Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Eun-Young Kang\textsuperscript{1,2}, Li Huang\textsuperscript{2} and Dongrui Mu\textsuperscript{2}

\textsuperscript{1}PReCISE Research Centre, University of Namur, Belgium
\textsuperscript{2}School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China

eykang@fundp.ac.be
\{huangl223, mudr\}@mail2.sysu.edu.cn
ABSTRACT

Modeling and analysis of nonfunctional requirements is crucial in automotive systems. EAST-ADL is an architectural language dedicated to safety-critical automotive system design. We have previously modified EAST-ADL to include energy constraints and transformed energy-aware timed (ET) behaviors modeled in SIMULINK/STATEFLOW into UPPAAL models amenable to formal verification. Previous work is extended in this paper by including support for SIMULINK DESIGN VERIFIER (SDV), i.e., the ET constraints are translated into proof objective models that can be verified using SDV. Furthermore, probabilistic extension of EAST-ADL constraints is defined and the semantics of the extended constraints is translated into verifiable UPPAAL models with stochastic semantics for formal verification. A set of mapping rules are proposed to facilitate the guarantee of translation. Verification & Validation are performed on the extended timing and energy constraints using SDV and UPPAAL-SMC. Our approach is demonstrated on a cooperative automotive system case study.

Keywords: EAST-ADL, UPPAAL-SMC, SIMULINK DESIGN VERIFIER
Contents

1 Introduction 1

2 Preliminary 3

3 Case Study 5

4 Modeling Non-functional Behaviors in SIMULINK & STATEFLOW 12
  4.1 Behavioral Modeling of $f_{ps}$ in SIMULINK & STATEFLOW . . . . . . 12
  4.2 Modeling Timing and Energy Constraints in Simulink/Stateflow . . . 18

5 Timing & Energy Requirements Translation in SDV 20
  5.1 LTL Properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
  5.2 Timing Constraints in SDV . . . . . . . . . . . . . . . . . . . . . . 22
  5.3 Energy Constraints in SDV . . . . . . . . . . . . . . . . . . . . . . 24

6 Modeling Functional Behaviours in UPPAAL-SMC 26

7 (Non)-functional Property Translation in UPPAAL-SMC 44
  7.1 Probabilistic Timing Constraints Translation . . . . . . . . . . . . 44
  7.2 Energy Constraint Estimation . . . . . . . . . . . . . . . . . . . . . 46

8 Experiment and Verification 48

9 Verification of Comparison Constraint in UPPAAL-SMC 63

10 Related Work 66

11 Conclusion 67

References 68
List of Figures

2.1 General verification models in SDV .......................... 3
2.2 Simulink Blocks used for property modeling ......................... 4
3.1 Cooperative Automotive System ................................ 5
3.2 EAST-ADL Architecture of CAS ................................. 7
4.1 Top-level diagram of CAS SIMULINK & STATEFLOW model .... 13
4.2 STATEFLOW chart of v1Controller ............................. 13
4.3 leader state in v1Controller ................................... 14
4.4 follower state in v1Controller .................................. 15
4.5 userCtrl state in v1Controller .................................. 15
4.6 SIMULINK & STATEFLOW model of localEnv .................. 16
4.7 SIMULINK & STATEFLOW model of DriverBehavior ........... 16
4.8 SIMULINK & STATEFLOW model of VehicleDynamic .......... 17
4.9 SIMULINK & STATEFLOW model of ComDevice ................ 17
4.10 Timing constraints modeling .................................... 19
4.11 Cumulative and Noncumulative constraint ....................... 19
4.12 Periodic constraint modeling ................................... 19
5.1 Bounded Always and Eventually property ......................... 20
5.2 p Until q: G (F[0, t] q ∧ (¬q ⟹ p)) ............................... 21
5.3 Functional Property: When auto is true and msg is false for 200ms, userCtrl should become true within 200ms. .................. 21
5.4 Property: If the dis12 is smaller than safeDis, v2dec should be true within 500ms. ................................................. 22
5.5 Synchronization timing constraint ................................ 22
5.6 Timing constraints translation .................................... 23
5.7 Sporadic timing constraint ....................................... 23
5.8 R30: if the message transmission is failed (msgNormal == false) when the vehicle runs in auto-mode, for the next 20 seconds, the vehicle should not run automatically, i.e., auto == false .......... 24
5.9 Cumulative Periodic Timing Constraint ........................... 24
5.10 Non-cumulative Periodic Timing Constraint ..................... 24
5.11 Energy constraint ................................................... 25
6.1 STA of SignRecognition ............................................. 26
6.2 STA of VehicleDynamic .............................................. 27
6.3 STA for modeling ComDevice ....................................... 28
6.4 STA for modeling DriverBehaviour ................................. 29
6.5 STA of Controller ..................................................... 30
6.6 STA of Control state .................................................. 30
6.7 STA for deciding whether the message is missing ................. 30
6.8 STA for modeling Communication state ............................ 31
6.9 STA for modeling leader state ...................................... 31
6.10 STA for modeling substates of leader state ....................... 32
6.11 STA for modeling substates of leader state ....................... 33
6.12 STA of turnLeft state ............................................... 34
6.13 STA of turnRight state ............................................. 35
6.14 STA of follower state .............................................. 36
6.15 STA of follower state ............................................... 37
6.16 STA for modeling substates of straight state .................... 38
6.17 STA for modeling substates of turn state ......................... 39
6.18 STA for modeling userctrl state .................................. 40
6.19 STA of UserCtrl state .............................................. 41
6.20 STA for modeling substates of straight state .................... 42
6.21 STA for modeling substates of turnleft state ..................... 42
6.22 STA for modeling substates of turnright state .................. 43

7.1 Probabilistic Extension of Timing Constraints in UPML 45
7.2 Energy estimation of v1Controller ................................. 47
7.3 Energy constraints on sensors in UPML ............................ 47

8.1 When v1 runs automatically and if there is no message received by the vehicle for a certain time, the running mode of the vehicle should be changed to manual mode within a certain time: \( \text{G} (G_{[0,200]} v1.Auto == true \land v1.msg == false \implies F_{[0,2000]} v1.userCtrl == true) \) 48
8.2 When v1 runs automatically with a constant speed, if it detects a stop sign, it should start to brake in 500ms: \( \text{G} (v1.Auto == true \land v1.const == true \land signType == 5 \implies F_{[0,500]} v1.stop == true) \) 48
8.3 When the lead vehicle (v1) runs automatically with a constant speed, if it detects left turn sign, it should start to turn left in 200ms: \( \text{G} (v1.auto == true \land v1.const == true \land signType == 4 \implies F_{[0,200]} v1.turnLeft == true) \) 49
8.4 If v1 is driven by the driver and it is running with a constant speed, the driver steers to the left, the vehicle will turn left within 200ms: \( \text{G} (v1.userCtrl == true \land v1.const == true \land v1.steerReq == 1 \implies F_{[0,200]} v1.turnLeft == true) \) 49
8.5 If v1 is driven by the driver and it is running at a constant speed, the driver brakes the car, the vehicle will slow down. \( G: (v1.userCtrl == true \land v1.const == true \land v1.brakeReq == 2 \implies F_{[0,200]} v1.brake == true) \)

8.6 When v1 is driven by the driver and it is running at a constant speed, if the driver gears up, the vehicle will accelerate. \( G (v1.userCtrl == true \land v1.const == true \land v1.gearReq > v1.gear \implies F_{[0,200]} v1.acc == true) \)

8.7 When the vehicles are running horizontally, following vehicle should not run ahead of lead vehicle. \( G (v1.dx == 1 \land v2.dx == 1 \land v3.dx == 1 \implies v1.x > v2.x) \)

8.8 When the lead vehicle detects a stop sign, the three vehicles should stop in 5s. \( G (v1.auto == true \land signType == 5 \implies F_{[0,5000]} v1.static == true \land v2.static == true \land v3.static==true) \)

8.9 Noncumulative Periodic timing constraint: A Periodic acquisition of VehicleDynamic of v1 must be carried out for every 50 ms with a jitter 10 ms. \( G (v1.velIn == true \implies v1.velIn == false U_{[40, 60]} v1.velIn == true) \)

8.10 Cumulative Periodic timing constraint of R27

8.11 Sporadic constraint: After the running mode of the lead vehicle is changed to manual mode because of the failed message transmission, the driver should not change it into auto mode within 20 seconds. \( G (v1.msgNormal == false \implies v1.auto == false U_{[2000, \infty]} v1.auto == true) \)

8.12 An Execution constraint applied on v3Controller, which measured from the inputs ports Avel to output ports gearReq, torqueReq, and brake, should be between 100 ms and 300 ms; \( G (v3.ctrlIn \implies F_{[100,300]} v3.ctrlOut) \)

8.13 An Execution constraint applied on v3Controller, which measured from the inputs ports Avel to output ports gearReq, torqueReq, and brake, should be between 100 ms and 300 ms; \( G (v3.ctrlIn \implies F_{[100,300]} v3.ctrlOut) \)

8.14 An Execution constraint applied on ComDevice of v1, which measured from the inputs ports torqueSet, gearSet and brake to output ports wheelSpeed, direct should be between 100 ms and 150 ms; \( G (v1.sdrIn \implies F_{[100,300]} v1.sdrOut) \)

8.15 Observer STA of R1 and its triggering

8.16 UPPAAL-SMC STAs for timing constraints verification

8.17 UPPAAL-SMC STAs for End-to-End timing constraints verification

8.18 UPPAAL-SMC STAs for End-to-End timing constraints verification
8.19 CE in SDV: In (b), \( \text{leadVe}.y \) equals to \( \text{followVe}.y \) from 0 to 8s, which indicates that the two vehicles run in the same lane. In (a), \( \text{signType} \) becomes “turn left (4)” at 8s and the lead vehicle begins to turn left. However, the following vehicle turns left at a different location (\( \text{leadVe}.x \) is not equal to \( \text{followVe}.x \) at 9s). Hence, the “x” is unequal after turning, which means that the two vehicles run on different lanes. Thus R23 and R24 is violated.

8.20 CE in SMC: At the beginning, three vehicles run towards direction (1,0). The three values of \( \text{posy} \) are the same, which indicates that the three vehicles are in the same lane. After a left turn sign is detected, the lead vehicle starts to turn left. Since the first follower turns left at (184, 0) and the second turns left at (157, 0), the three values of \( \text{posx} \) are not equal after turning, which indicates the three vehicle are not running in the same lane. Thus R5 is violated.

8.21 Probabilistic distribution of the expected energy consumption

8.22 Total energy consumption estimation of the lead vehicle
Chapter 1

Introduction

Model-driven development is rigorously applied in automotive systems in which the software controllers interact with physical environments. The continuous time behaviors (evolved with various energy rates) of those systems often rely on complex dynamics as well as on stochastic behaviors. Formal verification and validation (V&V) technologies are indispensable and highly recommended for development of safe and reliable automotive systems [1,2]. Conventional V&V, i.e., testing and model checking have limitations in terms of assessing the reliability of hybrid systems due to both the stochastic and non-linear dynamical features. To ensure the reliability of safety critical hybrid dynamic systems, statistical model checking (SMC) techniques have been proposed [3–5]. These techniques for fully stochastic models validate probabilistic performance properties of given deterministic (or stochastic) controllers in given stochastic environments.

EAST-ADL [6,7] is a domain specific language that provides support for architectural modeling of automotive embedded systems. A system in EAST-ADL is described by Functional Architectures (FA) at different abstraction levels. The FA are composed of a number of interconnected functionprototypes (fp s), and the fp s have ports and connectors for communication. EAST-ADL relies on external tools for the analysis of specifications related to requirements. For example, behavioral description in EAST-ADL is captured in external tools, i.e., SIMULINK/STATEFLOW [8]. The latest release of EAST-ADL has adopted the time model proposed in the Timing Augmented Description Language (TADL2) [9]. TADL2 allows for the expression and composition of timing constraints, e.g., periodic, end-to-end delay, and synchronization constraints on top of EAST-ADL models.

In this paper, we propose a model-driven approach to support formal analysis of energy and timed (ET) requirements for automotive systems at the design level: 1. The ET constraints in EAST-ADL are interpreted in SIMULINK/STATEFLOW (S/S) and provide corresponding modeling extensions in S/S; 2. The ET requirements, specified in temporal logics, are translated into proof objective models to perform formal verification using SDV; 3. Probabilistic extension of EAST-ADL/TADL constraints is defined and the semantics of the extended constraints (XTC) is translated into verifiable UPPAAL-SMC [10] models with stochastic semantics for formal veri-
fication; 4. A set of mapping rules are proposed to facilitate the guarantee of translation; 5. Simulation and V&V are performed on the Xtc and energy constraints using SDV and UPPAAL-SMC. Our approach is demonstrated on the cooperative automotive system (CAS) case study.

The paper is organized as follows: Chap. 2 presents an overview of SIMULINK, SDV, and UPPAAL-SMC. The CAS is introduced as a running example in Chap. 3. Chap. 4, 5, 6 and 7 describe a set of translation patterns and how our approaches provide support for formal analysis at the design level. The applicability of our method is demonstrated by performing verification on the CAS case study in Chap. 8. Chap. 10 and Chap. 11 present related work and conclusion.
Chapter 2

Preliminary

In our framework, Simulink and EMBEDDED MATLAB (EML) are utilized for modeling purposes. Verification and experiment are performed by SDV and UPPAAL-SMC.

Simulink and SDV: Simulink [8] is a synchronous data flow language, which provides different types of blocks, i.e., primitive-, control flow-, and temporal-blocks with predefined libraries for modeling and simulation of dynamic systems and code generation. Simulink supports the definition of custom blocks via Stateflow diagrams or user-defined function blocks written in EML, C, and C++. SDV is a plug-in to PROVER, which is a formal verification tool that performs reachability analysis on Simulink/Stateflow model. The satisfiability of each reachable state is determined by a SAT solver. A proof objective model is specified in Simulink/SDV and illustrated in Fig. 2.1. A set of data (predicates) on the input flows of System is constrained via ≪Proof Assumption≫ blocks during proof construction. A set of proof obligations is constructed via a function F block and the output of F is specified as input to a property P block. P passes its output signal to an ≪Proof Objective≫ block and returns true when the predicates set on the input data flows of the outline model are satisfied. Whenever ≪Proof Objective≫ (P block) is utilized, SDV verifies whether the specified input data flow is always true. The underlying PROVER engine allows the formal verification of properties for a given model. Any failed proof attempt ends in the generation of a counterexample representing an execution path to an invalid state. A harness model is generated to analyze the counterexample and refine the model.

Fig. 2.1: General verification models in SDV

Fig. 2.2 shows blocks in SDV that can be employed in properties specification.
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

≪Implies≫ block (Fig.2.2(a)) allows to specify a condition to produce a certain response: if A is true and B is false, the output is false; for all other pairs of A and B, the outputs are true. ≪Within Implies≫ block (Fig.2.2(b)) verifies the property that the response occurs within a given duration. This block captures the within implication by observing whether the Obs input is true for at least one step within each true duration of In. If Obs is always false within a particular input true duration of In, the output becomes false for one time step following the input true duration. As shown in Fig.2.2(c), the two blocks always appear in pairs with the same tags. The output signal from Block A goes to the ≪Goto≫ block tagged with A. ≪Goto≫ block then passes that signal to the ≪From≫ block tagged with A. Block B receives the signal from the ≪From≫ block.

Figure 2.2: Simulink Blocks used for property modeling

UPPAAL-SMC provides probabilistic analysis of properties based on statistical estimation of system models. By monitoring simulations of an hybrid system and using the statistic result to determine whether the properties are satisfied on top of the system model, an exhaustive exploration of the state-space of the system can be avoided. UPPAAL-SMC represents systems via networks of Timed Automata (TA) whose behaviours depend on both stochastic and non-linear dynamical features. Its clocks evolve with various rates, such rates can be specified with ordinary differential equations. UPPAAL-SMC provides a number of queries related to the stochastic interpretation of TA (STA) [5]. The following four queries are sufficient to perform V&V on timing and energy constraints in EAST-ADL, where N and bound indicate the number of simulations and the simulation time for each simulation respectively.

1. **Probability Estimation** estimates the probability that a requirement property \( \phi \) is satisfied for a given STA model within the time bound: \( \Pr[\text{bound}] \phi \);

2. **Hypothesis Testing** checks if the probability of \( \phi \) is satisfied over a given probability \( P_0 \): \( \Pr[\text{bound}] \phi \geq P_0 \);

3. **Simulations**: UPPAAL-SMC runs multiple simulations on the system model and the \( k \) (state-based) properties/expressions \( \phi_1, ..., \phi_k \) are monitored and visualized along the simulations: simulate \( N[\leq \text{bound}]\{\phi_1, ..., \phi_k\} \);

4. **Expected Value** evaluates the minimal or maximal value of a clock or an integer value while UPPAAL-SMC checks the STA model: \( E[\text{bound};N](\min : \phi) \) or \( E[\text{bound};N](\max : \phi) \).
Chapter 3

Case Study

A Cooperative Automotive System (CAS) is adopted as our running example where sensing and actuation are distributed over a number of vehicles, which is illustrated in Fig 3.1. A vehicle can be either a lead vehicle (leadVe) or a following vehicle (followVe) depending on its absolute position. We name the first vehicle as v1, the vehicle in the middle as v2 and the last vehicle v3. Communication between vehicles are realized by the Dedicated Short-Range Communication (DSRC), which is a wireless communication technology specifically dedicated for message transmission among automotive systems. The lead vehicle can either run automatically (auto mode) by detecting traffic signs on the road or be driven by the driver (userCtrl mode). Any two adjacent vehicles should be distant enough to avoid rear-end collision and close enough to guarantee the communication quality.

The requested vehicle motion is based on driver’s input, detected environment (e.g., traffic sign) and running situation. The vehicle movement can be either horizontal or vertical and the position of each vehicle is represented by two-dimension coordinates (x, y) in Cartesian coordinate system, where x and y are distances measured from vehicle to two fixed perpendicular directed lines respectively. Ideally, the following vehicle should maintain the same movement as the vehicle ahead and behaves similarly, e.g., when a lead vehicle accelerates, the following vehicles should be able to accelerate to a similar extent.

![Figure 3.1: Cooperative Automotive System](image)

Fig 3.2(a) shows the EAST-ADL model of the CAS, which consists of cooperative environment (CoopEnv) and local environments, left and middle, and the software controllers of three vehicles, right. The three LocalEnv $f_p$s receive information of positions and the traffic sign from the CoopEnv. Each lead vehicle
sends its own position and velocity to its adjacent following vehicle. The functionality of an individual vehicle, augmented with timing/energy constraints and viewed as FunctionDesignArchitecture (FDA) in [3.2(b)], consists of the following components: ComDevice represents the DSRC device for vehicle to send and receive messages. SignRecDevice analyzes the detected images and computes desired images (sign types). VeModeDevice reads the driver’s requests by means of pedals (brakeReq), steering wheel (steerReq), gear (gearReq) and a switch (driver-State) which indicates whether the driver intends to drive. v1Controller determines how the torque and gear of the vehicle are adjusted based on the received information. VeDynamicDevice specifies the kinematics behaviors of the vehicle based on engine torque, brake, steer and gear. The value of signType ranges from 0 to 5, which represents “straight”, “min/max speed limit”, “left/right turn ahead” and “stop” signs respectively.

We consider the following functional and non-functional properties, which includes Execution, End-to-End, Synchronization, Periodic, Sporadic and Comparison timing and Energy constraints, on top of the CAS EAST-ADL model, which are sufficient to capture the constraints described in Fig[3.2]. Energy constraint refers to the battery consumption of an individual vehicle in CAS.

R1. When v1 runs automatically and if there is no message received by the vehicle for a certain time, the running mode of the vehicle should be changed to manual mode within a certain time.

R2. When v2 runs automatically and if there is no message received by the vehicle for a certain time, the running mode of the vehicle should be changed to manual mode within a certain time.

R3. When v3 runs automatically and if there is no message received by the vehicle for a certain time, the running mode of the vehicle should be changed to manual mode within a certain time.

R4. When v1 runs automatically at a constant speed, if it detects a stop sign, it should start to brake in 500ms.

R5. When v1 runs automatically at a constant speed, if it detects left turn sign, it should start to turn left in 200ms.

R6. When v1 runs automatically at a constant speed, if it detects right turn sign, it should start to turn right in 200ms.

R7. If v1 is driven by the driver and it is running with a constant speed, the driver steers to the left, the vehicle will turn left within 200ms.

R8. If v1 is driven by the driver and it is running at a constant speed, the driver steers to the right, the vehicle will turn right within 200ms.
Figure 3.2: EAST-ADL Architecture of CAS
R9. If v1 is driven by the driver and it is running at a constant speed, the driver brake the car, the vehicle will slow down.

R10. When v1 is driven by the driver and it is running at a constant speed, if the driver gears up, the vehicle will accelerate.

R11. When v1 is driven by the driver and it is running at a constant speed, if the driver gears down, the vehicle will decelerate.

R12. v2 should not run ahead of v1.

R13. v3 should not run ahead of v2.

R14. When the lead vehicle detects a stop sign, the three vehicles should stop in 5s.

R15. When the vehicles are running straight with a constant speed, if v2 runs slower than v1, it should start to accelerate within a certain time.

R16. When the vehicles are running straight with a constant speed, if v3 runs slower than v2, it should start to accelerate within a certain time.

R17. When the vehicles are running straight with a constant speed, if v2 runs faster than v1, it should start to decelerate within a certain time.

R18. When the vehicles are running straight with a constant speed, if v3 runs faster than v2, it should start to decelerate within a certain time.

R19. If the distance between v1 and v2 is larger than 500m, v2 should accelerate within a certain time, e.g. 200ms.

R20. If the distance between v2 and v3 is larger than 500m, v3 should accelerate within a certain time, e.g. 200ms.

R21. If the distance between v1 and v2 is smaller than the safety distance, v2 should decelerate within a certain time, e.g. 500ms.

R22. If the distance between v2 and v3 is smaller than the safety distance, v3 should decelerate within a certain time, e.g. 500ms.

R23. When v1 starts to turn left, v1 and v2 should finish turning and run in the same lane within a given time.

R24. When v2 starts to turn left, v2 and v3 should finish turning and run in the same lane within a given time.

R25. When v1 starts to turn right, v1 and v2 should finish turning and run in the same lane within a given time.
R26. When v2 starts to turn right, v2 and v3 should finish turning and run in the same lane within a given time.

R27. A Periodic acquisition of VehicleDynamic of v1 must be carried out for every 50 ms with a jitter 10 ms.

R28. A Periodic acquisition of VehicleDynamic of v2 must be carried out for every 50 ms with a jitter 10 ms.

R29. A Periodic acquisition of VehicleDynamic of v3 must be carried out for every 50 ms with a jitter 10 ms.

R30. Sporadic constraint: After the running mode of v1 is changed to manual mode because of the failed message transmission, the driver should not change it into auto mode within 20 seconds.

R31. Sporadic constraint: After the running mode of v2 is changed to manual mode because of the failed message transmission, the driver should not change it into auto mode within 20 seconds.

R32. Sporadic constraint: After the running mode of v3 is changed to manual mode because of the failed message transmission, the driver should not change it into auto mode within 20 seconds.

R33. An Execution constraint applied on v1Controller, which measured from the input port Avel to output ports gearReq, torqueReq and brake, should be between 100 ms and 300 ms;

R34. An Execution constraint applied on v2Controller, which measured from the input port Avel to output ports gearReq, torqueReq and brake, should be between 100 ms and 300 ms;

R35. An Execution constraint applied on v3Controller, which measured from the input port Avel to output ports gearReq, torqueReq and brake, should be between 100 ms and 300 ms;

R36. An Execution constraint applied on ComDevice of v1, which measured from the input ports Avel, Apos, Adirect and pos to output ports Avel, Apos, Adirect and pos should be between 50 ms and 100 ms;

R37. An Execution constraint applied on ComDevice of v2, which measured from the input ports Avel, Apos, Adirect and pos to output ports Avel, Apos, Adirect and pos should be between 50 ms and 100 ms;

R38. An Execution constraint applied on ComDevice of v3, which measured from the input ports Avel, Apos, Adirect and pos to output ports Avel, Apos, Adirect and pos should be between 50 ms and 100 ms;
R39. An **End-to-End** is measured from v1Controller to v2Controller. The time interval is bounded with a minimum value of 300 ms and a maximum value of 700 ms;

R40. An **End-to-End** is measured from v2Controller to v3Controller. The time interval is bounded with a minimum value of 300 ms and a maximum value of 700 ms;

R41. An **End-to-End** constraint measured from input port **position** of v1LocalEnv to output ports **gearReq**, **torqueReq** and **brake** of v1Controller should be limited within [200, 500]ms;

R42. An **End-to-End** constraint measured from input port **position** of v2LocalEnv to output ports **gearReq**, **torqueReq** and **brake** of v2Controller should be limited within [200, 500]ms;

R43. An **End-to-End** constraint measured from input port **position** of v3LocalEnv to output ports **gearReq**, **torqueReq** and **brake** of v3Controller should be limited within [200, 500]ms;

R44. **Synchronization** constraint: The position and velocity of the vehicle and its lead vehicle (**pos**, **vel**, **Apos** and **Avel** ports on v1Controller) must be detected by v1Controller within a given time window, i.e., the tolerated maximum constraint = 200 ms;

R45. **Synchronization** constraint: The position and velocity of the vehicle and its lead vehicle (**pos**, **vel**, **Apos** and **Avel** ports on v2Controller) must be detected by v2Controller within a given time window, i.e., the tolerated maximum constraint = 200 ms;

R46. **Synchronization** constraint: The position and velocity of the vehicle and its lead vehicle (**pos**, **vel**, **Apos** and **Avel** ports on v3Controller) must be detected by v3Controller within a given time window, i.e., the tolerated maximum constraint = 200 ms;

R47. A **Comparison** constraint: The execution time interval measured from v1Controller to v2Controller needs to be larger than or equal to the sum of the worst-case execution times of the \( f_p \)s.

R48. **Energy** constraint: The maximum battery energy consumption for the braking of a vehicle should be less than 30kJ;

R49. **Energy** constraint: The maximum battery energy consumption for the controller to decide its movement for once should be less than 30J;

R50. **Energy** constraint: The maximum battery energy consumption for **ComDevice** to get/send signals for once should be less than 5J;
According to the EAST-ADL meta-model, the timing constraint describes a design constraint, but has the role of a property, requirement or validation result, based on its Context [6]. The TADL meta-model is integrated with the EAST-ADL meta-model and is supplemented with structural concepts from EAST-ADL. The EAST-ADL/TADL constraints contain the identifiable state changes as Events. The causality related events are contained as a pair by EventChains. Based on Event and EventChains, data dependencies, control flows, and critical execution paths are represented as additional constraints for the EAST-ADL functional architectural model, and apply timing constraints on these paths.

Sporadic constraint specifies a minimum delay min between any two consecutive occurrences of an event. Execution constraint restricts the time interval between inputs and outputs of a fp, which can be specified using lower, upper values given as Execution constraint. End-to-End constraint gives duration bounds (minimum and maximum) between two events source and target. The time intervals between source event and target event should satisfy lower, upper values given as End-to-End constraint. Synchronization constraint describes how tightly the occurrences of a group of events follow each other. All events must occur within a sliding window, specified by the tolerance attribute, i.e., the maximum time interval allowed between events. Comparison constraint specifies the comparison relation (including <, ≤, =, ≥ and >) among timing expressions. Periodic constraint states that the period of successive occurrences of a single event must have a length of at least a lower and at most an upper time interval. The interval is given as Periodic constraint. There are two types of Periodic timing constraints: cumulative (Cumu_Periodic) and non-cumulative timing constraints (Noncumu_Periodic). Those (non)-functional properties are formally specified and verified in the following sections.
Chapter 4

Modeling Non-functional Behaviors in SIMULINK & STATEFLOW

4.1 Behavioral Modeling of \( f_p \)s in SIMULINK & STATEFLOW

Based on the EAST-ADL architectural model of CAS, the functional and non-functional behaviours of \( f_p \)s is constructed with Stateflow chart and Simulink blocks. The architecture of Simulink/Stateflow model can be seen in Fig[4.1]. \( v1Controller \) \( f_p \) is modeled as a Stateflow chart while \( \text{localEnv} \), \( \text{CoopEnv} \) are modeled as Simulink subsystems with a number of mathematical and descriptive blocks provided in Simulink block library.

The top-level diagram of \( v1Controller \) Stateflow chart is illustrated in Fig[4.2], where three Matlab functions blocks are embedded: \( \text{checkGear()} \) is used to ensure that \textit{gear} is in the reasonable value range; \( \text{reDirect()} \) adjusts vehicle’s direction when vehicle turns; \( \text{checkDis} \) calculates the distance between the vehicle and the vehicle in front of it. A single Stateflow chart, consisting of two parallel states, implements the control logic of the system in its entirety. In \textit{Control} state, \textit{Auto} and \textit{userCtrl} refer two different running modes of a vehicle depend on whether the driver controls the vehicle.

If the vehicle runs automatically, either \( \text{leader} \) (shown in Fig[4.3]) or \( \text{follower} \) (shown in Fig[4.4]) will be activated according to the absolute positions of the three vehicles. Inside the \( \text{leader} \) state, there are six superstates \( \text{turn_right, turn_left, constSpeed, acc, dec and stop} \) together with their child states. The edges represent transitions between the states with the conditions on the edges. Initially, The chart transits to either \( \text{straight or stop} \) based on the initial speeds of the wheels. The transitions will be taken according to the current speed of the wheels and the value of \textit{signType}, whose various evaluation represents the different traffic sign. For example, when the vehicle is in \( \text{constSpeed} \) state and the detected \textit{signType} is 5 (stop sign is encountered), \textit{braking} and \textit{stop} will be activated and \textit{brake} will become 1 indicating that the velocity of vehicle should be decreased steeply. Whenever the vehicle detects a stop sign, the vehicle will start to brake. If the straight sign is detected (\textit{signType} is 0), the vehicle will maintain the current state and speed. If the
Figure 4.1: Top-level diagram of CAS SIMULINK & STATEFLOW model

Figure 4.2: STATEFLOW chart of v1Controller
vehicle recognizes a left turn sign, it will first decrease its speed to a low level (less than 30 km/h) and start to turn. After turning left, the vehicle will finally go straight and maintain the speed. *turnX* and *turnY* specify the location at which the vehicle turns.

Since a following vehicle should follow the traces and movements of the vehicle in front of it, in *follower* state, the transitions will be taken once the safety condition is violated, e.g., the distance between the two adjacent vehicles is either larger than 500m or less than the safety distance. As illustrated in *turn* state on the left of Fig 4.4, when the following vehicle is turning left/right, it will first accelerate to reach to the turning location at which the lead vehicle turns.

If the vehicle is driven by its driver, *userCtrl* state (as illustrated in Fig.4.5) will be active. The running mode (*stop, turnLeft, turnRight and straight*) of the vehicle will be changed based on the driver’s operations. *steer* represents the steering operation is conducted, i.e., steering to left (*steer = 1*), right (*steer = 2*) and straight (*steer = 3*).

Fig.4.6 illustrates the Simulink and Stateflow model of *localEnv*, whose functionality is captured in three subsystems: *ComDevice* is responsible for signal propagation and message receiving; *DriverBehavior* records the value of signals (*gear, steer, brake and DriverState*) that represent various operations conducted by the driver; *VehicleDynamic* specifies the how the velocity and running direction of the vehicle are changed according to the input *torque* and *gear* value. In this figure, blocks/entities in green format are employed to model timing constraints, whose modeling process will be explained detailly in the following section.

Fig.4.7 shows the Simulink and Stateflow model of *DriverBehavior fp*. This
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Figure 4.4: follower state in v1Controller

Figure 4.5: userCtrl state in v1Controller
subsystem obtains inputs from the driver by «Manual Switch» block, which represents a toggle switch where one of its two inputs (1 or 0) will be selected to pass through and become the value of the output.

Fig. 4.8 shows the Simulink and Stateflow model of VehicleDynamic $f_p$. In VehicleDynamic subsystem, a «Look Up Table» block maps inputs ($gear$ and $torqueSet$) to an output value ($wheelspeed$) by looking up or interpolating the predefined values in a table. «DataStore» block records the energy consumption in the duration of braking.
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Figure 4.8: SIMULINK & STATEFLOW model of VehicleDynamic

Figure 4.9: SIMULINK & STATEFLOW model of ComDevice
4.2 Modeling Timing and Energy Constraints in Simulink/Stateflow

In this section, we investigate how to model timing and energy constraints in S/S. In our previous work, we have shown how those constraints are interpreted in S/S and provided corresponding modeling extensions in S/S. However, the previous modeling extensions contain blocks that are unsupported in SDV, which leads to incompatibility problem. To perform verification of timing and energy constraints in SDV, the modeling extensions are modified to ensure the compatibility.

Sporadic timing constraint is modeled using “after(min, msec)” expression, which returns true after min milliseconds (msec) have elapsed since activation of the associated state. Synchronization timing constraint restricts the time duration among a set of events (i.e., maximum allowed time between the arrival of the event occurrences). We use In1, In2 and In3 (in Fig.4.10(a)) to indicate the arrivals of the three inputs of fp. To verify Synchronization constraint, we check whether the interval between occurrences of earliest and latest input event is within tolerance, which is described in Chap. 5.

As shown in Fig.4.10(a), to model Execution timing constraint of an fp, a Stateflow chart is used to create a true duration to execute the fp. The ≪Enable Subsystem≫ ES (representative of fp) executes when Exe state is active. t is the Execution constraint applied on the fp. id is a monotonically increased value. To ensure the one-to-one correspondence of input and output event, we use two variables, r and s, to represent the value of input and output event respectively. When input event occurs, fp will start to execute and r will be assigned by the value of current id. After t milliseconds, fp finishes execution and s becomes identical with r, which indicates the occurrence of output event.

Fig.4.10(b) shows an EventChain of n fps, we model each fp based on the pattern of Fig.4.10(a). For each fp, ri and si represent the values of input and output event. Hence in this EventChain, r1 is the value of source event and sn is the value of target event of End-to-End constraint. To identify the one-to-one correspondence of source and target event, r1 and sn should have the same assigned value.

We consider two types of Periodic timing constraints: cumulative constraint and noncumulative constraint. T and j indicate the period and jitter of Periodic constraint. In Fig.4.11, e(i) denotes the time point of the ith occurrence of a single event e. ei represents the ith occurrence of event e. Cumulative Periodic constraint limits that the time intervals between any two consecutive occurrences of e (e.g., |e(2)−e(1)| and |e(3)−e(2)|) should be within [T−j, T+j]. Noncumulative Periodic constraint limits that e(i) should be within [i*T−j, i*T+j].

Fig.4.12(a) shows the model of Cumulative Periodic timing constraint of fp. The behaviour of fp is modeled in TS (Trigger subsystem) The Stateflow chart
generates a signal $Trig$ to trigger $TS$. $rand$ is a random number within $[0, 2*j]$. Once $fp$ is triggered, it will be triggered again after $T-j+rand$ ms such that the time interval between any two consecutive triggerings of $fp$ is in $[T-j, T+j]$. Fig 4.12(b) shows the model of Noncumulative Periodic timing constraint of $fp$ (modeled in TS1). A ≪Pulse Generator≫ (PG) generates square waves based on Period and Phase delay parameters. Phase delay specifies the delay before the pulse is generated. PG generates a square wave signal whose Period is $T$ and Phase delay is $T-j$ such that the $i^{th}$ rising edge of the signal arrives at the time point of $i*T-j$. In the mean time, TS will be triggered to pass $rand$ and ≪Delay≫ block delays the rising edges by $rand$ ms s.t. the time point of the $i^{th}$ rising edge will be in $[i*T-j, i*T+j]$.

Energy constraints are modeled with ≪Data Store≫ and integrated MATLAB blocks, which calculate and record energy consumption in different modes of a vehicle. According to the various modes of vehicles in CAS, the amount of battery consumed for the mechanical motion of wheels is calculated: $energy = \int_{0}^{t} a * v \, dx$ where, $t$, $a$, and $v$ denote running time, coefficient reflects an energy rate associated to the current mode of an individual vehicle, and velocity respectively.
Chapter 5

Timing & Energy Requirements Translation in SDV

To enable the verification of the timing and energy requirements illustrated in Chap. 4, we specify timing and energy requirements as LTL formulas and provide the approach to model linear temporal logic (LTL) [11] formulas into proof objective models.

5.1 LTL Properties

An LTL formula usually consists of predicates and propositional connectives, together with temporal operators (e.g. Always, Eventually and Until).

**Always p** ($G\ p$) states that property $p$ always holds along the execution, which is modeled as the proof objective model described in Chap. 2.

**Always and Eventually within t time steps**: Always $p$ within $t$ time steps, denoted by $G_{[0,t]} p$, is valid if property $p$ holds over the first $t$ time steps during execution. Eventually $p$ within $t$ time steps, denoted as $F_{[0,t]} p$, holds if $p$ occurs within the first $t$ time steps during the execution. The model of Always property and Eventually are illustrated on the left and right of Fig. 5.1: $dur$ is a signal which is true within the first $t$ time steps. To checks whether $p$ always holds true within the true duration, $p$ is connected to $\langle$Implies$\rangle$ block. $\langle$Within Implies$\rangle$ block is employed to verify whether $p$ holds for at least one time step within the true duration $dur$. P block is applied to prove whether the property is satisfied.

![Figure 5.1: Bounded Always and Eventually property](image)

**p Until q within t time steps**. Denoted as $p U_{[0,t]} q$, this property states that $p$ should always hold before $q$ becomes true within the first $t$ time steps during the execution. The satisfaction of this property requires $\phi_1$: $q$ must hold at least one time step...
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

within the first $t$ time steps; $\varphi_2$: before $q$ holds, $p$ must always be true. This property can be interpreted as: $F_{[0, t]} q \land G_{[0, t]} (\neg q \implies p)$. The implementation of this property is shown in Fig. 5.2. $dur$ has a true duration with length $t$. First, a $\text{Within Implies}$ block check whether $\varphi_1$: $F_{[0, t]} q$ holds. An $\text{Extender}$ block is applied to extend the true duration of $q$ to $t$ time steps after $q$ occurs. $\text{Not}$ block is then employed to capture the true duration of $\neg q$. $\text{Implies}$ block checks whether “not $q$ implies $p$” holds. Afterwards, $\text{And}$ and $\text{Proof Objective}$ validate whether “$\varphi_1 \land \varphi_2$” is always true.

![Figure 5.2: p Until q: G (F_{[0, t]} q \land (\neg q \implies p))](image)

Fig. 5.3 illustrates the functional property (R1) which can be specified by using $\text{Always}$ and $\text{Eventually}$ operators: $G_{[0,200]} (\text{auto} == \text{true} \land \text{msgNormal} == \text{false}) \implies F_{[0,200]} \text{userCtrl} == \text{true}$. $\text{auto}$ ($\text{userCtrl}$) represents whether the vehicle is running automatically (driven manually). $\text{msgNormal}$ indicates whether the message is transmitted normally. In Fig. 5.3, $\text{Detector}$ constructs a signal with a 200ms true duration after “$\text{auto} == \text{true} \land \text{msg} == \text{false}$” becomes true for 200ms. Within this true duration, “$\text{userCtrl} == \text{true}$” should be satisfied at least one time step. To check this, the output signal from $\text{Detector}$ goes to the $\text{Goto}$ block tagged with $dur$. Then it is passed to the corresponding $dur$ tag on the right of Fig. 5.1. Similarly, the signal that represents “$\text{userCtrl} == \text{true}$” is passed to $p$ tag on the right of Fig. 5.1 (the input of $\text{Within Imply}$) block.

![Figure 5.3: Functional Property: When auto is true and msg is false for 200ms, userCtrl should become true within 200ms.](image)

Fig. 5.4 illustrates the model of the functional property (R21), which can be specified with $\text{Util}$ and $\text{Always}$ temporal operators: $G (\text{dist12} < \text{safeDis} \implies v2.\text{dec} == \text{false}) \implies U_{[0,500]} v2.\text{dec} == \text{true})$. $\text{dist12}$ is the distance between the lead vehicle ($v1$) and the following vehicle ($v2$). $\text{safeDis}$ is the required safety distance between them. $v2.\text{dec}$ represents whether $v2$ is decelerating. $\text{Detector}$ creates a signal with a 500ms true duration after “$\text{dist12} < \text{safeDis}$” becomes true. This signal then goes to the $\text{From}$ block tagged with $dur$ in Fig. 5.2. In this example, $p$ represents the property “$v2.\text{dec} == \text{false}$” and $q$ represents “$v2.\text{dec} == \text{true}$” as illustrated on the right of Fig. 5.4.
5.2 Timing Constraints in SDV

Timing constraints in EAST-ADL are modeled by means of constraints specified on Events and EventChains. A constraint is expressed as a proof objective model in SDV. To proof the correctness of the timing behaviours of Simulink/Stateflow model, we show how to construct the proof objective models for the design model based on the timing constraints. We focus on Synchronization, Execution, End-to-End, Periodic, Sporadic constraints (R27-R46) that are associated to fP in EAST-ADL.

Synchronization timing constraint can be interpreted as: all the input signals should be received within tolerance after the first (earliest) input signal arrives. The model of the timing constraint is shown in Fig.5.5. Assume that there are three input signals of an fP. firstIn represents whether the first (earliest) input signal is received. In1, In2 and In3 indicate the arrival of three input signals of fP respectively. ≪Extender≫ is employed to construct a true duration with length tolerance from the time step in which firstIn becomes true. Three ≪Within Implies≫ block are applied to capture whether In1, In2 and In3 becomes true within the true duration. ≪And≫ block then validates whether In1, In2 and In3 all become true within the true duration and pass the output signal to a ≪Proof Objective≫ block, which proves whether the constraint is valid.

Execution timing constraint limits that after the input signals are received, the output data should be sent out within upper time steps. The construct of this timing constraint is shown on the left of Fig.5.6 (a): dataIn (dataOut) indicates the input (output) signal received (sent out). ≪Extender≫ is applied to construct a true duration with length upper after dataIn becomes true. Afterwards ≪Within Implies≫ and ≪Proof Objective≫ blocks are used to verify whether the dataOut becomes true within the true duration.

End-to-End timing constraint restricts that after source event occurs, target event
should happen within \textit{tolerance}. The model of this timing constraint is illustrated on the right of Fig.5.6 (b): \textit{source} and \textit{target} indicate the \textit{source} and \textit{target} event occurs (true) or not (false). \textit{Extender} constructs a true duration with length \textit{tolerance} after \textit{source} becomes true. Afterwards \textit{Within Implies} and \textit{Proof Objective} blocks are used to verify whether the \textit{target} becomes true within the true duration.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.6.png}
\caption{Timing constraints translation}
\end{figure}

\textbf{Sporadic timing constraint} specifies that after the event occurs, the next occurrence should not happen within \textit{min} time steps. This timing constraint is constructed as the model in Fig.5.7 \textit{event} indicates the occurrence of an event. \textit{Detector} constructs a true duration with length \textit{min} after \textit{event} becomes true. Afterwards, \textit{Negation} (Not) and \textit{Implies} blocks are employed to validate whether \textit{p} is false over the true duration. At last, \textit{Proof Objective} block checks whether the property is satisfied. Fig.5.8 illustrates an example of sporadic timing constraint (R30), where the constraint is modeled in the dash box. In this example, \textit{event} is “auto == true”.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.7.png}
\caption{Sporadic timing constraint}
\end{figure}

\textbf{Periodic timing constraint} limits a single event \textit{f}_p should occur periodically with period \textit{T} and jitter \textit{j}. We consider two types of timing constraints in SDV: 1. \textbf{Cumulative}

\textbf{Periodic timing constraint} is modeled as the proof objective model illustrated in Fig.5.9. To verify this property, we check whether \textit{t}_i + 1 locates in \([\textit{t}_i+T-j, \textit{t}_i+T+j]\).

As shown on the left of Fig.5.9, a \textit{Delay} is to add a \textit{Tj} delay to the input signal \textit{event}. As illustrated on the right of Fig.5.9, an \textit{Extender} then generates a true duration with length \(2^k\textit{j}\) such that the range of true duration is \([\textit{t}_i+T-j, \textit{t}_i+T+j]\). Afterwards, \textit{Within Implies} checks whether \textit{event} occurs within each true duration. Take R27 as an example, if \textit{event} firstly occurs at 0.01s, a corresponding true
Figure 5.8: R30: if the message transmission is failed (msgNormal == false) when the vehicle runs in auto-mode, for the next 20 seconds, the vehicle should not run automatically, i.e., auto == false

duration [0.05, 0.07] is generated, where the existence of the second occurrence is checked.

2. Noncumulative Periodic timing constraint is constructed with the blocks illustrated in Fig 5.10, a «Pulse Generator» is applied to generate signals (square waves) based on Period, Pulse width, Phase delay parameters (see Fig 5.10). Pulse width is a duty cycle specified as a percentage of Period, i.e., Pulse width = Width/Period. Phase delay specifies the delay before the pulse is generated. «Convert» block is used to transform the signal type from double to boolean. The generated signal and event signal from the input port, are then passed to «Within Implies» block. «Within Implies» checks whether event becomes true within each true duration of the generated signal. Take R27 as an example, the Phase delay is specified as 0.04s, Period is 0.05s and Pulse width is 40%, which is illustrated on the right of Fig 5.10. To satisfy this timing constraint, VehicleDynamic should be triggered within each true duration of the generated signal.

Figure 5.9: Cumulative Periodic Timing Constraint

Figure 5.10: Non-cumulative Periodic Timing Constraint

5.3 Energy Constraints in SDV

According to the various modes of vehicles in CAS, the amount of battery consumed for the mechanical motion of wheels is calculated: energy = \textstyle \int_0^t a \cdot v \, dx$ where, $t,$
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

$a$, and $v$ denote running time, coefficient reflects an energy rate associated to the current mode of an individual vehicle, and wheel speed respectively.

**Energy constraint** demarcates that the energy consumption on different modes of an individual vehicle is within $[lower, upper]$. It is specified as $G(lower \leq energy \land energy \leq upper)$ and its POM is depicted in Fig. 5.11, where the value of the energy consumption is obtained through «Data Store Read» block and compared with $upper$ and $lower$.

![Energy constraint diagram](image-url)

**Figure 5.11:** Energy constraint
Chapter 6

Modeling Functional Behaviours in UPPAAL-SMC

We translate the EAST-ADL architectural model illustrated in Fig.3.2 and Simulink/Stateflow model in Fig.4.1 into a networks of STAs in UPPAAL-SMC. In the model, there are three vehicles receiving the information of traffic signs from cooperative environment $\text{CoopEnv}_f$. The functionality traffic sign recognition is modeled as the STA shown in Fig.6.1. The type of traffic signs include stop, left/right turn, straight, minimum/maximum speed limit. The dash line in Fig.6.1 indicates the probabilistic distribution of different traffic signs, i.e., traffic sign occurs randomly with a certain probability. The lead vehicle ($v_1$) will adjust its speed and running direction according to the detected sign types. The two follower vehicles ($v_2$ and $v_3$) will follow the lead vehicle.

![Figure 6.1: STA of SignRecognition](image)

The functionality of the three vehicles are similar. Each vehicle consists of a localEnv $f_p$ and a Controller. Each localEnv consists of three $f_p$s: ComDevice, VehicleDynamic and DriverBehaviour. ComDevice is modeled in Fig.6.3. The
inner behaviour of the vehicle calculating velocity according to the gears and torque is modeled in Fig. 6.2. This \( f_p \) will be triggered to execute periodically with period 50ms and jitter 10ms, i.e., STA of VehicleDynamic will be triggered to update the speed, gear and torque of the vehicle periodically. The computation is implemented as the function \( \text{speedCal()} \), which will be called to update the states and valuation of variables whenever the corresponding transition is taken.

![Figure 6.2: STA of VehicleDynamic](image)

Fig. 6.3(a) captures the behaviour of the sensor sending out signals of position, velocity and direction of vehicle. Fig. 6.3(b) models the behaviour of the sensor receiving signals from other vehicles. It receives information (the position, velocity and direction) from the lead vehicle, and detects whether the lead vehicle is braking periodically. The sensor will send out signals/information when the Controller of the vehicle finishes execution.

The behaviors of the driver is modeled as the STAs shown in Fig. 6.4. Brake Pedal STA represents the behaviors of braking pedal (Fig. 6.4(a)). Control status button STA records whether the driver intends to drive the vehicle manually (Fig. 6.4(b)). The behaviors of controlling gear and steer is modeled as the STA in Fig. 6.4(c) and Fig. 6.4(d) respectively. The driver can take charge of the running state of the vehicle by switching the running state to userCtrl mode using Control Button. The driver can control the vehicle by steer (steer the running direction of vehicle), gear (control the speeds of the wheels) and the brake pedal (to stop the vehicle).

The top view of the Controller modeled in Fig. 6.2 is shown in Fig. 6.5. The controller gets information from four different sensors. After receiving all input signals, the Controller will decide which state should be activated. Fig. 6.6 and Fig. 6.8 describe two states that represent auto mode and userCtrl mode of vehicle. If the driver intends to control the vehicle, the Control Button will transit to userCtrl, indicating that the vehicle is driven manually. When Auto location is active, and the vehicle cannot get the information from the ahead vehicle, then userCtrl will be activated automatically.

Fig. 6.8 describes the behavior that it is of 50% probability that the message transmission is failed. The message state is modeled by STA in Fig. 6.7. When the message is missed for 2s, userCtrl mode will be activated.

The behaviour of leader state in the Stateflow chart is captured by STA illustrated in Fig. 6.9. The leader gets image information from CoopEnv and changes its
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

(a) Sensor of sending signals

(b) Sensor of receiving signals

Running mode according to `signType`, the current running state and velocity. The substates of `leader` state is modeled in Fig. 6.10.

The inner behaviours of `Stop` can be captured in Fig. 6.10(a). Similarly, Fig 6.10(b) and Fig 6.10(c) describe the behaviour of `turnLeft`, `turnRight`. Fig 6.11(a) and Fig 6.11(b) illustrates the STA of `acc` and `dec` state respectively. When the leader vehicle detects a stop sign (`signType == 5`) and if it is in `Auto` mode, `Stop` state in `leader` STA will be activated and the torque will be set to maximum value (the velocity will be decreased rapidly). When the velocity of the vehicle is less than or equal to 0, it will stop and `static` state (shown in Fig. 6.10) will be active. When a minimum speed limit sign is detected or the driver takes control of the vehicle, the `straight` state will be active. When the lead vehicle detects a left turn sign and the `straight` state is active, `turnLeft` state will be active. The vehicle will first decelerate by decreasing the value of its gear until its velocity is less than or equal to 30km/h. Afterwards, the direction of the vehicle will be changed according to its current direction and the detected traffic sign. After turning, the vehicle will increase its velocity to the original velocity before turning.

The inner behaviour `speedUp` and `speedDown` is modeled in Fig 6.12. When the vehicle is decelerating, the torque (torq) of the vehicle will be increased and the gear (gear) will be decreased. Since the velocity of one vehicle is positively proportional to `gear`, the velocity will be increased when `gear` is increased. When the vehicle finishes turning, the torque of the vehicle will be decreased and the gear
Figure 6.4: STA for modeling DriverBehaviour
Figure 6.5: STA of Controller

Figure 6.6: STA of Control state

Figure 6.7: STA for deciding whether the message is missing
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Figure 6.8: STA for modeling Communication state

Figure 6.9: STA for modeling leader state
Figure 6.10: STA for modeling substates of leader state
will be increased to resume to the original velocity.

The inner behaviour of turnLeft state is captured in Fig.6.13. Similarly, when the vehicle detects a turn right sign, it will decelerate until its speed velocity is less than 30km/h. Afterwards, it will turn (its running directions will be changed) and increases its velocity. The acc and dec states (Fig.6.10(d), (e)) represent the states that the vehicle is accelerating or decelerating. When the lead vehicle (v1) detects a minimum speed limit, and the velocity of the vehicle is less than the limit, acc state will be active. The torque of the vehicle will be decreased to order to increase the velocity. In the mean time, the gear of the vehicle will be increased every 40ms until it reaches the maximum value. When the lead vehicle detects a maximum speed limit and the velocity is larger than the limit, it will activate dec state and its velocity will be decreased.

The behaviour of the follower is captured in Fig.6.14. The follow vehicle will decide its behaviour by three super-states: stop, turn and straight, which will be active according to the velocity and running mode of the lead vehicle. It is required to guarantee the distance between the following vehicle and the lead vehicle should be limited in order to ensure the safety. When the follower is in straight state (Fig.6.15(c)), it will always check the distance departing from the leader vehicle by executing checkDistance() function. If the distance is less than the safety distance, then it will activate dec state (Fig.6.15(c)) inside straight state. If the following vehicle is in dec state, which is shown in Fig.6.16(a), it will increase its torque or decrease its gear every 40ms until it reaches 0. If the distance between the following

---

**Figure 6.11:** STA for modeling substates of leader state

---
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Figure 6.12: STA of turnLeft state

(a) speedDown

(b) speedUp
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Figure 6.13: STA of `turnRight` state

(a) speedDown

(b) speedUp
vehicle and the leader vehicle is greater than the safety distance, the vehicle will stop decelerating. When the distance between the following vehicle and the leader vehicle is greater than 500m, in this case, the communication between the two vehicles may be lost. Hence the following vehicle should increase its speed in order to maintain the communication quality.

As presented in Fig. 6.16(b), when the vehicle accelerates, it will decrease its torque and increase its gear every 40ms. If the lead vehicle is braking, the follower will activate stop state (shown in Fig. 6.15(a)). When the follower stops, it will maintain the safety distance with the lead vehicle. When the follow vehicle detects that its running direction and the running direction of the lead vehicle are different, it will turn left/right in order to keep the same running direction with the lead vehicle. In this case, the following vehicle will first reach the turning point of the lead vehicle (recorded in judgeturnpoint), which is illustrated in Fig. 6.14. Similar to lead vehicle, the following vehicle will first decreases the velocity to turn and increase to finish turning. Detailed behaviors of turn state is captured in Fig. 6.17.

If the driver takes control of the vehicle, userCtrl (Fig. 6.18) state will be active. The movement of the vehicle will be controlled by the driver. Hence when the driver pressed the brake pedal, stop state in userCtrl will be active. Detailed behaviours of stop state is captured in Fig. 6.19(a).

However, when the driver increases the velocity of the vehicle, stop state will transit to straight state. In straight mode, the speed of the vehicle can be increased/decreased by changing the gear (gear up or gear down). As shown in Fig. 6.16, the vehicle will adjust its velocity according to the operation by the driver. If the driver steer to left or right, the vehicle will decrease its velocity to turn and increase the velocity to the original velocity after turning (as illustrated in Fig. 6.21 and Fig. 6.22).
Figure 6.15: STA of follower state
Figure 6.16: STA for modeling substates of straight state
Figure 6.17: STA for modeling substates of turn state
Figure 6.18: STA for modeling \textit{userctrl} state
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Figure 6.19: STA of UserCtrl state
Figure 6.20: STA for modeling substates of *straight* state

Figure 6.21: STA for modeling substates of *turnleft* state
Figure 6.22: STA for modeling substates of turnright state
Chapter 7

(Non)-functional Property Translation in UPPAAL-SMC

In this section, we discuss semantics of the extended Periodic, Sporadic, and Comparison timing constraints (XTC) with probabilistic parameters according to TADL [9]. Afterwards, we provide translation of XTC in UPPAAL-SMC. Finally, we present how the energy consumption of an individual vehicle in CAS can be modeled in UPPAAL-SMC.

7.1 Probabilistic Timing Constraints Translation

XTC that we verified follow the weakly-hard (WH) approach [9], which describes that a bounded number of occurrences among all are allowed to violate the constraints. The semantics of the WH \((c, m, k)\) is that a system behavior must satisfy a given timing constraint \(c\) at least \(m\) times out of \(k\) consecutive occurrences. We use object-oriented notation to define the attributes of occurrences of an event except for Comparison timing constraint, which does not related to any event. \(f_p.e\) refers to an event \(e\) of \(f_p\). we apply \(e_i\) and \(e(i)\) to represent the \(i^{th}\) occurrence of \(f_p.e\) and the time point of \(i^{th}\) occurrence respectively.

\(WH(\text{End-to-End}, m, k)\) limits the interval between the occurrences of \(f_p.source\) and corresponding \(f_p.target\). For any \(source_i \in f_p.source\) \((i \leq |f_p.source|)\), out of \(k\) consecutive occurrences of \(f_p.source\), i.e., \(source_i\) to \(source_{i+k}\), at least \(m\) satisfy \(\text{lower} \leq |target(i) - source(i)| \leq \text{upper}\), where \(|target(i) - source(i)|\) denotes the corresponding delay of \(f_p.source\). We have previously proposed the translation pattern of probabilistic End-to-End constraint, which is applicable for the system where \(target_i\) always happens prior to \(source_{i+1}\). To allow the case that \(source_{i+1}\) occurs prior to \(target_i\), we model End-to-End constraint as a spawnable STA [5]. As shown in Fig.7.1(a), whenever \(f_p.source\) occurs, Source STA will activate a spawned STA (STA\(_{sp}(i)\)) from End-to-End STA (in Fig.7.1(b)).

When STA\(_{sp}(i)\) is activated, the clock \(clk\) starts to count from 0 until \(target_i\) occurs. STA\(_{sp}(i)\) will judge whether the time interval from \(source_i\) to \(target_i\) is within the lower and upper bound. According to the judgment, either success or
fail location will be activated, representing that the End-to-End constraint is satisfied or not. Afterwards, \( \text{STA}_{\text{sp}}(i) \) will be inactive. To ensure the correspondence of \( f_p.\text{source} \) and \( f_p.\text{target} \), if target \( i \) doesn’t occur, \( \text{STA}_{\text{sp}}(i) \) will be inactive when target \( i+1 \) occurs. \( \text{WH}(\text{End-to-End}, m, k) \) is specified as: \( Pr[\text{bound}] (\[ \] \:\:\forall \text{source} \in f_p.\text{source} (\neg \text{STA}_{\text{sp}}(i).\text{fail})) \geq P. \)

**WH(Periodic, m, k)** limits the period of the successive occurrences of an event including jitter. As our previous work has provide probabilistic extension of Cumu_Periodic and translation rules to STA, we focus on Noncumu_Periodic. Noncumu_Periodic: For any \( e(i) \in f_p.\text{start} \), at least \( m \) times out of \( k \) occurrences, \( e(i) \) satisfies: \( (i*T-jitter \leq e(i) \leq i*T+jitter) \). In Fig.7.1(c), the STA enforces the \( i^{th} \) event \( \text{event} \) to occur within \([i*T-jitter, i*T+jitter] \). Initially, STA resets the local clock \( \text{clk} \) when it counts to \( \text{jitter} \), which indicates that the time moves backwards by \( \text{jitter} \) time on the time line. When event happen, STA will judge if \( \text{clk} \) is within \([T-2*\text{jitter}, T] \) and changes its current state to either (success or fail) based on the judgment. Afterwards, the STA will wait at reset till period \( T \) and reset \( \text{clk} \). Finally the STA repeats the calculation. \( \text{WH}(\text{Noncumu}_\text{Periodic}, m, k) \) is specified as: \( Pr[\text{bound}] (\[ ] \:\: \neg \text{STA}_{\text{Noncumu}_\text{Periodic}}.\text{fail}) \geq P. \)

**WH(Sporadic, m, k)** describes a single event \( f_p \) that occurs sporadically. For any \( e(i) \in f_p.\text{start} \) s.t. its successive occurrence \( e'(i) \) starts at \( e(i) \), out of \( k \) consecutive occurrences \( e(i) \), i.e., \( e(i) \) to \( e(i+k) \), at least \( m \) sequences satisfy \(|e'(i) - e(i)| \geq \text{min}\), where \( \text{min} \) denotes the minimum time interval between consecutive occurrences. In Fig.7.1(d), initially, STA resets the local clock \( \text{clk} \) when it receives signal \( \text{event} \), which indicates the first occurrence of an event. When STA receives \( \text{event} \) again, it will judge if \( \text{clk} \) is larger than \( \text{min} \) and changes its current state to either (success or fail) based on the judgment. Finally STA reset
clk and repeat calculation. WH(Sporadic,m,k) is specified as: \( Pr[\text{bound}] \left( \left[ \right] \rightarrow \text{STA}_\text{Sporadic}.\text{fail} \right) \geq P \).

WH(Comparison,m,k) limits the comparison relation (including \(<,\leq,\equiv,\geq\) and \(>)\) among timing expressions, where WH(Comparison,m,k) can be expressed as arithmetic. WH(Comparison,m,k) is defined as: out of \( k \) runs, at least \( m \) times the comparison relation between TimingExpr1 and TimingExpr2 is satisfied. WH(Comparison,m,k) is interpreted as: \( Pr[\text{bound}](\left[ \right] \text{TimingExpr1} \prec \text{TimingExpr2}) \geq P \), where \( \prec \in \{<,\leq,\equiv,\geq\text{or}\,\geq\} \).

### 7.2 Energy Constraint Estimation

Fig.7.2 shows the stochastic timed automata (STA) that estimates the energy consumption of the Controller of the lead vehicle with extended arithmetic on clocks: total_energy represents the total energy consumption of the Controller of the lead vehicle. Different modes of the vehicle are corresponded to different locations in the STA. As the energy consumption rate for turning right and turning left mode are the same, we combine the two modes into one location in the STA, i.e., turn-LeftorRight. Similarly, the acceleration and deceleration mode of the Controller are combined into decoracc. Since the energy consumption rate varies with the running mode of the vehicle, e.g., the energy consumes faster when the vehicle is in braking mode than when the vehicle drives in a constant speed, we define total_energy (the rate of total_energy) with ordinary differential equation (ODE) and assign different values to total_energy for different locations in the STA. Parameters \( a, b, c \) and \( d \) are user-defined coefficients to estimate energy consumption of the lead vehicle when it drives in a constant speed, brakes, turns or increases (decreases) its speed respectively. The values of the coefficients indicate the increasing rate of the energy consumption of the Controller and \( b > d > c > a \). The estimation of energy consumption requirement of Controller (R48) will be evaluated in Chap.8.

Also, the four sensors in Fig.5.2(b) will consume energy. As VeDynamicDevice is an imitation of how the velocity of an vehicle can be achieved, its energy consumption is more like the mechanical kinetic energy of the vehicle. While in our case, the velocity and the energy consumption of the vehicle is strongly related to the mode of the vehicle, thus we model the mechanical kinetic energy consumption in Controller (Fig.7.2). The energy of the other three \( f_p \)s, ComDevice, SignRecDevice and VeModeDevice, are modeled in Fig.7.3. In order to simplify calculation, we set constants as the increasing rate of energy consumption of each sensor. When the sensor is working, the energy consumption will increase uniformly. And when the sensors don’t work anymore, the energy consumption will not increase.
Figure 7.2: Energy estimation of \textit{v1Controller}
Chapter 8

Experiment and Verification

Formal verification of functional and non-functional properties are conducted by using SDV and UPPAAL-SMC. Since SDV only supports a sub-set of blocks in Simulink block library, several blocks need to be replaced from the original model, e.g., look-up table block and random value generator. The proof objective models that specify the functional and non-functional properties are constructed, which are illustrated in the following figures.

Figure 8.1: When \( v1 \) runs automatically and if there is no message received by the vehicle for a certain time, the running mode of the vehicle should be changed to manual mode within a certain time: \( G (G_{[0,200]} v1.Auto\text{==true} \land v1.msg\text{==false} \implies F_{[0,2000]} v1.userCtrl\text{==true}) \)

![Figure 8.1](image1.png)

Figure 8.2: When \( v1 \) runs automatically with a constant speed, if it detects a stop sign, it should start to brake in 500ms: \( G (v1.Auto == \text{true} \land v1.const == \text{true} \land \text{signType} == 5 \implies F_{[0,500]} v1.stop == \text{true}) \)

![Figure 8.2](image2.png)

Functional properties of CAS (R1-R26) can be measured using observer STAs. Observer STA describes the time delay between two transitions, i.e., two events. An event can be indicated by a synchronization channel, changes of states or data. Trigger STA represents the events that are represented by changes of states or data. Fig 8.15 shows how functional requirements can be modeled in observer STA. For
Figure 8.3: When the lead vehicle (v1) runs automatically with a constant speed, if it detects left turn sign, it should start to turn left in 200ms: \( G (v1.auton == true \land v1.const == true \land signType == 4 \Rightarrow F_{[0,200]} v1.turnLeft == true) \)

Figure 8.4: If v1 is driven by the driver and it is running with a constant speed, the driver steers to the left, the vehicle will turn left within 200ms: \( G (v1.userCtrl == true \land v1.const == true \land v1.steerReq == 1 \Rightarrow F_{[0,200]} v1.turnLeft == true) \)

Figure 8.5: If v1 is driven by the driver and it is running at a constant speed, the driver brakes the car, the vehicle will slow down. \( G (v1.userCtrl == true \land v1.const == true \land v1.brakeReq == 2 \Rightarrow F_{[0,200]} v1.brake == true) \)

Figure 8.6: When v1 is driven by the driver and it is running at a constant speed, if the driver gears up, the vehicle will accelerate. \( G (v1.userCtrl == true \land v1.const == true \land v1.gearReq > v1.gear \Rightarrow F_{[0,200]} v1.acc == true) \)
Figure 8.7: When the vehicles are running horizontally, following vehicle should not run ahead of lead vehicle. \( \text{G} (v_1.dx = 1 \land v_2.dx = 1 \land v_3.dx = 1 \implies v_1.x > v_2.x) \)

Figure 8.8: When the lead vehicle detects a stop sign, the three vehicles should stop in 5s. \( \text{G} (v_1.auto == \text{true} \land \text{signType} == 5 \implies \text{F}_{[0,5000]} v_1.static == \text{true} \land v_2.static == \text{true} \land v_3.static == \text{true}) \)

Figure 8.9: Noncumulative Periodic timing constraint: A Periodic acquisition of VehicleDynamic of \( v_1 \) must be carried out for every 50 ms with a jitter 10 ms. \( \text{G} (v_1.velIn == \text{true} \implies v_1.velIn == \text{false} \cup_{[40,60]} v_1.velIn == \text{true}) \)

Figure 8.10: Cumulative Periodic timing constraint of R27
Figure 8.11: Sporadic constraint: After the running mode of the lead vehicle is changed to manual mode because of the failed message transmission, the driver should not change it into auto mode within 20 seconds. \( G (v1.msgNormal == false \implies v1.auto == false) \cup_{[2000, \infty)} v1.auto == true \)

Figure 8.12: An Execution constraint applied on v3Controller, which measured from the inputs ports Avel to output ports gearReq, torqueReq, and brake, should be between 100 ms and 300 ms; \( G (v3.ctrlIn \implies F_{[100,300]} v3.ctrlOut) \)

Figure 8.13: An Execution constraint applied on v3Controller, which measured from the inputs ports Avel to output ports gearReq, torqueReq, and brake, should be between 100 ms and 300 ms; \( G (v3.ctrlIn \implies F_{[100,300]} v3.ctrlOut) \)
Figure 8.14: An execution constraint applied on ComDevice of v1, which measured from the inputs ports torqueSet, gearSet and brake to output ports wheelSpeed, direct should be between 100 ms and 150 ms; $G (v1.sdrIn \implies F_{[100,300]} v1.sdrOut)$
R1, the observer STA will start to count the elapsed time when communication failure occurs. It stops counting when the driver of this vehicle takes control of the vehicle. An STA is used to indicate the time instant at which the driver takes control of the car (the manual mode is activated). Whenever the user takes control (in_uc[0]==1), a signal will be sent to the STA in Fig. 8.15(a) and time counting will be ceased.

![Figure 8.15: Observer STA of R1 and its triggering](image)

Verification of timing properties (R27-R50) are done by adapting observer STAs in Fig 8.16. R27-R29 are verified by using the STAs in Fig 8.16(a) and (b), where the VeDynamicDevice is triggered periodically with jitter 10ms. R30-R32 are Sporadic constraints: if communication is missing, the driver should take control of the vehicle for at least 2s (Fig 8.16(c)). The Execution timing constraint of Controller and ComDevice are considered in STAs shown in Fig 8.16(d), (e). The End-to-End constraints measured from the Controller of the lead vehicles to the Controller of the follow vehicles (R39 and R40) can be checked by employing the STAs in Fig 8.16(f). R41-R43 measure the time duration from the Controller of each vehicle receiving signal from the sensors to sending information to other vehicles (Fig 8.16(g)). The synchronization of the input ports of the Controller can be modeled in STA in Fig 8.16(h).

Table 8.1 depicts the verification result of a list of functional (R1-R26) and non-functional (R27-R50) properties. The experiments are performed on a computer with Intel Core i3 CPU and 4GB of RAM. For Periodic timing constraints (R27-R29), results are provided in two categories: cumulative (R27.1, R28.1, R29.1) and non-cumulative (R27.2, R28.2, R29.2). The non-functional properties (R27-R50) are all satisfied in both UPPAAL-SMC and SDV. The functional properties R1-R22, R25-R26 are established as valid in SDV and are satisfied in UPPAAL-SMC with confidence greater than 95%, while R23-R24 is invalid in both SDV and UPPAAL-SMC.

The invalid property R23 is identified using SDV and UPPAAL-SMC. The counter-examples (CE) generated by SDV is shown in Fig 8.19. We perform the simulation in UPPAAL-SMC to obtain the error trace of CE, which is illustrated in
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

(a) Cumulative Periodic

(b) Noncumulative Periodic

(c) Sporadic

(d) Synchronization

Figure 8.16: UPPAAL-SMC STAs for timing constraints verification
Figure 8.17: UPPAAL-SMC STAs for End-to-End timing constraints verification

Figure 8.18: UPPAAL-SMC STAs for End-to-End timing constraints verification
Table 8.1: Logic expression and verification results of the requirements

| Req | Type | Expression | Result | Time (min) | Memory (Mb) | CPU (%) |
|-----|------|------------|--------|------------|-------------|---------|
| R1  | SDV  | G (G[0,2000] \ v1.auto==true ∧ v1.msgNormal==false ⇒ F[0,2000]) \ v1.userCtrl==true) | valid | 14.95 | 1675.47 | 0.55 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬Commufail(0).fail)) | [0.902, 1] | 7.61 | 38.48 | 22.40 |
| R2  | SDV  | G (G[0,200] \ v2.auto==true ∧ v2.msgNormal==false ⇒ F[0,200]) \ v2.userCtrl==true) | valid | 18.45 | 1782.39 | 0.29 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬Commufail(1).fail)) | [0.902, 1] | 6.83 | 40.49 | 22.49 |
| R3  | SDV  | G (G[0,200] \ v3.auto==true ∧ v3.msgNormal==false ⇒ F[0,200]) \ v3.userCtrl==true) | valid | 26.3 | 1932.35 | 0.69 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬Commufail(2).fail)) | [0.902, 1] | 7.89 | 42.88 | 22.62 |
| R4  | SDV  | G (v1.userCtrl==true ∧ v1.auto==true ∧ signType==5 ⇒ F[0,500]) \ v1.brake==true) | valid | 46.8 | 1964.57 | 10.25 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R4TR.fail)) | [0.902, 1] | 21.63 | 39.27 | 24.75 |
| R5  | SDV  | G (v1.userCtrl==true ∧ v1.const==true ∧ signType==4 ⇒ F[0,200]) \ v1.turnLeft==true) | valid | 40.08 | 2072.23 | 11.21 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R5TR.fail)) | [0.902, 1] | 25.33 | 39.20 | 24.52 |
| R6  | SDV  | G (v1.userCtrl==true ∧ v1.const==true ∧ signType==3 ⇒ F[0,200]) \ v1.turnRight==true) | valid | 40.28 | 2026.57 | 6.84 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R6TR.fail)) | [0.902, 1] | 25.67 | 39.20 | 24.78 |
| R7  | SDV  | G (v1.userCtrl==true ∧ v1.const==true ∧ v1.steerReq==1 ⇒ F[0,200]) \ v1.turnLeft==true) | valid | 99.95 | 2144.16 | 8.64 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R7TR.fail)) | [0.902, 1] | 27.6 | 39.10 | 24.67 |
| R8  | SDV  | G (v1.userCtrl==true ∧ v1.const==true ∧ v1.steerReq==2 ⇒ F[0,200]) \ v1.turnRight==true) | valid | 157.68 | 1852.91 | 1.78 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R8TR.fail)) | [0.902, 1] | 24.48 | 39.09 | 24.65 |
| R9  | SDV  | G (v1.userCtrl==true ∧ v1.const==true ∧ v1.brakeReq==2 ⇒ F[0,200]) \ v1.brake==true) | valid | 80.46 | 1667 | 1.46 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R9TR.fail)) | [0.902, 1] | 28.2 | 39.26 | 24.63 |
| R10 | SDV  | G (v1.userCtrl==true ∧ v1.const==true ∧ v1.gearReq>v1.gear ⇒ F[0,200]) \ v1.acc==true) | valid | 6.98 | 1882.97 | 2.95 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R10TR.fail)) | [0.902, 1] | 29.16 | 38.92 | 24.52 |
| R11 | SDV  | G (v1.userCtrl==true ∧ v1.const==true ∧ v1.gearReq<v1.gear ⇒ F[0,200]) \ v1.dec==true) | valid | 259.98 | 1430 | 1.85 |
|     | UPPAAL-SMC | Pr[≤3000] ([ ] (¬R11TR.fail)) | [0.902, 1] | 29.03 | 39.24 | 24.57 |
| Req | Type       | Expression                                                                                                                                                                                                 | Result | Time (min) | Memory (Mb) | CPU (%) |
|-----|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|------------|-------------|---------|
| R12 | SDV        | $G (v1.dx==1 \land v2.dx==1 \land v3.dx==1 \implies v1.x>v2.x)$                                                                                                                                          | valid  | 14.02     | 1320        | 3.65    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack dx[0]==1 \land dy[0]==0 \land dx[1]==1 \land dy[1]==0 \land (in.Str[0] \lor in.dec[0] \lor in.Str[0] \lor in.acc[0] \lor in_constSpeed[0]) \land (in.Str[1] \lor in.Str[1]) \implies x[0] > x[1]) \geq 0.95$ | valid  | 13.27     | 57.06       | 24.31   |
| R13 | SDV        | $G (v1.dx==1 \land v2.dx==1 \land v3.dx==1 \implies v2.x>v3.x)$                                                                                                                                              | valid  | 35.37     | 2085        | 3.05    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack dx[1]==1 \land dy[1]==0 \land dx[2]==1 \land dy[2]==0 \land (in.Str[2] \lor in.Str[2] \lor in_constSpeed[1]) \land (in.Str[2] \lor in.Str[2]) \implies x[1] > x[2]) \geq 0.95$ | valid  | 11.43     | 55.67       | 24.42   |
| R14 | SDV        | $G (v1auto \land signType==5 \implies F_{[0.5000]} (v1static \land v2static \land v3static))$                                                                                                           | valid  | 4.67      | 1903        | 4.91    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg allstop.fail) \geq 0.95$                                                                                                                                              | valid  | 20.28     | 52.47       | 23.10   |
| R15 | SDV        | $G (v1.const==true \land v2.const==true \land v1.vel>v2.vel \implies F_{[0.200]} (v2.acc==true))$                                                                                                     | valid  | 5.10      | 2095        | 4.20    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg acc(1).fail) \geq 0.95$                                                                                                                                              | valid  | 21.52     | 43.68       | 22.43   |
| R16 | SDV        | $G (v2.const==true \land v3.const==true \land v2.vel>v3.vel \implies F_{[0.200]} (v3.acc==true))$                                                                                                     | valid  | 0.15      | 2038        | 38.76   |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg acc(2).fail) \geq 0.95$                                                                                                                                              | valid  | 19.43     | 40.46       | 23.21   |
| R17 | SDV        | $G (v1.const==true \land v2.const==true \land v1.vel<v2.vel \implies F_{[0.200]} (v2.dec==true))$                                                                                                     | valid  | 0.15      | 1963        | 23.04   |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg R17TR.fail) \geq 0.95$                                                                                                                                              | valid  | 27.31     | 39.10       | 24.60   |
| R18 | SDV        | $G (v2.const==true \land v3.const==true \land v2.vel<v3.vel \implies F_{[0.200]} (v3.dec==true))$                                                                                                     | valid  | 11.52     | 2026        | 3.99    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg R18TR.fail) \geq 0.95$                                                                                                                                                | valid  | 26.89     | 39.07       | 24.73   |
| R19 | SDV        | $G (v1.const==true \land v2.const==true \land dist12>500 \implies v2.acc==true)$                                                                                                                            | valid  | 0.04      | 1961        | 4.38    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg R19TR.fail) \geq 0.95$                                                                                                                                                | valid  | 27.20     | 39.16       | 24.84   |
| R20 | SDV        | $G (v1.const==true \land v2.const==true \land dist12>500 \implies v3.acc==true)$                                                                                                                            | valid  | 0.01      | 1961        | 4.38    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg R20TR.fail) \geq 0.95$                                                                                                                                                | valid  | 26.53     | 38.92       | 24.65   |
| R21 | SDV        | $G (v1.const==true \land v2.const==true \land dist12<safeDis \implies v2.dec==true)$                                                                                                                      | valid  | 11.13     | 2054        | 4.09    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg dec(1).fail) \geq 0.95$                                                                                                                                              | valid  | 22.12     | 43.23       | 22.60   |
| R22 | SDV        | $G (v2.const==true \land v3.const==true \land dist23<safeDis \implies v3.dec==true)$                                                                                                                      | valid  | 0.03      | 2234        | 3.92    |
|     | UPPAAL-SMC | $Pr[\leq 3000][\lbrack \lbrack \neg dec(1).fail) \geq 0.95$                                                                                                                                              | valid  | 20.59     | 43.27       | 22.81   |
| Req | Type         | Expression                                                                 | Result | Time (min) | Memory (Mb) | CPU (%) |
|-----|--------------|----------------------------------------------------------------------------|--------|------------|-------------|---------|
| R23 | SDV          | G (v1TurnLeft \implies F_{[0.500]} RunInSameLine)                         | valid  | 15.49      | 1492.96     | 1.82    |
| R24 | UPPAAL-SMC   | Pr[\leq 3000] (\exists \turnLeft Time12.fail) \geq 0.95                  | valid  | 40.93      | 42.46       | 24.79   |
|     | SDV          | G (v2TurnLeft \implies F_{[0.500]} RunInSameLine)                        | valid  | 1.10       | 1918        | 3.48    |
| R25 | UPPAAL-SMC   | Pr[\leq 3000] (\exists \turnLeft Time23.fail) \geq 0.95                  | valid  | 42.71      | 47.23       | 23.80   |
| R26 | UPPAAL-SMC   | Pr[\leq 3000] (\exists \turnRight Time23.fail) \geq 0.95                 | valid  | 39.13      | 55.11       | 24.63   |
|     | SDV          | G (v1.turnRight==true \implies F_{[0.500]} v1.turnRight==false \land \neg v2.turnRight==false \land v1.x==v2.x \lor v1.y==v2.y) | valid  | 31.26      | 2253        | 3.86    |
|     | UPPAAL-SMC   | Pr[\leq 3000] (\exists \turnRight Time23.fail) \geq 0.95                 | valid  | 40.24      | 56.20       | 24.89   |
| R27 | UPPAAL-SMC   | Pr[\leq 3000] (\neg VehicleDynamic(0).cumulative.fail) \geq 0.95        | valid  | 7.18       | 51.57       | 24.64   |
|     | SDV          | G (period_{pre}==0 \land period==1 \implies F_{[0.20]} velIn)           | valid  | 8.42       | 1819.24     | 1.22    |
| R28 | UPPAAL-SMC   | Pr[\leq 3000] (\neg VehicleDynamic(0).noncumulative.fail) \geq 0.95    | valid  | 9.17       | 51.19       | 23.25   |
|     | SDV          | G (vel2In==1 \implies F_{[40.60]} vel2In==1)                             | valid  | 0.71       | 1285        | 0.5     |
|     | UPPAAL-SMC   | Pr[\leq 3000] (\neg VehicleDynamic(1).cumulative.fail) \geq 0.95        | valid  | 7.28       | 42.70       | 24.65   |
|     | SDV          | G (p2_{pre}==0 \land p2==1 \implies F_{[0.20]} vel2In)                  | valid  | 10.98      | 2002.17     | 8.10    |
| R29 | UPPAAL-SMC   | Pr[\leq 3000] (\neg VehicleDynamic(1).noncumulative.fail) \geq 0.95    | valid  | 5.6        | 51.79       | 24.66   |
|     | SDV          | G (vel3In==1 \implies F_{[40.60]} vel3In==1)                             | valid  | 0.65       | 54.69       | 8.3     |
|     | UPPAAL-SMC   | Pr[\leq 3000] (\neg VehicleDynamic(2).cumulative.fail) \geq 0.95        | valid  | 7.23       | 42.70       | 24.69   |
|     | SDV          | G (p3_{pre}==0 \land p3==1 \implies F_{[0.20]} vel3In)                  | valid  | 6.97       | 2084        | 8.30    |
| R30 | UPPAAL-SMC   | Pr[\leq 3000] (\neg VehicleDynamic(2).noncumulative.fail) \geq 0.95    | valid  | 6.13       | 51.79       | 24.49   |
|     | SDV          | G (\neg msgNormal \implies \neg autoU_{[2000,\infty]} auto==true)    | valid  | 4h         | 1588.14     | 7.27    |
| R31 | UPPAAL-SMC   | Pr[\leq 3000] (\neg sporadic(0).fail)) \geq 0.95                      | valid  | 0.001      | 30.96       | 0.9     |
|     | SDV          | G (\neg msgNormal2 \implies \neg auto2U_{[2000,\infty]} auto2==true)  | valid  | 49.15      | 2253        | 38.85   |
| R32 | UPPAAL-SMC   | Pr[\leq 3000] (\neg sporadic(1).fail)) \geq 0.95                      | valid  | 0.002      | 30.96       | 1.2     |
|     | SDV          | G (\neg msgNormal \implies \neg auto3U_{[2000,\infty]} auto3==true)   | valid  | 59.15      | 2125        | 36.40   |
| R33 | UPPAAL-SMC   | Pr[\leq 3000] (\neg sporadic(2).fail)) \geq 0.95                      | valid  | 23:03      | 1167.54     | 22.24   |
|     | SDV          | G (t_{pre}==0 \land r>0 \implies F_{[100,300]} s==r)                   | valid  | 19.7       | 1661        | 7.31    |
| R34 | UPPAAL-SMC   | Pr[\leq 3000] (\neg v1Controller\_execution\_fail) \geq 0.95           | valid  | 7.7        | 51.67       | 24.71   |
|     | SDV          | G (r2_{pre}==0 \land r2>0 \implies F_{[100,300]} s2==r2)               | valid  | 1.38       | 1493        | 3.63    |
| R35 | UPPAAL-SMC   | Pr[\leq 3000] (\neg v3Controller\_execution\_fail) \geq 0.95           | valid  | 6.5        | 42.61       | 24.72   |
|     | SDV          | G (r3_{pre}==0 \land r3>0 \implies F_{[100,300]} s3==r3)               | valid  | 1.08       | 1551        | 20.3    |
| R36 | UPPAAL-SMC   | Pr[\leq 3000] (\neg sensor1\_execution\_fail) \geq 0.95               | valid  | 7.12       | 43.24       | 24.79   |
| Req  | Type         | Expression                                                                 | Result | Time (min) | Memory (Mb) | CPU (%) |
|------|--------------|----------------------------------------------------------------------------|--------|------------|-------------|---------|
| R37  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor2\_execution\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 1.23 \quad 12.69 \quad 8.07 |        |            |             |         |
| R38  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor3\_execution\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 2.26 \quad 1515.98 \quad 3.41 |        |            |             |         |
| R39  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor3\_execution\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 9.91 \quad 35.12 \quad 24.56 |        |            |             |         |
| R40  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor3\_execution\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 10.08 \quad 1204.86 \quad 5.96 |        |            |             |         |
| R41  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{execv1\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 0.04 \quad 1961 \quad 4.38 |        |            |             |         |
| R42  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{execv2\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 7.28 \quad 42.70 \quad 24.76 |        |            |             |         |
| R43  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{execv3\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 7.47 \quad 42.70 \quad 24.67 |        |            |             |         |
| R44  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor\_controller\_synchronization\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 7.33 \quad 42.70 \quad 24.77 |        |            |             |         |
| R45  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor\_controller\_synchronization\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 5.98 \quad 50.84 \quad 24.51 |        |            |             |         |
| R46  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor\_controller\_synchronization\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 105.75 \quad 1197.67 \quad 6.56 |        |            |             |         |
| R47  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{sensor\_controller\_synchronization\_fail}\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 7.33 \quad 42.37 \quad 24.52 |        |            |             |         |
| R48  | SMC          | \(\text{E} \leq 3000\) \left(\text{max: energy\_braking\_energy}\right) \quad 7944\pm1172 \quad 8.7 \quad 51.44 \quad 24.58 |        |            |             |         |
| R49  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{energy\_con} \leq 30\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 18.05 \quad 1000.8 \quad 2.05 |        |            |             |         |
| R50  | ÜPPAAL-SMC   | \(\text{Pr} \leq 3000\) \left[\left(\neg \text{energy\_sen} \leq 5\right)\right] \geq 0.95 \quad \Rightarrow \quad \text{valid} \quad 20.2 \quad 80.9 \quad 24.02 |        |            |             |         |
Formal Analysis of Non-functional Properties for a Cooperative Automotive System

Fig 8.20. DirectX and DirectY represent the directions of the three vehicles and posx and posy indicate the position of the three vehicles. After analyzing CE, the cause of errors was found: After the lead vehicle turns left, the following vehicle begins to turn left after it detects that the running direction of the lead vehicle is changed. The two vehicles turn left at different locations. We modify SDV and UPPAAL-SMC models according to the CE: a turning location where the lead vehicle turns is recorded and sent to the following vehicle. Hence all the following vehicles should turn at the same turning location to ensure that the three vehicles are still in a lane after turning. After the refinement, R23 and R24 are satisfied in both SDV and UPPAAL-SMC.

Figure 8.19: CE in SDV: In (b), leadVe.y equals to followVe.y) from 0 to 8s, which indicates that the two vehicles run in the same lane. In (a), signType becomes “turn left (4)” at 8s and the lead vehicle begins to turn left. However, the following vehicle turns left at a different location (leadVe.x is not equal to followVe.x at 9s). Hence, the “x” is unequal after turning, which means that the two vehicles run on different lanes. Thus R23 and R24 is violated.

Probability Estimation query is applied on R1-R47, R49-R50, indicating that the property can be verified as valid within [0.902, 1]. For estimate the maximum energy consumption for braking the vehicles (R48), we adapt Expected value query
Figure 8.20: CE in SMC: At the beginning, three vehicles run towards direction (1,0). The three values of posy are the same, which indicates that the three vehicles are in the same lane. After a left turn sign is detected, the lead vehicle starts to turn left. Since the first follower turns left at (184, 0) and the second turns left at (157, 0), the three values of posx are not equal after turning, which indicates the three vehicle are not running in the same lane. Thus R5 is violated.

Figure 8.21: Probabilistic distribution of the expected energy consumption
and generate probability distribution diagram (Fig. 8.21) of the expected energy consumption. It illustrates that the average of energy consumption is approx. 7944J and the expected energy consumption is always less than 30kJ. The total energy of the lead vehicle is estimated with Simulation query for 100 times within 300 time units. Fig. 8.22 shows the result of one time simulation. At the beginning, the lead vehicle runs at constant speed, hence the total energy consumption increases constantly. When it recognizes the minimum speed limit sign (signType = 2), the vehicle starts to accelerate thus energy is consumed faster. The total energy then increases constantly with slight fluctuation due to energy consumption of the sensor and the camera. When the vehicle recognizes a turn right sign (signType = 3), it turns right from 1.55s to 1.88s, during which the energy is consumed faster than when the vehicle is running constantly. The vehicle finally recognizes a stop sign (signType = 5) at 4.46s and stops at 4.84s. The total energy consumed during braking is 18.33kJ, which is less than 30kJ, which validates R48. After the vehicle stops, the sensors and the camera still work, thus the consumed energy increases slowly.

Figure 8.22: Total energy consumption estimation of the lead vehicle

![Total energy consumption estimation of the lead vehicle](image_url)
Chapter 9

Verification of Comparison Constraint in UPPAAL-SMC

For comparison constraint (R47), which states that the execution time interval measured from v1Controller to v2Controller needs to be greater than or equal to the sum of the worst-case execution time of \( f_p \)’s, can be specified as:

\[
\phi = [ \text{et}_{\text{con}1} = = \text{w}_{\text{con}1} \land \text{et}_{\text{con}2} = = \text{w}_{\text{con}2} \land \text{et}_{\text{sen}1} = = \text{w}_{\text{sen}1} \land \text{et}_{\text{sen}2} = = \text{w}_{\text{sen}2} \implies \text{w}_{\text{con}1} + \text{w}_{\text{con}2} + \text{w}_{\text{sen}1} + \text{w}_{\text{sen}2} \geq \text{EndtoEnd} ].
\]

We verify this requirement in UPPAAL-SMC with the query:

\[
Pr[\leq 3000] \phi \geq 0.95 \tag{9.1}
\]

which means that the probability that \( \phi \) is satisfied should be greater than or equal to 0.95. However, UPPAAL-SMC cannot give the decidable result because of the large state space. Hence we transform the “always” operator with its dual operator “eventually” and then verify that the probability that the negation of \( \phi \) is satisfied is less than or equal to 0.05, i.e.,

\[
Pr[\leq 3000] \neg \phi \leq 0.05 \tag{9.2}
\]

In CTL, \( A[\ ] \psi \equiv E< > \neg \psi \). Here we want to validates the equivalence between (9.1) and (9.2) by specifying same requirements into the two different types of formulas and compare the verification results. We applied the existed case studies of Fischer protocol and train gate system provided in UPPAAL-SMC. In Fischer protocol example, we check the mutual exclusion property that two processes can get into the critical section at the same time. We verify

\[
Pr[\leq 3000](\forall (i : id) \forall (j : id) P(i).cs \land P(j).cs \implies i = j) \geq p \tag{9.3}
\]

\[
Pr[\leq 3000](< > \neg (\forall (i : id) \forall (j : id) P(i).cs \land P(j).cs \implies i = j)) \leq 1 - p \tag{9.4}
\]
Table 9.1: Verification Results of Mutual Exclusion in Fischer Protocol Example

| Expression                                                                 | Result | Time (s) | No. of Runs | Mem (Kb) |
|----------------------------------------------------------------------------|--------|----------|-------------|----------|
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.95$ | valid  | 0.003    | 140         | 26188    |
| $\Pr_{\leq 3000} (\langle > \neg (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j)) \leq 0.05$ | valid  | 8.672    | 140         | 26188    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.9$ | valid  | 0.009    | 67          | 26752    |
| $\Pr_{\leq 3000} (\langle > \neg (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j)) \leq 0.1$ | valid  | 4.175    | 67          | 26752    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.8$ | valid  | 0.007    | 60          | 27688    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.7$ | valid  | 3.081    | 60          | 27688    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.6$ | valid  | 0.002    | 53          | 26996    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.5$ | valid  | 3.263    | 53          | 26996    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.4$ | valid  | 0.007    | 31          | 27584    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.3$ | valid  | 1.755    | 31          | 27584    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.2$ | valid  | 0.005    | 26          | 27724    |
| $\Pr_{\leq 3000} (\forall i : \text{id} \forall j : \text{id} P(i).cs \land P(j).cs \implies i == j) \geq 0.1$ | valid  | 1.487    | 26          | 27740    |

The verification results of (9.3) and (9.4) are shown in Table 9.1. The verification results of (9.3) and (9.4) are the same. We then conduct the same verification for the mutual exclusion property in train gate system, i.e., only one train can cross the gate at one time.

$$Pr[\leq 100](\forall i : \text{id})\forall (j : \text{id})\text{Train}(i).\text{cross} \land \text{Train}(j).\text{cross} \implies i = j \geq p \text{ (9.5)}$$

$$Pr[\leq 100](\langle > \neg (\forall i : \text{id})\forall (j : \text{id})\text{Train}(i).\text{cross} \land \text{Train}(j).\text{cross} \implies i = j)) \leq 1 - p \text{ (9.6)}$$

Table 9.2 shows that verification result of (9.5) and (9.6) are the same. From the above experimental results, we can conclude that (9.1) and (9.2) are equivalent. By verifying the negation of R47 is satisfied with the probability at most 0.05, we can get the decidable result of R47.
### Table 9.2: Verification Results of Mutual Exclusion in Train Gate Example

| Expression                                                                 | Result | Time (s) | No. of Runs | Mem (Kb) |
|----------------------------------------------------------------------------|--------|----------|-------------|----------|
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.95 | valid  | 0.004    | 140         | 26988    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.05 | valid  | 0.078    | 140         | 26988    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.85 | valid  | 0.003    | 126         | 26980    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.15 | valid  | 0.003    | 126         | 26996    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.75 | valid  | 0.004    | 111         | 27080    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.25 | valid  | 0.003    | 96          | 27436    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.65 | valid  | 0.002    | 96          | 27260    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.35 | valid  | 0.003    | 81          | 27272    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.55 | valid  | 0.057    | 81          | 27272    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.45 | valid  | 0.003    | 67          | 27308    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.45 | valid  | 0.032    | 67          | 27328    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.55 | valid  | 0.005    | 52          | 26752    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.35 | valid  | 0.041    | 52          | 26752    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.65 | valid  | 0.003    | 37          | 27896    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.25 | valid  | 0.007    | 37          | 27840    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.75 | valid  | 0.007    | 23          | 27840    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.15 | valid  | 0.007    | 23          | 27840    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.85 | valid  | 0.0.007  | 8           | 27840    |
| Pr $\leq 100$ ([ ] for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\geq$ 0.05 | valid  | 0.007    | 8           | 27840    |
| Pr $\leq 100$ (< > ¬ (for all (i:id t) for all (j:id t) Train(i).Cross $\land$ Train(j).Cross $\implies$ i $\neq$ j) $\leq$ 0.95 | valid  | 0.0.007  | 8           | 27840    |
Chapter 10

Related Work

In the context of EAST-ADL, efforts on the integration of EAST-ADL and formal techniques based on timing constraints were investigated in several works [14–16], which are however, limited to the executonal aspects of system functions without addressing energy-aware behaviors. Kang [17], Goknil et al. [18] and Seceleanu et al. [19] defined the execution semantics of both the controller and the environment of industrial systems in CCSL [20] which are also given as mapping to UPPAAL models amenable to model checking. In contrast to our current work, those approaches lack precise stochastic annotations specifying continuous dynamics in particular regarding different energy consumption rates during execution. Though, Kang et al. [21, 22] and Marinescu et al. [23] present both simulation and model checking approaches of SIMULINK and UPPAAL-SMC on EAST-ADL models, neither formal specification nor verification of extended EAST-ADL timing constraints with probability were conducted. Our approach is a first application on the integration of EAST-ADL and formal V&V techniques based on a composition of energy- and probabilistic extension of EAST-ADL/TADL constraints. Gholami [24] extended the block library of SIMULINK in order to specify basic LTL properties and to transform them into assertions that are verifiable by SDV. However, this work is limited to functional properties (i.e., Neither energy nor timing constraints are addressed) and restricted to specify a subset of LTL.
Chapter 11

Conclusion

We present an approach to perform V&V on nonfunctional properties of (cooperative)-automotive systems using SDV and UPPAAL-SMC at the early design phase: 1. ET requirements are translated into POMs to perform formal verification using SDV; 2. Probabilistic extension of EAST-ADL constraints is defined and the semantics of the Xtc is translated into analyzable UPPAAL-SMC models for formal verification; 3. A set of mapping rules is proposed to facilitate the guarantee of translation and V&V are performed on the Xtc. As ongoing work, a dedicated plugin that directly provides translation of LTL2SIMULINK and of S/S to UPPAAL-SMC in fully automatic will supplement our tool chain, A-BeTA (Aβ: EAST-ADL Behavioral Modeling and Translation into Analyzable Model).
Acknowledgment

This work is supported by the National Natural Science Foundation of China and International Cooperation & Exchange Program (46000-41030005) within the project EASY.
References

[1] “ISO 26262-6: Road vehicles functional safety part 6. Product development at the software level. International organization for standardization, geneva,” 2011.

[2] “IEC 61508: Functional safety of electrical, electronic, programmable electronic safety related systems. International organization for standardization, geneva,” 2010.

[3] A. Legay and M. Viswanathan, “Statistical model checking: Challenges and perspectives,” STTT, vol. 17, no. 4, pp. 369–376, 2015.

[4] A. David, D. Du, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, and S. Sedwards, “Statistical model checking for stochastic hybrid systems,” in HSB, ser. EPTCS 92, 2012, pp. 122–136.

[5] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen, “Uppaal SMC tutorial,” STTT, vol. 17, no. 4, 2015, pp. 397–415.

[6] EAST-ADL Consortium, “EAST-ADL domain model specification v2.1.9,” Maenad European Project, Tech. Rep., 2011. http://www.maenad.eu/public/EAST-ADL-Specification_M2.1.9.1.pdf

[7] H. Blom, H. Lönn, F. Hagl, Y. Papadopoulos, M.-O. Reiser, C.-J. Sjöstedt, D.-J. Chen, F. Tagliabò, S. Torchiaro, S. Tucci et al., “EAST-ADL: An architecture description language for automotive software-intensive systems,” Embedded Computing Systems: Applications, Optimization, and Advanced Design, 2013, p. 456.

[8] “Simulink/StateFlow,” http://www.mathworks.nl/products/.

[9] H. Blom, L. Feng, H. Lönn, J. Nordlander, S. Kuntz, B. Lisper, S. Quinton, M. Hanke, M.-A. Peraldi-Frati, A. Goknil, J. Deanton, G. B. Defo, K. Klobedanz, M. zhan, and O. Honcharova, “Timing Model, Tools, algorithms, languages, methodology, USE cases,” TIMMO-2-USE, Tech. Rep., 2012.

[10] “UPPAAL-SMC,” http://people.cs.aau.dk/~adavid/smc/.

[11] C. Baier, J.-P. Katoen, and K. G. Larsen, Principles of Model Checking. MIT press, 2008.
[12] E. Y. Kang, D. Mu, and L. Huang, “Verification and validation of a cooperative automotive system,” Tech. Rep., 2017. [Online]. Available: https://sites.google.com/site/kangeu/home/publications

[13] E. Y. Kang, D. Mu, L. Huang, and Q. Lan, “Model-based analysis of timing and energy constraints in an autonomous vehicle system,” IEEE QRS-C, July 2017, pp. 525–532.

[14] T. N. Qureshi, D.-J. Chen, and M. Törngren, A Timed Automata-Based Method to Analyze EAST-ADL Timing Constraint Specifications. Springer, 2012.

[15] E. Kang, E. P. Enois, R. Marinescu, C. Seceleanu, P. Schobbens, and P. Pettersson, “A methodology for formal analysis and verification of EAST-ADL models,” Reliability Engineering and System Safety Journal, 2013, pp. 127 – 138.

[16] E. Kang, P. Schobbens, and P. Pettersson, “Verifying functional behaviors of automotive products in EAST-ADL2 using UPPAAL-PORT,” in SAFECOMP. Springer, vol 6894, 2011, pp. 243–256.

[17] E. Kang and P. Schobbens, “Schedulability analysis support for automotive systems: from requirement to implementation,” in 29th Symposium on Applied Computing, 2014, pp. 1080–1085.

[18] A. Goknil, J. Suryadevara, M.-A. Peraldi-Frati, and F. Mallet, “Analysis support for TADL 2 timing constraints on EAST-ADL models,” ECSA. Springer, 2013.

[19] C. Seceleanu, M. Johansson, J. Suryadevara, G. Sapienza, T. Seceleanu, S.-E. Ellevseth, and P. Pettersson, “Analyzing a wind turbine system: From simulation to formal verification,” Science of Computer Programming, vol. 133, 2017, pp. 216–242.

[20] F. Mallet, M.-A. Peraldi-Frati, and C. Andre, “MARTE CCSL to execute EAST-ADL timing requirements,” in 12th International Symposium on Object, Component, Service-oriented Real-time Distributed Computing, 2009, pp. 249 –253.

[21] E. Kang, J. Chen, L. Ke, and S. Chen, “Statistical analysis of energy-aware real-time automotive systems in EAST-ADL/Stateflow,” ICIEA. IEEE, 2016, pp. 1328–1333.

[22] E. Kang, L. Ke, M. Hua, and Y. Wang, “Verifying automotive systems in EAST-ADL/Stateflow using UPPAAL,” in 2015 22nd Asia-Pacific Software Engineering Conference. IEEE, 2015, pp. 143–150.
[23] R. Marinescu, H. Kaijser, M. Mikuĕionis, C. Seceleanu, H. Lönn, and A. David, “Analyzing industrial architectural models by simulation and model-checking,” in *3rd International Workshop on Formal Techniques for Safety-Critical Systems*. Springer, 2014, pp. 189 – 205.

[24] M.-R. Gholami, “Verifying timed LTL properties using simulink design verifier,” Ph.D. dissertation, École Polytechnique de Montréal, 2016.