the opponent vehicle's actions might recover the host vehicle from a violation and we will see some of these cases in our second model.

4.2 Problem Description

In our formalization we are trying to model a system where a vehicle moves on a straight road and the scenarios are related with its speed and distance in relation to other vehicle. Our assumption is that our focused host vehicle can do five things on the road. They are increase speed, decrease speed, maintain current speed (or continue speed), change lane and stop. In a normal situation, the host vehicle can perform any of these five operations but there are restrictions of certain kinds on executing an operation if the host vehicle violates any rule. For example, we know that every road has a fixed maximum speed limit and the vehicle must proceed under the limit. So, if the vehicle violates the speed limit, then it must reduce its speed to be within the limit. In this situation, the vehicle cannot increase its speed or continue its current speed. Some important terminology for our example is as follows.

Maximum speed: This is the maximum allowed speed for the road. Every vehicle must proceed within this speed limit and exceeding this limit is considered a violation.

Minimum speed: This is the minimum allowed speed for a road. Vehicles must proceed above this speed limit and having lower speed than this limit is considered a violation.

Minimum safety distance: Every vehicle must maintain a fixed safety distance with respect to its front and rear opponent vehicles. If a host vehicle has less distance than the minimum safety distance with respect to the front or rear opponent vehicle, then the host vehicle is in a violation situation.

Minimum collision distance: This is the least distance that every vehicle must maintain with respect to an opponent vehicle and we are assuming that the vehicle is about to collide with the opponent vehicle if it exceeds (gets closer than) this distance.

Increase Speed: Increasing speed is an operation which increases the vehicle's speed from its current speed. Our host vehicle is always allowed to increase speed unless it has exceeded the minimum safety distance or minimum collision distance with respect to the front opponent vehicle or has exceeded the maximum speed limit.

Decrease Speed: This is the operation which decreases the vehicle's speed from its current speed. The host vehicle can always decrease its speed if it has sufficient distance from the rear opponent vehicle and its current speed is no less than the minimum speed limit.

Continue Speed: This operation maintains the vehicle's current speed. The host vehicle can always maintain its current speed unless it has exceeded the maximum or minimum speed limit or exceeded any distance limitations. In some cases, the host vehicle can maintain its current speed even though it has exceeded distance limitations, but it must ensure that maintaining the current speed will increase the distance from the opponent vehicle.

Change Lane: This is the operation which changes the lane of the host vehicle; the host vehicle moves to an adjacent lane after executing this operation.

Stop: This is the operation which stops the vehicle. This operation will stop the host vehicle in any situation.

Violations: We have considered several violations here. Our first violation is related with excessive speed where the host vehicle exceeds the maximum speed limit. The host vehicle must decrease its speed to recover from this violation.

The second violation is related with the distance from the front opponent vehicle where the host vehicle exceeds the minimum safety distance. For this violation we are considering two cases. For the first case, if the host vehicle has greater speed than the front opponent vehicle, then it must decrease its speed so that it can achieve a safe distance. The host vehicle also can change lane to recover from this violation. If the host vehicle increases speed or continues its current speed, then it cannot increase the distance between them, so it cannot recover from the violation. The situation is also the same if the host vehicle has the same speed as the opponent vehicle. On the other hand, if the host vehicle has lower speed than the front opponent one, then it can decrease speed or continue its current speed and any of these operations will create more distance between the vehicles.

The third violation is the minimum collision distance violation which is the worst case scenario. We are assuming that the host vehicle must stop to avoid a collision when it is in this violation.

Note: In our model we have only considered the violations related with the front opponent vehicle and the maximum speed limitation. In a similar way, we can also model the speed violation related with the minimum speed limitation, and the distance violation related with the rear opponent vehicle or a combination of front and back opponents.

4.2.1 Requirements

In our modelling we will try to fullfill some requirements to ensure the safe speed and safe distance conditions. Here we list these requirements:

- If the host vehicle executes the continue speed action, then the speed after action execution and before action execution are the same.
- If the host vehicle executes the increase speed action, then it has greater speed after action execution than before action execution.
- If the host vehicle executes the decrease speed action, then it has less speed after action execution than before action execution.
- If the host vehicle executes the change lane action, then after action execution it reaches an adjacent lane.
- Executing the stop action stops the host vehicle in any situation.

- The host vehicle cannot execute more than one action at a time, all actions (except changing lane) are mutually exclusive.
- A stopped host vehicle cannot execute any action.
- If the host vehicle is stopped successfully, then it will not have collision with front opponent vehicle.
- If the host vehicle is not stopped, and is not in a violation situation, then it can execute any action from any state.
- If the opponent vehicle and the host vehicle have the same speed currently, then parallel execution of the continue speed action maintains the same distance.
- If the host vehicle currently has a safe speed, then, after executing any action, it has a safe speed or it violates the speed limitation.
- From a safe speed, the host vehicle violates the maximum speed limitation by increasing speed and violates the minimum speed limitation by decreasing speed.
- The host vehicle is not allowed to have a speed more than the *maximum speed* limitation and less than the *minimum speed* limitation. If it exceeds either of these limitations then it is in a speed violation.
- The host vehicle must maintain at least a minimum safety distance with respect to its front and rear opponent vehicles. If the actual distance is less than this distance limitation, then it is in violation.
- If the host vehicle violates the maximum speed limitation, then it is obliged to decrease its speed until it reaches a safe speed.
- If host vehicle violates the minimum speed limitation, then it is obliged to increase speed until it reaches a safe speed.
- If the host vehicle violates the minimum safety distance limitation with respect to the front vehicle and the host vehicle's speed is more than

the front vehicle's speed, then it must decrease speed until it achieves a minimum safety distance. The host vehicle might recover from this violation if the front vehicle executes an increase speed action.

- If the host vehicle violates the minimum safety distance limitation with respect to the front vehicle and the host vehicle's speed is less than the front vehicle's speed then it must maintain its current speed or decrease its speed until it achieves the minimum safety distance. The host vehicle might recover from this violation if the front vehicle executes the increase speed action.

- If the host vehicle violates the minimum safety distance limitation with respect to the rear vehicle and the host vehicle's speed is less than the rear vehicle's speed, then it must increase speed until it achieves a minimum safety distance. The host vehicle might recover from this violation if the rear vehicle executes the decrease speed action.

- If the host vehicle violates the minimum safety distance limitation with respect to the rear vehicle and the host vehicle's speed is more than the rear vehicle's speed, then it must maintain current speed or increase speed until it achieves a minimum safety distance. The host vehicle might recover from this violation if the rear vehicle executes the decrease speed action.

- If the host vehicle gets too close to the front vehicle and if a collision is imminent, then it is in the minimum collision distance violation, and is obliged to stop.

- If the host vehicle is in multiple violations, for example, speed and distance violations at the same time, then it must execute required actions until it recovers from both violations.

- If the host vehicle cannot recover from a violation after executing a recovery action, then it must execute that recovery action again or a different recovery action in order to recover from the violation.

- A cut-in vehicle does not improve the violation situation of a host vehicle. If the host vehicle is currently in a minimum safety distance violation with respect to its front opponent vehicle, then after a cut-in vehicle joins in front of the host vehicle, it either remains in the minimum safety distance violation or goes to the minimum collision distance violation.

- A cut-out vehicle may improve the violation situation and recover the host vehicle from distance violations. For example, if the host vehicle is currently in a minimum safety distance violation with respect to its front opponent vehicle, and if that front opponent vehicle changes the ego path and goes to an adjacent lane, then new the front opponent vehicle of the host vehicle will be the one which was the front vehicle of that cut-out vehicle before the action execution, and the actual distance between the host vehicle and the new front opponent vehicle might be greater than the minimum safety distance.

Chapter 5

Model in Propositional Deontic Logic

We have presented a brief discussion of Propositional Deontic Logic (PDL) in Chapter 3. Now in this chapter we will use this logic to build the first model of our example. We have presented sample scenarios and features of the problem in chapter 4. Now we will try to apply the PDL approach to model the safe speed and safe distance concept for a host vehicle.

The model in Deontic Propositional Logic is simpler, compared with the model in Deontic First Order Logic, and will help us to familiarize ourselves with the problem, as well as with the Logic. We also want to see to what extent we can use this more restricted logic approach to model a real world case study related with fault tolerance mechanisms. In the literature [Cas09], we have seen several fault tolerance modelling examples using the Propositional approach, but none of them are based on a real case study, focusing on a broader scenario. This model will also help us to understand the deficiencies of this logic that we can overcome through the First Order version.

In the PDL approach, we cannot define actions or propositions with parameters. So, in this model we are focusing on very basic actions and propositions. We have actions named $v_h.incspeed$ and $v_h.decspeed$ which increases and decreases speed of host vehicle, respectively, by some set amount that is not specified. Here we are only concerned with incrementing and decrementing speed after executing this action; we are not capturing the time to execute these actions or the value of the speed achieved. For example, decreasing the speed by $10Kmh^{-1}$ cannot be captured in our initial model.

5.1 Assumptions

We have made some of assumptions for this basic example.

ASM1 In this formalization, we are considering only the motion of the host vehicle. We are assuming that only the speed of the host vehicle increases or decreases and the opponent vehicle is moving at a constant speed. That is, only the host vehicle is responsible for all the violations that it creates with respect to the (front or rear) opponent vehicle.

ASM2 In this model, we are assuming that normal actions regarding increasing and decreasing speed will take the host vehicle from a violation state to a normal state.

ASM3 We are assuming that the host vehicle and the opponent vehicle are moving in the same direction on a straight road; any curvature of the road is not considered here. Multiple lanes are possible.

5.2 Problem formalization

5.2.1 Propositions

The following are the basic propositions used to characterise the scenarios we will encounter. The current state of the vehicles are represented in the model using these propositions.

- $v_h.speed_{cont}$: Host vehicle maintains its speed.
- $v_h.speed_{inc}$: Host vehicle's speed is increased.

- $v_h.speed_{dec}$: Host vehicle's speed is decreased.
- $v_h.speed_{safe}$: Host vehicle's speed is safe.
- $v_h.stopped$: Host vehicle is stopped.
- $v_h.reachedaddlane$: Host vehicle reaches an adjacent lane after executing lane changing action.
- $road.clear_{front}$: No obstacle in front of the host vehicle.
- $road.clear_{rear}$: No obstacle in rear of the host vehicle.
- $v_h.exceed_{maxspeed}$: Host vehicle exceeded maximum speed.
- $v_h.exceed_{minsafedistfront}$: Host vehicle exceeded minimum safety distance with respect to the front vehicle.
- $v_h.exceed_{minsafedistrear}$: Host vehicle's distance with rear opponent vehicle is less than minimum safety distance.
- $v_h.onsamelane(v_o)$: Host and opponent vehicle are on the same lane.
- $v_h.hasadjlane$: Host vehicle has adjacent lane.
- $v_h.onadjlane(v_o)$: Opponent vehicle is in the adjacent lane of the host vehicle.
- $v_h.posrelspeed(v_o)$: Host vehicle has positive relative speed with opponent vehicle.
- $v_h.negrelspeed(v_o)$: Host vehicle has negative relative speed with opponent vehicle.

5.2.2 Actions

- $v_h.incspeed$: Host vehicle increases speed.
- $v_h.decspeed$: Host vehicle decreases speed.
- $v_h.contspeed$: Host vehicle maintains or continues current speed.
- $v_h.changelane$: Host vehicle changes its lane.

5.2.3 Violations

- $v_h.v_1$: This violation is true when the host vehicle exceeds the maximum speed limit.
- $v_h.v_2$: Denotes the violation when the host vehicle exceeds the minimum safety distance with respect to its front opponent vehicle.
- $v_h.v_3$: Denotes the violation when the host vehicle's distance with respect to the rear opponent vehicle is less than the minimum safety distance.

5.2.4 Axioms

SAS1 $\neg Done(U) \longrightarrow v_h.speed_{safe} \wedge road.clear_{front} \wedge road.clear_{rear} \wedge \neg v_h.v_1 \wedge \neg v_h.v_2 \wedge \neg v_h.v_3$

SAS1 defines the initial state of the host vehicle and says that at the beginning the host vehicle's speed is safe, the front and rear end of the host vehicle is clear, i.e., it has not exceeded the minimum safety distance with respect to any vehicle in its ego path and the host vehicle is moving without any violation.

SAS2 $v_h.speed_{inc} \oplus v_h.speed_{dec} \oplus v_h.speed_{cont}$

SAS3 $v_h.speed_{safe} \oplus v_h.exceed_{maxspeed}$

SAS4	$v_h.stopped \longrightarrow$
	$[v_h.incspeed \sqcup v_h.decspeed \sqcup v_h.contspeed \sqcup v_h.changelane]\bot$
SAS5	$(road.clear_{front} \longleftrightarrow \neg v_h.v_2) \wedge (road.clear_{rear} \longleftrightarrow \neg v_h.v_3)$
SAS6	$v_h.speed_{inc} \vee v_h.speed_{dec} \vee v_h.speed_{cont} \longrightarrow$
	$[v_h.decspeed \sqcup v_h.incspeed \sqcup v_h.contspeed]\top$

The first axiom **SAS2** expresses a disjointness condition about the host vehicle's increase, decrease and continuing speed. We know that a vehicle executes an action either to increase or decrease or continue its speed. So in any state, more than one of these propositions cannot be true. Axiom **SAS3** says the host vehicle cannot be in a state where it has a safe speed and it also exceeds the maximum speed. For our model, we are assuming that if the host vehicle exceeds the maximum speed limit, then its speed is not safe. **SAS4** says a stopped host vehicle cannot increase, decrease or continue its speed or change lane. **SAS5** expresses that the front road is clear when the host vehicle is not in violation $v_h.v_2$ and the rear road is clear when the host vehicle is not in violation $v_h.v_3$. And **SAS6** shows when actions of increase, decrease or continue speed can occur.

Now we model the action $v_h.changelane$.

SAS7	$([v_h.changelane]v_h.reachedaddlane)$
	$\wedge (\neg v_h.reachedaddlane \longrightarrow [\overline{v_h.changelane}]\neg v_h.reachedaddlane)$
SAS8	$\neg v_h.exceed_{maxspeed} \wedge v_h.hasadjlane \wedge \neg v_h.onadjlane(v_o) \longrightarrow$
	$P_w(v_h.changelane)$

SAS7 expresses the effect of action $v_h.changelane$, i.e., after executing this action, the host vehicle goes to an adjacent lane. Axiom **SAS8** expresses that the host vehicle is permitted to change lane if there is no opponent vehicle in the adjacent lane and it has a safe speed. Here the action is given weak permission because, in some cases (like violation $v_h.v_1$), the host vehicle is not permitted to change lane.

The following axioms model the action $v_h.contspeed$ and specify the scenarios when host vehicle maintains its current speed.

SAS9	$([v_h.contspeed]v_h.speed_{cont}) \wedge (\neg v_h.speed_{cont} \longrightarrow [\overline{v_h.contspeed}]\neg v_h.speed_{cont})$
SAS10	$(\neg v_h.exceed_{maxspeed} \longrightarrow P(v_h.contspeed)) \wedge (\neg v_h.exceed_{minsafedistfront} \longrightarrow P(v_h.contspeed))$
SAS11	$v_h.speed_{safe} \longrightarrow [v_h.contspeed]v_h.speed_{safe}$
SAS12	$(\neg v_h.exceed_{minsafedistfront} \vee \neg v_h.exceed_{minsafedistrear}) \longrightarrow [v_h.contspeed](\neg v_h.exceed_{minsafedistfront} \vee \neg v_h.exceed_{minsafedistrear})$

Axiom **SAS9** models the action $v_h.contspeed$. Axiom **SAS10** says that the host vehicle is permitted to maintain its speed if it has not exceeded the maximum speed limitation or the minimum safety distance limitation. Axioms **SAS11** and **SAS12** require that the continue speed action preserves safe speed and safe distance. As we are assuming that the opponent vehicle always goes at a constant speed, so if the host vehicle currently has safe distance from the opponent vehicle, then executing this action will maintain the same distance with respect to an opponent vehicle.

The following axioms model the action $v_h.incspeed$ and specify the scenarios when the speed is increasing.

SAS13	$([v_h.incspeed]v_h.speed_{inc}) \wedge (\neg v_h.speed_{inc} \longrightarrow [\overline{v_h.incspeed}]\neg v_h.speed_{inc})$
SAS14	$road.clear_{front} \wedge (v_h.speed_{cont} \vee v_h.speed_{inc} \vee v_h.speed_{dec}) \longrightarrow \langle v_h.incspeed\rangle\top$
SAS15	$v_h.speed_{inc} \wedge \neg v_h.speed_{safe} \longrightarrow O(v_h.decspeed)$
SAS16	$P(v_h.incspeed)\ \mathcal{U}\ (v_h.exceed_{maxspeed} \vee v_h.exceed_{minsafedistfront})$
SAS17	$v_h.speed_{safe} \longrightarrow EN([v_h.incspeed]\neg v_h.speed_{safe})$
SAS18	$\neg v_h.exceed_{minsafedistfront} \longrightarrow EN([v_h.incspeed]v_h.exceed_{minsafedistfront})$

SAS13 expresses the effect of action $v_h.incspeed$ and says that the host vehicle's speed increases after executing this action. **SAS14** indicates when the host vehicle can increase speed and **SAS15** says if the host vehicle has increased speed and if the speed is not safe, then it is obliged to decrease speed. **SAS16** imposes permission on $v_h.incspeed$ to state that the host vehicle is permitted to increase speed until it exceeds the maximum speed limit or the minimum safety distance limit. Axiom **SAS17** says that, in some execution path, the host vehicle's speed is unsafe after executing this action, i.e., the host vehicle exceeds the maximum speed limitation. Similarly, axiom **SAS18** says that executing the increase speed action can bring about an unsafe distance from the front vehicle in some execution path.

The following axioms model the action $v_h.decspeed$ and specify the scenarios when speed is decreasing.

SAS19	$([v_h.decspeed]v_h.speed_{dec}) \land$ $(\neg v_h.speed_{dec} \longrightarrow [\overline{v_h.decspeed}]\neg v_h.speed_{dec})$
SAS20	$\neg v_h.stopped \longrightarrow EN([v_h.decspeed]v_h.stopped)$
SAS21	$road.clear_{rear} \land$ $(v_h.speed_{cont} \lor v_h.speed_{inc} \lor v_h.speed_{dec}) \longrightarrow \langle v_h.decspeed\rangle\top$
SAS22	$P(v_h.decspeed)\ \mathcal{U}\ (v_h.exceed_{minsafedistrear} \lor v_h.stop)$
SAS23	$\neg v_h.exceed_{minsafedistrear} \longrightarrow$ $EN([v_h.decspeed]v_h.exceed_{minsafedistrear})$
SAS24	$v_h.exceed_{minsafedistfront} \longrightarrow$ $EN([v_h.decspeed]\neg v_h.exceed_{minsafedistfront})$

Axiom **SAS19** expresses the behaviour of action $v_h.decspeed$ and says that the host vehicle's speed decreases after executing this action. Axiom **SAS20** indicates that, in some execution path, the host vehicle is stopped after decreasing speed. **SAS21** says when the host vehicle can decrease speed. Axiom **SAS22** shows that the host vehicle can decrease speed until it exceeds the minimum safety distance with respect to the rear vehicle or it has stopped. Axiom **SAS23** says that, in some execution path, after executing the decrease speed action, the host vehicle can achieve an unsafe distance with respect to the rear opponent vehicle. And the last axiom indicates that, in some execution path, host vehicle overcomes the front distance violation after executing this action.

5.2.5 Violations and Recovery Mechanisms

Finally we present a collection of axioms for modelling violations. Our violations are mainly categorized based on the speed limitations and distance limitations with respect to the front and rear opponent vehicles.

The first violation predicate $v_h.v_1$ is true when the host vehicle exceeds the maximum speed limit. To recover from this violation, the host vehicle must decrease its speed. In this formalization we are assuming that host vehicle can recover from any violation by the time it reaches the next state by executing an obligatory action in that one step. So by executing the decrease speed action, the host vehicle recovers from this violation in the next state.

The other violation $v_h.v_2$ defines cases when the host vehicle exceeds the minimum safety distance limitation with respect to the front opponent vehicle. We are considering two different situations: (1) The host vehicle has positive relative speed with respect to the front opponent vehicle and (2) the host vehicle has negative relative speed with respect to the front opponent vehicle. The host vehicle and the front opponent vehicle can have the same speed, i.e., a zero relative speed. Having the same speed can be considered as having a positive relative speed in case of this violation, because when the host vehicle is in violation $v_h.v_2$ with positive relative speed, it has to decrease its speed to recover from the violation. Similarly, to recover from violation $v_h.v_2$ with zero relative speed, the host vehicle has to decrease speed as the only option. It cannot recover from this violation situation if it increase speed or maintains current speed. As we said before, in this model we are assuming that the opponent vehicle's speed is constant, so only host vehicle's action can recover it from violation. But in our next model we will see that the opponent vehicle's action can also recover the host vehicle from a violation.

Now we model the speed violation $v_h.v_1$ and the recovery actions to overcome from this violation.

VS1	$\neg v_h.v_1 \ \wedge\ \neg v_h.exceed_{minsafedistfront} \ \wedge\ O(v_h.decspeed) \longrightarrow [\overline{v_h.decspeed}]v_h.v_1$
VS2	$v_h.v_1 \ \wedge\ O(v_h.decspeed) \longrightarrow [v_h.decspeed]\neg v_h.v_1$
VS3	$\neg v_h.v_1 \ \wedge\ \neg O(v_h.decspeed) \longrightarrow [U]\neg v_h.v_1$
VS4	$v_h.speed_{safe} \oplus v_h.v_1$

The first axiom defines when $v_h.v_1$ can be true, that is when the host vehicle has not yet exceeded the minimum distance limitation with respect to the front opponent vehicle and is obliged to reduce the speed but does not do that. Axiom **VS2** indicates that the host vehicle recovers from violation $v_h.v_1$ after executing the action to decrease speed. **VS3** says that the other actions do not affect $v_h.v_1$, i.e., if the host vehicle is not currently in violation $v_h.v_1$ and is not obliged to decrease speed, then after executing any action it will not be in that violation. And axiom **VS4** says that violation $v_h.v_1$ and safe speed are mutually exclusive.

Now we model violation $v_h.v_2$ with both positive and negative relative speed with the front opponent vehicle. At first we are defining the violation and showing the possible and obligated actions from a state where violation $v_h.v_2$ is true. The axioms are as follows:

VD1	$\neg v_h.v_2 \ \wedge\ O(v_h.decspeed \sqcup v_h.contspeed \sqcup v_h.changelane) \longrightarrow ([\overline{v_h.decspeed \sqcup v_h.contspeed \sqcup v_h.changelane}]v_h.v_2 \ \wedge\ [v_h.decspeed \sqcup v_h.contspeed \sqcup v_h.changelane]\neg v_h.v_2)$
VD2	$\neg v_h.v_2 \ \wedge\ \neg O(v_h.decspeed \sqcup v_h.contspeed \sqcup v_h.changelane) \longrightarrow [U]\neg v_h.v_2$
VD3	$v_h.v_2 \ \wedge\ v_h.posrelspeed(v_o)$ $\wedge\ (\neg v_h.hasadjlane \longrightarrow O(v_h.decspeed))$ $\wedge\ (v_h.hasadjlane \wedge v_h.onsamelane(v_o) \wedge \neg v_h.onadjlane(v_o) \longrightarrow O(v_h.decspeed \sqcup v_h.changelane))$ $\wedge\ (v_h.hasadjlane \wedge v_h.onsamelane(v_o) \wedge v_h.onadjlane(v_o) \longrightarrow O(v_h.decspeed))$

VD4 $v_h.v_2 \ \wedge \ v_h.negrelspeed(v_o)$
$\wedge \ (\neg v_h.hasadjlane \ \longrightarrow \ O(v_h.decspeed \ \sqcup \ v_h.contspeed))$
$\wedge \ (v_h.hasadjlane \wedge v_h.onsamelane(v_o) \wedge \neg v_h.onadjlane(v_o) \ \longrightarrow$
$O(v_h.decspeed \ \sqcup \ v_h.changelane \ \sqcup \ v_h.contspeed))$
$\wedge \ (v_h.hasadjlane \ \wedge \ v_h.onsamelane(v_o) \ \wedge \ v_h.onadjlane(v_o) \ \longrightarrow$
$O(v_h.decspeed \ \sqcup \ v_h.contspeed))$

Here **VD1** expresses that violation $v_h.v_2$ can be true if the host vehicle is obliged to reduce speed or maintain current speed or change lane but does not do that and is false otherwise. **VD2** says actions other than reducing or continuing speed or changing lane do not affect $v_h.v_2$. **VD3** defines different actions that the host vehicle is obliged to execute when it is in violation $v_h.v_2$ and it has positive relative speed with respect to the front opponent vehicle. This axiom clarifies three different situations for that violation. (1) It says that the host vehicle is obliged to reduce speed if it does not have any adjacent lane available. (2) The host vehicle is obliged to reduce speed or change lane if it has an adjacent lane and there is no opponent vehicle within the minimum safety distance in that adjacent lane. And (3) The host vehicle is obliged to reduce speed if it has an adjacent lane but there is an opponent vehicle within the minimum safety distance in that adjacent lane. Axiom **VD4** is similar to axiom **VD3** and indicates the obligated actions when the host vehicle is in violation $v_h.v_2$ with negative relative speed with respect to the front opponent vehicle.

Now we describe the recovery mechanism when the host vehicle is in violation $v_h.v_2$. The host vehicle recovers from this violation when it changes lane or creates more distance than the minimum safety distance limitation with respect to the front opponent vehicle. The axioms are as follows:

VD5 $v_h.v_2 \wedge v_h.posrelspeed(v_o) \wedge O(v_h.decspeed \sqcup v_h.changelane) \ \longrightarrow$
$[v_h.decspeed \ \sqcup \ v_h.changelane]\neg v_h.v_2$

VD6 $v_h.v_2 \ \wedge \ v_h.negrelspeed(v_o) \ \wedge$
$O(v_h.decspeed \ \sqcup \ v_h.contspeed \ \sqcup \ v_h.changelane) \ \longrightarrow$
$[v_h.decspeed \ \sqcup \ v_h.contspeed \ \sqcup \ v_h.changelane]\neg v_h.v_2$

Axiom **VD5** says that if the host vehicle is in violation $v_h.v_2$ with positive relative speed, then after executing the decrease speed or change lane action it recovers from the violation. Axiom **VD6** is similar to axiom **VD5** and shows the recovery mechanism for the same violation with negative relative speed. The only difference between these two situations is that when the host vehicle has negative relative speed, it can continue or maintain its current speed to recover from the violation $v_h.v_2$, because as the host vehicle has less speed than the front opponent vehicle, maintaining current speed will increase the distance.

Axioms **VS2**, **VD5** and **VD6** describes the recovery mechanisms from the violations $v_h.v_1$ and $v_h.v_2$. In all these cases the recovery mechanism takes only one transition according to our assumption and after executing the recovery action from the violation state the system goes to the next state where the system is safe. But it might happen that the recovery mechanism takes multiple transitions, i.e., after executing the recovery action multiple times, the system goes to a safe state. The temporal operator $\mathcal{U}$ can help us to formalize these sorts of scenarios. Using this operator, we can express that until the system reaches a safe state from a violation state, the recovery action has to be executed. Though, according to our assumption for this model, recovery mechanisms take only one step, but here we present same recovery mechanisms in multiple steps using the operator $\mathcal{U}$, in order to understand that how multi step recovery mechanisms can be defined in this logic.

VS2* $v_h.v_1 \wedge (O(v_h.decspeed)\, \mathcal{U}\, v_h.speed_{safe}) \longrightarrow$
$((v_h.speed_{dec}\, \mathcal{U}\, v_h.speed_{safe}) \longrightarrow \neg v_h.v_1)$

VD5* $v_h.v_2 \wedge v_h.posrelspeed(v_o) \wedge$
$((O(v_h.decspeed)\, \mathcal{U}\, \neg v_h.exceed_{minsafedistfront}) \vee O(v_h.changelane))$
$\longrightarrow (((v_h.speed_{dec}\, \mathcal{U}\, \neg v_h.exceed_{minsafedistfront}) \longrightarrow \neg v_h.v_2) \vee$
$([v_h.changelane]\neg v_h.v_2))$

VD6*	$v_h.v_2 \ \wedge \ v_h.negrelspeed(v_o) \ \wedge$ $((O(v_h.decspeed \ \sqcup \ v_h.contspeed) \ \mathcal{U} \ \neg v_h.exceed_{minsafedistfront})$ $\vee \ O(v_h.changelane)) \ \longrightarrow$ $((((v_h.speed_{dec} \ \vee \ v_h.speed_{cont}) \ \mathcal{U} \ \neg v_h.exceed_{minsafedistfront}) \ \longrightarrow$ $\neg v_h.v_2) \ \vee \ ([v_h.changelane]\neg v_h.v_2))$
VD7	$v_h.v_2 \ \wedge \ v_h.posrelspeed(v_o) \longrightarrow$ $EN([v_h.decspeed](v_h.v_2 \ \wedge \ v_h.negrelspeed(v_o)))$

Here axiom **VS2*** says that the host vehicle has to decrease speed until it attains a safe speed to recover from violation $v_h.v_1$. Axiom **VD5*** and **VD6*** describes the recovery mechanism from violation $v_h.v_2$. The first one says that if the host vehicle is in violation $v_h.v_2$ with positive relative speed, then to recover from this violation it has to change lane or decrease speed until it attains a safe distance. Axiom **VD6*** is similar to axiom **VD5*** and indicates that the host vehicle can also maintain its current speed until it attains safe distance to recover from violation $v_h.v_2$ with negative relative speed. And the last axiom describes how violation $v_h.v_2$ improves to a better state as we are assuming that having negative relative speed with this violation is a better situation than having positive relative speed.

Now we consider situations where the host vehicle is in both violation $v_h.v_1$ and $v_h.v_2$. That is the host vehicle is in a state where it has exceeded both the maximum speed and the minimum safety distance limit. When both violations are true, the reducing speed action will eventually lead the host vehicle out of both violations. In normal cases, when the host vehicle is in violation $v_h.v_2$, it can also execute the continue speed or the change lane action to recover from this violation. But in this scenario, the host vehicle cannot continue its current speed to recover from this violation as it has exceeded the maximum speed limitation. Also we are assuming that changing lanes with excessive speed can be dangerous, so in this scenario the host vehicle cannot execute the change lane action. But if after decreasing speed the host vehicle recovers from violation $v_h.v_1$ and still $v_h.v_2$ is true (though for this model they should recover at the same time), then the host vehicle can decrease or continue speed or change lane to recover from the second violation.

VSD1 $v_h.v_1 \wedge v_h.v_2 \longrightarrow O(v_h.decspeed)$

VSD2 $v_h.v_1 \wedge v_h.v_2 \longrightarrow$
$([v_h.decspeed]((v_h.speed_{safe} \wedge v_h.exceed_{minsafedistfront}) \longrightarrow$
$(\neg v_h.v_1 \wedge v_h.v_2)) \vee$
$[v_h.decspeed]((\ negv_h.speed_{safe} \wedge \neg v_h.exceed_{minsafedistfront}) \longrightarrow$
$(v_h.v_1 \wedge \neg v_h.v_2)))$

Axiom **VSD1** says that if the host vehicle is in both violations $v_h.v_1$ and $v_h.v_2$, then it is obliged to decrease speed. Axiom **VSD2** describes the recovery mechanism from violation $v_h.v_1$ and $v_h.v_2$. Though this axiom shows recovery from one violation after executing one decrease speed action, it can also happen that the host vehicle recovers from both violations after executing just one recovery action.

Now we describe the recovery mechanism mentioned in **VSD1** and **VSD2** requiring multiple states, using the temporal operator $\mathcal{U}$. Here recovery can also take multiple states and the until operator is useful to deal with this situation.

VSD1* $v_h.v_1 \wedge v_h.v_2 \longrightarrow$
$O(v_h.decspeed)\mathcal{U}(v_h.speed_{safe} \wedge \neg v_h.exceed_{minsafedistfront})$

VSD2* $v_h.v_1 \wedge v_h.v_2 \longrightarrow$
$(((v_h.speed_{dec}\, \mathcal{U}\ v_h.speed_{safe}) \longrightarrow \neg v_h.v_1 \wedge v_h.v_2)$
$\wedge\ ((v_h.speed_{dec}\, \mathcal{U}\ \neg v_h.exceed_{minsafedistfront}) \longrightarrow v_h.v_1 \wedge \neg v_h.v_2))$

Axiom **VSD1*** says that if the host vehicle is in violations $v_h.v_1$ and $v_h.v_2$, then it must decrease its speed until it attains a safe speed or safe distance. Axiom **VSD2*** indicates that reducing speed until attaining a safe speed takes the host vehicle out of violation $v_h.v_1$ and reducing speed until the host vehicle reaches the minimum safety distance from the front opponent vehicle will recover from the violation $v_h.v_2$.

Up-to this point, we have modelled violation $v_h.v_2$ and the related recovery mechanisms. In a similar approach we can also model violation $v_h.v_3$ where the host vehicle exceeds the minimum safety distance limitation with respect to its rear opponent vehicle.

5.3 Summary

In this chapter we have modelled a fault tolerance problem in Deontic Propositional Logic. During this modelling we have found that it is not possible or feasible to model some real aspects in this logical formalism as the notion of quantification is missing. So our main focus was to define very basic actions and model some simple cases. Here we have mainly focused on one step recovery mechanisms, though we have presented some multi step recovery approaches using the temporal operator $\mathcal{U}$. We also focused only on cases where the agent's action implements the recovery mechanism and omitted any environmental effect on the recovery process.

Chapter 6

Model in First Order Deontic Logic

In this chapter we will present a model in Deontic First Order Logic. We have already discussed this logical formalism in Chapter 3. Here we are also modelling the same problem discussed in Chapter 4, but in a more extended form than the model in Chapter 5. In our previous formalization in Propositional Deontic Logic, we captured actions only of the host vehicle. But here in this formalization we will focus on both the host and the opponent vehicle's actions. In another sense, we can say that in this example we are considering environmental effects (effects of an opponent vehicle) on the agent (host vehicle). In our last model, we basically focused on one step recovery, but here we will see how a vehicle recovers from violation in multiple steps. We know that Deontic First Order Logic (DFOL) allows us to quantify on data and we can define actions with parameters. So this Logical approach helps us to model complex real world scenarios where quantification is necessary. To make the example more realistic, we are considering another distance limitation called the minimum collision distance limitation. In real situations, vehicles crash when they get too close to the opponent vehicle and cannot control the distance. This minimum collision distance limitation represents the least distance necessary to avoid a collision.

6.1 Assumptions

Now we list all the assumptions that we have made for this basic example.

QASM1 Stopping the host vehicle is considered as the last recovery action if other recovery actions cannot recover the host vehicle from the violation(s).

QASM2 We are assuming that the host vehicle and the opponent vehicle are moving in the same direction on a straight road, any curvature of the road is not considered here. Multiple lanes can be possible in our scenarios, but, host vehicle does not change its lane. Other vehicles can come in or can go out from host vehicle's lane. The reason is we want to see how cut-in and cut-out vehicles affect different scenarios. Moreover, as we are assuming that the host vehicle does not change its lane, so in this respect, this model is not as general as the PDL one, where changing lanes to recover was possible.

QASM3 In our model, we are considering one particular road which has a fixed maximum speed limit and a fixed minimum speed limit. The host vehicle is not allowed to have a speed over the maximum limit and under the minimum limit.

QASM4 The host vehicle is bound to maintain certain safety distances from the opponent vehicle for safety purposes. We are assuming that the distance limitations are not fixed and depend on the current speed of the host and the opponent vehicles. For example, the minimum safety distance between the host and the front opponent vehicle are not the same when they move at a very high speed and at a slower speed. The host vehicle can execute any action (within a limited speed change) if it does not exceed this minimum safety distance.

QASM5 We are considering another distance limitation called the minimum collision distance, which is the least distance that the host vehicle must maintain with the opponent vehicle to avoid collision. If the distance between the host and the opponent vehicles is less than this distance, then host vehicle must stop to avoid collision.

QASM6 A vehicle can be stopped in two ways. One is the regular stop by decreasing speed (over time) and the other one is the emergency stop.

6.2 Some notation

We will use the notation v_h, v_{of} and v_{or} to denote the host vehicle, the front opponent vehicle and the rear opponent vehicle, respectively. In some cases (to define predicates, actions and axioms), instead of writing v_h, v_{of} or v_{or} separately, we will use the notation $v_{h(of,or)}$ where the notions of host, front and rear opponent vehicle are applicable. Also, to mention the non-deterministic execution of any action by the host, front or rear opponent vehicle, we will use the notation $v_h.U$, $v_{of}.U$ and $v_{or}.U$ respectively.

6.3 Problem formalization

Here in this problem formalization, we are considering types or sorts to make the specification more appealing and realistic. As usual, we can express them using predicates. We have the following types:

6.3.1 Types

- ***Speed*** : denotes the set of values that both host or opponent vehicle can take as its speed. *Speed* is the set of real numbers $\mathcal{R}$.
- ***Distance*** : denotes the set of values of the distance between the host and opponent vehicle. *Distance* is the set of real numbers $\mathcal{R}$.

6.3.2 Constants

- $maxspeed$: *Speed* denotes the constant $maxspeed$ which is the maximum allowed speed for the road.

- $minspeed : Speed$ denotes the constant $minspeed$ which is the minimum allowed speed for the road.

6.3.3 Variables

- $curntspeed_h$: $Speed$ variable denotes the current speed of the host vehicle.
- $curntspeed_{of}$: $Speed$ variable denotes the current speed of the front opponent vehicle.
- $curntspeed_{or}$: $Speed$ variable denotes the current speed of the rear opponent vehicle.
- $curntdist_{front}$: $Distance$ variable denotes the current distance between the host and the front opponent vehicle.
- $curntdist_{rear}$: $Distance$ variable denotes the current distance between the host and the rear opponent vehicle.

6.3.4 Functions

- $minsafedistfront(curntspeed_h : Speed, curntspeed_{of} : Speed) \rightarrow Distance$: denotes the function $minsafedistfront(-,-)$ that takes the current speed of both the host and front opponent vehicles as parameters and returns the minimum safety distance that the host vehicle must maintain with respect to the front opponent vehicle. Here we are using $curntspeed_h$ and $curntspeed_{of}$ as parameters which are of type $Speed$ to indicate that the first parameter is the host vehicle's current speed and the second one is the front opponent vehicle's current speed.
- $minsafedistrear(curntspeed_h : Speed, curntspeed_{or} : Speed) \rightarrow Distance$: denotes the function $minsafedistrear(-,-)$ that returns the minimum safety distance between the host and the rear opponent vehicles.

- $mincoldistfront(curntspeed_h : Speed, curntspeed_{of} : Speed) \rightarrow Distance$: denotes the function $mincoldist(-,-)$ which returns the minimum collision distance between the host and front opponent vehicles which we assume is the least distance that has to be maintained to avoid collision. This function also takes the current speed of the host and front opponent vehicles in sequence as parameters.

- $mincoldistrear(curntspeed_h : Speed, curntspeed_{or} : Speed) \rightarrow Distance$: denotes the function $mincoldist(-,-)$ which returns the minimum collision distance between the host and rear opponent vehicles.

6.3.5 Predicates

- $greaterSpeed(i : Speed, maxspeed : Speed)$: Speed i is greater than the maximum speed $maxspeed$.

- $lesserSpeed(i : Speed, minspeed : Speed)$: Speed i is less than the minimum speed $minspeed$.

- $greaterDistance(i : Distance, j : Distance)$: Distance i is greater than distance j.

- $v_{h(of,or)}.stopped$: Host (front or rear opponent) vehicle is stopped.

- $v_{h(of,or)}.speed_{cont}$: Host (front or rear opponent)vehicle continues or maintains current speed. The predicate is true in all those states where the vehicle's speed is equal to that in the previous state.

- $v_{h(of,or)}.speed_{inc}(i : Speed)$: Host (front or rear opponent) vehicle's speed is increased by amount i in a transition. This predicate is true in all those states where the vehicle's speed is i more than the previous state.

- $v_{h(of,or)}.speed_{dec}(i : Speed)$: Host (front or rear opponent) vehicle's speed is decreased by amount i in a transition. This predicate is true in all those states where the vehicle's speed is i less than the previous state.

- $v_{h(of,or)}.speed_{curnt}(i : Speed)$: Host (front or rear opponent) vehicle's current speed is i.

- $v_h.speed_{safe}(i : Speed)$: Host vehicle has safe speed i. This predicate is true when the host vehicle's speed i is neither more than the maximum speed limitation, nor less than the minimum speed limitation.

- $v_h.roadclear_{front}$: No obstacle in front of the host vehicle.

- $v_h.roadclear_{rear}$: No obstacle is behind of the host vehicle.

- $v_h.exceed_{maxspeed}$: Host vehicle exceeded maximum speed limit.

- $v_h.speedlowerthan_{minspeed}$: Host vehicle's speed is lower than minimum speed limit.

- $v_h.exceed(d : Distance)$: Host vehicle exceeded distance d with respect to the (front or rear) opponent vehicle.

- $v_h v_{of}.posrelspeed$: Host vehicle's current speed is more than the front opponent vehicle's current speed.

- $v_h v_{of}.negrelspeed$: Host vehicle's current speed is less than the front opponent vehicle's current speed.

- $v_h v_{or}.posrelspeed$: Host vehicle's current speed is more than the rear opponent vehicle's current speed.

- $v_h v_{or}.negrelspeed$: Host vehicle's current speed is less than the rear opponent vehicle's current speed.

- $collision(v_h, v_{(of,or)})$: Host vehicle has a collision with front or rear opponent vehicle.

6.3.6 Actions

- $v_{h(of,or)}.incspeed(i : Speed)$: Action to increase the host (opponent) vehicle's speed by amount i.
- $v_{h(of,or)}.decspeed(i : Speed)$: Action to decrease the host (opponent) vehicle's speed by amount i.
- $v_{h(of,or)}.contspeed$: Action to maintain the host (opponent) vehicle's current speed.
- $v_{h(of,or)}.stop$: Host (opponent) vehicle stops in emergency. Executing this action causes the vehicle to stop in one step.

6.3.7 Violations

Now we list the violations for this model. This will help to understand the axioms clearly.

- $v_h.v_1$: This violation is true when the host vehicle exceeds the maximum speed limit.
- $v_h.v_2$: This violation is true when the host vehicle's current speed is lower than the minimum speed limit.
- $v_h.v_3$: The host vehicle exceeds minimum safety distance with respect to the front opponent vehicle.
- $v_h.v_4$: The host vehicle exceeds minimum collision distance with respect to the front opponent vehicle.
- $v_h.v_5$: The host vehicle exceeds minimum safety distance with respect to the rear opponent vehicle.
- $v_h.v_6$: The host vehicle exceeds minimum collision distance with respect to the rear opponent vehicle.

6.3.8 Axioms

QSAS1 $\forall i \in Speed : \neg Done(v_h.U) \longrightarrow$
$(v_h.speed_{safe}(i) \wedge v_h.roadclear_{front} \wedge v_h.roadclear_{rear} \wedge \neg v_h.v_1 \wedge \neg v_h.v_2 \wedge \neg v_h.v_3 \wedge \neg v_h.v_4 \wedge \neg v_h.v_5 \wedge \neg v_h.v_6)$
$\vee\ (v_h.speed_{safe}(i) \wedge (\neg v_h.roadclear_{front} \vee \neg v_h.roadclear_{rear}) \wedge \neg v_h.v_1 \wedge \neg v_h.v_2 \wedge \neg v_h.v_3 \wedge \neg v_h.v_4 \wedge \neg v_h.v_5 \wedge \neg v_h.v_6)$
$\vee\ (v_h.speed_{safe}(i) \wedge \neg v_h.roadclear_{front} \wedge \neg v_h.roadclear_{rear} \wedge \neg v_h.v_1 \wedge \neg v_h.v_2 \wedge \neg v_h.v_3 \wedge \neg v_h.v_4 \wedge \neg v_h.v_5 \wedge \neg v_h.v_6)$
$\vee (\neg v_h.speed_{safe}(i) \wedge (v_h.v_1 \vee v_h.v_2) \wedge \neg v_h.v_3 \wedge \neg v_h.v_4 \wedge \neg v_h.v_5 \wedge \neg v_h.v_6)$
$\vee\ (v_h.speed_{safe}(i) \wedge \neg v_h.v_1 \wedge \neg v_h.v_2 \wedge (v_h.v_3 \vee v_h.v_4 \vee v_h.v_5 \vee v_h.v_6))$
$\vee (v_h.speed_{safe}(i) \wedge \neg v_h.v_1 \wedge \neg v_h.v_2) \wedge ((v_h.v_3 \vee v_h.v_4) \wedge (v_h.v_5 \vee v_h.v_6)))$
$\vee\ (\neg v_h.speed_{safe}(i) \wedge (v_h.v_1 \vee v_h.v_2) \wedge (v_h.v_3 \vee v_h.v_4 \vee v_h.v_5 \vee v_h.v_6))$
$\vee (\neg v_h.speed_{safe}(i) \wedge (v_h.v_1 \vee v_h.v_2) \wedge ((v_h.v_3 \vee v_h.v_4) \wedge (v_h.v_5 \vee v_h.v_6)))$

QSAS1 sets the initial state of the host vehicle. We are considering several situations that can happen at the beginning.

1. Host vehicle has safe speed, it has no front and rear opponent vehicle (i.e., front and rear road is clear) and it is not in any violation.

2. Host vehicle has safe speed, it has front or rear (or both) opponent vehicles, but it is not in any violation.

3. Host vehicle is in a single violation, i.e., there is a speed or distance (safety distance or collision distance) violation.

4. Host vehicle is in multiple violations, i.e., it is in a state where a speed violation and any or both of the distance violations are true.

QSAS2 $\forall i \in Speed :$
$(v_{h(of,or)}.speed_{inc}(i) \oplus v_{h(of,or)}.speed_{dec}(i) \oplus v_{h(of,or)}.speed_{cont}) \ \wedge$
$(v_{h(of,or)}.speed_{inc}(i) \oplus v_{h(of,or)}.speed_{cont} \oplus v_{h(of,or)}.stopped)$

QSAS3 $\forall i \in Speed :$
$v_{h(of,or)}.stopped \ \longrightarrow\ [v_{h(of,or)}.incspeed(i) \ \sqcup\ v_{h(of,or)}.decspeed(i) \ \sqcup\ v_{h(of,or)}.contspeed]\bot \ \wedge\ \neg collision(v_h, v_{(of,or)})$

QSAS4 $(v_h.roadclear_{front} \longrightarrow \neg v_h.v_3 \wedge \neg v_h.v_4) \wedge$
$(v_h.roadclear_{rear} \longrightarrow \neg v_h.v_5 \wedge \neg v_h.v_6)$

QSAS5 $([v_{h(of,or)}.stop]v_{h(of,or)}.stopped)$
$\wedge (\neg v_{h(of,or)}.stopped \longrightarrow [\overline{v_{h(of,or)}.stop}]\neg v_{h(of,or)}.stopped)$

The first axiom **QSAS2** expresses that the host (or opponent) vehicle cannot be in a state where it has done more than one of increase, decrease or continue speed. Axiom **QSAS3** says a stopped host (or opponent) vehicle cannot increase, decrease or continue speed. It also says that if the host vehicle is stopped, then it did not have a collision with its opponent vehicle. Axiom **QSAS4** says that if the road in front is clear, then the host vehicle is not in a safety distance or collision distance violation with the front opponent vehicle, and, if the road in rear is clear, then the host vehicle is not in a safety distance or collision distance violation with the rear opponent vehicle. **QSAS5** models the stopping action of the host and opponent vehicles.

QSAS6 $\forall i \in Speed : v_h.speed_{curnt}(i)$
$\wedge (greaterSpeed(i, maxspeed) \longleftrightarrow v_h.exceed_{maxspeed})$
$\wedge (lesserSpeed(i, minspeed) \longleftrightarrow v_h.speedlowerthan_{minspeed})$

QSAS7 $\forall i, j \in Speed, \forall k \in Distance :$
$curntspeed_h = i \wedge curntspeed_{of} = j \wedge curntdist_{front} = k \longrightarrow$
$((greaterDistance(minsafedistfront(i, j), k) \longleftrightarrow$
$v_h.exceed(minsafedistfront(i, j)))$
$\wedge (greaterDistance(mincoldistfront(i, j), k) \longleftrightarrow$
$v_h.exceed(mincoldistfront(i, j))))$

QSAS8 $\forall i, j \in Speed, \forall k \in Distance :$
$curntspeed_h = i \wedge curntspeed_{or} = j \wedge curntdist_{rear} = k \longrightarrow$
$((greaterDistance(minsafedistrear(i, j), k) \longleftrightarrow$
$v_h.exceed(minsafedistrear(i, j)))$
$\wedge (greaterDistance(mincoldistrear(i, j), k) \longleftrightarrow$
$v_h.exceed(mincoldistrear(i, j))))$

QSAS9 $\forall i \in Speed :$
$\neg v_h.exceed_{maxspeed} \wedge \neg v_h.speedlowerthan_{minspeed} \rightarrow v_h.speed_{safe}(i)$

Axiom **QSAS6** characterizes when the host vehicle exceeds the maximum and minimum speed limits based on its current speed. **QSAS7** shows when the host vehicle exceeds the minimum safety distance and minimum collision distance with respect to the front opponent vehicle, based on the current distance and minimum distance limit . The function $minsafedistfront(-,-)$ and $mincoldistfront(-,-)$ takes the current speed of both the host and the front opponent vehicles and returns the minimum safety distance and minimum collision distance. Axiom **QSAS8** is similar to axiom **QSAS7** and defines when the host vehicle exceeds the minimum safety distance and minimum collision distance with respect to the rear opponent vehicle. And axiom **QSAS9** says that if the host vehicle's speed is not more than the maximum speed limitation and not less than the minimum speed limitation, then it has a safe speed.

The following axioms model the action $v_h.contspeed$ and specify the scenarios when speed is increasing.

QSAS10	$[v_h.contspeed]v_h.speed_{cont} \ \wedge \ [\overline{v_h.contspeed}]\neg v_h.speed_{cont}$
QSAS11	$\forall i \in Speed:$ $(curntspeed_h = i) \ \longrightarrow$ $[v_h.contspeed](curntspeed_h = i) \wedge [\overline{v_h.contspeed}](curntspeed_h \neq i)$
QSAS12	$\forall i \in Speed:$ $v_h.speed_{cont} \vee v_h.speed_{inc}(i) \vee v_h.speed_{dec}(i) \ \longrightarrow \ \langle v_h.contspeed\rangle\top$
QSAS13	$\forall i \in Speed:$ $\neg v_h.exceed_{maxspeed} \ \vee \ \neg v_h.speedlowerthan_{minspeed} \ \longrightarrow$ $P^1(v_h.contspeed)$
QSAS14	$\forall i,j,k \in Speed:$ $(curntspeed_h = i) \ \wedge \ (curntspeed_{of} = j) \ \wedge \ (curntspeed_{or} = k) \ \wedge$ $(\neg v_h.exceed(minsafedistfront(i,j)) \vee$ $\neg v_h.exceed(mincoldistfront(i,j)) \ \vee$ $\neg v_h.exceed(minsafedistrear(i,k)) \vee$ $\neg v_h.exceed(mincoldistrear(i,k))) \longrightarrow P^1(v_h.contspeed)$
QSAS15	$(\neg v_h.exceed_{maxspeed} \ \wedge \ \neg v_h.speedlowerthan_{minspeed}) \ \longrightarrow$ $[v_h.contspeed](\neg v_h.exceed_{maxspeed} \ \wedge \ \neg v_h.speedlowerthan_{minspeed})$

Axiom **QSAS10** models the action $v_h.contspeed$ and axiom **QSAS11** shows how the current speed of the host vehicle varies based on the action $v_h.contspeed$. This action maintains the current speed after execution. Axioms **QSAS12** says that this action can be executed from any state where it has increased, decreased or maintained the current speed. Axiom **QSAS13** and **QSAS14** shows the permissions associated with this action, based on the speed and distance limitation. The action is not permitted to execute when it violates any of the speed or distance limitations as this action maintains current speed. The last axiom says that if the host vehicle is in a state where it has not exceeded the maximum or minimum speed limits, then after executing the action $v_h.contspeed$ it will not exceed those limitations.

The following axioms model the action $v_h.incspeed(-)$ and specify the scenarios when speed is increasing.

QSAS16 $\forall i \in Speed:$
$[v_h.incspeed(i)](v_h.speed_{inc}(i)) \ \wedge\ [\overline{v_h.incspeed(i)}](\neg v_h.speed_{inc}(i))$

QSAS17 $\forall i, j \in Speed:$
$(curntspeed_h = i) \ \longrightarrow\ [v_h.incspeed(j)](curntspeed_h = i + j) \ \wedge$
$[\overline{v_h.incspeed(j)}](curntspeed_h = i)$

QSAS18 $\forall i, j \in Speed:$
$v_h.speed_{cont} \vee v_h.speed_{inc}(i) \vee v_h.speed_{dec}(i) \ \longrightarrow\ \langle v_h.incspeed(j)\rangle\top$

QSAS19 $\forall i \in Speed : \neg v_h.exceed_{maxspeed} \longrightarrow P^1(v_h.incspeed(i))$

QSAS20 $\forall i, j, k \in Speed : (curntspeed_h = i) \ \wedge\ (curntspeed_{of} = j) \ \wedge$
$\neg v_h.exceed(minsafedistfront(i, j)) \longrightarrow P^1(v_h.incspeed(k))$

QSAS21 $\exists i \in Speed:$
$(\neg v_h.exceed_{maxspeed} \ \longrightarrow\ [v_h.incspeed(i)]EN(v_h.exceed_{maxspeed}))$
$\wedge\ (v_h.speedlowerthan_{minspeed} \ \longrightarrow$
$[v_h.incspeed(i)]EN(\neg v_h.speedlowerthan_{minspeed}))$

The axioms **QSAS16** and **QSAS17** expresses the behaviour of the action $v_h.incspeed(-)$ and shows how the current speed changes after executing this action. **QSAS19** says the host vehicle is permitted to increase speed if it has not exceeded the maximum speed limit. Now, as we assign permissions to this action, it might happen that executing this action can exceed the maximum speed limit. In that case, the host vehicle has to take a recovery action to overcome that situation. Axiom **QSAS20** says the host vehicle is permitted to increase speed if it has not exceeded the minimum safety distance. And the last axiom shows that if the host vehicle has not yet exceeded the maximum speed limit in a state, then after executing the increase speed action with some parameter value, it can exceed that limit in the next state. This axiom also shows that, this action takes the host vehicle from a state where the current speed of the host vehicle is less than the minimum speed limit, to a state where the current speed is more than the minimum speed limit.

The following axioms model the action $v_h.decspeed(-)$ and specify the scenarios when speed is decreasing.

QSAS22	$\forall i \in Speed :$ $[v_h.decspeed(i)](v_h.speed_{dec}(i)) \;\wedge\; [\overline{v_h.decspeed(i)}](\neg v_h.speed_{dec}(i))$
QSAS23	$\forall i, j \in Speed : (curntspeed_h = i) \;\longrightarrow$ $[v_h.decspeed(j)](curntspeed_h = i - j)$ $\wedge\; [\overline{v_h.decspeed(j)}](curntspeed_h = i)$
QSAS24	$\forall i, j \in Speed :$ $v_h.speed_{cont} \vee v_h.speed_{inc}(i) \vee v_h.speed_{dec}(i) \;\longrightarrow\; \langle v_h.decspeed(j)\rangle\top$
QSAS25	$\forall i \in Speed : \neg v_h.speedlowerthan_{minspeed} \longrightarrow P^1(v_h.decspeed(i))$
QSAS26	$\forall i, j, k \in Speed : (curntspeed_h = i) \;\wedge\; (curntspeed_{or} = j) \;\wedge$ $\neg v_h.exceed(minsafedistrear(i, j)) \longrightarrow P^1(v_h.decspeed(k))$
QSAS27	$\exists i \in Speed :$ $(\neg v_h.speedlowerthan_{minspeed} \;\longrightarrow$ $[v_h.decspeed(i)]EN(v_h.speedlowerthan_{minspeed})) \;\wedge$ $(v_h.exceed_{maxspeed} \;\longrightarrow\; [v_h.decspeed(i)]EN(\neg v_h.exceed_{maxspeed})) \wedge$ $(\neg v_h.stopped \;\longrightarrow\; [v_h.decspeed(i)]EN(v_h.stopped))$

The axioms **QSAS22** to **QSAS27** are similar to the axioms **QSAS16** to **QSAS21** and express the behaviour of action $v_h.decspeed(-)$. But we want to clarify the last three axioms. Axiom **QSAS25** says that the host vehicle is permitted to decrease speed if its current speed is not less than the minimum speed limit. Axiom **QSAS26** assigns permission to execute this action when the host vehicle has not exceeded the minimum safety distance limitation with a rear opponent vehicle. And the last axiom shows that in some execution path after executing this action, the host vehicle recovers from exceeding the maximum speed limitation and sometimes the host vehicle's speed gets lower than the minimum speed limitation and sometimes it ends up stopped.

In the previous axioms we saw how the current speed changes after executing actions of increasing, decreasing and continuing the speed. In the following axioms we will see how the distance changes after executing actions of the host and opponent vehicles.

QSAS28 $\forall i, j \in Speed, \forall k \in Distance : i = j \ \wedge$
$curntspeed_h = i \ \wedge \ curntspeed_{of} = j \ \wedge \ curntdist_{front} = k \ \longrightarrow$
$[v_h.U \ \sqcap \ v_{of}.U] curntdist_{front} = k$

QSAS29 $\forall i, j \in Speed, \forall k \in Distance :$
$curntspeed_h = i \ \wedge \ curntspeed_{of} = j \ \wedge \ curntdist_{front} = k \ \longrightarrow$
$(([v_h.U] curntdist_{front} = k - i) \ \wedge$
$([v_{of}.U] curntdist_{front} = k + j) \ \wedge$
$([v_h.U \ \sqcap \ v_{of}.U] curntdist_{front} = k - i + j))$

Axiom **QSAS28** says that if the host and front opponent vehicle's current speed is the same, then parallel execution of any action (not necessarily the same actions) will maintain the current distance. Axiom **QSAS29** shows the change of distance after single or parallel execution of action(s) in host and front opponent vehicle.

The change of distance after executing actions(s) (single or parallel) always depends on the current speed of the host and opponent vehicle. We know that the execution of an action takes one instant of time (according to our Deontic Action Logic framework), so after executing an action, a vehicle covers the distance that it can go at one time instant by its current speed. The following figures will make the previous two axioms clearer where front opponent vehicle is followed by host vehicle.

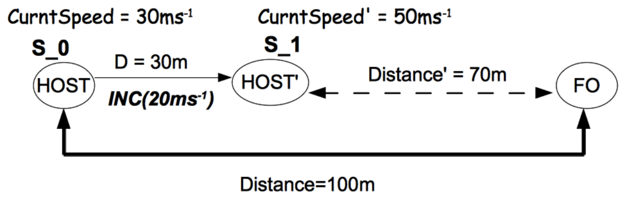

Figure 6.1: Distance after single action execution

Figure: 6.1 shows the situation where the host vehicle executes an action to increase its speed, but the front opponent vehicle (FO) is not executing any action. Here we can see that the initial distance (before execution of an action) between the host and front opponent vehicles is $100m$ (shown by the **dark left-right arrow**). In state S_0, the host vehicle's current speed is $30ms^{-1}$. Then it executes an action to increase the speed by amount $20ms^{-1}$ and reaches state S_1 where the vehicle's current speed is $50ms^{-1}$. As the action takes one instant of time, for example $1s$ in this case, then the distance covered by the host vehicle in reaching state S_1 from S_0 is $30m$ ($30ms^{-1} \times 1s$). So the distance between the host and opponent vehicles after executing action $INC(20ms^{-1})$ is $70m$ (shown by the **dashed left-right arrow**).

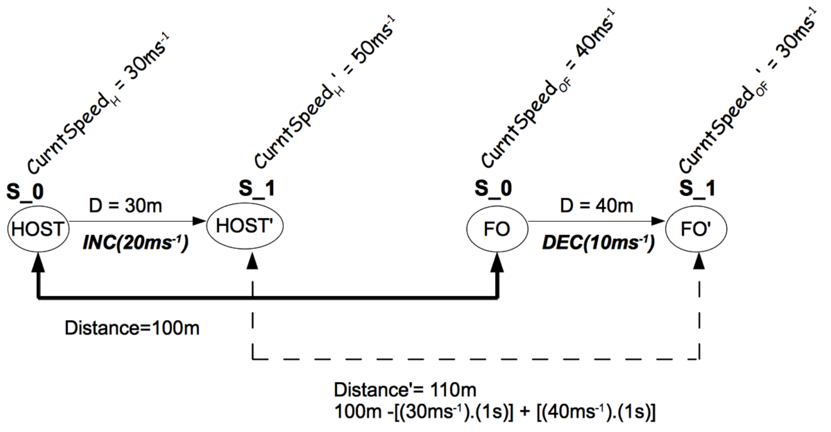

Figure 6.2: Distance after parallel action execution

Figure: 6.2 is similar to the Figure: 6.1 but here both the host and front opponent vehicles execute actions in parallel and step from state S_0 to S_1. The distance between the host and front opponent vehicles after parallel execution of two actions depends on the speed which was the current speed of the host and opponent vehicles before executing the actions. So the equation to calculate the distance between the host and front opponent vehicle after parallel execution of actions is: $Distance_{Init} - [CurntSpeed_H \times Time] + [CurntSpeed_{FO} \times Time]$. In our picture the **dark left-right arrow** shows the initial distance (100m) between the host and front opponent vehicles before executing the parallel actions. The figure shows that in state S_0 the host and front opponent vehicles have initial speeds $30ms^{-1}$ and $40ms^{-1}$, respectively. Then after executing parallel actions $INC(20ms^{-1})$ and $DEC(10ms^{-1})$, respectively, the host and front opponent vehicles reach state S_1 where they have resultant speeds $50ms^{-1}$ and $30ms^{-1}$. But the distance (shown by **dashed left-right arrow**) after parallel execution of actions in state S_1 is 110m.

QSAS30	$\forall i, j \in Speed, \forall k \in Distance : i = j \ \wedge$ $curntspeed_h = i \ \wedge \ curntspeed_{or} = j \ \wedge \ curntdist_{rear} = k \ \longrightarrow$ $[v_h.contspeed \ \sqcap \ v_{or}.contspeed]curntdist_{rear} = k$
QSAS31	$\forall i, j \in Speed, \forall k \in Distance :$ $curntspeed_h = i \ \wedge \ curntspeed_{or} = j \ \wedge \ curntdist_{rear} = k \ \longrightarrow$ $(([v_h.U]curntdist_{rear} = k + i) \ \wedge$ $([v_{of}.U]curntdist_{rear} = k - j) \ \wedge$ $([v_h.U \ \sqcap \ v_{of}.U]curntdist_{rear} = k + i - j))$

Axiom **QSAS30** and **QSAS31** are similar to the axioms **QSAS28** and **QSAS29** and show the change of distance between the host and rear opponent vehicles after executing an action.

Now we describe axioms modelling the situations when the front opponent vehicle changes speed. Any opponent vehicle is not potentially dangerous if there is sufficient distance between the host and opponent vehicles after the opponent vehicle executes any action. For example, if the front opponent vehicle decreases its speed but after decreasing speed the distance between the host and front opponent vehicles is no less than the minimum safety distance or minimum collision distance, then the host vehicle can execute any action. Otherwise, there must be some restriction imposed on the host vehicle's actions.

QSAS32 $\forall i, j \in Speed : i \neq j \ \wedge$
$v_h.speed_{curnt}(i) \ \wedge \ v_{of}.speed_{curnt}(j)$
$\wedge \ (greaterSpeed(i,j) \ \longrightarrow \ v_h v_{of}.posrelspeed)$
$\wedge \ (\neg greaterSpeed(i,j) \ \longrightarrow \ v_h v_{of}.negrelspeed)$

QSAS33 $\forall i, j, k \in Speed :$
$v_h.speed_{curnt}(i) \ \wedge \ v_{of}.speed_{curnt}(j) \ \wedge \ v_h v_{of}.posrelspeed$
$\wedge \ \big(\neg v_h.exceed(minsafedistfront(i,j)) \longrightarrow [v_h.incspeed(k) \ \sqcup$
$v_h.decspeed(k) \ \sqcup \ v_h.contspeed]\top\big)$
$\wedge \ \big(v_h.exceed(minsafedistfront(i,j)) \ \wedge$
$\neg v_h.exceed(mincoldistfront(i,j)) \longrightarrow \neg P^2(v_h.incspeed(k) \ \sqcup$
$v_h.contspeed)\big)$
$\wedge \ \big(v_h.exceed(mincoldistfront(i,j)) \longrightarrow \neg P^4(v_h.incspeed(k) \ \wedge$
$v_h.decspeed(k) \ \wedge \ v_h.contspeed)\big)$

QSAS34 $\forall i, j, k \in Speed :$
$v_h.speed_{curnt}(i) \ \wedge \ v_{of}.speed_{curnt}(j) \ \wedge \ v_h v_{of}.negrelspeed$
$\wedge \ \big(\neg v_h.exceed(minsafedistfront(i,j)) \longrightarrow [v_h.incspeed(k) \ \sqcup$
$v_h.decspeed(k) \ \sqcup \ v_h.contspeed]\top\big)$
$\wedge \ \big(v_h.exceed(minsafedistfront(i,j) \ \wedge$
$\neg v_h.exceed(mincoldistfront(i,j)) \longrightarrow \neg P^2(v_h.incspeed(k)\big)$
$\wedge \ \big(v_h.exceed(mincoldistfront(i,j)) \longrightarrow \neg P^4(v_h.incspeed(k) \ \wedge$
$v_h.decspeed(k) \ \wedge \ v_h.contspeed\big)$

QSAS35 $\forall i, j \in Speed : v_h.speed_{curnt}(i) \ \wedge \ v_{of}.speed_{curnt}(j) \ \wedge$
$\neg v_h.exceed(minsafedistfront(i,j)) \ \longrightarrow$
$\exists k \in Speed : ([v_h.incspeed(k) \ \sqcup \ v_{of}.decspeed(k)]$
$EN\big(v_h.exceed(minsafedistfront(i+k,j)) \ \vee$
$v_h.exceed(minsafedistfront(i,j-k)))\big)$

QSAS36 $\forall i, j \in Speed : v_h.speed_{curnt}(i) \wedge v_{of}.speed_{curnt}(j) \wedge$
$v_h.exceed(minsafedistfront(i, j)) \wedge$
$\neg v_h.exceed(mincoldistfront(i, j)) \longrightarrow$
$\exists k \in Speed : ([v_h.incspeed(k) \sqcup v_{of}.decspeed(k)]$
$EN(v_h.exceed(mincoldistfront(i + k, j)) \vee$
$v_h.exceed(mincoldistfront(i, j - k))))$

QSAS37 $\forall i, j \in Speed : i = j \wedge v_h.speed_{curnt}(i) \wedge v_{of}.speed_{curnt}(j)$
$\wedge (v_h.exceed(minsafedistfront(i, j))$
$\vee v_h.exceed(mincoldistfront(i, j))) \longrightarrow$
$([v_h.contspeed \sqcap v_{of}.contspeed]$
$AN(v_h.exceed(minsafedistfront(i, j))$
$\vee v_h.exceed(mincoldistfront(i, j))))$

Axiom **QSAS32** describes situations in which there is positive or negative relative speed between the host and front opponent vehicles, and says that if the host vehicle's speed is more than the front opponent vehicle's speed, then the host vehicle has positive relative speed and if the host vehicle's speed is less than the front opponent vehicle's speed, then it has negative relative speed. Here we are not measuring how much positive or negative relative speed the host vehicle has in comparison with the front opponent vehicle's speed, rather we are only trying to find out whether it has positive or negative relative speed. Axiom **QSAS33** shows the permitted or possible actions of the host vehicle when it has positive relative speed with the front opponent vehicle. When the host vehicle has positive relative speed but has not exceeded the minimum safety distance with respect to the front vehicle, then the host vehicle can execute any action to increase, decrease and continue speed. If the host vehicle exceeds the minimum safety distance and has positive relative speed, then it is not permitted to increase or continue speed. And if it exceeds the minimum collision distance, then all actions regarding increasing, decreasing and continuing speed are prohibited. Axiom **QSAS34** is similar to axiom **QSAS33** and model the possible actions when the host vehicle has negative relative speed. The only difference is that if the host vehicle exceeds the minimum safety distance with the front opponent vehicle with negative relative speed, then the host vehicle has permission to continue speed along with decreasing speed. Axiom **QSAS35** says that if the host vehicle increases speed or the front opponent vehicle decreases speed with some value, then the host vehicle may exceed the minimum safety distance in the next state. Axiom **QSAS36** shows how the host vehicle exceeds the minimum collision distance in the next state from a state where it already exceeded the minimum safety distance. And the last axiom says that in the current state if the host and the front opponent vehicles have same speed and if they have exceeded the minimum safety distance or minimum collision distance limitation, then; after parallel execution of the continue speed actions they reach the next state where they still exceed those limitations.

6.3.9 Violations and Recovery Mechanism

Now we present a collection of axioms for modelling violations, i.e., the occurrence of faults. First we are modelling violation $v_h.v_1$ where the host vehicle exceeds the maximum speed limit. The host vehicle must decrease its speed to recover from this violation.

QVS1	$\forall j, k, \exists i \in Speed : \neg v_h.v_1 \wedge \neg(v_h.exceed(minsafedistfront(j,k)) \wedge v_h.exceed(mincoldistfront(j,k))) \wedge O^1(v_h.decspeed(i)) \longrightarrow [\overline{v_h.decspeed(i)}]v_h.v_1$
QVS2	$\forall i \in Speed : \neg v_h.v_1 \;\wedge\; \neg O^1(v_h.decspeed(i)) \;\longrightarrow\; [v_h.U]\neg v_h.v_1$
QVS3	$v_h.exceed_{maxspeed} \;\longleftrightarrow\; v_h.v_1$
QVS4	$\forall i \in Speed : v_h.v_1 \;\longrightarrow\; \neg P^2(v_h.contspeed) \wedge \neg P^2(v_h.incspeed(i))$
QVS5	$v_h.v_1 \;\longrightarrow\; \exists i \in Speed : (O^2(v_h.decspeed(i)) \;\wedge\; [v_h.decspeed(i)](\neg v_h.exceed_{maxspeed} \;\longleftrightarrow\; \neg v_h.v_1))$
QVS6	$v_h.v_1 \;\longrightarrow\; \exists i \in Speed : (O^2(v_h.decspeed(i)) \;\wedge\; [v_h.decspeed(i)](v_h.v_1 \;\longrightarrow\; O^2(v_h.decspeed(i))))$
QVS7	$\forall i \in Speed :$ $v_h.v_1 \;\longrightarrow\; [\overline{v_h.decspeed(i)}](v_h.v_1 \;\wedge\; O^3(v_h.decspeed(i)))$
QVS8	$v_h.v_1 \;\longrightarrow\; \exists i \in Speed : EN([v_h.decspeed(i)]\neg v_h.v_1)$
QVS9	$\forall i \in Speed : v_h.speed_{safe}(i) \;\oplus\; v_h.v_1$

The first axiom defines when $v_h.v_1$ can become true, that is, when the host vehicle has not yet exceeded the minimum safety distance or minimum collision distance with respect to the front opponent vehicle and is obliged to reduce the speed, but does not do that. **QVS2** says that executing the other actions does not affect $v_h.v_1$. **QVS3** shows the relation between exceeding the maximum speed limit and violation $v_h.v_1$ and **QVS4** expresses that the host vehicle is not permitted to continue at the same speed or to increase speed by any amount if it is in this violation state. Axiom **QVS5** describes how the host vehicle can recover from the violation $v_h.v_1$ after executing the decrease speed action by a specified amount. It says that if after executing the decrease speed action, the host vehicle is in a state where it does not exceed the maximum speed limit, then it is not in violation $v_h.v_1$. Axiom **QVS6** imposes an obligation on further execution of the decrease speed action if the previous decrease speed action has not caused recovery from violation $v_h.v_1$. Axiom **QVS7** shows how the level of obligation changes when the obligated action is not performed. It says that if an obligation of level 2 (to decrease speed) is not fulfilled (i.e., any action other than the obligated action has been executed) when host vehicle is in violation $v_h.v_1$, then an obligation of level 3 is assigned on the same action in the next state. Axiom **QVS8** says that in some execution path, after decreasing speed with some specified amount, the host vehicle recovers from the violation $v_h.v_1$ in the next state. And the last axiom shows the disjointness between maintaining a safe speed and violation $v_h.v_1$.

In a similar approach, we can also model the violation $v_h.v_2$, where host vehicle's current speed is less than the minimum speed limit. As a recovery mechanism from this violation state, host vehicle is obliged to increase its speed.

Now we model the distance violations $v_h.v_3$ and $v_h.v_4$. The violations $v_h.v_3$ or $v_h.v_4$ are true when the host vehicle exceeds the minimum safety distance or the minimum collision distance, respectively, with respect to the front opponent vehicle. If the host vehicle is in violation $v_h.v_3$, then, because of the situation, it has to decrease speed or continue the current speed to recover from this violation. If the host vehicle has negative relative speed when it is in this violation, then it can decrease or continue the speed, otherwise it must decrease speed as the recovery action. When the host vehicle is in violation $v_h.v_4$, which we assume to be the worst case scenario, the host vehicle must stop to recover from that violation. As we have stated in axiom **QVS6**, for violation $v_h.v_3$, if one recovery action cannot the take host vehicle from a violation state to a safe state, then host vehicle must execute another recovery action. But for $v_h.v_4$, we are assuming that only one emergency stop action will take the host vehicle from this violation state to a safe state (even if vehicle has excessive speed). Here we model both violations together as they are both distance violations of a certain kind.

QVD1	$\forall i \in Speed : \neg v_h.exceed_{maxspeed} \wedge$ $(v_h v_{of}.posrelspeed \wedge O^1(v_h.decspeed(i)) \rightarrow [\overline{v_h.decspeed(i)}]v_h.v_3) \wedge$ $(v_h v_{of}.negrelspeed \wedge O^1(v_h.decspeed(i) \vee v_h.contspeed) \longrightarrow$ $[\overline{v_h.decspeed(i) \sqcup v_h.contspeed}]v_h.v_3) \wedge$
QVD2	$\forall i \in Speed :$ $\neg v_h.v_3 \wedge \neg O^1(v_h.decspeed(i) \vee v_h.contspeed) \longrightarrow [v_h.U]\neg v_h.v_3$
QVD3	$\neg v_h.v_4 \wedge O^1(v_h.stop) \longrightarrow [v_h.stop]\neg v_h.v_4 \wedge [\overline{v_h.stop}]v_h.v_4$
QVD4	$\neg v_h.v_4 \wedge \neg O^1(v_h.stop) \longrightarrow [v_h.U]\neg v_h.v_4$
QVD5	$\forall i, j \in Speed : v_h.speed_{curnt}(i) \wedge v_{of}.speed_{curnt}(j)$ $\wedge (v_h.exceed(minsafedistfront(i,j)) \longleftrightarrow v_h.v_3)$ $\wedge (v_h.exceed(mincoldistfront(i,j)) \longleftrightarrow v_h.v_4)$
QVD6	$(\neg v_h.v_3 \longrightarrow \neg v_h.v_4) \wedge (v_h.v_3 \wedge v_h.v_4 \longrightarrow v_h.v_4)$
QVD7	$v_h.v_3 \vee v_h.v_4 \longrightarrow v_h v_{of}.posrelspeed \vee v_h v_{of}.negrelspeed$

Axiom **QVD1 - QVD4** define when violation $v_h.v_3$ and $v_h.v_4$ can be true. Axiom **QVD5** says that if the host vehicle is in a state where it has exceeded the minimum safety distance, then it is in violation $v_h.v_3$ and if it has exceeded the minimum collision distance, then it is in violation $v_h.v_4$. Axiom **QVD6** shows the relation between violations $v_h.v_3$ and $v_h.v_4$. As we know that the minimum collision distance is less than the minimum safety distance, if any vehicle has exceeded the minimum collision distance, then it has also exceeded the minimum safety distance and, again, if any vehicle has not exceeded the minimum safety distance, then it has not exceeded the minimum collision distance. In our model, if both violations $v_h.v_3$ and $v_h.v_4$ are true in a state, then we will deal with violation $v_h.v_4$ as it describes the more severe situation. And axiom **QVD7** says that the host vehicle has either positive relative speed or negative relative speed with respect to the opponent vehicle when it is in violation $v_h.v_3$ or $v_h.v_4$.

QVD8 $\forall i \in \mathit{Speed}:$
$$(v_h.v_3 \wedge v_h v_{of}.\mathit{posrelspeed} \longrightarrow \neg P^2(v_h.\mathit{contspeed}) \wedge \neg P^2(v_h.\mathit{incspeed}(i))) \wedge (v_h.v_3 \wedge v_h v_{of}.\mathit{negrelspeed} \longrightarrow \neg P^2(v_h.\mathit{incspeed}(i)))$$

QVD9 $\exists i, \forall j, k \in \mathit{Speed}:$
$$v_h.v_3 \wedge O^2(v_h.\mathit{decspeed}(i) \vee v_h.\mathit{contspeed}) \longrightarrow [v_h.\mathit{decspeed}(i) \sqcup v_h.\mathit{contspeed}](v_h.\mathit{speed}_{\mathit{curnt}}(j) \wedge v_{of}.\mathit{speed}_{\mathit{curnt}}(k) \wedge (\neg v_h.\mathit{exceed}(\mathit{minsafedistfront}(j,k)) \longrightarrow \neg v_h.v_3) \wedge (v_h.\mathit{exceed}(\mathit{minsafedistfront}(j,k)) \longrightarrow O^2(v_h.\mathit{decspeed}(i) \vee v_h.\mathit{contspeed})))$$

QVD10 $v_h.v_4 \longrightarrow \neg P^4_w(\overline{v_h.\mathit{stop}})$

QVD11 $v_h.v_4 \wedge O^4(v_h.\mathit{stop}) \longrightarrow [v_h.\mathit{stop}]\neg v_h.v_4$

QVD12 $\forall i \in \mathit{Speed}: v_h.v_3$
$$\wedge (v_h v_{of}.\mathit{posrelspeed} \longrightarrow [\overline{v_h.\mathit{decspeed}(i)}](v_h.v_3 \wedge O^3(v_h.\mathit{decspeed}(i)))) \wedge (v_h v_{of}.\mathit{negrelspeed} \longrightarrow [\overline{v_h.\mathit{decspeed}(i) \sqcup v_h.\mathit{contspeed}}](v_h.v_3 \wedge O^3(v_h.\mathit{decspeed}(i) \vee v_h.\mathit{contspeed})))$$

QVD13 $v_h.v_4 \longrightarrow [\overline{v_h.\mathit{stop}}](v_h.v_4 \wedge O^5(v_h.\mathit{stop}))$

QVD14 $\exists i \in \mathit{Speed}: v_h.v_3$
$$\wedge (v_h v_{of}.\mathit{posrelspeed} \longrightarrow [v_h.\mathit{decspeed}(i)]EN(v_h.v_3)) \wedge (v_h v_{of}.\mathit{negrelspeed} \longrightarrow [v_h.\mathit{decspeed}(i) \sqcup v_h.\mathit{contspeed}]EN(v_h.v_3))$$

QVD15 $v_h.v_4 \longrightarrow \forall i \in \mathit{Speed}:$
$$[\overline{v_h.\mathit{stop}} \sqcap (v_{of}.\mathit{incspeed}(i) \sqcup v_{of}.\mathit{decspeed}(i) \sqcup v_{of}.\mathit{contspeed} \sqcup v_{of}.\mathit{stop})]EN(\mathit{collision}(v_h, v_{of}))$$

Axiom **QVD8** defines the actions that are not permitted for violation $v_h.v_3$, based on having positive and negative relative speed with respect to the front opponent vehicle. Axiom **QVD9** shows how the host vehicle can recover from violation $v_h.v_3$. If after executing the obligatory action to decrease or continue speed, the host vehicle creates more distance between itself and the front opponent vehicle than the minimum safety distance, then it recovers from that violation. Otherwise, it is obliged to perform those actions again. Axiom **QVD10** says that all actions except stopping the vehicle are not permitted when the vehicle is in violation $v_h.v_4$. Axiom **QVD11** says that after executing the stop action the host vehicle recovers from violation $v_h.v_4$. Axioms **QVD12** and **QVD13** show that the level of obligation changes when the host vehicle does not perform the obligated action when it is in violation $v_h.v_3$ or $v_h.v_4$. **QVD14** states that, after executing the obligated action, the vehicle recovers from violation $v_h.v_3$ in the next state in some execution path. The last axiom says that in some execution path, in the next state the host vehicle can have a collision with the front opponent vehicle if the front opponent vehicle executes any action and host vehicle does not stop.

In this formalization we have described the violations and recovery mechanisms for the front opponent vehicle. In a similar approach, we can also model violations $v_h.v_5$ and $v_h.v_6$, where the host vehicle fails to maintain the minimum safety distance and minimum collision distance, respectively, with the rear opponent vehicle.

6.3.10 Cut-in and Cut-out vehicle

In this section we will examine the scenarios when a vehicle comes into or goes out from the host vehicle's lane. In our model our main focus is on the host vehicle, along with the front and rear opponent vehicles. The opponent of our host vehicle is changed when a vehicle comes into or goes out from the host vehicle's lane. This sort of action also changes the current speed and distance of the opponent vehicle from the host vehicle. To formalize this scenario, we add a few more actions, predicates, functions etc. to our model.

Type:

- ***Vehicles***: denotes the set of vehicles.

Constants:

- $host$: $Vehicles$ denotes the constant *host* which is the host vehicle in our formalization.

Predicates:

- $opponentfront(x : Vehicle, y : Vehicle)$ denotes that x is the front opponent of vehicle y.
- $opponentrear(x : Vehicle, y : Vehicle)$ denotes that x is the rear opponent of vehicle y.

Functions:

- $opponent_f(x : Vehicle)$: returns the front opponent vehicle of x.
- $opponent_r(x : Vehicle)$: returns the rear opponent vehicle of x.
- $distanceof(x : Vehicle)$: returns the current distance of the opponent vehicle from the host vehicle.
- $speedof(x : Vehicle)$: returns the current speed of the opponent vehicle
.

Actions:

- $cutinfront(x : Vehicle)$: vehicle x moves in front of the host vehicle from another lane.
- $cutoutfront(x : Vehicle)$: vehicle x moves from in front of the host vehicle.
- $cutinrear(x : Vehicle)$: vehicle x comes in directly behind the host vehicle.

- $cutoutrear(x : Vehicle)$: vehicle x leaves from behind the host vehicle.

Axioms:

C1 $\forall i, j \in Vehicle : opponentfront(i, host) \longrightarrow$
$[cutinfront(j)](opponentfront(j, host) \wedge opponentfront(i, j))$

C2 $\forall i, j \in Vehicle, \forall s \in Speed, \forall l \in Distance :$
$opponentfront(i, host) \wedge (curntdist_{front} = l) \longrightarrow [cutinfront(j)]$
$((curntdist_{front} = distanceof(j)) \wedge (curntspeed_{of} = speedof(j)))$

C3 $\forall i \in Vehicle : opponentfront(i, host) \longrightarrow$
$[cutoutfront(i)](opponentfront(opponent_f(i), host)$

C4 $\forall i \in Vehicle, \forall s \in Speed, \forall l \in Distance :$
$opponentfront(i, host) \wedge (curntdist_{front} = l) \longrightarrow$
$[cutoutfront(i)]((curntdist_{front} = distanceof(opponent_f(i))) \wedge$
$(curntspeed_{of} = speedof(opponent_f(i))))$

Axiom **C1** states that the cut-in vehicle becomes the new front opponent of the host vehicle. **C2** says that when a new cut-in vehicle appears just in front of the host vehicle, the current speed and distance of the front opponent vehicle changes according to the newly appearing vehicle. **C3** shows the change of front opponent vehicle when the immediate front vehicle leaves the lane. And the last axiom is the same as axiom **C2** but shows the changes of the front vehicle's speed and distance when one front opponent vehicle cuts out of the lane.

C5 $\forall i \in Vehicle :$
$\neg v_h.v_3 \vee \neg v_h.v_4 \longrightarrow [cutinfront(i)](EN(v_h.v_3 \vee v_h.v_4))$

C6 $\forall i \in Vehicle :$
$(v_h.v_3 \longrightarrow [cutinfront(i)](AN(v_h.v_3 \vee v_h.v_4))) \wedge$
$(v_h.v_4 \longrightarrow [cutinfront(i)](AN(v_h.v_4)))$

C7 $\forall i \in Vehicle :$
$\neg v_h.v_3 \vee \neg v_h.v_4 \longrightarrow [cutoutfront(i)](AN(\neg v_h.v_3 \vee \neg v_h.v_4))$

C8 $\forall i \in Vehicle :$
$v_h.v_3 \vee v_h.v_4 \longrightarrow [cutoutfront(i)](EN(\neg v_h.v_3 \vee \neg v_h.v_4))$

Axiom **C5** says that if the host vehicle is not in violation $v_h.v_3$ or $v_h.v_4$, then, in some execution path, it can be in any of these violations after a cut-in vehicle comes in front of the host vehicle. Axiom **C6** states that distance violations are preserved after a cut-in vehicle comes in front of the host vehicle. It says that, if the host vehicle is in violation $v_h.v_3$, then in all execution paths it will be either in violation $v_h.v_3$ or in $v_h.v_4$ after that event happens. In another case, if the host vehicle is in violation $v_h.v_4$, then it will still be in that violation after a vehicle comes in front of that vehicle. Axiom **C7** shows that cut-out vehicles do not introduce any violation for the host vehicle. The last axiom states how the host vehicle recovers from a violation or improves the violation condition when the front opponent vehicle leaves the lane.

6.3.11 Properties

In this section, we present some expected system properties and demonstrate that they can be proven from the axioms of the model presented above. Here we exhibit one proof and the rest of the proofs can be found in the Appendix of this thesis.

Property 1 The host vehicle cannot recover from violation $v_h.v_1$ after executing the increase speed action.

1.	$v_h.v_1 \longrightarrow \exists i \in \mathit{Speed} : EN[v_h.\mathit{decspeed}(i)]\neg v_h.v_1$	**QVS8**
2.	$v_h.v_1 \longrightarrow EN[v_h.\mathit{decspeed}(s)]\neg v_h.v_1$	FOL, 1
3.	$\forall i \in \mathit{Speed} :$ $v_h.v_1 \wedge O^2(v_h.\mathit{decspeed}(i)) \longrightarrow [\overline{v_h.\mathit{decspeed}(i)}]v_h.v_1$	**QVS7**
4.	$v_h.v_1 \wedge O^2(v_h.\mathit{decspeed}(s)) \longrightarrow [\overline{v_h.\mathit{decspeed}(s)}]v_h.v_1$	FOL, 3
5.	$\forall i \in \mathit{Speed} : AN([v_h.\mathit{decspeed}(i)]v_h.\mathit{speed}_{\mathit{dec}}(i))$	**TempRule2**, **QSAS21**
6.	$AN([v_h.\mathit{decspeed}(s)]v_h.\mathit{speed}_{\mathit{dec}}(s))$	FOL, 5
7.	$v_h.v_1 \longrightarrow AN([v_h.\mathit{decspeed}(s)]v_h.\mathit{speed}_{\mathit{dec}}(s))$	PL, 6
8.	$v_h.v_1 \longrightarrow EN([v_h.\mathit{decspeed}(s)]v_h.\mathit{speed}_{\mathit{dec}}(s))$	CTL, 7
9.	$v_h.v_1 \longrightarrow EN([v_h.\mathit{decspeed}(s)](v_h.\mathit{speed}_{\mathit{dec}}(s) \wedge \neg v_h.v_1))$	ML, 8, 2

10.	$v_h.v_1 \longrightarrow EN([v_h.decspeed(s)]v_h.speed_{dec}(s))$	PL, 9
11.	$v_h.v_1 \longrightarrow AN([v_h.decspeed(s)](v_h.speed_{dec}(s) \vee v_h.v_1))$	PL, 7
12.	$v_h.v_1 \wedge O^2(v_h.decspeed(s)) \longrightarrow AN([v_h.decspeed(s)](v_h.speed_{dec}(s) \vee v_h.v_1))$	PL, 11
13.	$v_h.v_1 \wedge O^2(v_h.decspeed(s)) \longrightarrow AN([U](v_h.speed_{dec}(s) \vee v_h.v_1))$	BA, 4, 12
14.	$\forall i \in Speed : v_h.speed_{dec}(i) \longrightarrow \neg\exists j \in Speed : v_h.speed_{inc}(j)$	**QSAS2**
15.	$\forall i \in Speed : v_h.speed_{dec}(i) \longrightarrow \forall j \in Speed : \neg v_h.speed_{inc}(j)$	FOL, 14
16.	$v_h.speed_{dec}(s) \longrightarrow \neg v_h.speed_{inc}(s)$	FOL, 15
17.	$v_h.v_1 \wedge O^2(v_h.decspeed(s)) \longrightarrow AN([U](\neg v_h.speed_{inc}(s) \vee v_h.v_1))$	PL, 13, 16
18.	$v_h.v_1 \wedge O^2(v_h.decspeed(s)) \longrightarrow AN([U](v_h.speed_{inc}(s) \longrightarrow v_h.v_1))$	PL, 17
19.	$\forall i \in Speed : (v_h.v_1 \wedge O^2(v_h.decspeed(i)) \longrightarrow AN([U](v_h.speed_{inc}(i) \longrightarrow v_h.v_1))$	PL, FOL, 18

Property 2 If the host vehicle is in a minimum collision distance violation, then, after executing any action, the host vehicle either recovers from the violation or stays in that violation or has a collision with the front opponent vehicle.

$$v_h.v_4 \wedge O^4(v_h.stop) \longrightarrow [v_h.U \sqcap v_{of}.U](v_h.v_4 \vee \neg v_h.v_4 \vee EN(collision(v_h, v_{of})))$$

Property 3 After executing any action, the host vehicle either has a safe speed or it violates the maximum or minimum speed limitations.

$$\forall i \in Speed : v_h.speed_{safe}(i) \longrightarrow [U]AN(v_h.speed_{safe}(i) \vee v_h.exceed_{maxspeed} \vee v_h.speedlowerthan_{minspeed})$$

Property 4 If the host vehicle increases speed from a state where it is in a minimum safety distance violation with respect to the front opponent vehicle, then in the next state it is not permitted to increase speed or is obliged to stop.

$$v_h.v_3 \ \wedge \ (v_h v_{of}.negrelspeed \ \vee \ v_h v_{of}.posrelspeed) \ \longrightarrow$$
$$\exists i, j, \forall k : [v_h.incspeed(i) \ \sqcap \ \overline{v_{of}.incspeed(j)}]$$
$$(AN(\neg P^2(v_h.incspeed(k))) \ \wedge \ EN(\neg P_w^4(\overline{v_h.stop})))$$

Property 5 The host vehicle cannot change the current situation after executing a continue speed action unless the host vehicle is in a minimum safety distance violation with negative relative speed.

$$\forall i \in Speed :$$
$$((v_h.speed_{safe}(i) \ \vee \ v_h.v_1 \ \vee \ v_h.v_4 \ \vee$$
$$(v_h.v_3 \ \wedge \ v_h v_{of}.posrelspeed)) \ \longrightarrow$$
$$[v_h.contspeed](v_h.speed_{safe}(i) \ \vee \ v_h.v_1 \ \vee \ v_h.v_4 \ \vee$$
$$(v_h.v_3 \ \wedge \ v_h v_{of}.posrelspeed)))$$
$$\wedge$$
$$((v_h.v_3 \ \wedge \ v_h v_{of}.negrelspeed) \ \longrightarrow \ [v_h.contspeed]EN(v_h.v_3))$$

In this formalization, we have modelled the scenarios that the host vehicle encounters with the front opponent vehicle, the violations that occur when the host vehicle fails to maintain the maximum speed limitation and the minimum safety distance or minimum collision distance limitations, respectively, with respect to the front opponent vehicle; and the recovery mechanisms for these violations. Accordingly, we have chosen properties that relates with the violations, related with the maximum speed limitation and the minimum safety distance or minimum collision distance limitation. Based on our model, we have tried to prove that, in what conditions the host vehicle cannot recover from a violation; what are the possible situations arise for the host vehicle, from a safe state or a violations state after executing any action or a specified action. In a similar approach, we can also prove system properties related with the violations regarding minimum speed limitation, or the distance violations with respect to the rear opponent vehicle. We can also take into account of proving properties, related with the scenarios that cut-in and cut-out vehicles exhibit.

6.4 Summary

In this chapter we have modelled an automotive example concerning safe speed and safe distance properties with Deontic First Order Logic. We could not represent some real world aspects in Propositional Deontic Logic, such as change of speed and distance. But the Deontic First Order Logic approach helped us to overcome this obstacle. In this example we also took into account the environment's effect on the agent to model fault tolerance mechanisms. Here we have exhibited multi step recovery mechanisms which also provide a clear understanding of different levels of deontic operators and different levels of violation resulting from faults. As an extension of our last model, we have also modelled the cut-in and cut-out vehicle concept and saw how it effects different violations. And finally, we have presented some properties of our model and their proofs. The properties that presented here, are proved in hand, which is relatively a hard task. So, automated tool to prove system properties in this logic would be very interesting and effective.

Chapter 7

Engineering Lessons and Discussion

7.1 Engineering Lessons and Discussion

One of the major objectives defined for this work was to determine the obstacles that arise during high level modelling of fault tolerant systems and ideas for possible ways to overcome them. In this chapter, we describe some important lessons that we have learned during our analysis and modelling of the case study problem.

7.1.1 How easy is it to write the specifications?

In this thesis, we have examined an existing real world problem and defined some high level specifications. Obviously, a very general and important question is: how easy was it to write the specifications? Based on our work, we can say that writing specifications from a problem description is not an easy task, as formal specifications must reflect the details of a system which encodes many design decisions. Our specification writing stage can be broken down into two stages: (1) Extracting requirements from the problem description, (2) Converting requirements into a high level specification.

7.1.1.1 Extracting requirements from the problem description

The first stage of writing specifications starts with obtaining the requirements of the problem. In some cases, it is possible to get the requirements correct, concrete and organized, make the specification process much easier. But in most cases, the specifier faces problems to generate the specifications from the requirements, because the requirements can be ambiguous, and understanding of the requirements may vary from the requirement writer to the specifier. As the requirements are usually composed of natural language sentences, the same concept can be written in completely different ways, which can be misleading for the specifier. Moreover, the specifier may not get all the related requirements together. So, even if the specifier obtains the requirements without extracting them from the problem description, organise them in terms of related task can be vital.

On the other hand, we had to start modelling our problem from scratch with a problem description, and did not have any specific or well defined set of requirements. So, as a first step in specification writing, we had to determine the requirements of the problem. To do this, we broke down the problem into smaller sub-problems (based on scenarios) to analyse and to determine the sub-problem requirements. Isolating all the sub-problems from the problem description is useful to obtain all the related requirements together.

The way we generated the requirements was based on the problem analysis, and we followed some defined steps to produce requirements from the problem description:

1. At first, we tried to find the defining set of scenarios from the problem description, and considered them as sub-problems. A sub-problem can be the details of any operation (e.g., the process of increasing speed), or a violation (e.g., speed violation) etc.
2. We tried to understand what the sub-problem is all about, in order to get the possible requirements of that sub-problem.
3. We tried to find all the events that occur within the sub-problem.
4. We tried to isolate the components that participate in an event, and

the interaction between the components.

5. We elicited the requirements of the sub-problem based on the associated event(s). In most cases, each requirement consists of three pieces of information: (a) the current situation of the system, (b) the possible next action(s) based on the current situation and (c) the result(s) of each such action.

Though we tried to write requirements based on sub-problems, still our requirements lacked any modularisation, and the requirements can be still ambiguous to other people. More in depth analysis of the sub-problems to create (much) better isolation between them, and, more analysis of language construction and strong documentation can potentially overcome these problems.

Refining or improving requirements in order to make them more correct and consistent in terms of properties is very important. In this thesis, we could not focus that much on improving requirements, as we had to prove required properties of the system by hand, and as we could not attempt to prove all the required properties, finding inconsistencies in requirements was a hard task. Some automated tool to discharge proofs of the required properties will be very effective for a large model; from the interactive proving with the tool and attempting to discharge all the proof obligations, we can detect missing and wrong requirements, this will be very interesting future work.

7.1.1.2 Converting requirements into a high level specification

In our modelling stage, we did not use any tool, and had to write the specifications by hand. Therefore, we had to be very focused to maintain proper syntax and semantics of Deontic Logic, so that (in later stage) we can effectively prove the desired properties of the system. Though requirements are written in natural language, converting them to specifications is a difficult task. As our modelling approach was Deontic Logic, we had to address several key issues while writing specifications :

1. Assigning the right normative property to an action, i.e., defining whether an action (occurring in a requirement) is permitted or forbidden was a major task. We know that a permitted action can be executed in every possible way and should lead the system to a good successor state (the resulting expected behaviours), while execution of a forbidden action will take the system to a bad successor state (the resulting unexpected behaviours). So while defining specification, we had to be aware of assigning the normative modalities to actions in order to isolate good and bad behaviours.

2. From the description of our requirements, we had to distinguish the events of single and parallel action(s) executions and their effects. We know that when an action is not affected by another parallel action, the behaviour of the system when the relevant action is executed as a single action and in parallel with other action(s) are the same. But, if execution of another action affects the relevant action, (like, an action neutralizes or does something negative of the relevant action) then the resulting behaviour of the single and parallel action(s) execution could be different. So, defining the effects of single and parallel execution of action(s) in the specification was a critical task.

3. Violation and recovery are major issues for a fault tolerant system's specification. While writing the specification of a violation scenario, we tried to capture the possible (single and multi step) recovery mechanisms, and, distinguished them in terms of different priorities of violation and levels of obligation (on recovery action(s)). It is also important to isolate single and multiple violations on the basis of the requirement, as the required recovery mechanisms for a single violation V_1 and multiple violations V_1 and V_2 may be different. Moreover, in case of multiple violations, each violation may require a different recovery action, or all violations may require the same single recovery action. So, it was our prime concern to obtain the violation(s) and the corresponding recovery mechanism(s), to convert into the specification.

4. The behaviours that an agent exhibits while it is isolated from the environment, compared to when it interacts with the environment, can be different. Moreover, the environment can create a violation for the agent. So, from the requirements, we tried to capture the effects of the environment on

the agent, and then, in the specification, we draw a clear distinction on the agent's behaviour based on when it interacts with the environment, and when it does not.

5. During specification, we ourselves had to analyse each requirement to extract the essence, so that we could write axioms with the correct syntax incorporating proper connectives and quantifiers. Moreover, we had to incorporate temporal operators in our axioms in order to reason about states and the execution paths. We know that, writing axioms with wrong connectors, quantifiers or temporal operators can make it harder to prove desired system properties. Subsequently, this will also create problem in improving requirements.

So, we can summarise by saying that writing specifications is a systematic task with some prerequisite steps and prerequisite knowledge, and requires an extensive analysis on the problem description and the requirements. Moreover, while writing a specification in Deontic Logic, a modeller must be always aware of some key aspects, concerning violation and recovery, the environment etc. Therefore, converting natural language to high level specifications is a difficult task. In future, we also want to work on improving specifications, to create stronger and more solid through improving the requirements, making them more specific and concrete.

7.1.2 How easy is it to define and prove properties?

Every system must have some obvious and expected properties that can be proved from the system specifications. In this thesis we had to find some expected properties of the system from the problem description, so that we could prove them and gain confidence that our model is correct in terms of the requirements captured in the specification. Finding and defining properties that a system must satisfy relies on the proper analysis of the problem description, and requires considerable effort. Moreover, discharging all the required properties related with each sub-system is important in building an effective model in terms of improving requirements.

In this thesis, we had to prove our desired properties by hand, and undoubtedly proving properties of a real world problem by hand is a difficult and error-prone task. Moreover, this effort is time consuming and finding errors in a hand written proof is hard. To produce an efficient proof in this logic, a specifier must have a solid grasp of several topics in logic: propositional logic, first order logic, modal logic etc., along with deontic and temporal operators. Therefore, an automatic or semi-automatic tool for proving system properties would be very effective. These software tools guide a modeller during the task of proving system properties; sometimes they are fully automated, and sometimes they are automated for simple cases, but require human intervention for more difficult ones. Another interesting outcome of using tools is to obtain counterexamples in the case of unsuccessful proofs. These counterexamples can be used to investigate which possible runs of the system (to be built) can be dangerous and must be taken into account to improve the requirements.

The current theorem provers can be useful to prove properties in this logical formalism, but this requires a significant effort to encode the logic in the theorem prover, and this can be a very interesting future exercise. Moreover, Castro described a tableaux deductive system in [Cas09] and conjectured that this system would be useful for automatic theorem proving. An automatic or semi-automatic tool based on the tableaux system will be very effective and helpful for modelling systems, and will encourage modellers to use this logic in practice.

7.1.3 Possible recovery scenarios from a violation state

Having a solid understanding of the possible scenarios arising from a violation state is useful in term of analysing the problem requirement, and high level modelling of a fault tolerant system. Based on our work, we have come to the conclusion that there are four possible scenarios that can arise from a violation.

1. If the obligatory action (for recovery purposes) is executed successfully in one step after a violation arises, then after the step the system recovers from

that violation state and goes to a safe state, perhaps fully recovering from a fault.

2. But, in some cases, the obligated recovery mechanism has to be performed in multiple steps to recover from the violation. In those cases, after executing the obligated actions in one or more steps the system goes from a violation state to a "better" state. The recovery transaction constitutes several actions until the system goes into a safe state and, finally, in a future state the system recovers (partly or wholly) from the violation(s).

3. In some situations it can also happen that, after executing the obligated recovery action from a violation state, the system remains in the same level of violation. These situations occur when the action is not performed successfully or one component depends on or is interrupted by another component. Even if the component successfully executes the recovery action, it (the action) will not remain successful if the system does not show the expected behaviour, or the expected outcome is altered for the environmental intervention; will result the system to remain in the same level of violation.

4. The last possibility arises when a system does not execute the obligated action at all from a violation state. In that case, the system can go to a state where the level of violation in the next state is (much) higher than in the previous state.

Now, how can we use these recovery scenarios to model and analyse a fault tolerant system? To model the recovery scenarios discussed above, we have to apply different levels of obligation during the recovery process. If the system recovers from any violation after executing a recovery action, the corresponding obligation has to be removed from the relevant component. If the component executes the required recovery action, (or if it executes any action other than the required recovery action), and after execution the system moves to a (much) worse state; then in the next state, we have to assign a (much) higher level of obligation to the recovery action. Contrarily, If the execution of a recovery action takes the system to a better state, and the system has not fully recovered from the violation, then in the next state, we have to assign a much lower level of obligation to the recovery action. So, the process is: as long as the system's current condition is getting worse in each transition, in specification, we have to keep incrementing the obligation level, and when the opposite happens, we have to decrement the obligation level. If the level of violation remains same after execution of an action from the violation state, then we have to keep the same level of obligation in the next state.

In a general way, we can classify the states of a fault tolerant system as: normal, degraded and critical. Reasoning about a fault tolerant system's current state, i.e., when the system is in safe, degraded or critical condition is very crucial. During execution, a system shifts between these states, and, by analysing the various recovery scenarios we can reason about the transition between good and bad states, and the current condition of the system. Sometimes, these scenarios are also useful to analyse the environmental effect on the system. Moreover, in most cases, obtaining all the possible recovery mechanisms from a system requirement may not be possible; therefore, these scenarios can help a modeller to reason about all the paths starting from a violation state.

7.1.4 Modelling the interaction between the agent and the environment

Every component must execute obligated action(s) to recover from a violation, and in the next or some future state, the component reaches a safe situation. But, from our modelling experience, we have come to realise that, in case of environmental effects or interaction between components, the relevant component might recover from a violation without even executing any recovery action.

1. This situation might introduce a confusion while writing a specification. Questions may arise: should we always consider a recovery action as an obligation? We have to say "yes". If the violation depends on two or more components, then every component must have its own obligated action(s) for

recovery, and one component should not rely on another one, because it cannot be guaranteed that the other component's recovery action(s) will recover the relevant component from a violation, and the reliance of one component on the other to bring the system to a safe state may be unwarranted (e.g., the other component may violate it's own obligation).

2. What happens to the obligation (imposed on a component) when the violation is fixed by another component? Generally, when an obligation imposed on a component is not fulfilled, the obligation (on the relevant component) should be retained or a new obligation should be introduced. But, in some cases of environmental effect, even if the relevant component cannot fulfil the assigned obligation (in order to recover from the violation), the system might withdraw that obligation without assigning a new one.

The above two issues are important when writing specifications of a fault tolerant system that interact with their environment. By analysing the requirement, we can capture the interaction between the environment and the system, but, a requirement does not explicitly express the assignment and removal of an obligation, and the change of violation, considering the environment. Moreover, to effectively prove a desired system property which focuses on an intervention of the environment or interaction between components, assigning and withdrawing normative modalities (in the specification) in the correct way (based on the environmental effect) is very essential, and generally a complicated issue.

Moreover, while modelling the interaction between the agent and the environment, it is also important to determine how and to what extent the agent's action is interrupted by the other action(s), as parallel execution of action(s) might alter the expected behaviour (of the agent).

We know that, the effects of the environment on the agent is an area of common interest in fault tolerance community. As Deontic Logic is equipped enough to model fault tolerant system, by using proper deontic operators (based on the situation) and changing the level of violation, we can effectively model the environmental effect on the agent.

7.1.5 How can we model a scenario with multiple violations?

We know that a system can be in multiple violations in a given system state, and executing a single recovery action might not recover the system from all the violations at the same time. Moreover, it can also happen that a system cannot execute multiple parallel recovery actions from a violation state. In this sort of case, we have to prioritize the violations for recovery purposes.

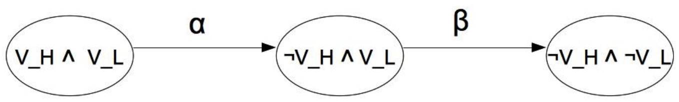

Figure 7.1: Multiple violations and different recovery action

The figure 7.1 above shows two violations V_H (higher priority) and V_L (lower priority) being true in a state. There are two recovery actions, where action α and β recover the system from violation V_H and V_L, respectively. Here we are assuming that the system cannot execute parallel recovery actions. So the system executes the recovery actions α and β in sequence, in order to recover from the violations V_H and V_L, respectively.

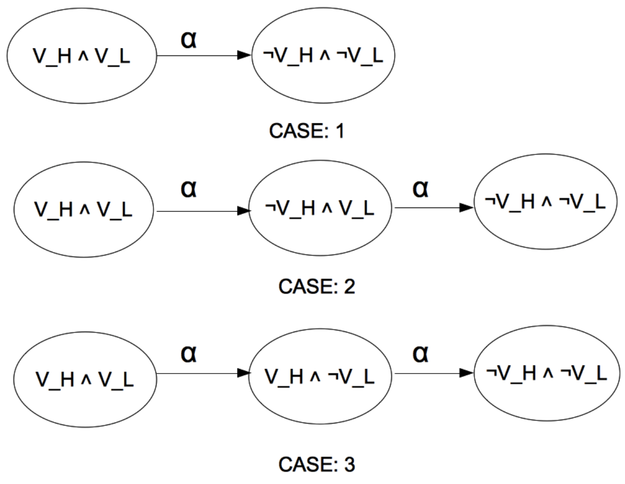

Figure 7.2: Multiple violations and single recovery action

Figure 7.2 shows three cases where a single recovery action α recovers the system from two violations V_H and V_L. In the first case the system recovers from both violations at the same time. The second case shows that the system recovers from the higher priority violation first, as is expected; and in the last case, in an unintended way the system recovers from the lower priority violation, before it recovers from the higher priority violation.

So, we can say that, if a system is in multiple violations, different violations have different recovery actions and the system can execute them in parallel, then prioritization of violations is not necessary. But, if the system cannot execute different recovery actions in parallel to recover from multiple violations, or, a single recovery action cannot recover the system from all the violations at the same time, then prioritization of violations is important.

Now, the obvious question is: how to specify multiple violations based on prioritization? By analysing a system requirement, we can effectively determine the level of severity of each violation, and the individual or parallel recovery process. Therefore, we can determine the priority level of each violation. Now, if the system can execute single action from a violation state, and the recovery action for each violation is different, then the specification should reflect that, the system at first executes the recovery action for the higher priority violation; when it recovers from that violation, it executes the recovery action for the lower priority violation. On the other hand, if all violations have the same recovery action, then the specification should reflect two things: (1) in case of expected execution, the system at first recovers from the higher priority violation, then it recovers lower priority violation, and (2) if the execution is unexpected, then the system might recover from the lower priority violation before it recovers from the higher priority violation.

Finally we can say that, the discussion above is useful to analyse the scenarios of multiple violations, and to decide when and how to prioritize them based on single and parallel recovery action(s), and to convert the requirements of multiple violations to the specifications.

7.1.6 Comparing different sub-problems

While converting requirements to the high level specifications, we might generate many redundant specifications. By comparing and matching different sub-problems while specifying a system, we can significantly reduce the redundant specifications. So, while writing specifications (of a new sub-problem) we might ask ourselves: how can we compare and match the new sub-problem with an earlier one (which is already being specified)?

A possible solution is to try to match the possible actions, that can happen from possible states of these sub-problems. Usually while modelling different sub-problems, we will see that the execution paths of two sub-problems are different, i.e., the states and the actions (to be executed) cannot be matched. But sometimes by analysing the requirement, we can directly match the possible, permitted and forbidden actions of two sub-problems, even though they represent two different situations (in the problem description); and in both cases, the execution of each possible action from a state takes the system to the same next state. In this sort of case, in order to avoid redundant specifications, we can specify one sub-problem and can encode the new sub-problem in the earlier one in a way that the same specifications can represent both sub-problems.

The benefit of comparing sub-problems and minimising redundant specification is, it makes the proving process (of desired system properties) much easier, as we can achieve better control and focus on the specifications. Moreover, having concrete specifications can help us to avoid confusions during system analysis. Also, modification of specifications becomes easier and reduces the chance of introducing inconsistency.

7.1.7 The first order approach versus the propositional approach

The First Order Deontic Logic approach introduced in [CM10] incorporates standard quantifiers of First Order Logic, and has similar algebraic operators for actions as for the Propositional Deontic Logic described in [Cas09]. In the Propositional Deontic Logic approach, we can define very basic actions and propositions and these actions cannot take a range of values as parameters. But in real world situations, an action has to take some value as a parameter and sometimes an infinite range of values. To formalize this sort of cases, in particular when we need to reason about infinite domain or complex data structures, we have to use the First Order Deontic Logic. This logic approach allows us to capture the usual notion of commands and procedures of programming languages.

In the literature, we have seen several examples of specifications and proofs in the Deontic Propositional Logic. But, no proof using the First Order approach is presented in the literature based on any case study. As a result, to structure proofs in the First Order approach, we had to follow the proofs of the Propositional approach along with incorporating quantifier over variables. In addition to this, we applied the generalisation and elimination rules (over quantifiers) of first order logic in our proving process. So based on our experience we can say that, a modeller can study the proofs of the Propositional Deontic Logic presented in the literature, in order to effectively produce future proofs in the First Order Deontic Logic approach, and undoubtedly an automatic or semi-automatic tool will make the proving process (of desired system properties) much easier.

7.2 Summary

In this chapter we have presented a discussion on: how we obtained the requirements from the problem description, what was our approach to convert the requirements to the high level specification, how did we define and prove different desired properties of our system, and how can we improve all these process in future. We have also discussed on the modelling approach of the environmental effect on the agent, different levels of violation and single and multi step recovery approaches. Here we also demonstrated, when and how can we achieve violation prioritisation, and how can we reduce redundant specifications by comparing sub-problems. The motivation of discussing all these engineering lessons is to assist future modelling of fault tolerance systems.

Chapter 8

Conclusion and Future Work

8.1 Contribution

This thesis presents a practical case study of an existing fault tolerance problem in Deontic Logic. Our main intention in this thesis is to test the strengths of Deontic Logic for modelling fault tolerant systems, i.e., how it deals with different kinds of violations and a variety of recovery mechanisms. This thesis is made up of three parts.

The first part of this thesis is the background knowledge description, which includes a discussion of different fault tolerance mechanisms in the low level implementation phase and the high level design phase. Here we have seen that existing work mostly focuses on implementation level fault tolerance mechanisms through the use of replication and voting. We have also presented some high level fault tolerance mechanisms along with some modelling examples. This part also presents an overview of Deontic Action Logic and mostly focuses on Propositional and First Order versions of Deontic Logic presented in [Cas09] and [CM10]. This part also covers a brief description of our case study problem. We have chosen a real world example of a moving vehicle and captured the scenarios based on the its speed and distance with respect to the immediate front and rear vehicles. We have applied Deontic Logic approach to model scenarios exhibiting faults from this example, as we have strong evidence that this logic has strong support for designing fault tolerance systems at an abstract design level.

The second part of the thesis focuses on the usage of Deontic Logic in a practical example of a fault tolerant system. The main contributions of this thesis is described in this section. Here we have presented two models. One is in Propositional Deontic Logic, which is significantly simpler than the model in First Order Deontic Logic. The reason for using the First Order version of the logic after the Propositional version is that we could not express some practical aspects in the latter language. In our first model in Propositional Deontic Logic, we omitted the effects of the environment on the agent, while in the second model we presented how the environment presents different situations for the agent to deal with. In these models we have shown how violations occur at different stages of a system's behaviour and how recovery actions take the system from a violation state to a safe state. In the first model, we mostly focused on one step recovery, but in the second model we exhibited multi step recovery mechanisms. Here we have also modelled situations where the agent is in multiple violations in a system state and the related sequential recovery mechanism. At the end of this section, we have presented some expected system properties along with their proofs.

The last part of this thesis focuses on the engineering lessons and ideas that we have gained through this work. During the modelling phase, we have faced many obstacles in designing fault tolerance mechanisms and applied different ideas to overcome them. In this section we have discussed those approaches that we followed to deal with these situations. The motivation of this part is to help and assist in abstract fault tolerant system modelling.

8.2 Future Work

As we said before, we have performed our fault tolerance modelling in Deontic Logic and found some disadvantages in this logic that became obstacles during specification of the system. The existing Deontic Logic deals with atomic actions and we can model and reason about situations that occur after one time instant. But for modelling some practical scenarios, we might need actions or transitions that execute over a time period and the current logic cannot model this sort of action. So it is important to investigate if the current logic can be extended in some ways to define transactions that have some duration.

At this stage, Deontic Logic is not equipped to deal with faults related to strong timing constraints of real time systems. Real Time Logic [Lan98] has strong mechanisms for modelling real time systems as it can define an action with duration that associates an action with a Request Time, an Activation Time and a Termination Time. But this logic has no special construct to deal with faults. It would be very interesting to see if it is possible to relate Deontic Logic with Real Time Logic to model fault tolerance of real time systems.

In this thesis we have not used any automated tool for specifying the system in Deontic Logic and we proved properties by hand, not an easy task. Automated tools and techniques for defining specifications and proving system properties will make the modelling easier and faster. Extensive research is necessary to develop efficient tools to make the logic more easily applicable and acceptable.

Appendix A

Properties and proofs

Property 2 If the host vehicle is in a minimum collision distance violation, then, after executing any action, the host vehicle either recovers from the violation or stays in that violation or has a collision with the front opponent vehicle.

1.	$v_h.v_4 \ \wedge\ O^4(v_h.stop) \ \longrightarrow\ [v_h.stop]\neg v_h.v_4$	**QVD10**
2.	$v_h.v_4 \ \wedge\ O^4(v_h.stop) \ \longrightarrow\ [v_h.stop \ \sqcap\ v_{of}.stop]\neg v_h.v_4$	ML, 1
3.	$v_h.v_4 \ \longrightarrow\ \exists i \in Speed :$ $[\overline{v_h.stop} \ \sqcap\ v_{of}.incspeed(i)]EN(collision(v_h, v_{of}))$	**QVD15**
4.	$v_h.v_4 \ \longrightarrow\ [\overline{v_h.stop} \ \sqcap\ v_{of}.incspeed(s)]EN(collision(v_h, v_{of}))$	FOL, 3
5.	$[v_h.stop]v_h.stopped$	**QSAS5**
6.	$[v_{of}.stop]v_{of}.stopped$	**QSAS5**
7.	$v_h.v_4 \ \wedge\ O^4(v_h.stop) \ \longrightarrow$ $[v_h.stop \ \sqcap\ v_{of}.stop](\neg v_h.v_4 \ \wedge\ v_h.stopped \ \wedge\ v_{of}.stopped)$	**T5**, 2, 5, 6
8.	$\forall i \in Speed : v_{of}.stopped \ \longrightarrow\ \neg v_{of}.speed_{inc}(i)$	**QSAS2**
9.	$v_{of}.stopped \ \longrightarrow\ \neg v_{of}.speed_{inc}(s)$	FOL, 8
10.	$\forall i \in Speed : [\overline{v_{of}.incspeed(i)}](\neg v_{of}.speed_{inc}(i))$	**QSAS15**
11.	$[\overline{v_{of}.incspeed(s)}](\neg v_{of}.speed_{inc}(s))$	FOL, 10
12.	$[v_{of}.stop]\top \ \longrightarrow\ [v_{of}.incspeed(s)]\bot$	ML 6, 9, 11
13.	$v_h.v_4 \ \wedge\ O^4(v_h.stop) \ \longrightarrow$ $[v_h.stop \sqcap \overline{v_{of}.incspeed(s)}](\neg v_h.v_4 \wedge v_h.stopped \wedge v_{of}.stopped)$	PL, 7, 12

14.	$v_h.v_4 \wedge O^4(v_h.stop) \longrightarrow [v_h.stop \sqcap \overline{v_{of}.incspeed(s)}](\neg v_h.v_4)$	PL, 13
15.	$v_h.v_4 \wedge O^4(v_h.stop) \longrightarrow$ $[v_h.U \sqcap v_{of}.U](\neg v_h.v_4 \vee EN(collision(v_h, v_{of})))$	BA, PL, 4, 14
16.	$v_h.v_4 \wedge O^4(v_h.stop) \longrightarrow [\overline{v_h.stop}](v_h.v_4 \wedge O^5(v_h.stop))$	**QVD13**
17.	$v_h.v_4 \wedge O^4(v_h.stop) \longrightarrow [\overline{v_h.stop} \sqcap v_{of}.U]v_h.v_4$	PL, A4, 16
18.	$v_h.v_4 \wedge O^4(v_h.stop) \longrightarrow$ $[v_h.U \sqcap v_{of}.U](v_h.v_4 \vee \neg v_h.v_4 \vee EN(collision(v_h, v_{of})))$	PL, 15, 17

Property 3 After executing any action, the host vehicle either has a safe speed or it violates the maximum or minimum speed limitations.

1.	$\neg v_h.exceed_{maxspeed} \longrightarrow [v_h.contspeed]AN(\neg v_h.exceed_{maxspeed})$	**QSAS15**
2.	$\exists i \in Speed:$ $\neg v_h.exceed_{maxspeed} \longrightarrow [v_h.incspeed(i)]EN(v_h.exceed_{maxspeed})$	**QSAS21**
3.	$\neg v_h.exceed_{maxspeed} \longrightarrow [v_h.incspeed(s_1)]EN(v_h.exceed_{maxspeed})$	FOL, 2
4.	$\forall i \in Speed: v_h.speed_{inc}(i) \longrightarrow \neg v_h.speed_{cont}$	**QSAS2**
5.	$v_h.speed_{inc}(s_1) \longrightarrow \neg v_h.speed_{cont}$	FOL, 4
6.	$[v_h.contspeed]v_h.speed_{cont} \wedge [\overline{v_h.contspeed}]\neg v_h.speed_{cont}$	**QSAS10**
7.	$\forall i \in Speed: [v_h.incspeed(i)]v_h.speed_{inc}(i)$	**QSAS16**
8.	$[v_h.incspeed(s_1)]v_h.speed_{inc}(s_1)$	FOL, 7
9.	$Done(v_h.incspeed(s_1)) \longrightarrow \neg Done(v_h.contspeed)$	ML, DTL, 5, 6, 8
10.	$\neg v_h.exceed_{maxspeed} \longrightarrow [v_h.incspeed(s_1)](Done(v_h.incspeed(s_1))$ $\wedge EN(v_h.exceed_{maxspeed}))$	ML, 3, 9
11.	$\neg v_h.exceed_{maxspeed} \longrightarrow [v_h.incspeed(s_1)](\neg Done(v_h.contspeed)$ $\wedge EN(v_h.exceed_{maxspeed}))$	PL, 9, 10
12.	$\neg v_h.exceed_{maxspeed} \longrightarrow ([\overline{v_h.contspeed}]EN(v_h.exceed_{maxspeed})$	ML, 11
13.	$\neg v_h.exceed_{maxspeed} \longrightarrow$ $[U]AN(\neg v_h.exceed_{maxspeed} \vee v_h.exceed_{maxspeed})$	BA, PL, 1, 12
14.	$\neg v_h.speedlowerthan_{minspeed} \longrightarrow$ $[v_h.contspeed]AN(\neg v_h.speedlowerthan_{minspeed})$	**QSAS15**
15.	$\exists i \in Speed: \neg v_h.speedlowerthan_{minspeed} \longrightarrow$ $[v_h.decspeed(i)]EN(v_h.speedlowerthan_{minspeed})$	**QSAS27**

16.	$\neg v_h.speedlowerthan_{minspeed} \longrightarrow [v_h.decspeed(s_2)]EN(v_h.speedlowerthan_{minspeed})$	FOL, 15
17.	$\forall i \in Speed : [v_h.decspeed(i)]v_h.speed_{dec}(i)$	**QSAS22**
18.	$[v_h.decspeed(s_2)]v_h.speed_{dec}(s_2)$	FOL, 17
19.	$\forall i \in Speed : v_h.speed_{dec}(i) \longrightarrow \neg v_h.speed_{cont}$	**QSAS2**
20.	$v_h.speed_{dec}(s_2) \longrightarrow \neg v_h.speed_{cont}$	FOL, 19
21.	$Done(v_h.decspeed(s_2)) \longrightarrow \neg Done(v_h.contspeed)$	ML, DTL, 6, 18
22.	$\neg v_h.speedlowerthan_{minspeed} \longrightarrow [v_h.decspeed(s_2)](\neg Done(v_h.contspeed) \wedge EN(v_h.speedlowerthan_{minspeed}))$	ML, PL, 16, 21
23.	$\neg v_h.speedlowerthan_{minspeed} \longrightarrow [\overline{v_h.contspeed}]EN(v_h.speedlowerthan_{minspeed})$	ML, 22
24.	$\neg v_h.speedlowerthan_{minspeed} \longrightarrow [U]AN(\neg v_h.speedlowerthan_{minspeed} \vee v_h.speedlowerthan_{minspeed})$	BA, PL, 14, 23
25.	$\neg v_h.exceed_{maxspeed} \wedge \neg v_h.speedlowerthan_{minspeed} \longrightarrow [U]AN((\neg v_h.speedlowerthan_{minspeed} \vee v_h.speedlowerthan_{minspeed}) \wedge (\neg v_h.exceed_{maxspeed} \vee v_h.exceed_{maxspeed}))$	PL, 13, 24
26.	$\neg v_h.exceed_{maxspeed} \wedge \neg v_h.speedlowerthan_{minspeed} \longrightarrow [U]AN((\neg v_h.speedlowerthan_{minspeed} \wedge \neg v_h.exceed_{maxspeed}) \vee (\neg v_h.speedlowerthan_{minspeed} \wedge v_h.exceed_{maxspeed}) \vee (v_h.speedlowerthan_{minspeed} \wedge \neg v_h.exceed_{maxspeed}) \vee (v_h.speedlowerthan_{minspeed} \wedge v_h.exceed_{maxspeed}))$	PL, 25
27.	Assmp: $[v_h.speedlowerthan_{minspeed} \wedge v_h.exceed_{maxspeed} \to F]$ $\forall i \in Speed : v_h.speed_{safe}(i) \longrightarrow [U]AN(v_h.speed_{safe}(i) \vee (\neg v_h.speedlowerthan_{minspeed} \wedge v_h.exceed_{maxspeed}) \vee (v_h.speedlowerthan_{minspeed} \wedge \neg v_h.exceed_{maxspeed}))$	**QSAS9**, PL, 26
28.	$\forall i \in Speed : v_h.speed_{safe}(i) \longrightarrow [U]AN(v_h.speed_{safe}(i) \vee v_h.exceed_{maxspeed} \vee v_h.speedlowerthan_{minspeed})$	PL, 27

Property 4 If the host vehicle increases speed from a state where it is in a minimum safety distance violation with respect to the front opponent vehicle, then in the next state it is not permitted to increase speed or is obliged to stop.

1.	$\forall i,j \in Speed : v_h.speed_{curnt}(i) \ \wedge \ v_{of}.speed_{curnt}(j) \ \wedge$ $v_h.exceed(minsafedistfront(i,j)) \ \longrightarrow \ \exists k \in Speed :$ $([v_h.incspeed(k)]EN(v_h.exceed(mincoldistfront(i+k,j))))$	**QSAS36**
2.	$v_h.speed_{curnt}(s_1) \ \wedge \ v_{of}.speed_{curnt}(s_2) \ \wedge$ $v_h.exceed(minsafedistfront(s_1,s_2)) \ \longrightarrow$ $[v_h.incspeed(c_1)]EN(v_h.exceed(mincoldistfront(s_1+c_1,s_2)))$	FOL, 1
3.	$v_h.speed_{curnt}(s_1) \ \wedge \ v_{of}.speed_{curnt}(s_2) \ \wedge$ $v_h.exceed(minsafedistfront(s_1,s_2)) \ \longrightarrow \ [v_h.incspeed(c_1) \sqcap \overline{v_{of}.incspeed(c_2)}]$ $EN(v_h.exceed(mincoldistfront(s_1+c_1,s_2)))$	ML, 2
4.	$\forall i,j \in Speed : (curntspeed_{h(of)} = i) \ \longrightarrow$ $[v_{h(of)}.incspeed(j)](curntspeed_{h(of)} = i+j) \ \wedge$ $[\overline{v_{h(of)}.incspeed(j)}](curntspeed_{h(of)} = i)$	**QSAS17**
5.	$((curntspeed_h = s_1) \ \longrightarrow$ $[v_h.incspeed(c_1)](curntspeed_h = s_1 + c_1)) \ \wedge$ $((curntspeed_{of} = s_2) \ \longrightarrow$ $[\overline{v_{of}.incspeed(c_2)}](curntspeed_{of} = s_2))$	FOL, 4
6.	$v_h.speed_{curnt}(s_1) \ \wedge \ v_{of}.speed_{curnt}(s_2) \ \wedge$ $v_h.exceed(minsafedistfront(s_1,s_2)) \ \longrightarrow \ [v_h.incspeed(c_1) \sqcap \overline{v_{of}.incspeed(c_2)}]$ $EN((curntspeed_h = s_1 + c_1) \ \wedge \ (curntspeed_{of} = s_2) \ \wedge$ $v_h.exceed(mincoldistfront(s_1+c_1,s_2)))$	PL, 3, 5
7.	$\forall i,j \in Speed :$ $(v_h.exceed(minsafedistfront(i,j)) \ \longleftrightarrow \ v_h.v_3)$ $\wedge \ (v_h.exceed(mincoldistfront(i,j)) \ \longleftrightarrow \ v_h.v_4)$	**QVD5**
8.	$\forall i,j \in Speed : v_h.speed_{curnt}(i) \ \wedge \ v_{of}.speed_{curnt}(j) \ \wedge$ $(v_h.exceed(minsafedistfront(i,j)) \ \longleftrightarrow \ v_h.v_3)$ $\wedge \ (v_h.exceed(mincoldistfront(i,j)) \ \longleftrightarrow \ v_h.v_4)$	PL, 7
9.	$v_h.speed_{curnt}(s_1) \ \wedge \ v_{of}.speed_{curnt}(s_2) \ \wedge$ $(v_h.exceed(minsafedistfront(s_1,s_2)) \ \longleftrightarrow \ v_h.v_3)$ $\wedge \ (v_h.exceed(mincoldistfront(s_1+c_1,s_2)) \ \longleftrightarrow \ v_h.v_4)$	FOL, 8
10.	$v_h.v_3 \ \longrightarrow \ [v_h.incspeed(c_1) \ \sqcap \ \overline{v_{of}.incspeed(c_2)}]EN(v_h.v_4)$	PL, 6, 9
11.	$v_h.v_3 \ \longrightarrow$ $[v_h.incspeed(c_1) \ \sqcap \ \overline{v_{of}.incspeed(c_2)}]EN(\neg P^4_w(\overline{v_h.stop}))$	PL, 10, **QVD9**

12.	$\forall i \in Speed :$ $v_h.v_3 \wedge (v_h v_{of}.posrelspeed \vee v_h v_{of}.negrelspeed) \longrightarrow \neg P^2(v_h.incspeed(i))$	**QVD7**
13.	$v_h.v_3 \wedge (v_h v_{of}.negrelspeed \vee v_h v_{of}.posrelspeed) \longrightarrow \neg P^2(v_h.incspeed(c_1))$	FOL, 12
14.	$v_h.v_3 \wedge (v_h v_{of}.negrelspeed \vee v_h v_{of}.posrelspeed) \longrightarrow [v_h.incspeed(c_1)]AN(v_h.v_3)$	DPL, 13, **QVD12**
15.	$v_h.v_3 \wedge (v_h v_{of}.negrelspeed \vee v_h v_{of}.posrelspeed) \longrightarrow [v_h.incspeed(c_1) \sqcap \overline{v_{of}.incspeed(c_2)}](AN(v_h.v_3) \wedge EN(\neg P^4_w(\overline{v_h.stop})))$	ML, 11, 14
16.	$v_h.v_3 \wedge (v_h v_{of}.negrelspeed \vee v_h v_{of}.posrelspeed) \longrightarrow \exists i, j, \forall k : [v_h.incspeed(i) \sqcap \overline{v_{of}.incspeed(j)}] (AN(\neg P^2(v_h.incspeed(k))) \wedge EN(\neg P^4_w(\overline{v_h.stop})))$	FOL, **QVD7**, **QVD8**

Property 5 The host vehicle cannot change the current situation after executing a continue speed action unless the host vehicle is in a minimum safety distance violation with negative relative speed.

1.	$(\neg v_h.exceed_{maxspeed} \wedge \neg v_h.speedlowerthan_{minspeed}) \longrightarrow [v_h.contspeed](\neg v_h.exceed_{maxspeed} \wedge \neg v_h.speedlowerthan_{minspeed})$	**QSAS15**
2.	$\forall i \in Speed :$ $\neg v_h.exceed_{maxspeed} \wedge \neg v_h.speedlowerthan_{minspeed} \longrightarrow v_h.speed_{safe}(i)$	**QSAS9**
3.	$\neg v_h.exceed_{maxspeed} \wedge \neg v_h.speedlowerthan_{minspeed} \longrightarrow v_h.speed_{safe}(s_1)$	FOL
4.	$v_h.speed_{safe}(s_1) \longrightarrow [v_h.contspeed]v_h.speed_{safe}(s_1)$	PL , 1, 3
5.	$\forall i \in Speed :$ $v_h.v_1 \longrightarrow [\overline{v_h.decspeed(i)}](v_h.v_1 \wedge O^3(v_h.decspeed(i)))$	**QVS7**
6.	$v_h.v_1 \longrightarrow [\overline{v_h.decspeed(s_2)}]v_h.v_1$	PL, FOL, 5
7.	$[v_h.contspeed]v_h.speed_{cont} \wedge [v_h.decspeed(s_2)]v_h.speed_{dec}(s_2)$	**QSAS10**, FOL, **QSAS22**
8.	$\forall i \in Speed : (v_h.speed_{dec}(i) \longleftrightarrow \neg v_h.speed_{cont})$	**QSAS2**
9.	$\neg v_h.speed_{dec}(s_2) \longrightarrow v_h.speed_{cont}$	FOL, 8
10.	$\neg Done(v_h.decspeed(s_2)) \longrightarrow Done(v_h.contspeed)$	DTL, 7, 9

11.	$v_h.v_1 \longrightarrow [\overline{v_h.decspeed(s_2)}](\neg Done(v_h.decspeed(s_2)) \wedge v_h.v_1)$	ML, DTL, 6
12.	$v_h.v_1 \longrightarrow [\overline{v_h.decspeed(s_2)}](Done(v_h.contspeed) \wedge v_h.v_1)$	PL, 10, 11
13.	$v_h.v_1 \longrightarrow [v_h.contspeed](Done(v_h.contspeed) \wedge v_h.v_1)$	ML, DTL, 12
14.	$v_h.v_4 \longrightarrow [\overline{v_h.stop}](v_h.v_4 \wedge O^5(v_h.stop))$	**QVD13**
15.	$\neg v_h.stopped \longrightarrow v_h.speed_{cont}$	**QSAS2**
16.	$\neg Done(v_h.stop) \longrightarrow Done(v_h.contspeed)$	DTL, **QSAS5**, 7, 15
17.	$v_h.v_4 \longrightarrow [v_h.contspeed]v_h.v_4$	ML, DTL, 14, 16
18.	$(v_h.speed_{safe}(s) \vee v_h.v_1 \vee v_h.v_4) \longrightarrow [v_h.contspeed](v_h.speed_{safe}(s) \vee v_h.v_1 \vee v_h.v_4)$	PL, 4, 13, 17
19.	$v_h.v_3 \wedge v_hv_{of}.posrelspeed \longrightarrow \neg P^2(v_h.contspeed)$	**QVD8**
20.	$\forall i \in Speed : v_h.v_3 \wedge v_hv_{of}.posrelspeed \longrightarrow [\overline{v_h.decspeed(i)}](v_h.v_3 \wedge O^3(v_h.decspeed(i))$	**QVD12**
21.	$v_h.v_3 \wedge v_hv_{of}.posrelspeed \longrightarrow [\overline{v_h.decspeed(s_2)}](v_h.v_3 \wedge O^3(v_h.decspeed(s_2))$	FOL
22.	$v_h.v_3 \wedge v_hv_{of}.posrelspeed \longrightarrow [v_h.contspeed](v_h.v_3 \wedge O^3(v_h.decspeed(s_2))$	ML, 16, 21
23.	$v_h.v_3 \wedge v_hv_{of}.negrelspeed \longrightarrow [v_h.contspeed]EN(v_h.v_3)$	**QVD14**
24.	$\forall i \in Speed : ((v_h.speed_{safe}(i) \vee v_h.v_1 \vee v_h.v_4 \vee (v_h.v_3 \wedge v_hv_{of}.posrelspeed)) \longrightarrow [v_h.contspeed](v_h.speed_{safe}(i) \vee v_h.v_1 \vee v_h.v_4 \vee (v_h.v_3 \wedge v_hv_{of}.posrelspeed))) \wedge ((v_h.v_3 \wedge v_hv_{of}.negrelspeed) \longrightarrow [v_h.contspeed]EN(v_h.v_3))$	FOL, 18, 22, 23

Bibliography

[AB08] Zair Abdelouahab and Isaias Braga. An adaptive train traffic controller. In *Springer Netheralands*, pages 550–555, 2008.

[ABH$^+$10] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

[Abr06] Jean-Raymond Abrial. Train systems. In *RODIN Book. Springer*, 2006.

[AK] Anish Arora and Sandeep S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components.

[AK98] Anish Arora and Sandeep S. Kulkarni. Component based design of multitolerant systems. *IEEE Trans. Software Eng.*, 24(1):63–78, 1998.

[BFG02] Cinzia Bernardeschi, Alessandro Fantechi, and Stefania Gnesi. Model checking fault tolerant systems. *Softw. Test., Verif. Reliab.*, 12(4):251–275, 2002.

[BFS00] Cinzia Bernardeschi, Alessandro Fantechi, and Luca Simoncini. Formally verifying fault tolerant system designs. *Comput. J.*, 43(3):191–205, 2000.

[Bro03] Jan Broersen. *Modal Action Logics for Reasoning about Reactive Systems.* PhD thesis, Vrije University, August 2003.

[Cas09] Pablo F. Castro. *DEONTIC ACTION LOGICS FOR SPECIFICATION AND ANALYSIS OF FAULT-TOLERANCE*. PhD thesis, McMaster University, 2009.

[CJ96] José Carmo and Andrew J. I. Jones. Deontic database constraints, violation and recovery. *Studia Logica*, 57(1):139–165, 1996.

[CKAA11] Pablo F. Castro, Cecilia Kilmurray, Araceli Acosta, and Nazareno Aguirre. dctl: A branching time temporal logic for fault-tolerant system verification. In *SEFM*, pages 106–121, 2011.

[CM07a] Pablo F. Castro and T. S. E. Maibaum. A complete and compact propositional deontic logic. pages 109–123, 2007.

[CM07b] Pablo F. Castro and T. S. E. Maibaum. An ought-to-do deontic logic for reasoning about fault-tolerance: the diarrheic philosophers. In *SEFM*, pages 151–160, 2007.

[CM09] Pablo F. Castro and T. S. E. Maibaum. Reasoning about system-degradation and fault-recovery with deontic logic. In *Methods, Models and Tools for Fault Tolerance*, pages 25–43. 2009.

[CM10] Pablo F. Castro and T. S. E. Maibaum. Towards a first-order deontic action logic. In *WADT*, pages 61–75, 2010.

[Cri85] F. Cristian. A rigorous approach to fault-tolerant programming. *IEEE Trans. Softw. Eng.*, 11:23–31, January 1985.

[FM92] Jose Luiz Fiadeiro and T.S.E. Maibaum. Temporal theories as modular- ization units for concurrent system specification. *Formal Aspects of Computing*, 4:239–2172, 1992.

[FTO05] M. Alonso F. Tango, A. Saroldi and A. Oyaide. Towards a new approach in supporting drivers function: Specifications of the saspence system. PReVENT, 2005.

[Gär98] Felix C. Gärtner. Specifications for fault tolerance: A comedy of failures. Technical report, 1998.

[Gär99] Felix C. Gärtner. Transformational approaches to the specification and verification of fault-tolerant systems: Formal background and classification. Technical report, 1999.

[GLM05] Stefania Gnesi, Gabriele Lenzini, and Fabio Martinelli. Logical specification and analysis of fault tolerant systems through partial model checking. *Electr. Notes Theor. Comput. Sci.*, 118:57–70, 2005.

[HG93] Claude Hennebert and Gérard D. Guiho. Sacem: A fault tolerant system for train speed control. In *FTCS*, pages 624–628, 1993.

[JJFN07] Sandor Szabo John J. Ference and Wasim G. Najm. Objective test scenarios for integrated vehicle based safety systems. National Highway Traffic Safety Administration, National Institute of Standards and Technology and Volpe National Transportation Systems Center, 2007.

[JL93] Mathai Joseph and Zhiming Liu. Specifying and verifying recovery in asynchronous communicating systems. *J. (ed), Formal Techeniques in Real-Time and Fault Tolerant Systems*, 1993.

[Kho98] Samit Khosla. *System Specification: A Deontic Approach.* PhD thesis, Imperial College, 1998.

[KM87] Samit Khosla and T. S. E. Maibaum. The prescription and description of state based systems. In *Temporal Logic in Specification*, pages 243–294, 1987.

[KQM91] S. Kent, B. Quirk, and T.S.E. Maibaum. Specifying deontic behaviour in modal action logic. Technical report, Forest Research Project, 1991.

[Lan98] Kevin Lano. Logical specification of reactive and real-time systems. *J. Log. Comput.*, 8(5):679–711, 1998.

[LJ92] Zhiming Liu and Mathai Joseph. Transformation of programs for fault-tolerance. *Formal Asp. Comput.*, 4(5):442–469, 1992.

[LJ93] Zhiming Liu and Mathai Joseph. Specification and verification of recovery asynchornous communication systems. *In J. Vytopil, editor, Formal techniques in real-time and fault-tolerant systems*, 6:137–145, 1993.

[LM94] Leslie Lamport and Stephan Merz. Specifying and verifying fault-tolerant systems. In *FTRTFT*, pages 41–76, 1994.

[LS04] Alessio Lomuscio and Marek J. Sergot. A formalisation of violation, error recovery, and enforcement in the bit transmission problem. *J. Applied Logic*, 2(1):93–116, 2004.

[MAH05] Pedro Garayo Maria Alonso and Loli Herran. Defining safe speed and safe distance towards improved longitudinal control using advanced driver assistance systems: Functional requirements of the saspence system. PReVENT, 2005.

[Mcn] Paul Mcnamara. Kripke-style semantics for sdl. Technical report, Stanford Encyclopedia of Philosophy.

[Mcn06] Paul Mcnamara. Deontic logic. Technical report, Stanford Encyclopedia of Philosophy, April 2006.

[Mey88] J. J. Meyer. A different approach to deontic logic: Deontic logic viewed as variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29:109–136, 1988.

[MT84] T. S. E. Maibaum and Wladyslaw M. Turski. On what exactly is going on when software is developed step-by-step. In *ICSE*, pages 528–533, 1984.

[PJ94] Doron Peled and Mathai Joseph. A compositional framework for fault tolerance by specification transformation. *Theor. Comput. Sci.*, 128(1&2):99–125, 1994.

[Ram07] Bruno L. C. Ramos. Challenging malicious inputs with fault tolerance techniques. Black Hat Europe, 2007.

[SKQ93] T.S.E. Maibaum S. Kent and W. Quirk. Formally specifying temporal constraints and error recovery. *In Proceedings of IEEE International Symposium on Requirements Engineering*, pages 208–215, 1993.

[Tp00] Wilfredo Torres-pomales. Software fault tolerance: A tutorial. Technical report, 2000.

[WM93] Roel J. Wieringa and John-Jules Meyer. Applications of deontic logic in computer science: A concise overview. In *Deontic Logic in Computer Science, Normative System Specification*, 1993.

[XSS] Zaipeng Xie, Hongyu Sun, and Kewal Saluja. A survey of software fault tolerance techniques. University of Wisconsin-Madison/Department of Electrical and Computer Engineering.