Formal Modelling of Safety Architecture for Responsibility-Aware Autonomous Vehicle via Event-B Refinement

Tsutomu Kobayashi[1(✉)], Martin Bondu[2], and Fuyuki Ishikawa[3]

[1] Japan Aerospace Exploration Agency, Tsukuba, Japan
kobayashi.tsutomu@jaxa.jp
[2] Sorbonne University, Paris, France
martin.bondu@etu.sorbonne-universite.fr
[3] National Institute of Informatics, Tokyo, Japan
f-ishikawa@nii.ac.jp

Abstract. Ensuring the safety of autonomous vehicles (AVs) is the key requisite for their acceptance in society. This complexity is the core challenge in formally proving their safety conditions with AI-based black-box controllers and surrounding objects under various traffic scenarios. This paper describes our strategy and experience in modelling, deriving, and proving the safety conditions of AVs with the Event-B refinement mechanism to reduce complexity. Our case study targets the state-of-the-art model of goal-aware responsibility-sensitive safety to argue over interactions with surrounding vehicles. We also employ the Simplex architecture to involve advanced black-box AI controllers. Our experience has demonstrated that the refinement mechanism can be effectively used to gradually develop the complex system over scenario variations.

Keywords: Autonomous driving · AI safety · Responsibility-sensitive safety · Safety architecture · Event-B · Refinement

1 Introduction

The safety of automated vehicles has been attracting increased interest in society. In addition to the intensive effort of simulation-based testing, there is a key approach based on formal reasoning called responsibility-sensitive safety (RSS) [13]. RSS defines the minimum rules that traffic participants should comply with for safety, i.e., no collisions. This rule-based approach has recently been extended to goal-aware RSS (GA-RSS) to deal with the goal-achievement, i.e., the driving goal of the ego-vehicle is eventually achieved such as pulling over upon emergency [7]. GARSS is effective for formally limiting liabilities, which is vital for AV manufacturers.

The first author is supported by JSPS KAKENHI grant number 19K20249 and JST ERATO-MMSD (JPMJER1603) project. The third author is supported by JST MIRAI-eAI (JPMJMI20B8) project.

M. Chechik et al. (Eds.): FM 2023, LNCS 14000, pp. 533–549, 2023.
https://doi.org/10.1007/978-3-031-27481-7_30

The challenge lies in deriving the necessary GARSS conditions and formally checking the compliance of the design of the ego vehicle over various scenarios under different environmental conditions. In addition, there is increasing demand to consider complex behaviours of black-box AI-based advanced controllers backed up with safety-ensured controllers, e.g., the Simplex architecture [10].

Existing efforts have clarified the principles to derive and argue conditions that ego-vehicles should comply with in example scenarios. However, the engineering aspect has yet to be investigated. Specifically, we need a systematic modelling design that accepts the flexibility to mitigate the complexity in dealing with multiple aspects of scenario variations and architectural design.

To this end, we report our experience in modelling, deriving, and proving the safety conditions of autonomous vehicles (AVs). We follow the GA-RSS approach to define and derive the safety conditions to be checked with architectural design with black-box advanced controllers. We propose a strategy for using the refinement mechanism of Event-B [2] to gradually argue the complex aspects including the scenario variations. Our experience has shown the potential of the refinement mechanism for the flexible design of models and proofs to mitigate the complexity in a gradual manner. To the best of our knowledge, this is the first attempt to focus on the model engineering aspect over scenario variations in the deductive approach for AV safety.

The rest of this paper is structured as follows: In Sect. 2, we describe the safety architecture, RSS, and Event-B. Section 3 introduces GA-RSS and a case study example. We elaborate on our approach and its application to the case studies in Sect. 4–5. We discuss the approach in Sect. 6 before concluding the paper in Sect. 7.

2 Preliminaries

2.1 Safety Architecture

Contemporary software systems often have black-box modules, such as machine learning modules, in which their safety is essentially difficult to verify.

A safety architecture, such as Simplex architecture (Fig. 1) [10], is a fundamental approach to guaranteeing the safety of such systems while benefitting from the high performance and functionality of black-box modules. It models interactions between a controller and a plant. The controller part has two different controllers: the baseline controller (BC), which is designed to force safe behaviour, and the advanced controller (AC), which aims at satisfying various requirements (e.g., comfort and progress) in addition to safety. The decision module (DM) switches between the BC and AC in accordance with the state of the plant. BC may fail to satisfy requirements other than safety, but it has a simple white-box behaviour enabling the safety to be easily verified. In contrast, although AC usually gives better user experiences, guaranteeing its safety is difficult due to its complicated black-box behaviour. For example, a typical BC for an AV may drive by following a predefined rule that is guaranteed to be

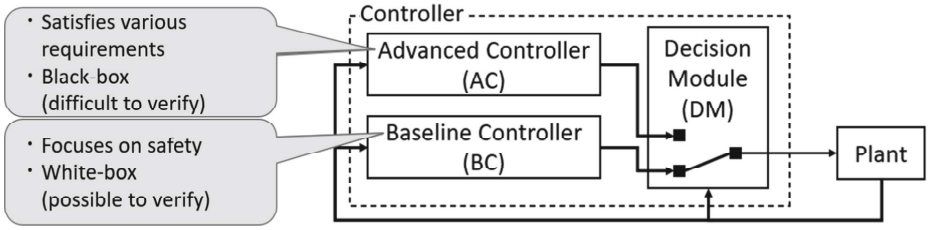

Fig. 1. Component-based simplex architecture [10]

safe in certain situations. A typical AC, on the other hand, would be one that uses machine learning for motion planning.

2.2 Responsibility-Sensitive Safety (RSS)

RSS is an approach to determining the safety of AVs by formal proof. The core idea is to derive conditions that should be satisfied by the current state of the traffic participants such that safety, or no collisions, is ensured in the future.

An RSS rule consists of an assertion ϕ called an RSS condition and a control strategy α called a proper response. They are defined for particular traffic scenarios. For example, a subject vehicle (SV), i.e., the ego vehicle, is following a preceding vehicle on a one-way road. We consider this preceding vehicle as the sole traffic participant called a principal other vehicle (POV). The SV must satisfy the RSS condition ϕ regarding the minimum relative distance from the POV. The distance is defined by considering the response time for braking and the distance necessary for the maximum comfortable braking to stop. The proper response α of the SV is to engage the maximum comfortable braking when the distance condition ϕ is about to be violated. The proof should show the RSS condition ϕ is preserved through the execution with the proper response α.

In a general setting, RSS considers the SV and POV in the target scenario and determines the RSS condition and proper response. To prove the condition is preserved through the execution, a certain set of constraints must be satisfied by not only the SV but also all traffic participants (POVs), called RSS responsibility principles. Examples of the principles include "do not cut in recklessly" and "be cautious in areas with limited visibility", intuitively.

Our focus is not on the core responsibility principles of RSS but on the RSS-driven framework for proving safety of AVs. We are interested in the formal engineering aspect to model and verify scenario variations.

2.3 Modelling and Proving in Event-B

In this section, we describe the concepts of modelling and theorem proving in Event-B [2] that are used in our case study[1].

[1] For simplicity, we do not cover the "full" Event-B (described in [2]). For instance, our concrete machines inherit all variables and parameters from abstract machines, which is not necessary in general Event-B machines.

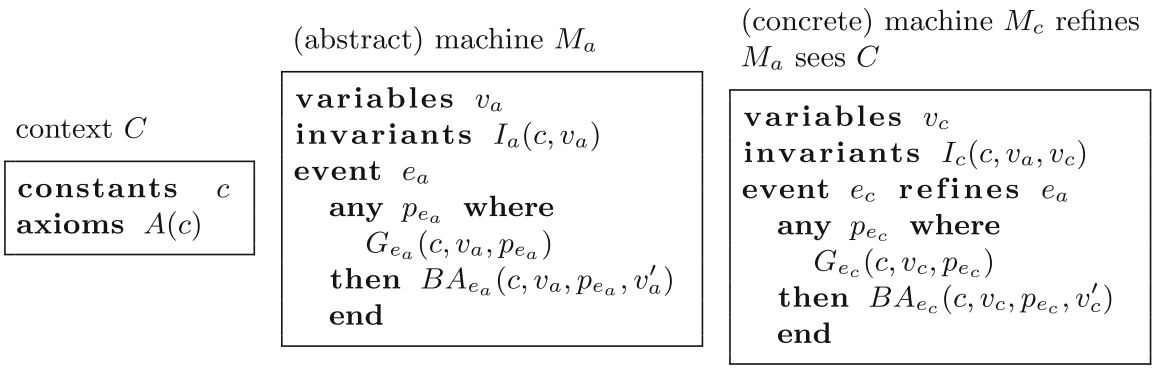

Fig. 2. Structure of Event-B model components

Event-B Model Components. Event-B models are structured as shown in Fig. 2. The static aspects of the target system are specified as contexts, which consist of constants and their properties (axioms). The dynamic aspects are specified as machines, which consist of variables, invariant predicates, and a set of events. An event e has parameters p_e, guard condition G_e, and before-after predicate BA_e that explains the assignment performed in e in terms of variables' current values v and next values v'. A significant feature of Event-B is a flexible refinement mechanism that enables declaring a machine M_c as a refinement of another machine M_a. Every event in M_c should be seen as a refinement of events in M_a (including the implicit *skip* event). M_c does not need to inherit predicates of M_a, but those two machines should be compatible as described in the following.

Proving Consistency of Models. Constructed models should be verified by discharging *proof obligations* (POs) generated with predicates in the models. Primary POs include the following:

- **Invariant Preservation** (for an abstract machine): Invariant predicates are inductive ones, i.e., they must hold after every occurrence of events, given that they hold beforehand. Formally, invariant preservation by an event e_a is: $A(c) \land I_a(c, v_a) \land G_{e_a}(c, v_a, p_{e_a}) \land BA_{e_a}(c, v_a, p_{e_a}, v'_a) \land \ldots \Longrightarrow I_a(c, v'_a)$.
- **Invariant Preservation** (for concrete machines): Formally, invariant preservation by an event e_c is: $A(c) \land I_a(c, v_a) \land I_c(c, v_a, v_c) \land G_{e_c}(c, v_c, p_{e_c}) \land BA_{e_c}(c, v_c, p_{e_c}, v'_c) \land \ldots \Longrightarrow I_c(c, v'_a, v'_c)$.
- **Guard Strengthening**: For an event e_c to be a refinement of an event e_a, the guard of e_c must be stronger than that of e_a's. Formally, guard strengthening of e_c is: $A(c) \land I_c(c, v_a, v_c) \land I_a(c, v_a) \land G_{e_c}(c, v_c, p_{e_c}) \land \ldots \Longrightarrow G_{e_a}(c, v_a, p_{e_a})$.

3 Example: Goal-Aware RSS for Pull over Scenario

Goal-aware RSS (GA-RSS) [7] is an extension of RSS for dealing with complex scenarios that require planning over multiple manoeuvres to achieve particular

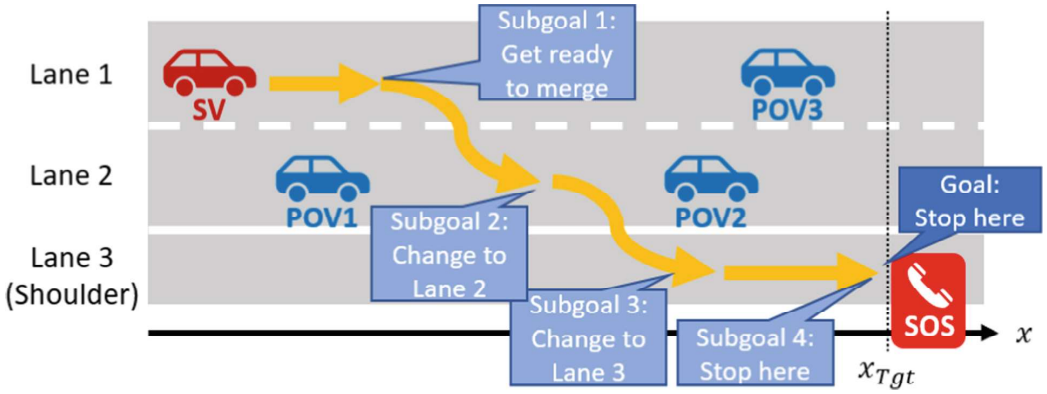

Fig. 3. Pull over scenario [7]

goals. For instance, consider the scenario shown in Fig. 3 (pull over scenario) [7]: the SV needs to stop at a designated location (x_{Tgt}) on the shoulder lane while keeping safe distances from POVs as required by RSS. Following only the original RSS rules for avoiding collisions is necessary but not enough to achieve the goal. The goal should be decomposed into several subgoals, such as (1) getting ready to merge between two POVs by changing the velocity, (2–3) changing lanes, and (4) stopping at x_{Tgt}. Different proper responses are required for different subgoals as well. However, for example, the SV can be trapped in Lane 1 if it is concerned about only the distance from the car ahead.

The workflow of GA-RSS is based on their extension of Floyd-Hoare logic. Given a driving scenario $\mathcal{S}$ composed of the goal condition Goal and safety condition Safety, the workflow is first used to decompose $\mathcal{S}$ into *subscenarios* $\mathcal{S}_{1,\ldots,n}$ and identify the *proper response* α_i for each subscenario $\mathcal{S}_i$.[2]

Then, the *precondition* ϕ_i for each subscenario is calculated as the precondition for establishing $\mathsf{Goal}_i \wedge \phi_{i+1}$ while satisfying Safety_i, by performing α_i. Here, by seeing the (grand) goal of $\mathcal{S}$ as the postcondition of the final subscenario $\mathcal{S}_n$, the preconditions of all subscenarios are derived in a backward manner, à la Floyd-Hoare logic, and then integrated into the precondition of $\mathcal{S}$.

For instance, Fig. 4 shows the subgoals, safety conditions, proper responses, and preconditions of a subscenario chain (defined and derived in [7]) where the SV goes between POV1 and POV2 and changes lanes.

Variables are as follows: x_{SV} and $x_{1,2,3}$ are the lateral positions of the SV and the three POVs; v_{SV} and $v_{1,2,3}$ are their lateral velocities; a_{SV} and $a_{1,2,3}$ are their lateral acceleration rates; L and $L_{1,2,3}$ for set of lanes they are on. Constants are as follows: x_{Tgt} is the position of the final goal position; v_{min} and v_{max} are the legal speed limits; b_{min} and b_{max} are the minimum (comfortable) and maximum (emergency) braking deceleration rates; a_{max} is the maximum acceleration rate.

[2] To be precise, with case distinctions, a tree of subscenarios is derived.

Subscenario $\mathcal{S}_4$ (Stop at the target) **Subgoal** Goal_4: $x_{SV} = x_{Tgt} \wedge v_{SV} = 0$

Safety Safety_4: $L = \{3\} \wedge 0 \leq v_{SV} \leq v_{max} \wedge -b_{min} \leq a_{SV} \leq a_{max}$
Proper Response α_4: Cruise for $timeToCruise_4(x_{SV0}, v_{SV0})$, then brake with b_{min} for $timeToBrake_4(v_{SV0})$
Precondition ϕ_4: $\mathsf{Env} \wedge L = \{3\} \wedge x_{Tgt} - x_{SV} \geq v_{SV}^2/2b_{min}$

Subscenario $\mathcal{S}_3$ (Change to Lane 3) **Subgoal** Goal_3: $L = \{3\}$

Safety Safety_3: $(L = \{3\} \vee L = \{2,3\}) \wedge 0 \leq v_{SV} \leq v_2 \wedge x_2 - x_{SV} \geq \mathsf{dRSS}(v_2, v_{SV}) \wedge -b_{min} \leq a_{SV} \leq a_{max}$
Proper Response α_3: Cruise for $timeToCruise_3(x_{SV0}, v_{SV0}, \ldots)$, then brake with b_{min} for $timeToBrake_3(x_{SV0}, v_{SV0}, \ldots)$
Precondition ϕ_3: $\mathsf{Env} \wedge L = \{2\} \wedge 0 < v_{SV} \leq v_2 \wedge x_2 - x_{SV} \geq \mathsf{dRSS}(v_2, v_{SV}) \wedge x_{Tgt} - x_{SV} \geq v_{SV}^2/2b_{min}$

Subscenario $\mathcal{S}_2$ (Change to Lane 2) **Subgoal** Goal_2: $lanes = \{2\}$

Safety Safety_2: $(L = \{2\} \vee L = \{1,2\}) \wedge 0 \leq v_{SV} \leq v_2 \wedge x_2 - x_{SV} \geq \mathsf{dRSS}(v_2, v_{SV}) \wedge x_3 - x_{SV} \geq \mathsf{dRSS}(v_3, v_{SV}) \wedge -b_{min} \leq a_{SV} \leq a_{max}$
Proper Response α_2: Cruise for $timeToCruise_2(x_{SV0}, v_{SV0}, \ldots)$, then brake with b_{min} for $timeToBrake_2(x_{SV0}, v_{SV0}, \ldots)$
Precondition ϕ_2: ...

Subscenario $\mathcal{S}_1$ (Get ready to merge)

Subgoal Goal_1: $x_2 - x_{SV} \geq \mathsf{dRSS}(v_2, v_{SV}) \wedge x_{SV} - x_1 \geq \mathsf{dRSS}(v_{SV}, v_1) \wedge v_2 = v_{SV}$
Safety Safety_1: $L = \{1\} \wedge x_3 - x_{SV} \geq \mathsf{dRSS}(v_3, v_{SV}) \wedge 0 \leq v_{SV} \leq v_{max} \wedge -b_{min} \leq a_{SV} \leq a_{max}$
Proper Response α_1: Combinations of acceleration, cruising, and braking depending on the situation. See § 6.2 for details.
Precondition ϕ_1: ...

Fig. 4. Subscenarios of pull over scenario with proper response and precondition

The condition of environment Env is as follows:

$$\mathsf{Env} = \bigwedge_{i=1,2,3} (v_{min} \leq v_i \leq v_{max} \wedge a_i = 0) \wedge L_1 = \{2\} \wedge L_2 = \{2\} \wedge L_3 = \{1\} \wedge x_2 > x_1.$$

This condition includes the assumption that POVs are supposed to run at constant velocity.

The RSS safety distance that the SV running at v_{SV} should keep from the POVi ahead running at v_i is defined as follows:

$$\mathsf{dRSS}(v_i, v_{SV}) = \max\left(0, \frac{v_{SV}^2}{2b_{min}} - \frac{v_i^2}{2b_{max}}\right). \quad (1)$$

The times the SV should cruise, brake, or accelerate in subscenario $\mathcal{S}_i$ for proper response α_i are derived in the GA-RSS workflow [7]. For instance,

$$timeToCruise_4(x_{SV0}, v_{SV0}) = \frac{x_{Tgt} - x_{SV0}}{v_{SV0}} - \frac{v_{SV0}}{2b_{min}}, \tag{2}$$

$$timeToBrake_4(v_{SV0}) = \frac{v_{SV0}}{b_{min}}, \tag{3}$$

where x_{SV0} and v_{SV0} are the position and velocity of the SV, respectively, when the switching occurs.

GA-RSS is designed to be integrated with the Simplex architecture. The identified scenarios are used to construct the BC that performs the derived proper response α in the situation compatible with the scenario, and thus the BC is guaranteed to be safe and goal-achieving. While the correctness of the DM is not covered with the method in [7], their experiment used their implementation of a Simplex-based controller, where the AC is black-box.

Motivation of Our Case Study. Even with the BC specifications identified with the GA-RSS workflow, a formal model of the whole Simplex architecture closer to the implementation is desired to construct safe and goal-achieving controllers of AVs. Such models should at least take into account the behaviour of the DM and the monitor-decide-control loop (Fig. 1).

The challenge here is the model's *complexity*; for example, in addition to DM-related elements, we need to take switching time delays into consideration.

To overcome this, we exploit the refinement mechanism of Event-B, which distributes the complexity of modelling and verification over multiple steps.

The rest of this paper discusses our case study, where we constructed and verified Event-B models of Simplex-based controllers for pull over subscenarios.

4 Case Study 1: Modelling Subscenario $\mathcal{S}_4$

In this section, we introduce our modelling strategy, where elements of systems should be specified in each refinement step by using our model for subscenario $\mathcal{S}_4$ of the pull over scenario as an example. We model the entire safety architecture and verify its safety in three refinement steps as follows:

Machine $M_{4,0}$: Whole controller-level. This is the most abstract machine. The properties of the whole controller's (AC+BC+DM) behaviour at every cycle are modelled. We focus on physical requirements that should be satisfied due to the controller's behaviour.

Machine $M_{4,1}$: Module-level. This machine refines $M_{4,0}$. This machine is aware of the safety architecture; behavioural properties of AC, BC, and DM are specified separately. We checked that the switching by the DM satisfies the requirements in $M_{4,0}$ by proving the correctness of $M_{4,0}$–$M_{4,1}$ refinement.

Machine $M_{4,2}$: Manoeuvre-level. This machine refines $M_{4,1}$. Details of the BC's behaviour (proper responses) are specified. By checking the correctness of $M_{4,1}$–$M_{4,2}$ refinement, we check that the proper responses satisfy the requirements.

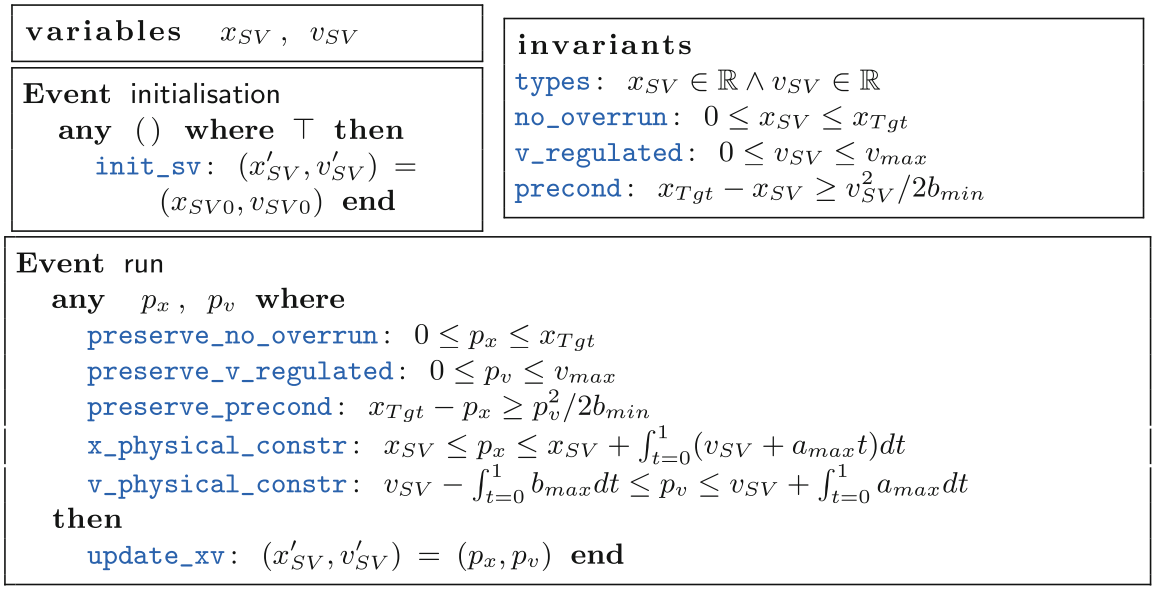

Fig. 5. $M_{4,0}$: Abstract, whole controller-level machine for subscenario $\mathcal{S}_4$

4.1 Machine $M_{4,0}$: Whole Controller-Level Behaviour

Machine $M_{4,0}$ is shown in Fig. 5. In this machine, we abstract away details of the controller and focus on the SV's position (x_{SV}) and velocity (v_{SV}) as the result of the controller's behaviour.

Invariant predicates `no_overrun` and `v_regulated` express basic requirements.

The precondition ϕ_4 derived from the GA-RSS workflow is designed to be an invariant that the safety architecture should preserve; the DM enables using the AC while ϕ_4 is *robustly* satisfied, but it switches to the control using the BC once ϕ_4 is *about to be* violated. Therefore, we specify ϕ_4 as an invariant predicate (`precond`).

There is only a single non-initialisation event named run. It has parameters p_x and p_v, which are specified as values of x_{SV} and v_{SV} at the next cycle (`update_xv`). The parameters are constrained by the guard predicates `preserve_*` required for the event's invariant preservation and those for the constraints related to physics (`*_physical_constr`). With these constraints as guard predicates of the event, we declare that every detailed behavioural description specified as events in concrete machines ($M_{4,1}$ and $M_{4,2}$) should satisfy the constraints.

The guard predicate `preserve_precond` states that the controller *somehow* produces the result (i.e., p_x and p_v) such that `precond` is satisfied. Indeed, the preservation of the precondition ϕ_4 is trivial because:

$$(x_{Tgt} - p_x \geq p_v^2/2b_{min}) \land \ldots \land ((x'_{SV}, v'_{SV}) = (p_x, p_v))$$
$$\implies x_{Tgt} - x'_{SV} \geq v'^2_{SV}/2b_{min}.$$

Note that *how* the controller works to produce the invariant-satisfying result is not yet specified and deferred to concrete machines; how the DM prevents the

variables x_{SV} , v_{SV} , $ctrl$, v_{BC0}

invariants
types: $ctrl \in \{AC, BC\} \land v_{BC0} \in \mathbb{R}$
vsvbcinit_regulated: $0 \leq v_{BC0} \leq v_{max}$
bc_no_accel: $ctrl = BC \implies v_{SV} \leq v_{BC0}$
switching: $ctrl = AC \implies \phi_4(x_{SV} + \int_{t=0}^{1}(v_{SV} + a_{max}t)dt,\ v_{SV} + \int_{t=0}^{1} a_{max}dt)$

Event AC → BC **refines** run
any p_x , p_v **where**
... (guard predicates of run except preserve_precond) ...
AC_operating: $ctrl = AC$
maybe_unsafe_next: $\neg\phi_4(x_{SV} + \int_{t=0}^{2}(v_{SV} + a_{max})dt,\ v_{SV} + \int_{t=0}^{2} a_{max}dt)$
then
... (actions of run) ...
switch_to_bc: $ctrl' = BC$
vsvbcinit_update: $v'_{BC0} = p_v$ **end**

Event BC → AC **refines** run
any p_x , p_v **where**
... (guard predicates of run) ...
BC_operating: $ctrl = BC$
no_acceleration: $p_v \leq v_{BC0}$
surely_safe_next: $\phi_4(x_{SV} + \int_{t=0}^{2}(v_{SV} + a_{max})dt,\ v_{SV} + \int_{t=0}^{2} a_{max}dt)$
then
... (actions of run) ...
switch_to_ac: $ctrl' = AC$ **end**

Fig. 6. (A part of) $M_{4,1}$: Intermediate, module-level machine for subscenario $\mathcal{S}_4$

AC from violating it is specified in machine $M_{4,1}$, and how the BC's behaviour (proper responses) satisfies it is specified in machine $M_{4,2}$.

4.2 Machine $M_{4,1}$: Module-Level Behaviour

In machine $M_{4,1}$ (Fig. 6), which refines $M_{4,0}$, we focus on the requirements on white-box modules of the architecture, namely the BC and DM, particularly the condition for switching; through the proof attempt, we *derived* the switching condition such that the precondition is always satisfied. Note that we assume that the AC's behaviour is arbitrary as long as it satisfies run's guard. Details of the BC's behaviour that should be specified using the time spent for each manoeuvre are introduced in machine $M_{4,2}$.

There are two new variables: $ctrl$, for the currently active controller, and v_{BC0}, which stores the velocity at the time when switching to the BC occurs.

Invariant predicates are in regard to the requirements on the BC and DM. vsvbcinit_regulated requests that v_{BC0} should not exceed v_{max} like v_{SV}, and bc_no_accel expresses that the BC does not accelerate in the proper response. switching states that if the AC is active, then the SV will be goal-achieving and

safe after a cycle even if the SV accelerated with the maximum rate a_{max}. The contraposition of `switching` means that the BC is used if the precondition ϕ_4 may be violated at the next cycle.

There are four events for cases of switching: AC → AC, AC → BC, BC → BC, and BC → AC. They all refine the run event of the previous machine $M_{4,0}$. For instance, AC → BC is for the case where the current controller is the AC (`AC_operating`) and `switching` can be violated after the event (`maybe_unsafe_next`; note that the integrals are from $t = 0$ to 2 to look ahead for two cycles). Note that, however, `switching` is guaranteed to hold before the event since it is an invariant predicate. In addition to actions of run, the controller is switched to the BC (`switch_to_bc`) and v_{BC0} is updated (`vsvbcinit_update`). On the other hand, BC → AC is the case where the controller is switched from the BC to AC because the invariant `switching` will be satisfied after the occurrence of the event (`surely_safe_next`).

The main POs are as follows:

1. **Do the events AC → ∗ preserve the invariant** `precond`? This corresponds to the guard strengthening PO of AC → ∗. The intuition of the proof is because the AC is operating only if the precondition is guaranteed to hold after two cycles (`surely_safe_next`), and it is guaranteed to hold after one cycle as well.
2. **Do events ∗ → AC preserve the invariant** `switching`? It is preserved because the AC will be used only if `surely_safe_next` holds at the current state. In fact, we *derived* the switching condition `surely_safe_next` through the attempt to discharge this PO.

4.3 Machine $M_{4,2}$: Manoeuvre-Level Behaviour

In machine $M_{4,2}$ (Fig. 7), which refines $M_{4,1}$, we focus on the details of the behaviour with the notion of time to spend on each manoeuvre to verify that the BC's behaviour satisfies the requirements specified in machines $M_{4,0}$ and $M_{4,1}$.

Two new variables about the remaining time for cruising ($t_{BCCruise}$) and braking ($t_{BCBrake}$) are introduced. The unit of time here is the cycle, e.g., the value of $t_{BCCruise}$ is the number of the controller's cycles spent for cruising.

Invariant predicates are in regard to the detailed properties of the BC's behaviour: `cruise_before_brake` expresses that the proper response α_4 is cruising and then braking, and `*_in_BC*` states that the velocity and position should follow the proper response α_4 as shown in Fig. 8.

Events of $M_{4,2}$ refine those of $M_{4,1}$ as shown in Fig. 9.

Three events that refine AC → ∗ are mostly the same as $M_{4,1}$, but events regarding switching to the BC (such as AC_run → BC) are extended with actions of calculating $t_{BCCruise}$ and $t_{BCBrake}$ as Eqs. 2 and 3 (derived in the GA-RSS workflow) because the BC should calculate them every time it get activated.

Unlike events that refine AC → ∗, six events that refine BC → ∗ do not inherit all of the guard predicates and actions of corresponding events in machine $M_{4,1}$. For example, the differences between the event BC_cruise → AC in $M_{4,2}$ and the corresponding event BC → AC in $M_{4,1}$ is as shown in Fig. 10. The removed

variables x_{SV} , v_{SV} , $ctrl$, v_{BC0} , $t_{BCCruise}$, $t_{BCBrake}$

invariants
types: $t_{BCCruise} \in \mathbb{R}_{\geq 0} \land t_{BCBrake} \in \mathbb{R}_{\geq 0}$
cruise_before_brake: $0 < t_{BCCruise} \implies 0 < t_{BCBrake}$
v_in_BC: $ctrl = BC \implies v_{SV} = t_{BCBrake} \cdot b_{min}$
v_in_BC_cruise: $(ctrl = BC \land 0 < t_{BCCruise}) \implies v_{SV} = v_{BC0}$
x_in_BC: $(ctrl = BC \land v_{SV} \neq 0)$
$\implies x_{Tgt} - x_{SV} = \int_{t=0}^{t_{BCCruise}} v_{SV} dt + \int_{t=0}^{t_{BCBrake}} (v_{SV} - b_{min} t) dt$

axioms t_bccruise_def: $timeToCruise_4(x, v) = (2b_{min}(x_{Tgt} - x) - v^2)/(2b_{min}v)$

Event AC· run → BC **refines** AC → BC
any p_x , p_v **where**
... (guard predicates of AC → BC) ...
will_run_more: $0 < p_v$
then
... (actions of AC → BC) ...
update_tcruisebc: $t'_{BCCruise} = timeToCruise_4(p_x, p_v)$
update_tbrakebc: $t'_{BCBrake} = timeToBrake_4(p_v)$
end

Event BC· cruise → AC **refines** BC → AC
any p_x , p_v **where**
BC_operating: $ctrl = BC$
surely_safe_next: $\phi_4(x_{SV} + \int_{t=0}^{2}(v_{SV} + a_{max})dt,\ v_{SV} + \int_{t=0}^{2} a_{max} dt)$
will_cruise_more: $1 \leq t_{BCCruise}$
cruise_xv: $p_x = x_{SV} + \int_{t=0}^{1} v_{SV} dt \land p_v = v_{SV} + \int_{t=0}^{1} 0 dt$
then
... (actions of BC → AC) ...
tcruise_pass: $t'_{BCCruise} = t_{BCCruise} - 1$
end

Fig. 7. (A part of) $M_{4,2}$: Concrete, manoeuvre-level machine for subscenario $\mathcal{S}_4$

guard predicates (lines with red background) are requirements on the values of the SV's position and velocity after the occurrence of the event (p_x and p_v), while introduced guard predicates (lines with green background) include the concrete behaviour of the BC (cruise_xv), namely running with the constant velocity. By changing events in this way and checking that the guard of BC_cruise → AC is stronger than that of BC → AC, we can verify that the BC's concrete behaviour satisfies the requirements specified in machines $M_{4,0}$ and $M_{4,1}$.

In addition to the consistency between the BC's concrete behaviour specified in $M_{4,2}$ and requirements on the BC specified in $M_{4,1}$, we checked that events $* \rightarrow$ BC and BC $\rightarrow *$ preserve the invariant.

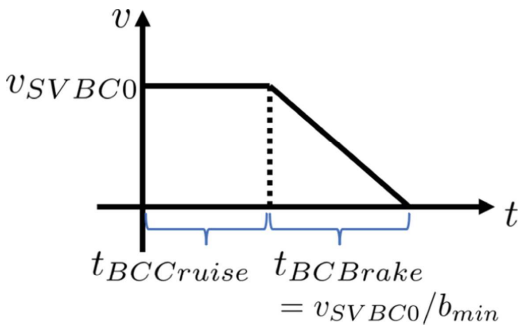

Fig. 8. Proper response α_4

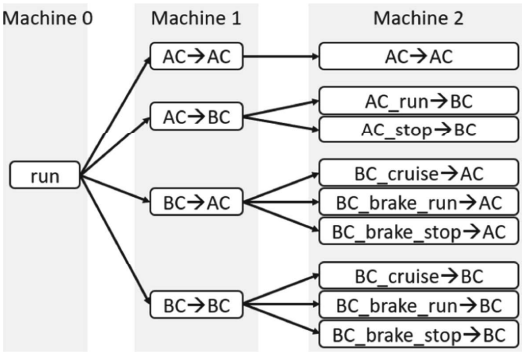

Fig. 9. Event refinement relationship

5 Case Study 2: Modelling Subscenario $\mathcal{S}_3$

In this section, we use subscenario $\mathcal{S}_3$ to demonstrate how our modelling strategy (Sect. 4) is applicable to other subscenarios. subscenario $\mathcal{S}_3$ has new aspects; the SV is changing lanes and the leading vehicle POV2.

5.1 Machine $M_{3,0}$: Whole Controller-Level Behaviour

Following machine $M_{4,0}$ of subscenario $\mathcal{S}_4$, we focus only on the physical results of the controller behaviour.

POV2's variable position (x_2) and constant velocity (v_2) are used in addition to SV's position and velocity.

As the SV is changing lanes, we assume that this action will be done in an exact amount of time modelled as a constant t_{LC} (the time for lane changing), and therefore we introduce another variable t_{LCe} (the time for lane changing elapsed) so that when the time elapsed reaches t_{LC}, the SV should have finished switching lanes and the subscenario is over. We modelled lanes in this style instead of introducing another physical coordinate for simplicity.

A new invariant predicate `no_overtime` regarding the time limit of this subscenario is also introduced as a replacement for `no_overrun` of subscenario $\mathcal{S}_4$. The corresponding guard predicates of the event run are specified so that no event can occur once the lane switching is over.

$$\boxed{\texttt{no\_overtime}\colon t_{LCe} \leq t_{LC}}$$

The precondition for subscenario $\mathcal{S}_3$ (ϕ_3 derived in [7]) takes into consideration the RSS safety distance between the SV and the leading vehicle POV2.

$$\boxed{\begin{array}{c}\texttt{precond}\colon x_{Tgt} - x_{SV} \geq v_{SV}^2/2b_{min} \wedge x_{SV} < x_2 \\ \wedge 2(x_{SV} - x_2) + \frac{v_{SV}^2}{b_{min}} \leq \frac{v_2^2}{b_{max}}\end{array}}$$

As in subscenario $\mathcal{S}_4$, the run event has guard predicates to preserve invariant predicates. The event also has new actions for updating x_2 and t_{LCe}:

update_xLead: $x'_2 = x_2 + \int_{t=0}^{1}(v_2 t)dt$
update_xLCe: $t'_{LCe} = \min(t_{LC}, t_{LCe} + 1)$

```
–Event BC → AC refines run
+Event BC_cruise → AC refines BC → AC
  any  p_x , p_v  where
–   preserve_no_overrun: 0 ≤ p_x ≤ x_Tgt
–   preserve_v_regulated: 0 ≤ p_v ≤ v_max
–   preserve_precond: x_Tgt − p_x ≥ p_v²/2b_min
–   x_physical_constr: x_SV ≤ p_x ≤ x_SV + ∫_{t=0}^{1}(v_SV + a_max t)dt
–   v_physical_constr: v_SV − ∫_{t=0}^{1} b_max dt ≤ p_v ≤ v_SV + ∫_{t=0}^{1} a_max dt
    BC_operating: ctrl = BC
–   no_acceleration: p_v ≤ v_BC0
    surely_safe_next: φ4(x_SV + ∫_{t=0}^{2}(v_SV + a_max)dt, v_SV + ∫_{t=0}^{2} a_max dt)
+   will_cruise_more: 1 ≤ t_BCCruise
+   cruise_xv: p_x = x_SV + ∫_{t=0}^{1} v_SV dt ∧ p_v = v_SV + ∫_{t=0}^{1} 0dt
  then
    update_xv: (x'_SV, v'_SV) = (p_x, p_v)
    switch_to_ac: ctrl' = AC
+   tcruise_pass: t'_BCCruise = t_BCCruise − 1
  end
```

Fig. 10. Differences between BC → AC (in $M_{4,1}$) and BC_cruise → AC (in $M_{4,2}$)

5.2 Machine $M_{3,1}$: Module-Level Behaviour

This machine is also similar to $M_{4,1}$, but the invariant switching and guard predicates surely_safe_next (and its negation maybe_unsafe_next) take into account the distance between the SV and POV2.

switching: $ctrl = AC \implies \phi_3(x_{SV} + \int_{t=0}^{1}(v_{SV} + a_{max}t)dt, v_{SV} + \int_{t=0}^{1} a_{max}dt, x_2 + \int_{t=0}^{1}(v_2 t)dt, v_2)$

surely_safe_next: $\phi_3(x_{SV} + \int_{t=0}^{2}(v_{SV} + a_{max}t)dt, v_{SV} + \int_{t=0}^{2} a_{max}dt, x_2 + \int_{t=0}^{2}(v_2 t)dt, v_2)$

As subscenario $\mathcal{S}_4$ (Sect. 4.2), the POs are in regard to the preservations of invariants precond and switching.

5.3 Machine $M_{3,2}$: Manoeuvre-Level Behaviour

Compared with $M_{4,2}$ for subscenario $\mathcal{S}_4$, there are two major differences: when switching to the BC, the calculation of $t_{BCCruise}$ and $t_{BCBrake}$ (derived in [7]) is different because the velocity of the SV should not be zero by the end of the subscenario $\mathcal{S}_3$ but only low enough to satisfy the goal invariant.

$$\boxed{\texttt{tBrake\_update}\text{: } t'_{BCBrake} = (t_{LC} - t_{LCe}) + \frac{p_v}{2.b_{min}} + \frac{p_x - x_{Tgt}}{p_v}}$$

The six events that refine $\mathsf{BC\_}* \rightarrow *$ have to satisfy machine $M_{3,0}$'s `precond` that now includes the safety distance to the leading vehicle POV2.

The POs in regard to this invariant were discharged in the following way:

1. $\mathsf{BC\_}* \rightarrow \mathsf{BC}$. The idea behind this proof is that BC's proper response does not include accelerating and the leading vehicle's velocity is constant, so the distance between these two may only increase.
2. $\mathsf{BC\_}* \rightarrow \mathsf{AC}$. The guard predicate `surely_safe_next` states that the invariant will be satisfied in two cycles without having to break in the next cycle because the controller will be in the AC.

6 Discussion

6.1 Model Engineering

In the case studies, we have used the refinement mechanism of Event-B to gradually model and verify the different aspects. Specifically, we separated the argument over the definition of safe and goal-achieving behaviour, architecture for switching behaviours, and concrete behaviour design. The refinement mechanism limits the complexity of modelling and proof in each step, which was essential in handling the increasing complexity in proving continuous properties.

We did not directly reuse the models between subscenarios, e.g., sharing the abstract steps between subscenarios. This is our explicit choice as the key safety properties and involved variables for the POVs are unique to each subscenario. We instead used the common refinement strategy as well as the model representations. We believe this experience enables us to demonstrate the know-how for scenarios other than the pull over scenario. The generality of the approach is further discussed in the following.

6.2 Generality of Approach

We have described how the same refinement strategy can deal with subscenarios $\mathcal{S}_3$ and $\mathcal{S}_4$. We describe how the other subscenarios can be modelled as well as the omitted aspect of perception errors.

Subscenario $\mathcal{S}_2$. The machines for subscenario $\mathcal{S}_2$ are similar to that for subscenario $\mathcal{S}_3$. The main difference between them is the presence of a leading vehicle in the next lane in subscenario $\mathcal{S}_2$ while there is none in subscenario $\mathcal{S}_3$.

Subscenario $\mathcal{S}_1$. In this subscenario, the SV needs to prepare to switch lanes and merge into the next lane. There are three POVs to take into account: one ahead of the SV in the current lane (POV3) and two others in the next lane (POV 1 and 2). This subscenario thus involves multiple (in this case, four) proper responses: an example is accelerating to pass POV1 in the next lane, and another example is decelerating to match the velocity of POV2 in the next lane.

To handle multiple proper responses in a unified manner, we modelled them as a sequence of proper responses with variable durations as follows: (1) Accelerate for $t_{BCAccel}$ (2) Cruise for $t_{BCCruise}$ (3) Brake for $t_{BCBrake}$. Moreover, we needed to take into account different precondition for each proper response. Therefore, we introduced a variable to record which proper response was taken the last time the BC got activated.

Perceptual Uncertainty. Another aspect not included in the case studies is perceptual uncertainty or the possibility of errors in sensing. A basic approach to this issue would be adding safety margins to the behaviour of the controller. For instance, introducing a variable $\widehat{x_{Tgt}}$ for the perceived value of target location (x_{Tgt}) and discussing assumptions on the difference between x_{Tgt} and $\widehat{x_{Tgt}}$ enables us to derive the appropriate amount of the safety margin for this uncertainty.

6.3 Using Event-B for Modelling and Proving

Features of Event-B and its modelling environment Rodin [1] were useful for modelling and proving the safety architecture for GA-RSS. Rodin generated POs and helped interactive proof of them. The refinement mechanism of Event-B was effective for distributing the complexity of modelling and proving over multiple steps. In addition, as we discussed in Sect. 4.2, we derived the correct behaviour of DM from generated POs.

Our contributions in this paper, namely strategies of modelling and refinement, provide a guide to the effective use of Event-B's features for the rigorous and systematic construction of controllers for different subscenarios.

On the other hand, although Rodin has proof tactics and provers for automatically discharging POs, we had to manually discharge all POs. It is because we needed an extension of Event-B language [4] to use real numbers in models, and Rodin's current automatic proof functionalities are not strong when the language is extended. However, we expect that this problem will be solved; for instance, there are studies aiming at assisting automatic proof of hybrid systems by bridging Rodin with external solvers [3].

6.4 Related Work

RSS was originally proposed as the formal approach for AVs, but the paper did not include any machine-processible models [13]. The work on GA-RSS extended the framework of RSS with formal specifications and partial calculations supported by Mathematica [7]. Other studies only used the resulting RSS conditions, for example, encoding them in signal temporal logic for runtime verification [8]. To the best of our knowledge, this is the first attempt to make use of formal modelling for the RSS scheme. The study in [12] demonstrated the difficulty in checking RSS properties with automated "one button" tools for reachability analysis and model checking.

Other formal attempts for AVs include proofs with the Isabelle/HOL prover [11] with support of MATLAB. The focus was on the detailed computation including floating-point errors while the driving behaviour was rather simple; avoidance of one static object with a white-box controller.

Verification over RSS is intrinsically hybrid, i.e., including continuous aspects such as velocity and distance. Proofs over hybrid models have been actively investigated in the Hoare-style reasoning, not only for Event-B but also in other formalisms such as KeYmaera X [6]. Our case study did not focus on the continuous aspects and used rather simple theories for handling real arithmetic. Our future work includes the use of more sophisticated support for discharging the proof obligations. It is notable that refining continuous models in the physics world into discrete software controllers has been actively investigated for Event-B, e.g., [5]. Models obtained in our approach can be further refined with such techniques into concrete designs of discrete software controllers.

Guidelines with a focus on refinement strategies have been considered useful for Event-B as reusable know-how for specific types of systems [14]. Our case study has the potential to be elaborated into such guidelines. Although the effectiveness of refinement strategies has been discussed qualitatively in most cases, there have been efforts on quantitative analysis [9]. Our future work will include analysis of refinement strategies in this work in a more systematic way.

7 Conclusion

In this paper, we reported our case study to model, derive, and prove the safety conditions of AVs in the RSS scheme. We target a state-of-the-art problem with the goal-aware version of RSS as well as the Simplex architecture to consider black-box AI controllers. We proposed a strategy for leveraging the refinement mechanism of Event-B and demonstrated how it mitigates the complexity over scenario variations. We will continue studying other scenarios to convert the obtained lessons into more concrete and general guidelines for formal modelling and verification of AVs.

Acknowledgements. We thank our industrial partner Mazda for discussions of realistic problems in the safety assurance of autonomous driving. We also thank members of JST ERATO HASUO Metamathematics for Systems Design Project for discussions of Goal-Aware RSS and the safety architecture.

References

1. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transf. **12**(6), 447–466 (2010). https://doi.org/10.1007/s10009-010-0145-y
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)

3. Afendi, M., Mammar, A., Laleau, R.: Building correct hybrid systems using Event-B and sagemath: illustration by the hybrid smart heating system case study. In: 26th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 91–96. Hiroshima, Japan (2022). https://doi.org/10.1109/ICECCS54210.2022.00019
4. Butler, M., Maamria, I.: Practical theory extension in Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 67–81. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39698-4_5
5. Dupont, G., Ait-Ameur, Y., Singh, N.K., Pantel, M.: Event-B hybridation: a proof and refinement-based framework for modelling hybrid systems. ACM Trans. Embed. Comput. Syst, **20**(4), 1–37 (2021). https://doi.org/10.1145/3448270
6. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 527–538. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_36
7. Hasuo, I., et al.: Goal-aware RSS for complex scenarios via program logic. In: IEEE Transactions on Intelligent Vehicles, pp. 1–33 (2022). https://doi.org/10.1109/TIV.2022.3169762
8. Hekmatnejad, M., et al.: Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic. In: 17th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE). ACM, New York, NY, USA (2019). https://doi.org/10.1145/3359986.3361203
9. Kobayashi, T., Ishikawa, F.: Analysis on strategies of superposition refinement of Event-B specifications. In: Sun, J., Sun, M. (eds.) ICFEM 2018. LNCS, vol. 11232, pp. 357–372. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02450-5_21
10. Phan, D., et al.: A component-based simplex architecture for high-assurance cyber-physical systems. In: 17th International Conference on Application of Concurrency to System Design (ACSD), pp. 49–58. Zaragoza, Spain (2017). https://doi.org/10.1109/ACSD.2017.23
11. Rizaldi, A., Immler, F., Schürmann, B., Althoff, M.: A formally verified motion planner for autonomous vehicles. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 75–90. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_5
12. Roohi, N., Kaur, R., Weimer, J., Sokolsky, O., Lee, I.: Self-driving vehicle verification towards a benchmark. CoRR **abs/1806.08810** (2018). http://arxiv.org/abs/1806.08810
13. Shalev-Shwartz, S., Shammah, S., Shashua, A.: On a formal model of safe and scalable self-driving cars. CoRR **abs/1708.06374** (2017). http://arxiv.org/abs/1708.06374
14. Yeganefard, S., Butler, M.J., Rezazadeh, A.: Evaluation of a guideline by formal modelling of cruise control system in Event-B. In: Muñoz, C.A. (ed.) The 2nd NASA Formal Methods Symposium (NFM). NASA Conference Proceedings, vol. NASA/CP-2010-216215, pp. 182–191 (2010)