Goal-Aware RSS for Complex Scenarios via Program Logic

Ichiro Hasuo, *Member, IEEE*, Clovis Eberhart, James Haydon, Jérémy Dubut, Rose Bohrer, Tsutomu Kobayashi, Sasinee Pruekprasert, Xiao-Yi Zhang, Erik André Pallas, Akihisa Yamada, Kohei Suenaga, Fuyuki Ishikawa, *Member, IEEE*, Kenji Kamijo, Yoshiyuki Shinya, and Takamasa Suetomi

***Abstract*—We introduce a *goal-aware* extension of responsibility-sensitive safety (RSS), a recent methodology for rule-based safety guarantee for automated driving systems (ADS). Making RSS rules guarantee goal achievement—in addition to collision avoidance as in the original RSS—requires complex planning over long sequences of manoeuvres. To deal with the complexity, we introduce a compositional reasoning framework based on program logic, in which one can systematically develop RSS rules for smaller subscenarios and combine them to obtain RSS rules for bigger scenarios. As the basis of the framework, we introduce a program logic dFHL that accommodates continuous dynamics and safety conditions. Our framework presents a dFHL-based workflow for deriving goal-aware RSS rules; we discuss its software support, too. We conducted experimental evaluation using RSS rules in a safety architecture. Its results show that goal-aware RSS is indeed effective in realising both collision avoidance and goal achievement.**

Manuscript received 26 January 2022; revised 8 March 2022; accepted 15 March 2022. Date of publication 5 July 2022; date of current version 19 May 2023. This work was supported in part by JST under the ERATO HASUO Metamathematics for Systems Design Project (JPMJER1603,) and under ACT-I (JPMJPR17UA), and in part by JSPS under Grants-in Aid under Grant 19K20215 and Grant 19K20249. *(Ichiro Hasuo, Clovis Eberhart, and James Haydon contributed equally to this work.) (Corresponding author: Ichiro Hasuo.)*

Ichiro Hasuo is with the National Institute of Informatics, Tokyo 101-8430, Japan, and also with SOKENDAI (The Graduate University for Advanced Studies) 101-8430, Japan (e-mail: i.hasuo@acm.org).

Clovis Eberhart and Jérémy Dubut are with the National Institute of Informatics, Tokyo 101-8430, Japan, and also with the Japanese-French Laboratory for Informatics (IRL 3527), Tokyo 135-0064, Japan (e-mail: eberhart@nii.ac.jp; dubut@nii.ac.jp).

James Haydon, Tsutomu Kobayashi, Sasinee Pruekprasert, Xiao-Yi Zhang, and Fuyuki Ishikawa are with the National Institute of Informatics, Tokyo 101-8430, Japan (e-mail: jhaydon@nii.ac.jp; t-kobayashi@nii.ac.jp; sasinee@nii.ac.jp; xiaoyi@nii.ac.jp; f-ishikawa@nii.ac.jp).

Rose Bohrer is with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609-2280 USA (e-mail: rose.bohrer.cs@gmail.com).

Erik André Pallas is with the Institute of Software and Systems Engineering, University of Augsburg, D-86135 Augsburg, Germany (e-mail: erik.pallas@outlook.com).

Akihisa Yamada is with the National Institute of Informatics, Tokyo 101-8430, Japan, and also with Cyber Physical Security Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 135-0064, Japan (e-mail: akihisa.yamada@aist.go.jp).

Kohei Suenaga is with the National Institute of Informatics, Tokyo 101-8430, Japan, and also with the Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan (e-mail: ksuenaga@fos.kuis.kyoto-u.ac.jp).

Kenji Kamijo, Yoshiyuki Shinya, and Takamasa Suetomi are with Mazda Motor Corporation, Fuchu 730-8670, Japan (e-mail: kamijyo.k@mazda.co.jp; shinya.y@mazda.co.jp; suetomi.t@mazda.co.jp).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TIV.2022.3169762.

Digital Object Identifier 10.1109/TIV.2022.3169762

***Index Terms*—Automated driving, differential dynamics, Floyd–Hoare logic, program logic, responsibility-sensitive safety (RSS), rule-based safety, safety, simplex architecture.**

I. Introduction

SAFETY of automated driving systems (ADS) is a problem of growing industrial and social interest. New technologies in sensing and planning (such as lidars and deep neural networks) are making ADS technologically possible. However, towards the social acceptance of ADS, their safety should be guaranteed, explained, and agreed upon.

This paper is about *responsibility-sensitive safety* (RSS) [1]—a recent rule-based approach to ADS safety. Our contribution is to make the RSS framework *goal-aware*, so that logical "safety rules" in RSS

- not only guarantee collision avoidance (as in the original RSS [1]),
- but also guarantee *goal achievement*, such as changing lanes and stopping at a designated position on the highway shoulder (Example 1.5).

Goal-aware RSS rules typically involve multiple manoeuvres (accelerating, braking, changing lanes, etc.); deriving goal-aware RSS rules and proving their correctness is therefore much more complex compared to the original RSS. As technical contribution, we introduce logical, methodological and software infrastructures that realise goal-aware RSS. They are namely 1) a program logic suited for our purpose (called dFHL, Section II), 2) a logical workflow for compositional derivation of goal-aware RSS rules (Section IV), and 3) software support for the workflow (Section V). We demonstrate the value of our goal-aware RSS by experiments in a safety architecture (Section VI).

A. (Collision-Avoiding) Responsibility-Sensitive Safety

(The original) responsibility-sensitive safety (RSS) [1] is an approach to ADS safety that has been attracting growing attention. RSS aims to provide *safety rules* that are rigorously formulated in mathematical terms. Unlike most algorithms and techniques studied for ADS, RSS is not so much about *how to drive safely*; it is rather about breaking down the ultimate goal (namely safety *in the future*) into concrete conditions that only depend on the *current system state*. The core idea of RSS is that those safety rules should guarantee ADS safety in the rigorous form of *mathematical proofs*.

Here is an outline of original RSS [1]. We will often call the original RSS [1] *collision-avoiding RSS (CA-RSS)*, in contrast to our extension that we call *goal-aware RSS (GA-RSS)*. When we simply say RSS, the argument should apply to both CA- and GA-RSS.[1]

CA-RSS introduces *RSS rules* in a manner specific to different driving scenarios (driving in a single lane, changing lanes, other vehicles in front or behind, etc.). An RSS rule is a pair (A, α) of

- a logical assertion A called an *RSS condition*, and
- a control strategy α called a *proper response*.

An RSS rule is subject to the following requirements.

Requirements 1.1 (requirements on RSS rules, in CA-RSS): Let (A, α) be an RSS rule. Consider an arbitrary execution E of the proper response α; assume that the RSS condition A is satisfied at the beginning of E. Then

- (the *collision avoidance* requirement) the execution E in question must exhibit no collision; and
- (the *responsibility* requirement) the execution E must satisfy the RSS responsibility principles.

The last *RSS responsibility principles*, taken literally from [1], are listed below (cf. Remark 1.2).

1) Don't hit the car in front of you.
2) Don't cut in recklessly.
3) Right of way is given, not taken.
4) Be cautious in areas with limited visibility.
5) If you can avoid a crash without causing another one, you must.

One significance of the RSS framework is that the safety *in the future* (that is, safety during the whole execution E of α) is reduced to the RSS condition A *at present* (that is, one that can be checked at the beginning of E). In other words, the truth of A at present guarantees the safety in the future. This means that, in particular, A and α must take into account all possible future evolutions of the driving situation, such as sudden braking or acceleration of other vehicles, etc.

Another significance of RSS is the *assume-guarantee reasoning* via responsibilities. Establishing the collision avoidance requirement (Requirements 1.1) for the subject vehicle (SV) is usually impossible without suitable assumptions on other vehicles' behaviours—imagine a malicious vehicle that actively chases others and hits them. In RSS, one can impose the RSS responsibility principles on other vehicles and limit their behaviours; reciprocally, SV must obey the same principles, too.

Remark 1.2: The five RSS responsibility principles (as we call them) are often called "safety rules" and "common sense rules" in the RSS literature such as [1]. These principles are often presented as the main concept of RSS—especially in the presentation to the general public, such as Mobileye/Intel's webpage.[2] However, we believe that the logical framework of RSS (including reduction of the future to the present and assume-guarantee reasoning, as discussed above) is at least as important. The focus of the current paper is formalising and extending this logical framework of RSS.

[1]Our introduction of CA-RSS here adapts some terminologies for our purpose of extending it later; the terminologies can therefore differ from those used in [1]. Another logic-oriented introduction to CA-RSS is found in [2].

[2]https://www.mobileye.com/responsibility-sensitive-safety

Fig. 1. The one-way traffic scenario.

Example 1.3 (a CA-RSS rule for one-way traffic): Consider the one-way traffic scenario shown in Fig. 1 , where the subject vehicle (SV, car_{rear}) drives behind another car ($\text{car}_{\text{front}}$). The (collision-avoiding) RSS rule for this simple scenario, presented in [1], is (A, α) defined as follows.

The RSS condition A: The RSS condition A is

$$A = \big(y_f - y_r > \mathsf{dRSS}(v_f, v_r)\big), \tag{1}$$

where $\mathsf{dRSS}(v_f, v_r)$ is the *RSS safety distance* defined by

$$\mathsf{dRSS}(v_f, v_r) := \max\left(0, v_r\rho + \frac{1}{2}a_{\max}\rho^2 + \frac{(v_r + a_{\max}\rho)^2}{2b_{\min}} - \frac{v_f^2}{2b_{\max}}\right). \tag{2}$$

Here y_f, y_r are the positions of the two cars, and v_f, v_r are their velocities (their dynamics are modelled in the 1-dimensional lane coordinate). The other parameters are as follows: ρ is the maximum *response time* that car_{rear} might take to initiate the required braking; $a_{\max}$ is the maximum (forward) acceleration rate of car_{rear}; $b_{\min}$ is the maximum comfortable braking rate for car_{rear}; and $b_{\max}$ is the maximum emergency braking rate for $\text{car}_{\text{front}}$.

The proper response α: The proper response α dictates SV (car_{rear}) to engage the maximum comfortable braking (at rate $b_{\min}$) when condition (1) is about to be violated.

That the RSS rule (A, α) satisfies the collision avoidance requirement (Requirements 1.1) is proved in the original RSS paper [1]. We also give a formal proof later in Example 2.15, using the logic dFHL we introduce for our purpose of formalising reasoning in RSS.

B. Usages of RSS

Before introducing our goal-aware extension of RSS, we discuss some usages of (CA- and GA-)RSS, hoping that the discussion further illustrates the goals and features of RSS.

A distinguishing feature of RSS is that it gives *a priori* rules for rigorous safety guarantee. This is in contrast with

- many optimisation- and learning-based planning algorithms for safe driving, such as [3] (they do not offer rigorous safety guarantee),
- testing-based approaches for ADS safety, such as [4] (they do not offer rigorous safety guarantee, either), and
- runtime verification approaches for ADS safety by reachability analysis, such as [5], [6].

(See Section I-G for further discussion.) This feature has enabled multiple unique usages of RSS, as we discuss below. These usages have been already pursued in the literature for CA-RSS; we expect similar usages for our GA-RSS as well.

One usage of RSS is for *attribution of liability* [7], that is, to identify culpable parties in accidents. RSS rules are designed so

that there is no collision as long as all parties comply with them (Requirements 1.1); therefore, in an accident, at least one party was *not* compliant and is therefore culpable.

Another usage is as a *safety metric* (discussed and/or used in [8]–[12]). Here, the risk of a given situation can be measured by either 1) the degree with which the RSS condition of a relevant RSS rule is violated, or 2) whether the proper response of the RSS rule is not engaged while it should.

Another obvious usage of RSS is for *formal reasoning* about ADS safety: by proving that SV complies with RSS rules, one can prove *a priori* that SV is never responsible for accidents. Often one does not go so far as formally proving SV's compliance with RSS rules. Even in that case, collecting empirical evidences for RSS compliance, e.g. by testing, allows one to establish logical *safety cases*. The importance of such safety cases are emphasised in standards such as UL 4600 [13]; the use of RSS is advocated in the current efforts towards the IEEE 2846 standard.

Yet another usage of RSS is as part of a *safety architecture*, whose detailed introduction is deferred to Section I-E. This is a variation of the last usage (formal safety reasoning), but is more widely and easily deployable, and is therefore attracting a lot of attention (see e.g. [14, Fig. 1]). Our experimental evaluation (Section VI) follows this usage.

After all, RSS rules are not only for *making ADS safer* but also for *limiting liabilities*. In the dawn of automated driving today, ADS vendors are under a lot of pressure to ensure the safety of their products, fearing the possibilities of unexpected or excessive liabilities. RSS rules cut clear mathematical bounds of the vendors' liabilities, easing their safety assurance efforts.

C. Goal-Aware RSS

We seek a *goal-aware* extension of the original (collision-avoiding) RSS [1], so that the RSS rules are not only concerned with collision avoidance but also with achieving a goal.

Requirements 1.4 (requirements on RSS rules, in GA-RSS): In *goal-aware RSS (GA-RSS)*, an RSS rule (A, α) must satisfy the following: for any execution E of the proper response α that starts at a state where the RSS condition A is true,

- (*collision avoidance*, the same as in Requirements 1.1),
- (*responsibility*, the same as in Requirements 1.1), and
- (the *goal achievement* requirement) the specified goal is achieved at the end of E.

Deriving such goal-aware RSS rules and establishing their correctness (in the sense of Requirements 1.4) pose multiple technical challenges. They include

- the formalisation of goals to be achieved,
- the identification of proper responses α, which would involve multiple manoeuvres (accelerating, braking, changing lanes, etc.),
- the identification of RSS conditions A that guarantee both collision avoidance and goal achievement along/after complex controls described by α,

and so on. The technical contribution of the current paper is a program logic framework that addresses these challenges. It enables compositional derivation of goal-aware RSS rules, as we will describe in Section I-D.

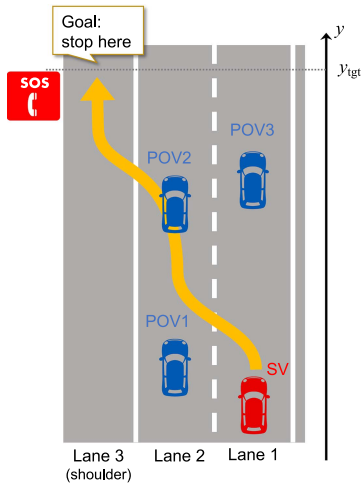


Fig. 2. The pull over scenario.

The following is our leading example for goal-aware RSS.

Example 1.5 (the pull over scenario): Consider the scenario shown in Fig. 2.[3] Here SV is initially in Lane 1; its goal is to pull over to Lane 3 (the shoulder) at the specified position y_{tgt}. There are three principal other vehicles (POVs); two POVs are in Lane 2 and the other is in Lane 1. This scenario is relevant to automated emergency stop, an important example of level-4 ADS conditions.

Our aim here is to design an RSS condition A and a proper response α that satisfy Requirements 1.4. We find that the design of such (A, α) is harder than in the collision-avoiding case in Example 1.3. Major challenges include the following.

- (Complexity of a scenario) Achieving the ultimate goal (stopping in Lane 3 at y_{tgt}) is achieved by a series of subgoals, such as changing lanes.
- (High-level manoeuvre planning) There can be multiple high-level manoeuvre sequences that are feasible. In the current scenario, they are specifically 1) to merge between POV2 and POV1, and 2) to merge after POV1. These will have different corresponding RSS conditions, and we have to systematically compute them.
- (Multiple constraints at odds) To merge between POV2 and POV1, SV may need to accelerate in Lane 1, in order to make enough space behind. However, doing so incurs the risk of driving too fast to stop at y_{tgt} in Lane 3.
- (Safety vs. goal-achievement) Proper responses should achieve both goal achievement and collision avoidance, which may be at odds as well. For example, the acceleration discussed above should also take into account the distance from POV3.

It is obvious that collision-avoiding RSS rules do not suffice to ensure goal achievement. For example, avoiding collision without an eye to the ultimate goal can trap SV in

[3]We assume that cars drive on the left, as in Japan, U.K. and other countries.

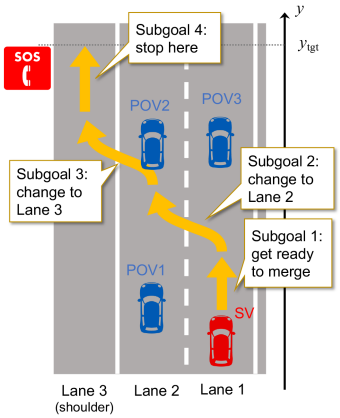


Fig. 3. Subgoals in the pull over scenario.

Lane 1, making it reach the position y_{tgt} without changing lanes. We experimentally show that this can indeed happen (Section VI).

D. Compositional Derivation of GA-RSS Rules by Program Logic

Reasoning under the level of complexity in Example 1.5 is hardly seen in the existing RSS literature. To address the challenge, in this paper, we propose a structured and compositional approach by the application of *program logic*.

More specifically, our approach in this work is

- firstly to decompose a scenario into *subscenarios*, along subgoals such as those shown in Fig. 3,
- to identify proper responses for each subscenario (which is easier since subscenarios are simpler, see Fig. 3), and
- to identify *preconditions* of those subscenario proper responses so that each precondition guarantee both 1) the goal of the subscenario and 2) the precondition of the subsequent proper response. Here we reason backwards along a sequence of subscenarios (from Subgoal 4 to Subgoal 1 in Fig. 3), much like backward predicate transformers in program logic [15].

A proper response for the whole scenario is then obtained by combining the proper responses for subscenarios; so is the corresponding RSS condition for the whole scenario.

In Section IV, we formulate the above workflow in terms of the program logic that we introduce in Section II—the latter is called *differential Floyd–Hoare logic* dFHL. The logic dFHL extends classic *Floyd–Hoare logic* [16] in 1) accommodation of continuous-time dynamics specified by ODEs and 2) what we call *safety conditions* that must hold all the time during execution. In Section II, we introduce derivation rules for dFHL that addresses these extensions; we prove their soundness too.

The second extension discussed above (safety conditions) makes the logic dFHL use *Hoare quadruples* $\{A\}\ \alpha\ \{B\} : S$, instead of triples $\{A\}\ \alpha\ \{B\}$ in the original Floyd–Hoare logic. This extension follows the idea formally presented in [17]; see Section I-G for further discussions. Explicating a safety condition S allows us to reason simultaneously about goal achievement (modelled by the postcondition B) and collision avoidance (modelled by S).

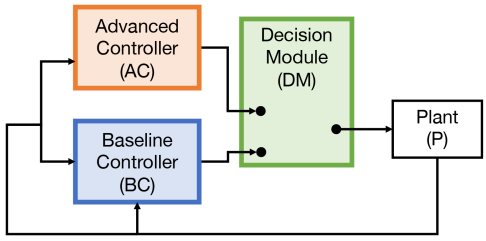


Fig. 4. The simplex architecture.

Our logic dFHL can be seen as a variant of Platzer's *differential dynamic logic dL* [18]—we believe that embedding of dFHL in dL is possible. Among a number of differences, a major one is our choice of the Hoare-style syntax ($\{A\}\ \alpha\ \{B\}$ with the safety extension $: S$ discussed above) rather than the dynamic logic-style one ($A \Rightarrow [\alpha]B$, as in dL). This syntactic choice fits the purpose of formalising our workflow (Section IV), where the emphasis is on compositional reasoning along sequential compositions.

We also discuss software support for the workflow in Section V. Our current implementation is partially formalised in the sense that 1) rule applications in dFHL are not formalised, but 2) symbolic reasoning about real numbers (such as solving quadratic equations) is formalised in Mathematica. Moreover, the interactive features of Mathematica notebooks are exploited so that even the informal part of reasoning is well-documented and thus trackable. We also discuss prospects of full formalisation.

E. RSS-Supervised Controller: RSS in a Safety Architecture

We continue Section I-B and discuss the usage of (CA- and GA-)RSS that is the most relevant to us, namely in a safety architecture. We use GA-RSS in this way to experimentally demonstrate its significance (Section VI).

A prototypical safety architecture is the *simplex architecture* shown in Fig. 4 [19], [20]. Here, the *advanced controller* (AC) is a complex controller that pursues not only safety but also other performance measures (such as comfort, progress, and fuel efficiency); the *baseline controller* (BC) is a simpler controller with a strong emphasis on safety; and the *decision module* (DM) switches between the two controllers. DM tries to use AC as often as possible for its superior performance. However, when DM finds that the current situation is safety critical, it switches to BC, whose behaviours are more predictable and easier to analyse.

The point of the simplex architecture is that the system's safety can be formally verified even if AC is a black box. Logically, DM enforces *contracts* that AC should respect. The safety of the whole system can then be established by formally reasoning about DM, the plant (P), and BC (that takes over the control in case AC cannot comply with the contracts). Specifically, to formally prove that a system guided by a simplex architecture is safe, it is enough to show that BC is, and that DM gives control

to BC soon enough. Safety of AC is irrelevant to the proof here. This point is especially appealing for ADS, whose AC typically involves a number of learning and optimisation components and thus is very hard to formally analyse.

The components of an RSS rule (A, α) map naturally to the simplex architecture:

- DM can be made so that it implements the RSS condition A. It uses AC as long as A is robustly satisfied; however, when A is about to be violated, it switches the control to BC.
- BC can implement the proper response α. Safety of its execution is then guaranteed by Requirements 1.1.
- If it happens that the robust satisfaction of the RSS condition A is restored during BC's execution, DM can switch back from BC to AC.

In this way, a possibly unsafe AC can be made safe, via suitable intervention of RSS-based DM and BC. The whole system built this way will be called an *RSS-supervised controller*.

In Section VI, we present our implementation of

- the GA-RSS rule for Example 1.5 (pull over) that we derive in Section IV, and
- the CA-RSS rule in Example 1.3 (one-way traffic), naturally adapted to the multi-lane setting of Example 1.5,

both in the simplex architecture (AC is a trajectory planner based on sampling and optimisation). Our experiments for the scenario in Example 1.5 demonstrate that the GA-RSS rule indeed achieves the goal safely, while the CA-RSS rule fails to do so.

F. Contributions

This paper introduces the idea of *goal-aware RSS* (GA-RSS). We claim that goal-aware RSS rules may be developed for complex scenarios, and that they are suited for use in the simplex architecture. These claims are backed up by the following technical contributions; they collectively establish a program logic framework for GA-RSS.

- We introduce a program logic dFHL (*differential Floyd–Hoare logic*) as a logical foundation for GA-RSS (Section II). It extends classic Floyd–Hoare logic by 1) differential dynamics and 2) safety conditions (S in our *Hoare quadruples* $\{A\}\,\alpha\,\{B\} : S$). We introduce derivation rules for dFHL and prove their soundness.
  The main novelty here is dealing with the combination of the two extensions, specifically in the (DWH) rule in Fig. 7.
- We develop a compositional workflow for deriving GA-RSS rules. It is formulated in terms of dFHL, exploiting the organising power of the logic. We also describe software support for the workflow by Mathematica.
- We run the workflow for the pull over scenario (Example 1.5). The resulting GA-RSS rule is implemented in the simplex architecture. Its experimental comparison with 1) no simplex and 2) a CA-RSS rule demonstrates the value of GA-RSS.

G. Related and Future Work

Much of the related work on RSS has been already discussed. Recent extensions of RSS include a risk-aware one [14] and one that allows swerves as evasive manoeuvres [21]. These extensions shall be pursued in our current goal-aware framework. In particular, allowing swerves should be possible, and it will significantly improve the progress of a RSS-supervised controller.

Inclusion of safety conditions in the Floyd–Hoare logic—in the form of Hoare quadruples—is also pursued in [17], in the context of verification of concurrent systems. Our logic dFHL combines the idea with the machinery of dL [18] for handling continuous dynamics. In particular, our main technical novelty—namely an inference rule for continuous dynamics and safety (Section II-B)—does not appear in [17], [18].

Some RSS rules have been implemented and are offered as a library [22]. Integration of the goal-aware RSS rules we derive in this paper, in the library, is future work. One advantage of doing so is that the GA-RSS rules will then accommodate varying road shapes.

This paper studies logical derivation of GA-RSS rules, with a prospect of *fully formal* derivation (see Section V-C). The problem of formally verifying correctness of RSS rules is formulated and investigated in [23]. Their formulation is based on a rigorous notion of signal; they argue that none of the existing *automated* verification tools is suited for the verification problem. This concurs with our experience so far—in particular, formal treatment of other participants' responsibilities (in the RSS sense) seems to require human intervention. At the same time, in our preliminary manual verification experience in KeYmaera X, we see a lot of automation opportunities. Developing proof tactics dedicated to those will ease manual verification efforts.

An idea similar to that of RSS-supervised controllers (Section I-E) is found in [24]. BC in [24] is a learning-based controller that is realised by an RNN and is trained to follow given safety rules. This is unlike our RSS-based BC that executes explicitly RSS proper responses. There is no statistical learning, hence no uncertainties from black-box learning, in our BC.

In [6], a rigorous guarantee of ADS safety is pursued via the notion of *invariably safe set*. The latter is defined in terms of backward reachability analysis, and in that sense, the work is similar to RSS and the current work. The biggest difference is that the approach in [6] is about *runtime* and *numeric* verification while RSS is about *static*, *a priori* and *symbolic* rules. Consequently, many usages of RSS discussed in Section I-B do not apply to [6]. Moreover, the symbolic nature of RSS is what allows compositional derivation of rules, the key contribution of this work. At the same time, the numeric and online nature of [6] will probably yield less conservative control actions. Overall, it seems that the two works target at different classes of driving situations: [6] for urban scenarios (less structure, shorter-term control); this work is for highway scenarios (more structure, longer-term control).

Formal (logical, deductive) verification of ADS safety is also pursued in [25] using the interactive theorem prover Isabelle/HOL [26]. The work uses a white-box model of a controller, and a controller must be very simple. This is unlike RSS and the current work, which allows black-box ACs and thus accommodates various real-world controllers such as sampling-based path planners (Section I-E).

In the presence of perceptual uncertainties (such as errors in position measurement and object recognition), it becomes harder for BCs and DMs to ensure safety. Making BCs tolerant of perceptual uncertainties is pursued in [27], [28]. One way to adapt DMs is to enrich their input so that they can better detect potential hazards. Feeding DNNs' confidence scores is proposed in [29]; in [30], it is proposed for DMs to look at inconsistencies between perceptual data of different modes.

H. Organisation of the Paper

In Section II, we introduce our program logic dFHL, introducing its syntax, semantics, and derivation rules. We prove the soundness of the derivation rules, too (Theorem 2.14). In Section III, we formulate our problem of deriving GA-RSS rules, based on the mathematical notion of driving scenario that we also introduce there. Our workflow for compositional derivation of GA-RSS rules is presented in Section IV, where our main theorem is the correctness of the workflow (Theorem 4.11) assuming the correctness in each subscenario (the condition (15)). We use the pull over scenario (Example 1.5) as a leading example, and derive a GA-RSS rule for it. Software support for the workflow is discussed in Section V (the current partially formal one and the prospects of full formalisation). Our experimental evaluation is discussed in Section VI, where our implementation of the GA-RSS rules for Example 1.5 in the simplex architecture is compared with those without BC and with CA-RSS. In Section VII we conclude.

II. Differential Floyd-Hoare Logic dFHL

A. The Syntax of dFHL: Assertions, Hybrid Programs, and Hoare Quadruples

Notation 2.1: In this paper, we let $[1, N]$ denote the set $\{1, 2, \dots, N\}$ of integers, where N is a positive integer.

1) Overview of the Syntax: In this section, we describe three ingredients to formalise our rules: assertions, hybrid programs, and Hoare quadruples. Let us give some intuition before we delve into formal definitions.

Assertions are logical objects describing qualitative properties of states. For example, in the one-way traffic scenario of Example 1.3, if we denote by y_f and v_f the position and velocity of the front car and y_r and v_r those of the rear car, we will write an assertion $\neg(v_f = 0 \wedge v_r = 0) \wedge y_r < y_f$ to describe the configurations where at least one car is not stopped and for which there is no collision.

Hybrid programs are a combination of usual programs of imperative languages (such as IMP [31]) and differential equations that express continuous dynamics. Our syntax therefore contains assignments, if-branchings, etc., but also constructs allowing the state to change following the solutions of differential equations. The terminology "hybrid program" comes from differential dynamic logic [18], which uses a slightly different syntax, but to which our syntax can be translated.

Finally, *Hoare quadruples* $\{A\}\,\alpha\,\{B\} : S$ relate both assertions and hybrid programs to formally specify and prove correctness of the latter. The traditional Floyd–Hoare logic [16] uses *Hoare triples* $\{A\}\ \alpha\ \{B\}$ that roughly means the truth of a *precondition* A guarantees the truth of a *postcondition* B after the execution of a program α. In dFHL, following [17], we extend the above classic syntax and write

$$\{A\}\ \alpha\ \{B\} : S$$

with the intention that

- every execution of the hybrid program α, if it starts from a state satisfying the assertion A (the precondition),
- terminates in a state satisfying the assertion B (the postcondition), and
- moreover, respects the assertion S (the *safety condition*) at all times during the execution.

The addition of a safety condition S allows us to reason about collision avoidance in RSS, while the goal of a scenario is naturally modelled as a postcondition. Later in Section III, driving scenarios and GA-RSS rules (cf. Section I-C) are modelled as components of Hoare quadruples.

Assertions, hybrid programs, and Hoare quadruples form the syntax of *differential Floyd-Hoare logic* (dFHL for short).

2) Formal Definition: We formally define dFHL. An example is in Example 2.10.

Definition 2.2 ((dFHL) assertions): A *term* is a rational polynomial on a fixed infinite set V of variables. *dFHL assertions* are generated by the grammar

$$A, B ::= \mathsf{true} \mid e \sim f \mid A \wedge B \mid A \vee B \mid \neg A \mid A \Rightarrow B$$

where e, f are terms and $\sim \in \{=, \leq, <, \neq\}$.

A dFHL assertion can be *open* or *closed* (or both, or none). *Openness* and *closedness* are defined recursively: true is both open and closed, $e < f$ and $e \neq f$ are open, $e \leq f$ and $e = f$ are closed, $A \wedge B$ and $A \vee B$ are open (resp. closed) if both components are, $\neg A$ is open (resp. closed) if A is closed (resp. open), and $A \Rightarrow B$ is open (resp. closed) if A is closed and B open (resp. A open and B closed). Note that open dFHL assertions describe open subsets of $\mathbb{R}^V$.

Definition 2.3: Hybrid programs (or *dFHL programs*) are given by the syntax:

$$\begin{aligned}\alpha, \beta \quad ::= \quad & \mathsf{skip} \mid \alpha; \beta \mid x := e \mid \mathsf{if}\,(A)\,\alpha\,\mathsf{else}\,\beta \mid \\ & \mathsf{while}\,(A)\,\alpha \mid \mathsf{dwhile}\,(A)\,\{\dot{\mathbf{x}} = \mathbf{f}\}\,.\end{aligned}$$

We sometimes drop the braces in $\mathsf{dwhile}\,(A)\{\dot{\mathbf{x}} = \mathbf{f}\}$ for readability. In $\mathsf{dwhile}\,(A)\,\{\dot{\mathbf{x}} = \mathbf{f}\}$, $\mathbf{x}$ and $\mathbf{f}$ are lists of the same length, respectively of (distinct) variables and terms, and A is open.

All constructs are usual ones from imperative programming, except for the *differential while* construct dwhile. It encodes the differential dynamics: $\dot{\mathbf{x}} = \mathbf{f}$ denotes a system of differential equations, and $\mathsf{dwhile}\,(A)\,\{\dot{\mathbf{x}} = \mathbf{f}\}$ denotes a dynamical system following the differential equations until the condition A is falsified. Openness of A ensures that, if A is falsified at some point, then there is the smallest time t_0 when it is falsified, and the system follows the dynamics for time t_0.

Remark 2.4: It is possible to extend the language of terms, by allowing more functions than just polynomials. In that case, the syntax $\dot{\mathbf{x}} = \mathbf{f}$ is only allowed when $\mathbf{f}$ is locally Lipschitz

$$\frac{}{\langle \mathsf{skip}; \beta, \rho\rangle \to \langle \beta, \rho\rangle} \qquad \frac{\langle \alpha, \rho\rangle \to \langle \alpha', \rho'\rangle}{\langle \alpha; \beta, \rho\rangle \to \langle \alpha'; \beta, \rho'\rangle} \qquad \frac{}{\langle x := e, \rho\rangle \to \langle \mathsf{skip}, \rho[x \to [\![e]\!]_\rho]\rangle} \qquad \frac{\rho \vDash A}{\langle \mathsf{if}\ (A)\ \alpha\ \mathsf{else}\ \beta, \rho\rangle \to \langle \alpha, \rho\rangle}$$

$$\frac{\rho \nvDash A}{\langle \mathsf{if}\ (A)\ \alpha\ \mathsf{else}\ \beta, \rho\rangle \to \langle \beta, \rho\rangle} \qquad \frac{\rho \nvDash A}{\langle \mathsf{while}\ (A)\ \alpha, \rho\rangle \to \langle \mathsf{skip}, \rho\rangle} \qquad \frac{\rho \vDash A}{\langle \mathsf{while}\ (A)\ \alpha, \rho\rangle \to \langle \alpha; \mathsf{while}\ (A)\ \alpha, \rho\rangle}$$

$$\frac{t \geq 0 \quad \hat{x}(0) = \rho \quad \frac{d\hat{x}}{dt}(t) = [\![\mathbf{f}]\!]_{\hat{x}(t)} \quad \rho' = \hat{x}(t) \quad \forall t' \leq t.\, \hat{x}(t') \vDash A}{\langle \mathsf{dwhile}\ (A)\ \{\, \dot{\mathbf{x}} = \mathbf{f}\, \}, \rho\rangle \to \langle \mathsf{dwhile}\ (A)\ \{\, \dot{\mathbf{x}} = \mathbf{f}\, \}, \rho'\rangle}\ (*)$$

$$\frac{t \geq 0 \quad \hat{x}(0) = \rho \quad \frac{d\hat{x}}{dt}(t) = [\![\mathbf{f}]\!]_{\hat{x}(t)} \quad \rho' = \hat{x}(t) \quad \forall t' < t.\, \hat{x}(t') \vDash A \quad \hat{x}(t) \nvDash A}{\langle \mathsf{dwhile}\ (A)\ \{\, \dot{\mathbf{x}} = \mathbf{f}\, \}, \rho\rangle \to \langle \mathsf{skip}, \rho'\rangle}\ (*)$$

Fig. 5. Reduction relation $\to$ on states; rules annotated with $(*)$ are discussed in Remark 2.6.

continuous to ensure existence and uniqueness of solutions, by the Picard-Lindelöf theorem. One should also make sure that any term of the syntax possesses partial derivatives with respect to all variables in order to use the rules of Section II-B.

Our programming language syntax (Definition 2.3) is inspired by that in dL [18], but comes with significant changes. It is imperative and deterministic (see Lemma 2.8), which makes it easier to use for practitioners, while expressive enough to encode interesting models. This also makes it more suited to Hoare logic and total correctness, which is crucial for applications to automated driving.

We define an operational semantics for our syntax:

Definition 2.5 (semantics): A *store* is a function from variables to reals. *Store update* is denoted $\rho[x \to v]$; it maps x to v and any other variable x' to $\rho(x')$. The *value* $[\![e]\!]_\rho$ of a term e in a store ρ is a real defined as usual by induction on e (see for example [31, Section 2.2]). The *satisfaction* relation between stores ρ and dFHL assertions A, denoted $\rho \vDash A$, is also defined as usual (see [31, Section 2.3]).

A *state* is a pair $\langle \alpha, \rho\rangle$ of a hybrid program and a store. The *reduction* relation on states is defined in Fig. 5. A state s *reduces* to s' if $s \to^* s'$, where $\to^*$ is the reflexive transitive closure of $\to$. A state s *converges* to ρ, denoted $s \Downarrow \rho$, if there exists a reduction sequence $s \to^* \langle \mathsf{skip}, \rho\rangle$.

Let us explain how to read Fig. 5: hypotheses are listed above the horizontal line, and the conclusion below it. For example, $\langle \mathsf{skip}; \beta, \rho\rangle$ can always reduce to $\langle \beta, \rho\rangle$ (there are no hypotheses), and if $\langle \alpha, \rho\rangle$ reduces to $\langle \alpha', \rho'\rangle$, then $\langle \alpha; \beta, \rho\rangle$ reduces to $\langle \alpha'; \beta, \rho'\rangle$.

Remark 2.6: In the reduction rules for dwhile, $\hat{x}$ is the global solution to the differential equation $\dot{\mathbf{x}} = \mathbf{f}$ with initial condition $\hat{x}(0) = \rho$. As a side condition (left untold for readability in the rule), we assume that all variables not mentioned in $\mathbf{x}$ are left untouched during the transition, so if y is not in $\mathbf{x}$, then $\rho'(y) = \rho(y)$.

Convergence $\Downarrow$ corresponds to complete executions of programs (until termination), while reduction $\to^*$ corresponds to potentially partial executions, which can reach any intermediate state of the computation.

Example 2.7: The state $\langle \alpha, \rho\rangle$, where

$$\alpha = (\mathsf{dwhile}(x > 0)\, \{\dot{x} = -1\}\,; \quad x := x - 1)\,,$$

and $\rho(x) = 2$, can reduce

- to $\langle \alpha, \rho[x \to v]\rangle$ for any $v \in (0, 2]$,
- to $\langle x := x - 1, \rho[x \to 0]\rangle$,
- and to $\langle \mathsf{skip}, \rho[x \to -1]\rangle$,

but only the last one corresponds to convergence (namely $\langle \alpha, \rho\rangle \Downarrow \rho[x \to -1]$).

Lemma 2.8 (confluence): Our language of hybrid programs is confluent. That is, if $s \to^* s_1$ and $s \to^* s_2$, then there exists s' such that $s_1 \to^* s'$ and $s_2 \to^* s'$. In particular, if $s \Downarrow \rho$, s cannot converge to any other store $\rho' \neq \rho$.

Confluence basically means that the language is deterministic in the sense that, no matter the reduction sequence, a program always converges to the same value. This holds because reduction in our language is mainly deterministic, except for the dwhile rules, in which case the reduction that has run for the smaller amount of time can be reduced again to catch up with the other reduction.

Finally, we define validity of Hoare quadruples:

Definition 2.9 (Hoare quadruples): A *Hoare quadruple* is a quadruple $\{A\}\ \alpha\ \{B\} : S$ of three dFHL assertions A, B, and S, and a hybrid program α. It is *valid* if, for all stores ρ such that $\rho \vDash A$,

- there exists ρ' such that $\langle \alpha, \rho\rangle \Downarrow \rho'$ and $\rho' \vDash B$, and
- for all reduction sequences $\langle \alpha, \rho\rangle \to^* \langle \beta, \rho'\rangle$, $\rho' \vDash S$.

Hoare quadruples have safety conditions S in addition to the usual components of Hoare triples. They are required to specify safety properties, which must hold at all times. In traditional programming, one is usually only interested in input-output behaviours: as long as a program returns a valid value, it does not matter which intermediate states it went through. In contrast, the intermediate states matter in our case, since there may be a collision or safety violation halfway through an execution that reaches the desired target.

The safety of all intermediate states is ensured by the definition of $s \to s'$. The interesting case is that of the differential dynamics, where the dynamics can be stopped at any point in time, and thus s' can be the state reached at any point of the dynamics.

Also note that this semantics is *total correctness*, rather than *partial correctness*. A Hoare triple $\{A\}\ \alpha\ \{B\}$ is valid for partial correctness if, roughly, any terminating execution of α under the precondition A satisfies the postcondition B. In particular, if α is not terminating, then the Hoare triple is trivially true, regardless of the truth of B. This is not desired

$$
\begin{array}{ll}
1 & t := 0\ ; \\
2 & \mathsf{dwhile}\,(v_f > 0 \wedge t < \rho)\left\{ \boxed{\delta_f}\ ,\ \boxed{\delta_r^1} \right\}\ ; \\
3 & \mathsf{if}\,(v_f = 0)\left[\mathsf{dwhile}\,(t < \rho)\boxed{\delta_r^1}\ ; \mathsf{dwhile}\,(v_r > 0)\boxed{\delta_r^2} \right] \\
4 & \mathsf{else} \\
5 & \quad \mathsf{dwhile}\,(v_f > 0 \wedge v_r > 0)\left\{ \boxed{\delta_f}\ ,\ \boxed{\delta_r^2} \right\}\ ; \\
6 & \quad \mathsf{if}\,(v_f = 0)\,\mathsf{dwhile}\,(v_r > 0)\boxed{\delta_r^2} \\
7 & \quad \mathsf{else}\ \mathsf{dwhile}\,(v_f > 0)\boxed{\delta_f}
\end{array}
$$

Fig. 6. Hybrid program α for the one-way traffic scenario (Example 2.10).

since we want to ensure goal achievement (modelled by the postcondition B). In contrast, total correctness additionally requires the existence of a terminating execution, which suits our purpose. See [31] for more details on partial and total correctness.

Example 2.10 (the one-way traffic scenario): The scenario for Example 1.3 can be modelled in dFHL. We start by modelling the dynamics of the different agents involved in the scenario (the front and rear cars) as a hybrid program α (see Fig. 6). We then model the property that we want to show (namely, that the cars can stop without colliding if they are far enough apart) into a Hoare quadruple (see (4)). In what follows we explain the modelling (α in Fig. 6 and the Hoare quadruple in (4)). We defer the proof of validity of (4) to Example 2.15.

In this scenario, we aim to show that whatever the front car is doing, the rear car can properly respond without colliding with the front car, as long as it respects the RSS safety distance ($\mathsf{dRSS}(v_f, v_r)$ in (2)).

The worst case is when the front car breaks at the maximal braking rate $b_{\max}$, while the rear car is accelerating with the maximal acceleration rate $a_{\max}$ during the reaction time ρ before engaging the proper response α (namely decelerating with the maximal comfortable braking rate $b_{\min}$). If we denote by y_f and v_f the position and velocity of the front car and by y_r and v_r those of the rear car, then these behaviours of the cars can be written as the hybrid programs

$$
\begin{aligned}
\alpha_f &= \left(\mathsf{dwhile}\,(v_f > 0)\boxed{\delta_f}\right), \\
\alpha_r &= \left(t := 0; \mathsf{dwhile}\,(t < \rho)\boxed{\delta_r^1}; \mathsf{dwhile}\,(v_r > 0)\boxed{\delta_r^2}\right),
\end{aligned}
$$

where the δ's represent the dynamics of each car:

$$
\begin{aligned}
\delta_f &= (\dot{y_f} = v_f, \dot{v_f} = -b_{\max}), \\
\delta_r^1 &= \left(\dot{y_r} = v_r, \dot{v_r} = a_{\max}, \dot{t} = 1\right), \\
\delta_r^2 &= (\dot{y_r} = v_r, \dot{v_r} = -b_{\min}). \qquad (3)
\end{aligned}
$$

By manually combining the two hybrid programs α_f, α_r—letting them run in parallel—we obtain the hybrid program α in Fig. 6.

This hybrid program α models the situation in which both the front and rear cars are following their worst case behaviours as long as they are not both stopped. The logical structure of α enumerates all the different states the scenario can be in: whether the front car has stopped braking or not, and whether the rear car is still accelerating, has engaged the proper response, or finished braking. For example, the dwhile on Line 2 of α in Fig. 6 corresponds to a state where the front car is braking and the rear car is still accelerating, and the if on Line 3 corresponds to the case where the front car has stopped braking before the rear car starts engaging the proper response.

We can then model the whole scenario as the following Hoare quadruple.

$$
\left\{ \begin{array}{l} v_r \geq 0 \wedge v_f \geq 0 \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r) \end{array} \right\}\ \alpha\ \begin{array}{r} \{v_r = 0 \wedge v_f = 0\} \\ : y_r < y_f. \end{array} \qquad (4)
$$

The postcondition states that both cars have stopped, while the precondition models the situations in which we want to prove that there is no collision (namely, when the cars are farther than an RSS safety distance apart). The safety condition $y_r < y_f$ models the fact that there is no collision along the dynamics. We will prove that this dFHL quadruple is valid, using derivation rules for dFHL, in Example 2.15.

B. Derivation Rules in dFHL

We present a set of rules to derive valid Hoare quadruples, listed in Fig. 7. Like the rules in Fig. 5, hypotheses are listed above the horizontal line, and the conclusion below it. For example, the (SEQ) rule can be read as: if $\{A\}\ \alpha\ \{B\} : S$ and $\{B\}\,\beta\,\{C\} : S$ are provable in dFHL, then so is $\{A\}\,\alpha;\beta\,\{C\} : S$.

Assumption 2.11: In the (WH) rule, $\gtrsim\ \in \{>, \geq\}$ (meaning that all the occurrences of $\gtrsim$ should be replaced with the same $>$ or $\geq$) and x is a fresh variable. In the (DWH) rule, $(\sim, \simeq) \in \{(=,=), (>,\geq), (\geq,\geq)\}$, and the dynamics $\dot{\mathbf{x}} = \mathbf{f}$ is assumed to have a global solution.

Note that the existence of global solutions is not a constraint in practice, since their non-existence would imply that some physical quantity diverges to infinity, which is impossible in a physical system.

Some hypotheses of the (DWH) and (LIMP) rules are dFHL assertions, by which we mean that these assertions must be *valid*, that is, satisfied by all stores. For example, the precondition $A \Rightarrow e_{\mathsf{var}} \geq 0$ means that, for any ρ, if $\rho \vDash A$, then $[\![e_{\mathsf{var}}]\!]_\rho \geq 0$.

Most of the rules in Fig. 7 are standard Hoare logic rules when stripped of their safety conditions, so we only discuss the exceptions: (WH), (DWH) and (DWH-SOL).

1) The (WH) Rule: The first exception is the (WH) rule, which is an alternative used for *total* correctness (similar for example to the one found in [32]), while Hoare logic is more often used for *partial* correctness. The parts that prove total correctness are those that involve the *variant* e_{var}, which decreases with each iteration by at least 1, and must be positive (or non-negative), so the loop must stop at some point. This notion of variant is similar to those of *ranking function* [33] and *Lyapunov function* [34].

2) The (DWH) Rule: The second exception is the (DWH) rule, which uses the notion of *Lie derivative*. The term $\mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e$ is called the *Lie derivative of* e *with respect to the dynamics* $\dot{\mathbf{x}} = \mathbf{f}$. If $\mathbf{x}$ is the list of variables $x_1, \ldots, x_n$, and $\mathbf{f}$ is the list of terms

$$\frac{}{\{A\}\ \mathsf{skip}\ \{A\} : A}\ (\textsc{Skip}) \qquad \frac{\{A\}\ \alpha\ \{B\} : S \qquad \{B\}\ \beta\ \{C\} : S}{\{A\}\ \alpha;\beta\ \{C\} : S}\ (\textsc{Seq}) \qquad \frac{}{\{A[e/x]\}\ x := e\ \{A\} : A \vee A[e/x]}\ (\textsc{Assign})$$

$$\frac{\begin{array}{c}\{A \wedge B\}\ \alpha\ \{C\} : S \\ \{\neg A \wedge B\}\ \beta\ \{C\} : S\end{array}}{\{B\}\ \mathsf{if}\,(A)\,\alpha\,\mathsf{else}\,\beta\ \{C\} : S}\ (\textsc{If}) \qquad \frac{\{A \wedge B \wedge e_{\mathsf{var}} \gtrsim 0 \wedge e_{\mathsf{var}} = x\}\ \alpha\ \{B \wedge e_{\mathsf{var}} \gtrsim 0 \wedge e_{\mathsf{var}} \leq x - 1\} : S}{\{B \wedge e_{\mathsf{var}} \gtrsim 0\}\ \mathsf{while}\,(A)\,\alpha\ \{\neg A \wedge B \wedge e_{\mathsf{var}} \gtrsim 0\} : S}\ (\textsc{Wh})^{\dagger}$$

$$\frac{\begin{array}{ll}\mathsf{inv}: & A \Rightarrow e_{\mathsf{inv}} \sim 0 \quad e_{\mathsf{var}} \geq 0 \wedge e_{\mathsf{inv}} \sim 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{inv}} \simeq 0 \\ \mathsf{var}: & A \Rightarrow e_{\mathsf{var}} \geq 0 \quad e_{\mathsf{var}} \geq 0 \wedge e_{\mathsf{inv}} \sim 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{var}} \leq e_{\mathsf{ter}} \\ \mathsf{ter}: & A \Rightarrow e_{\mathsf{ter}} < 0 \quad e_{\mathsf{var}} \geq 0 \wedge e_{\mathsf{inv}} \sim 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{ter}} \leq 0\end{array}}{\{A\}\ \mathsf{dwhile}\,(e_{\mathsf{var}} > 0)\,\dot{\mathbf{x}} = \mathbf{f}\ \{e_{\mathsf{var}} = 0 \wedge e_{\mathsf{inv}} \sim 0\} : e_{\mathsf{inv}} \sim 0 \wedge e_{\mathsf{var}} \geq 0}\ (\textsc{DWh})^{\dagger} \qquad \frac{\begin{array}{c}A \Rightarrow A' \\ \{A'\}\ \alpha\ \{B'\} : S' \quad S' \wedge B' \Rightarrow B \\ S' \Rightarrow S\end{array}}{\{A\}\ \alpha\ \{B\} : S}\ (\textsc{Limp})$$

$$\frac{\{A\}\ \alpha\ \{B\} : S \qquad \{A\}\ \alpha\ \{B'\} : S'}{\{A\}\ \alpha\ \{B \wedge B'\} : S \wedge S'}\ (\textsc{Conj}) \qquad \frac{A_0 \Rightarrow (\exists t \geq 0.\ C_{<t} \wedge \neg C_t \wedge B_t \wedge S_{\leq t})}{\{A\}\ \mathsf{dwhile}\,(C)\,\dot{\mathbf{x}} = \mathbf{f}\ \{B\} : S}\ (\textsc{DWh-Sol})$$

Fig. 7. dFHL rules for total correctness; rules with † have side conditions discussed in Assumption 2.11. Rule (DWH-SOL) is discussed separately in Section II-B3.

$f_1, \ldots, f_n$, its formal definition is

$$\mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e = \sum_{i=1}^{n} \frac{\partial e}{\partial x_i} f_i,$$

where the terms $\frac{\partial e}{\partial x_i}$ are the *partial derivatives of* e whose definitions are by induction on the structure of the term e as usual. The fundamental lemma of the Lie derivative (see [35]), crucial for proving the soundness of the (DWH) rule is the following:

Lemma 2.12: Assume given any solution $\hat{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ of the differential equations $\dot{\mathbf{x}} = \mathbf{f}$. Then the derivative of the function $t \mapsto [\![e]\!]_{\hat{x}(t)}$ is given by $t \mapsto [\![\mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e]\!]_{\hat{x}(t)}$.

Proof: This is just an application of the chain rule. ■

The (DWH) rule is similar to the (WH) rule, in that it contains an *invariant* e_{inv} (B in (WH)), a *variant* e_{var}, and a *terminator* e_{ter} (decreasing by 1 in (WH)). The inv condition states that the invariant holds at the start and is preserved by the dynamics, so it must hold at all times along the dynamics.

The other conditions are only present to ensure that the loop eventually terminates. The var condition essentially means that the variant e_{var} must decrease along the dynamics (if the terminator e_{ter} is always negative). But this is not enough, as the variant could get asymptotically closer to 0, without ever reaching it. The condition ter ensures that this never happens, by showing that the terminator e_{ter} is not only negative, but below a fixed negative value.

Note that, even though the (DWH) rule may look like it is about a single variable, e_{var} is typically a term that contains several variables, which makes it expressive enough to prove interesting properties of driving systems. For example, in (18) in Section IV-D, e_{var} depends on both y and v.

Remark 2.13: We note that (WH) and (DWH) can be made more general. For example, instead of the usual order on a single term e_{var}, (WH) could use a lexicographic order on several terms, or any well-founded order. This is also true for (DWH), where we could use more general forms than $e_{\mathsf{var}} > 0$ as the variant.

Indeed, in Section IV, we will use a "multiple-invariant multiple-variant" generalization of (DWH); it is presented in Fig. 19 in Appendix A1. In this section, we use the current simpler forms of the rules that are easy to describe and manipulate, and yet share their essence with the generalized forms.

3) The (DWH-SOL) Rule: Finally, let us discuss the (DWH-SOL) rule in detail. It uses explicit solutions, which makes it further from the spirit of Hoare logic, but it is still valid. Let us assume that $\dot{\mathbf{x}} = \mathbf{f}$ has a closed form solution, that is, a function $\hat{x} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ such that $\hat{x}(x_0, 0) = x_0$ and $\frac{d\hat{x}}{dt}(x_0, t) = [\![\mathbf{f}]\!]_{\hat{x}(x_0,t)}$. The only premise of (DWH-SOL) is the dFHL assertion shown in Fig. 7, where A_t is a shorthand for the assertion $A[\hat{x}(x_0, t)/x]$ (and similarly for B_t and C_t), while $C_{<t}$ is a shorthand for $\forall s \in [0, t).l[\hat{x}(x_0, s)/x]$ (and similarly for $S_{\leq t}$). Intuitively, this rule means that for all states x where the assertion A holds, there is some time t when the condition C just becomes false, and it is enough to prove the assertion B holds at time t, and that S holds for all times from 0 to t.

4) The case Construct: If we denote by $\mathsf{case}\,(A_1)\,\alpha_1 \ldots (A_n)\,\alpha_n$ the obvious nesting of if constructs, then the following rule can be derived from repeated uses of (IF) and (LIMP):

$$\frac{\{A_1\}\,\alpha_1\,\{B\} : S \quad \ldots \quad \{A_n\}\,\alpha_n\,\{B\} : S}{\{\bigvee_{i=1}^{n} A_i\}\,\mathsf{case}(A_1)\alpha_1 \ldots (A_n)\,\alpha_n\,\{B\} : S}(\textsc{Case}) \qquad (5)$$

It is useful in our framework for automated driving: given hybrid programs $\alpha_1, \ldots, \alpha_n$ that satisfy the same postcondition B and safety condition S, but with different preconditions A_i, $\mathsf{case}\,(A_1)\,\alpha_1 \ldots (A_n)\,\alpha_n$ also satisfies B and S, but on the more general precondition $A_1 \vee \ldots \vee A_n$, as demonstrated in Section IV-E.

5) Soundness: Soundness of dFHL can be proved:

Theorem 2.14: Only valid Hoare quadruples can be proved in dFHL.

Proof: The proof is done by induction on the size of the proof tree and case analysis of the first rule used. All cases are rather standard except for the additional requirement of the safety condition, so let us develop only the case when the last rule is (DWH) in details.

Let us assume the premises of (DWH) are valid, and assume given a store ρ such that $\rho \vDash A$. The goal is to prove that $\langle \mathsf{dwhile}\,(e_{\mathsf{var}} > 0)\{\dot{\mathbf{x}} = \mathbf{f}\}, \rho\rangle$ converges to a store ρ' with $\rho' \vDash e_{\mathsf{var}} = 0 \wedge e_{\mathsf{inv}} \sim 0$, and for all reduction sequences $\langle \mathsf{dwhile}\,(e_{\mathsf{var}} > 0)\{\dot{\mathbf{x}} = \mathbf{f}\}, \rho\rangle \to^* \langle \alpha, \rho''\rangle, \rho'' \vDash e_{\mathsf{inv}} \sim 0 \wedge e_{\mathsf{var}} \geq 0$. Let $\hat{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ be the solution of $\dot{\mathbf{x}} = \mathbf{f}$ with $\hat{x}(0) = \rho$, $K = \{t \geq 0 \mid \forall t' \leq t.[\![e_{\mathsf{var}}]\!]_{\hat{x}(t)} > 0\}$, and $K' =$

$\{t \geq 0 | \forall t' \leq t. [\![e_{\mathsf{var}}]\!]_{\hat{x}(t)} > 0 \wedge [\![e_{\mathsf{inv}}]\!]_{\hat{x}(t)} \sim 0\}$. Let T be the supremum of K'.

Step 1. $K = K'$ and $[\![e_{\mathsf{inv}}]\!]_{\hat{x}(T)} \sim 0$: For the first point, since $K' \subseteq K$, it is sufficient to show that $e_{\mathsf{inv}} \sim 0$ holds in K. Let us assume that $\sim$ is $=$ (other cases are similar). By inv and Lemma 2.12, the function $t \in K \mapsto [\![e_{\mathsf{inv}}]\!]_{\hat{x}(t)}$ is 0 at $t = 0$ and of constant derivative 0. This means its value at $t \in K$ is also 0, that is, $[\![e_{\mathsf{inv}}]\!]_{\hat{x}(t)} = 0$. For the second point, because the function above is continuous and constantly equal to 0 on K, we also have $[\![e_{\mathsf{inv}}]\!]_{\hat{x}(T)} \sim 0$ (other cases are similar).

Step 2. K is bounded: By Step 1, $[\![e_{\mathsf{inv}}]\!]_{\hat{x}(t)} \sim 0$ for all $t \in K$. By ter and Lemma 2.12, the function $t \mapsto [\![e_{\mathsf{ter}}]\!]_{\hat{x}(t)}$ is negative at $t = 0$ and of non-positive derivative, i.e., non-increasing, on K. This means that $[\![e_{\mathsf{ter}}]\!]_{\hat{x}(t)} \leq [\![e_{\mathsf{ter}}]\!]_{\rho}$. Similarly, by var and Lemma 2.12, the derivative of the function $v : t \mapsto [\![e_{\mathsf{var}}]\!]_{\hat{x}(t)}$ is bounded by $[\![e_{\mathsf{ter}}]\!]_{\hat{x}(t)}$. By monotonicity of integrals:

$$v(t) - v(0) = \int_{[0,t]} \dot{v}(t)dt \leq \int_{[0,t]} [\![e_{\mathsf{ter}}]\!]_{\rho} dt = t \cdot [\![e_{\mathsf{ter}}]\!]_{\rho},$$

so for $t > -\frac{[\![e_{\mathsf{var}}]\!]_{\rho}}{[\![e_{\mathsf{ter}}]\!]_{\rho}} \geq 0$, $[\![e_{\mathsf{var}}]\!]_{\hat{x}(t)} < 0$, so $t \notin K$, hence K is bounded by $-\frac{[\![e_{\mathsf{var}}]\!]_{\rho}}{[\![e_{\mathsf{ter}}]\!]_{\rho}}$.

Step 3. analysis of the exit time: By definition of K', the loop ends at time T (since T is finite by Step 2). By Step 1, T is also the supremum of K. Furthermore, by openness of the condition $e_{\mathsf{var}} > 0$ and the continuity of the solution $\hat{x}$, if T belonged to K, then there would exist $\epsilon > 0$, such that for all $t' \leq T + \epsilon, t' \in K$, which would contradict the supremality of T. This means that $[\![e_{\mathsf{var}}]\!]_{\hat{x}(T)} \leq 0$. Again by continuity of the solution, $[\![e_{\mathsf{var}}]\!]_{\hat{x}(T)}$ is in the closure of K, which is included in $\mathbb{R}_{\geq 0}$. Consequently, $[\![e_{\mathsf{var}}]\!]_{\hat{x}(T)} = 0$.

Step 4. convergence: The previous analysis implies that $\langle \mathsf{dwhile}\,(e_{\mathsf{var}} > 0)\{\dot{x} = f\}, \rho\rangle$ converges to $\rho' = \hat{x}(T)$, for which $\rho' \vDash e_{\mathsf{var}} = 0$ (by Step 3) and $\rho' \vDash e_{\mathsf{inv}} \sim 0$ (by Step 1).

Step 5. safety: By uniqueness of the solutions of $\dot{\mathbf{x}} = \mathbf{f}$, we can prove by induction on the number of reduction steps that if $\langle \mathsf{dwhile}\,(e_{\mathsf{var}} > 0)\{\dot{\mathbf{x}} = \mathbf{f}\}, \rho\rangle \to^* \langle \alpha, \rho''\rangle$, then $\rho'' = \hat{x}(t)$ for some $0 \leq t \leq T$. By Step 3, $\rho'' \vDash e_{\mathsf{var}} \geq 0$, and by Step 1, $\rho'' \vDash e_{\mathsf{inv}} \sim 0$. ∎

6) Example: We exemplify formal reasoning in dFHL using the one-way traffic scenario (Examples 1.3 and 2.10). We only show a typical part of the proof here, and refer the interested reader to Appendix A1 for the complete formal proof.

Example 2.15 (proving safety of the one-way traffic scenario): We show how to prove the validity of the Hoare quadruple (4)—which we shall write as $\{A\}\,\alpha\,\{B\} : y_r < y_f$—for the one-way traffic scenario in Example 2.10.

Here we use a slightly extended version of the (DWH) rule, namely one that combines several variants and invariants. See Remark 2.13. The exact form of the rule is given inFig. 19 in Appendix A1.

The proof then relies on finding an invariant that implies $y_r < y_f$ and that is preserved by the dynamics α. As always with proofs in program logics, finding a suitable invariant is difficult. In our case, a suitable invariant turns out to be

$$y_f - y_r - \mathsf{dRSS}(v_f, v_r, \rho - t) > 0, \tag{6}$$

where t is the current time (note that we make the ρ parameter explicit throughout the proof, because it is important there). Explicit use of the assertion (6) as an invariant is not common in the literature—it is not used in [1] for example—showing the subtlety of finding invariants.

Once a suitable invariant is found, constructing a dFHL proof is relatively simple: for each program construct, we apply the corresponding rule of dFHL. We present only part of the validity proof here, focusing on Line 2 of the program α (Fig. 6). The rest of the proof is similar.

We let α' denote Line 2, that is,

$$\alpha' = \left(\mathsf{dwhile}\,(v_f > 0 \wedge v_r > 0 \wedge t < \rho)\left\{\boxed{\delta_f}, \boxed{\delta_r^1}\right\}\right).$$

Here we used snippets δ_f, δ_r^1 from (3).

Then we want to prove that the following Hoare quadruple is valid:

$$\{A'\}\,\alpha'\,\{B'\} : y_f - y_r - \mathsf{dRSS}(v_f, v_r, \rho - t) > 0, \tag{7}$$

where

$$A' = \begin{pmatrix} v_r \geq 0 \wedge v_f \geq 0 \wedge t = 0 \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho) \end{pmatrix},$$

$$B' = \begin{pmatrix} ((v_f \geq 0 \wedge t = \rho) \vee (v_f = 0 \wedge t \leq \rho)) \wedge \\ v_r \geq 0 \wedge y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

This quadruple can be directly proved by applying the (DWH) rule (and the (LIMP) rule) with the following variants and invariants:

- $e_{\mathsf{inv},1} = (v_r \geq 0)$,
- $e_{\mathsf{inv},2} = (y_f - y_r - \mathsf{dRSS}\,(v_f, v_r, \rho - t) > 0)$,
- $e_{\mathsf{var},1} = v_f, \quad e_{\mathsf{ter},1} = -b_{\max}$,
- $e_{\mathsf{var},2} = \rho - t, \quad e_{\mathsf{ter},2} = -1$.

The only non-obvious point is that $e_{\mathsf{inv},2}$ is preserved by the dynamics. We first observe

$$\mathcal{L}_{\delta_f, \delta_r^1} e_{\mathsf{inv},2} = \begin{cases} 0 & \text{if } \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) \geq 0 \\ v_f - v_r & \text{otherwise,} \end{cases}$$

where $\mathsf{dRSS}_{\pm}(v_f, v_r, \rho)$ is given by

$$\mathsf{dRSS}_{\pm}(v_f, v_r, \rho) = v_r\rho + \frac{a_{\max}\rho^2}{2} + \frac{(v_r + a_{\max}\rho)^2}{2b_{\min}} - \frac{v_f^2}{2b_{\max}}.$$

Therefore, we can infer as follows.

$$\begin{aligned} &\mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) < 0 \\ &\iff v_r(\rho - t) + \frac{a_{\max}(\rho - t)^2}{2} \\ &\qquad + \frac{(v_r + a_{\max}(\rho - t))^2}{2b_{\min}} - \frac{v_f^2}{2b_{\max}} < 0 \\ &\implies \frac{(v_r + a_{\max}(\rho - t))^2}{2b_{\min}} - \frac{v_f^2}{2b_{\max}} < 0 && \text{(i)} \\ &\implies (v_r + a_{\max}(\rho - t))^2 < v_f^2 && \text{(ii)} \\ &\implies v_r^2 < v_f^2 && \text{(iii)} \\ &\implies v_r < v_f && \text{(iv)} \end{aligned}$$

Here (i) and (iii) are because $v_r \geq 0, t \leq \rho$ (by $e_{\mathsf{inv},1}$ and $e_{\mathsf{var},2}$), and $a_{\max} \geq 0$; (ii) because $0 \leq b_{\min} \leq b_{\max}$; and (iv) because $v_r \geq 0$ and $v_f \geq 0$ (the latter by $e_{\mathsf{var},1}$).

The argument above concludes that $e_{\mathsf{inv},2}$ is indeed an invariant, which establishes the validity of the Hoare quadruple (7) on Line 2 of α. Combining similar arguments, we prove the validity of the Hoare quadruple $\{A\}\ \alpha\ \{B\} : y_r < y_f$ (from (4)) for the one-way traffic scenario. The rest of the proof can be found in Appendix A1.

In the last example, in order to define dRSS (2) in dFHL, we needed to add the $\max$ operator to the syntax for terms. This is straightforward.

III. Problem Formulation

A. Modelling of Physical Components: Roads, Lanes, Occupancy, and Vehicle Dynamics

We use the double integrator model as done in the original RSS work [1]. Occupancy is lane-based. In changing lanes, a vehicle occupies two lanes—this modelling is reasonable in less-congested highway situations. This way we do not need to consider lateral positions of vehicles within a lane; this modelling is even simpler than the lane-based one in [6].

Concretely, we use integers to express lanes ($l = 1, 2, 3, \ldots$). A vehicle changing lanes from Lane 1 to 2 is expressed by $l = 1.5$; it means that 1) the vehicle occupies both Lanes 1 & 2, as discussed above, and 2) the vehicle is hence subject to the RSS distance responsibilities (Example 1.3) with respect to preceding vehicles both in Lanes 1 & 2.

Our scope here is driving situations that are highly structured and thus allow abstract modelling in terms of lane occupancy. This is the case typically with highway traffic situations. Many other works, such as [36], study less structured driving situations; their scope is therefore different from ours.

B. Scenario Modelling

What constitutes a mathematical notion of "driving scenario" is a difficult question—its answer can change depending on the intended model granularity and the goal of modelling. For our purpose of compositional derivation of goal-aware RSS rules in dFHL, we propose the following definition.

Definition 3.1 ((driving) scenario): A *(driving) scenario* is a quadruple $\mathcal{S} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal})$, where

- $\mathbf{Var}$ is a finite set of variables;
- Safe is a dFHL assertion called a *safety condition*;
- Env is a dFHL assertion called a *environmental condition*; and
- Goal is a dFHL assertion called a *goal*.

It is required that all the variables occurring in Safe, Env, and Goal belong to $\mathbf{Var}$.

The set $\mathbf{Var}$ should cover all the variables that are used for rule derivation; it is a modelling of the physical components involved in the driving scenario in question. We follow Section III-A in deciding $\mathbf{Var}$.

The three assertions $\mathsf{Safe}, \mathsf{Env}, \mathsf{Goal}$ describe different aspects of a driving scenario, and are thus used differently in our rule derivation workflow (Section IV).

- Safe describes safety conditions for collision avoidance. SV should satisfy them *all the time* while it drives.
  Typically Safe requires the RSS safety distance (Example 1.3) between SV and some of POVs. To be precise, the latter POVs are those which are ahead of SV in the same lane. (According to the RSS principles, the distance between SV and a POV *behind* it is SV's concern only if SV is cutting in—see $\mathsf{Goal}^{(1)}$ in (11), Section IV-B.)
- Env describes additional environmental conditions in driving—these conditions must be satisfied *all the time* during driving, too, but ensuring them is not SV's responsibility but the environment's.
  Environmental conditions typically include 1) assumptions on POVs' dynamics (e.g. they maintain their speed), and 2) other assumptions imposed in the scenario, such as "$\mathsf{POV}1$ is behind SV" ($\mathcal{T}_{111}$ in Example 4.5). See Example 3.2.
- Goal describes the goal condition of a driving scenario. It must be true *at the end of* driving.
  Devising proper responses that achieve Goal—and proving that they do so safely—is a major feature of our framework that the original (goal-unaware) RSS [1] lacks.

Example 3.2: For the pull over scenario (Example 1.5), a scenario $\mathcal{S} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal})$ is defined as follows.

The set $\mathbf{Var}$ of variables for the pull over scenario, following Section III-A, are

- l, l_1, l_2, l_3 for the lanes of SV and the three POVs;
- y, y_1, y_2, y_3 for their (longitudinal) positions;
- v, v_1, v_2, v_3 for their (longitudinal) velocities; and
- a, a_1, a_2, a_3 for their (longitudinal) acceleration rates.

The safety condition Safe is

$$\mathsf{Safe} = \textstyle\bigwedge_{i=1,2,3} \big(\mathsf{aheadSL}_i \Longrightarrow y_i - y > \mathsf{dRSS}(v_i, v)\big), \\ \wedge\ 0 \leq v \leq v_{\max} \wedge -b_{\min} \leq a \leq a_{\max}. \tag{8}$$

- The first conjunct requires that SV maintain the RSS safety distance dRSS (Example 1.3) from the preceding vehicle. We used the following abbreviation ("ahead in the same lane").

$$\mathsf{aheadSL}_i = y_i > y \wedge |l_i - l| \leq 0.5. \tag{9}$$

  Recall, e.g., that $l = 1.5$ means SV's occupancy of both Lanes 1 and 2 (Section III-A).
- The second conjunct imposes the legal maximum velocity on SV. In this paper, for simplicity, we do not impose the legal minimum speed on SV ($v_{\min} \leq v$) because of its emergency. In practice, this can be justified by turning on SV's hazard lights.
- The third conjunct bounds SV's acceleration, where we require that it brakes comfortably (within $b_{\min}$) and does not engage emergency braking (not within $b_{\max}$), much like in Example 1.3.

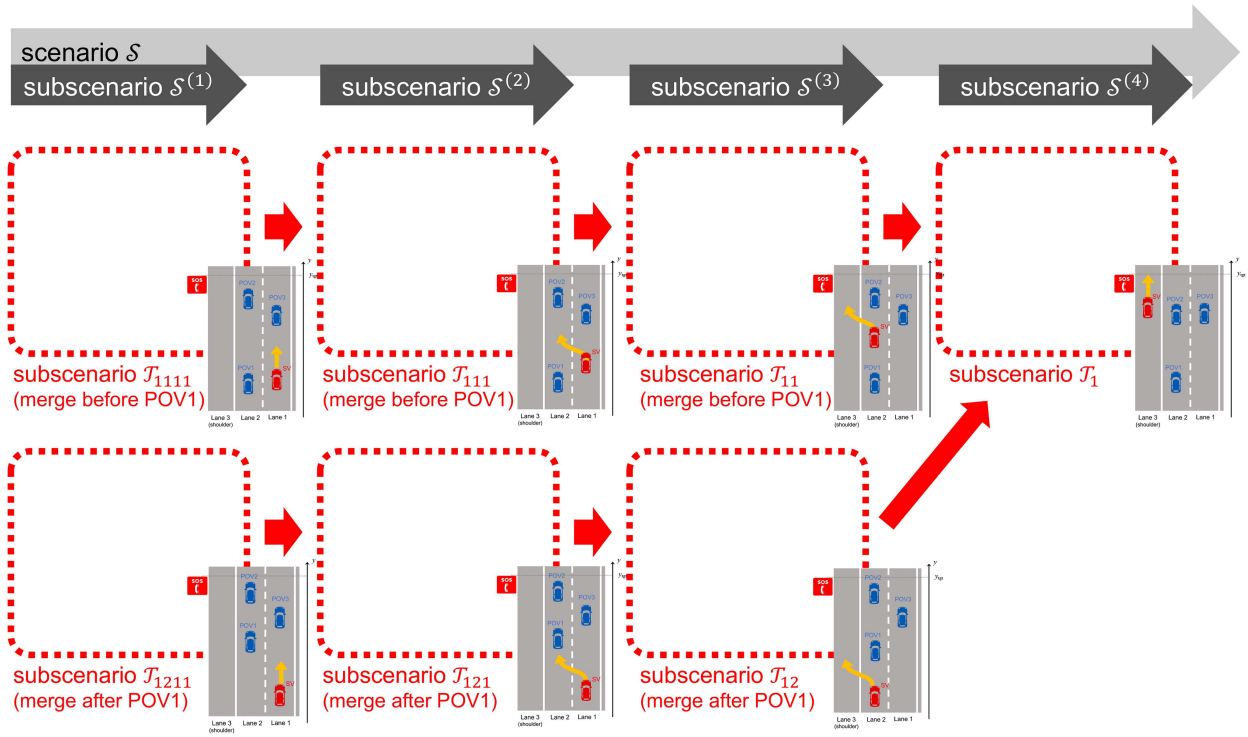


Fig. 8. The subscenario tree $\mathcal{T}$ in Example 4.5, obtained on Line 3 of Procedure 1, for the pull over scenario $\mathcal{S}$ (Example 3.2). The subscenarios $\mathcal{T}_1, \mathcal{T}_{11}, \ldots$ are defined in Fig. 12

The environmental condition Env is as follows.

$$\mathsf{Env} = \textstyle\bigwedge_{i=1,2,3}\big(v_{\min} \le v_i \le v_{\max} \wedge a_i = 0\big) \\ \wedge\, l_1 = 2 \wedge l_2 = 2 \wedge l_3 = 1 \wedge y_2 > y_1.$$

We assume that POVs do not change their speed ($a_i = 0$)—an assumption we adopt in this paper to simplify arguments. Violation of this assumption can affect goal achievement (i.e. reaching y_{tgt} in Lane 3), but it does not endanger collision avoidance. See Section IV-G3.

The goal Goal is to stop at the intended position, that is,

$$\mathsf{Goal} \quad = \quad l = 3 \wedge y = y_{\text{tgt}} \wedge v = 0.$$

Remark 3.3 (distinguishing Safe and Env): It turns out that the mathematical positions of Safe and Env are the same in our workflow in Section IV. Therefore there is no theoretical need of separating them.

We nevertheless distinguish them for their conceptual difference: Safe is an invariant that SV must maintain, while Env is an invariant that SV can assume. Separating Safe and Env also helps modelling the scenario, because treating them separately restricts the modeller's focus to specific agents.

C. Our Problem: Goal-Aware RSS Rules as dFHL Quadruples

Using Definition 3.1, we can formalise what we are after:

Definition 3.4 (goal-aware RSS rule): Let $\mathcal{S} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal})$ be a driving scenario. A *goal-aware RSS rule* (or *GA-RSS rule*) is a pair (A, α) of

- a dFHL assertion A (called an *RSS condition*), and
- a dFHL program α (called a *proper response*),

such that the quadruple

$$\{A\}\ \alpha\ \{\mathsf{Goal}\} : \mathsf{Safe} \wedge \mathsf{Env} \tag{10}$$

is valid.

Note that an RSS condition A is in the position of a precondition in the quadruple (10).

Remark 3.5: As a convention, in Definition 3.4, the dFHL program α controls only SV. The actual dynamics of the whole driving situation includes parts that model POVs' dynamics too—they are described by $\dot{y}_i = v_i, \dot{v}_i = a_i$, where a_i is typically constrained in Env.

We use this convention throughout the paper, describing only the control of SV and leaving POVs' dynamics implicit in dFHL programs. We do so e.g. in Example 4.6.

IV. Compositional Derivation of Goal-Aware RSS Rules: A General Workflow

In this section, we present a general workflow that compositionally derives a goal-aware RSS rule (A, α). In the workflow, the original scenario $\mathcal{S}$ is split up into a tree $\mathcal{T}$ of subscenarios—such as one shown in Fig. 8. Each subscenario is simplified and has a more specific scope, which allows one to come up with proper responses and their preconditions more easily. These *subscenario* proper responses and preconditions get bundled up, using dFHL rules such as (Seq) and (Case), to finally yield a goal-aware RSS rule for the original scenario.

The outline of our rule derivation workflow is Procedure 1. Some steps of the workflow are illustrated in Figs. 8–11.

Procedure 1: Our Workflow for Compositional Derivation of Goal-Aware RSS Rules.

Input: a driving scenario
Output: a goal-aware RSS rule (A, α) (Definition III.4)
1 *Scenario modelling*: mathematically model the driving scenario, obtaining $\mathcal{S} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal})$ (cf. Section III-B);
2 *Goal decomposition*: identify subgoals $\mathsf{Goal}^{(1)}, \dots, \mathsf{Goal}^{(N)}$ and decompose the scenario $\mathcal{S}$ into subscenarios $\mathcal{S}^{(1)}, \dots, \mathcal{S}^{(N)}$ where $\mathcal{S}^{(i)} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal}^{(i)})$;
3 *Subscenario refinement*: refine subscenarios, distinguishing cases and strengthening safety and environmental conditions. Then express their causal relationships and obtain a subscenario tree $\mathcal{T}$;
4 **foreach** *subscenario* $\mathcal{T}_w = (\mathbf{Var}, \mathsf{Safe}_w, \mathsf{Env}_w, \mathsf{Goal}_w)$ *in* $\mathcal{T}$ **do**
5 Identify *subscenario proper responses*: find programs $\alpha_{w,1}, \dots, \alpha_{w,K_w}$ that achieve Goal_w and maintain $\mathsf{Safe}_w \wedge \mathsf{Env}_w$ during their execution. (The programs $\alpha_{w,i}$ may include syntactic parameters $F, G, \dots$; they are instantiated by concrete expressions on Line 7);
6 **end**
7 Identify *subscenario preconditions*: find dFHL assertions $(A_{w,u})_{w,u}$ for each subscenario proper response $\alpha_{w,i}$, reasoning backwards from shorter w to longer, so that they guarantee subgoal achievement as well as the next preconditions (formalised in (15));
8 Compute *global proper response and precondition*: combine the subscenario proper responses $\alpha_{w,i}$ and the subscenario preconditions $A_{w,u}$ obtained so far, to obtain A and α such that $\{A\}\ \alpha\ \{\mathsf{Goal}\} : \mathsf{Safe} \wedge \mathsf{Env}$ is valid;
9 **return** (A, α)

Each step of the workflow is described in detail below. We use the pull over scenario (Example 1.5) as a leading example in its course. Another example scenario, which is more complex, is discussed later in Section IV-F.

A. Scenario Modelling (Line 1)

We assume that the input driving scenario is given only in informal terms. In this step, we identify its mathematical modelling $(\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal})$ in the sense of Section III-B. See Example 3.2 for a concrete example for the pull over scenario.

B. Subscenario Identification (Lines 2–3)

In the two steps on Lines 2–3, we decompose the original problem (namely, to find A and α such that $\{A\}\,\alpha\,\{\mathsf{Goal}\} : \mathsf{Safe} \wedge \mathsf{Env}$) into problems about smaller subscenarios. We aim to identify subscenarios such that 1) they make local objectives and case distinctions explicit, 2) each subscenario is simpler and more homogeneous, and 3) the safety and environmental conditions for each subscenario are more concrete and specific. These features will make it easier to devise proper responses and preconditions for those subscenarios.

The following formal definition will be justified in the course of the explanation below, notably in the proof of Theorem 4.11.

Definition 4.1 (subscenario, subgoal): Let $\mathcal{S} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal})$ and $\mathcal{S}' = (\mathbf{Var}, \mathsf{Safe}', \mathsf{Env}', \mathsf{Goal}')$ be scenarios with the same variable set. We say that $\mathcal{S}'$ is a *subscenario* of $\mathcal{S}$ if both of the logical implications $\mathsf{Safe}' \wedge \mathsf{Env}' \Rightarrow \mathsf{Safe}$ and $\mathsf{Safe}' \wedge \mathsf{Env}' \Rightarrow \mathsf{Env}$ are valid. In this case, Goal' is called a *subgoal*.

We separate the task of subscenario identification into *g*oal decomposition (Line 2) and *s*ubscenario refinement (Line 3). The separation is not a necessity from the theoretical point of view. We nevertheless explicate the separation for conceptual and practical reasons: in our experience, the two-step workflow (Lines 2–3) is the way we came up with useful subscenarios.

1) Goal Decomposition (Line 2): On Line 2, we aim at a series $\mathsf{Goal}^{(1)}, \dots, \mathsf{Goal}^{(N)}$ of subgoals that naturally paves the way to the original goal Goal. More specifically, we expect the subgoals to be such that their achievement in the given order leads to the achievement of Goal. On Line 2, note that the resulting subscenarios $\mathcal{S}^{(i)} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal}^{(i)})$ all have the same safety and environmental conditions $\mathsf{Safe}, \mathsf{Env}$ as the original scenario $\mathcal{S}$. Strengthening those conditions is left to the next step (Line 3).

Note that, in fact, any sequence $\mathsf{Goal}^{(1)}, \dots, \mathsf{Goal}^{(N)}$ of dFHL assertions qualifies as the outcome of Line 2—Definition 4.1 does not constrain Goal'. However, a good choice of subgoals $\mathsf{Goal}^{(1)}, \dots, \mathsf{Goal}^{(N)}$ eases the rest of the workflow by making local objectives explicit. It is usually easy to come up with a natural series of subgoals, too, as we demonstrate now.

Example 4.2: For the pull over scenario (Example 1.5 and 3.2), we use the goal decomposition that we informally described in Section I (Subscenario 1–4). These subscenarios arise from 1) coming to a halt (Subscenario 4), 2) changing lanes (Subscenarios 2–3), and 3) preparing for lane changes, in case there are vehicles in the destination lane (Subscenario 1).

The corresponding subgoals are formalised as follows.

$$\begin{aligned}
\mathsf{Goal}^{(1)} &= \begin{pmatrix} y_2 - y \geq \mathsf{dRSS}(v_2, v) \\ \wedge\, y - y_1 \geq \mathsf{dRSS}(v, v_1) \\ \wedge\, v_2 = v \end{pmatrix} \\
&\qquad \vee \begin{pmatrix} y_l - y \geq \mathsf{dRSS}(v_1, v) \\ \wedge\, v_{\min} = v \end{pmatrix} \\
\mathsf{Goal}^{(2)} &= (l = 2) \\
\mathsf{Goal}^{(3)} &= (l = 3) \\
\mathsf{Goal}^{(4)} &= (l = 3 \wedge y = y_{\mathrm{tgt}} \wedge v = 0)
\end{aligned} \tag{11}$$

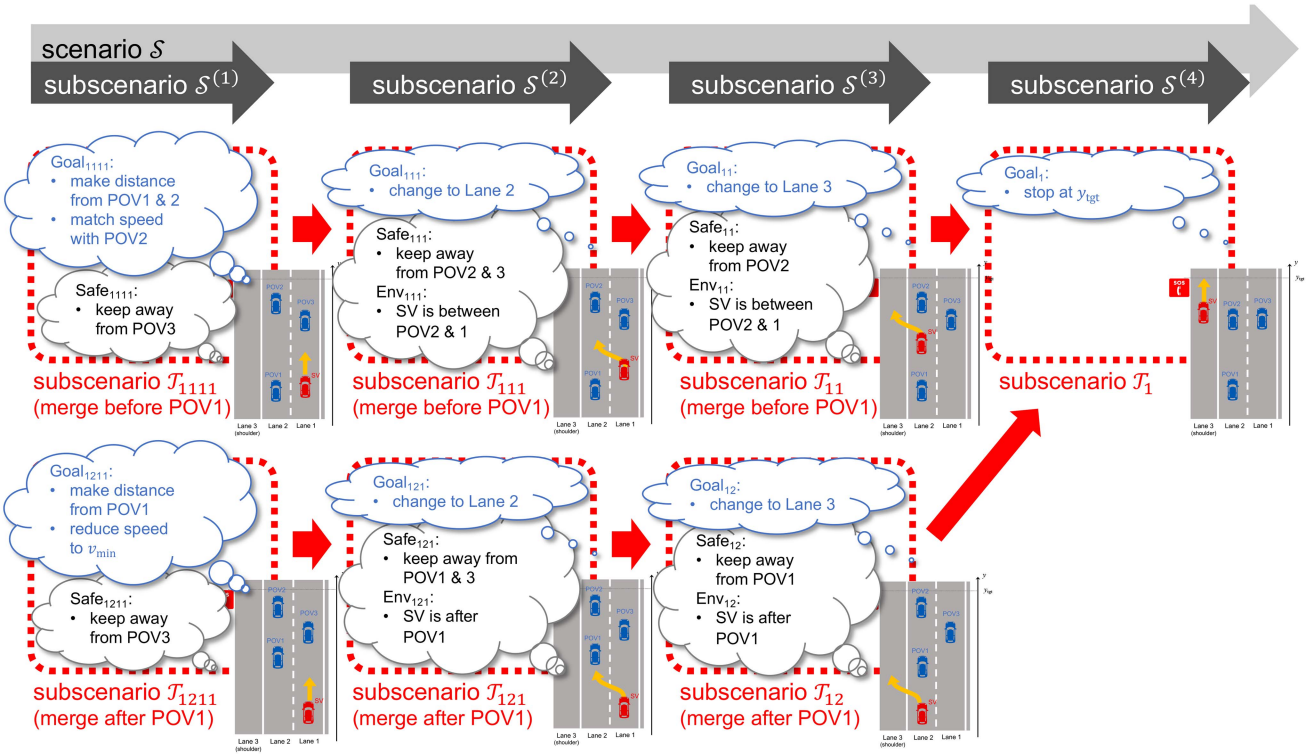


Fig. 9. The subscenario tree $\mathcal{T}$ in Example 4.5, highlighting (informally) the goal Goal_w, the safety condition Safe_w, and the environmental condition Env_w of each subscenario $\mathcal{T}_w$. The full formal definition is in Fig. 12

We define $\mathcal{S}^{(i)} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal}^{(i)})$ (for $i = 1, \dots, 4$), where $\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}$ are the ones in Example 3.2. These subscenarios $\mathcal{S}^{(i)}$ appear at the top of Fig. 8.

The two disjuncts in $\mathsf{Goal}^{(1)}$ represent 1) the case of SV merging between POV2 and POV1, and 2) that of merging behind POV1, respectively. (For simplicity, we ignore the case of merging in front of POV2.) In the former case, keeping enough distance from POV1 is deemed to be the responsibility of SV—although POV1 is behind SV, it is SV's lane-changing manoeuvre that creates the duty of distance keeping. One can also see this responsibility as an instance of the RSS responsibility principle 2) "Don't cut in recklessly"—see Section I-A.

In the first disjunct of $\mathsf{Goal}^{(1)}$, we additionally require that SV's velocity matches that of the preceding vehicle. We do so because 1) it is a natural driving practice, and 2) it eases the safety analysis of the later subscenarios (see the case for $\mathcal{T}_{11}$ in Example 4.9, for example). For the second disjunct, for similar reasons, we require that SV's velocity is the legal minimum.

2) Subscenario Refinement (Line 3): The case distinction in $\mathsf{Goal}^{(1)}$ of Example 4.2 (to merge before or after POV1) is typical in our workflow: there are different possible inter-vehicle relationships; distinguishing cases with respect to them makes each case simpler and more focused.

On Line 3, we make such case distinction explicit as different subscenarios. Relating the resulting subscenarios by their causal relationship, we obtain a *t*ree of subscenarios. See Fig. 8 for an example.

Notation 4.3: We use words $w \in (\mathbb{Z}_{>0})^*$ to designate nodes of a tree $\mathcal{T}$, as is common in the literature. Specifically, 1) the root of $\mathcal{T}$ is denoted by ε (where ε stands for the empty word), and 2) the k-th child of a node w is denoted by wk.

Definition 4.4 (subscenario tree): Let $\mathcal{S} = (\mathbf{Var}, \mathsf{Safe}, \mathsf{Env}, \mathsf{Goal})$ be a scenario. A *s*ubscenario tree $\mathcal{T}$ for $\mathcal{S}$ is a finite tree

- whose root is not labelled (we write $\bullet$ for the root label),
- whose non-root node w is labelled by a subscenario $\mathcal{T}_w$ of $\mathcal{S}$ (cf. Definition 4.1), and
- additionally, for every node of depth 1 (i.e. $\mathcal{T}_w$ with $|w| = 1$), the corresponding subscenario $\mathcal{T}_w = (\mathbf{Var}, \mathsf{Safe}_w, \mathsf{Env}_w, \mathsf{Goal}_w)$ satisfies $\mathsf{Safe}_w \wedge \mathsf{Env}_w \wedge \mathsf{Goal}_w \Rightarrow \mathsf{Goal}$, where Goal is the goal of $\mathcal{S}$.

Hence $\mathcal{T}_\varepsilon = \bullet$ for the root, and $\mathcal{T}_w$ is a subscenario for $w \neq \varepsilon$.

In the third item above, a subscenario $\mathcal{T}_w$ with $|w| = 1$ is one of those which are executed at the end (see $\mathcal{T}_1$ in Fig. 8 for an example). The item is a natural requirement that its goal Goal_w implies the goal Goal of the whole scenario $\mathcal{S}$, potentially with the help of Safe_w and Env_w.

A subscenario tree $\mathcal{T}$ arises naturally from the outcome of Line 2 $\big($namely $\mathcal{S}^{(1)}, \dots, \mathcal{S}^{(N)}\big)$ by distinguishing cases, as demonstrated below. Note that case distinction also helps concretising safety conditions.

Example 4.5: Continuing Example 4.2, we obtain the subscenario tree $\mathcal{T}$ shown in Fig. 8 as a possible outcome of Line 3. We do so by distinguishing cases of SV merging before or after POV1. The subscenarios $\mathcal{T}_w$ in $\mathcal{T}$ are defined in Fig. 12,

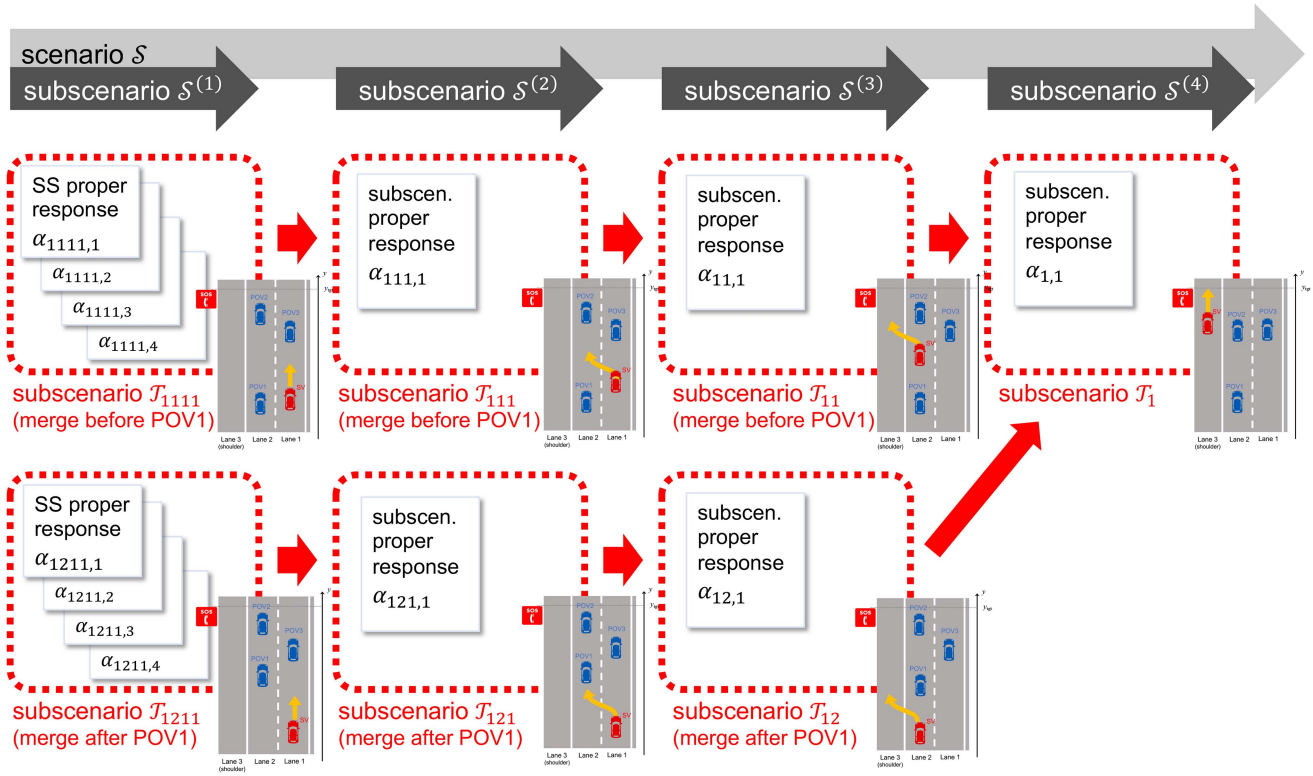


Fig. 10. The subscenario proper responses $\alpha_{1,1}, \alpha_{11,1}, \ldots$ in Example 4.6, obtained on Line 5 of Procedure 1, for the pull over scenario $\mathcal{S}$ (Example 3.2).

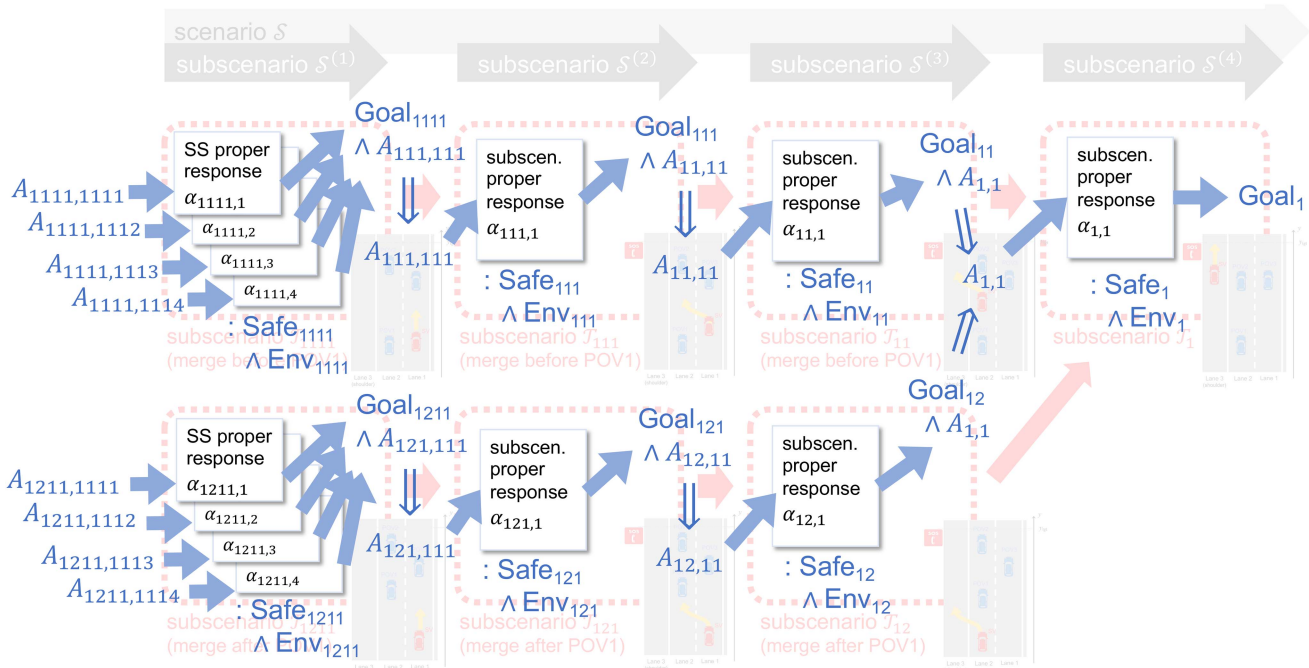


Fig. 11. The subscenario preconditions $A_{1,1}, A_{11,11}, \ldots$ in Example 4.9, obtained on Line 7 of Procedure 1, for the pull over scenario $\mathcal{S}$ (Example 3.2). Here the solid arrows represent Hoare quadruples (with a program in the middle and a safety condition below); note that they realise the condition (15). The double arrows $\Longrightarrow$ represent logical implication.

$$
\begin{aligned}
\mathcal{T}_1:\ \mathsf{Safe}_1 &= l=3 \land 0 \le v \le v_{\max} \land -b_{\min} \le a \le a_{\max}\\
\mathsf{Env}_1 &= \mathsf{Env}\\
\mathsf{Goal}_1 &= y = y_{\mathrm{tgt}} \land v = 0\\
\hline
\mathcal{T}_{11}:\ \mathsf{Safe}_{11} &= (l=2.5 \lor l=3) \land 0 \le v \le v_2 \land y_2 - y \ge \mathsf{dRSS}(v_2, v)\\
&\quad \land -b_{\min} \le a \le a_{\max}\\
\mathsf{Env}_{11} &= \mathsf{Env} \land (l=2.5 \Rightarrow \mathsf{behindSL}_1 \land \mathsf{aheadSL}_2)\\
\mathsf{Goal}_{11} &= l=3\\
\hline
\mathcal{T}_{12}:\ \mathsf{Safe}_{12} &= (l=2.5 \lor l=3) \land 0 \le v \le v_1 \land y_1 - y \ge \mathsf{dRSS}(v_1, v)\\
&\quad \land -b_{\min} \le a \le a_{\max}\\
\mathsf{Env}_{12} &= \mathsf{Env} \land (l=2.5 \Rightarrow \mathsf{aheadSL}_1)\\
\mathsf{Goal}_{12} &= l=3\\
\hline
\mathcal{T}_{111}:\ \mathsf{Safe}_{111} &= (l=1.5 \lor l=2) \land 0 \le v \le v_2\\
&\quad \land y_2 - y \ge \mathsf{dRSS}(v_2, v) \land y_3 - y \ge \mathsf{dRSS}(v_3, v)\\
&\quad \land -b_{\min} \le a \le a_{\max}\\
\mathsf{Env}_{111} &= \mathsf{Env} \land \mathsf{behindSL}_1 \land \mathsf{aheadSL}_2\\
\mathsf{Goal}_{111} &= l=2\\
\hline
\mathcal{T}_{121}:\ \mathsf{Safe}_{121} &= (l=1.5 \lor l=2) \land 0 \le v \le v_1\\
&\quad \land y_1 - y \ge \mathsf{dRSS}(v_1, v) \land y_3 - y \ge \mathsf{dRSS}(v_3, v)\\
&\quad \land -b_{\min} \le a \le a_{\max}\\
\mathsf{Env}_{121} &= \mathsf{Env} \land \mathsf{aheadSL}_1\\
\mathsf{Goal}_{121} &= l=2\\
\hline
\mathcal{T}_{1111}:\ \mathsf{Safe}_{1111} &= l=1 \land y_3 - y \ge \mathsf{dRSS}(v_3, v)\\
&\quad \land 0 \le v \le v_{\max} \land -b_{\min} \le a \le a_{\max}\\
\mathsf{Env}_{1111} &= \mathsf{Env}\\
\mathsf{Goal}_{1111} &= y_2 - y \ge \mathsf{dRSS}(v_2, v) \land y - y_1 \ge \mathsf{dRSS}(v, v_1)\\
&\quad \land v_2 = v\\
\hline
\mathcal{T}_{1211}:\ \mathsf{Safe}_{1211} &= l=1 \land y_3 - y \ge \mathsf{dRSS}(v_3, v)\\
&\quad \land 0 \le v \le v_{\max} \land -b_{\min} \le a \le a_{\max}\\
\mathsf{Env}_{1211} &= \mathsf{Env}\\
\mathsf{Goal}_{1211} &= y_1 - y \ge \mathsf{dRSS}(v_1, v) \land v_{\min} = v
\end{aligned}
$$

Fig. 12. The subscenarios $\mathcal{T}_w$ in Fig. 9 for the pull over scenario. See Example 4.5

where $\mathcal{T}_w = (\mathbf{Var}, \mathsf{Safe}_w, \mathsf{Env}_w, \mathsf{Goal}_w)$. We use the following abbreviation; it is much like $\mathsf{aheadSL}_i$ in (9).

$$\mathsf{behindSL}_i \ = \ y_i < y \land |l_i - l| \le 0.5.$$

The design of the subscenarios $\mathcal{T}_w$ is described below. Some key conditions therein are highlighted in Fig. 9.

The subscenario $\mathcal{T}_1$: This comes from $\mathcal{S}^{(4)}$ in Example 4.2. The condition $l = 3$ in the original goal $\mathsf{Goal}^{(4)}$ is moved to the safety condition Safe_1 since it has to be maintained throughout rather than achieved at the end. Requiring $l = 3$ in $\mathcal{S}^{(4)}$ automatically discharges the RSS safety distance requirement in the overall safety condition Safe (see (8)) since $\mathsf{aheadSL}_i$ is false. As a result, the subscenario safety condition Safe_1 is much simplified.

The subscenario $\mathcal{T}_{11}$: This comes from $\mathcal{S}^{(3)}$ in Example 4.2, and assumes that SV has merged between POV1 and POV2. The last assumption is found in the environmental condition Env_{11}. Consequently, the RSS distance requirement is simplified: in Safe_{11}, only the one for POV2 is required.

Note that we also assume $0 \le v \le v_2$ as part of the safety condition. This assumption may not be necessary but simplifies the subsequent reasoning a lot, especially when it comes to proving maintenance of the RSS safety distance. This assumption can be enforced, too, by requiring velocity matching in our subgoals ($v_2 = v$ and $v_{\min} = v$ in $\mathsf{Goal}^{(1)}$ in (11), and thus in $\mathsf{Goal}_{1111}, \mathsf{Goal}_{1211}$ in Fig. 12).

The subscenarios $\underline{\boldsymbol{\mathcal{T}_{12}}}, \underline{\boldsymbol{\mathcal{T}_{111}}}, \underline{\boldsymbol{\mathcal{T}_{121}}}$: Similarly to $\mathcal{T}_{11}$, we 1) explicate case distinction in the environmental conditions Env_w, and 2) simplify the safety conditions Safe_w, adding some extra assumptions (such as $v \le v_2$) if we find them useful.

The subscenarios $\underline{\boldsymbol{\mathcal{T}_{1111}}}, \underline{\boldsymbol{\mathcal{T}_{1211}}}$: These come from the two disjuncts of $\mathsf{Goal}^{(1)}$ (see (11)): their goals are precisely those disjuncts; and the safety conditions $\mathsf{Safe}_{1111}, \mathsf{Safe}_{1211}$ are the original safety condition Safe simplified using $l = 1$.

Each $\mathcal{T}_w$ *is indeed a subscenario*: It is not hard to show that each $\mathcal{T}_w$ is indeed a subscenario of $\mathcal{S}$ from Example 3.2, in the sense of Definition 4.1, as required in Definition 4.4.

- For $\mathcal{T}_1$, we have to show that $\mathsf{Safe}_1 \land \mathsf{Env}_1 \Rightarrow \mathsf{Safe}$ holds. Since $l = 3$ is in Safe_1 and $l_1 = 2, l_2 = 2, l_3 = 1$ are in Env_1, we see that $\mathsf{aheadSL}_i$ is false for each $i = 1, 2, 3$; this makes Safe in (8) trivially true.
- For $\mathcal{T}_{11}$, $\mathsf{Safe}_{11} \land \mathsf{Env}_{11} \Rightarrow \mathsf{Safe}$ can be shown as follows. Note first that $0 \le v \le v_{\max}$ is inferred from $0 \le v \le v_2$ (in Safe_{11}) and $v_2 \le v_{\max}$ (in Env_{11}).
  If $l = 3$ then Safe is trivially true, much like in the above. Otherwise $l = 2.5$ holds, which forces $\mathsf{behindSL}_1$ to hold (by Env_{11}). Therefore $\mathsf{aheadSL}_1$ is false (it contradicts with $\mathsf{behindSL}_1$), and Safe is equivalent to $y_2 - y > \mathsf{dRSS}(v_2, v)$. The last is required in Safe_{11}.
- Proofs for $\mathcal{T}_{12}, \mathcal{T}_{111}, \mathcal{T}_{121}$ are similar to the one for $\mathcal{T}_{11}$.

Definition 4.4 additionally requires $\mathsf{Safe}_1 \land \mathsf{Env}_1 \land \mathsf{Goal}_1 \Rightarrow \mathsf{Goal}$, whose validity is obvious.

C. Identifying Subscenario Proper Responses (Line 5)

On Line 5, for each subscenario $\mathcal{T}_w = (\mathbf{Var}, \mathsf{Safe}_w, \mathsf{Env}_w, \mathsf{Goal}_w)$ in the subscenario tree $\mathcal{T}$, we find dFHL programs $\alpha_{w,1}, \dots, \alpha_{w,K_w}$ such that each $\alpha_{w,i}$ achieves the goal Goal_w maintaining $\mathsf{Safe}_w \land \mathsf{Env}_w$ under a certain precondition. These programs $\alpha_{w,1}, \dots, \alpha_{w,K_w}$ are *p*roper responses for the subscenario $\mathcal{T}_w$.

There can be a number of such proper responses: collecting more of them and thus being more comprehensive is desired here, but it is not mandatory. As we will shortly see in Section IV-E, missing some proper responses may lead to a stronger precondition (i.e. a stronger RSS condition, Definition 3.4) than necessary, but the resulting precondition may still be weak enough to be useful.

The above requirement on proper responses $\alpha_{w,i}$—that they "achieve Goal_w maintaining $\mathsf{Safe}_w \land \mathsf{Env}_w$"—is made precise as follows.

Under some precondition $A_{w,i}$, the Hoare quadruple

$$\{A_{w,i}\}\ \alpha_{w,i}\ \{\mathsf{Goal}_w\} : \mathsf{Safe}_w \land \mathsf{Env}_w$$

should be valid. Moreover, it is desired that $A_{w,i}$ is weak. (12)

Note that this is not a mathematical condition—while weak $A_{w,i}$ is desired, nothing prevents to have false as $A_{w,i}$, in which case any program qualifies as a proper response $\alpha_{w,i}$.

However, finding "better" $\alpha_{w,i}$ leads to weaker (and more widely applicable) RSS conditions. See Section IV-E.

We allow the proper responses $\alpha_{w,1}, \ldots, \alpha_{w,K_w}$ to have syntactic parameters $F, G, \ldots$; they are instantiated by concrete expressions later on Line 7. The use of this flexibility is demonstrated below in Examples 4.6 and 4.9.

Example 4.6: Continuing Example 4.5, for each subscenario $\mathcal{T}_w$ (Fig. 12), we aim to find proper responses $\alpha_{w,1}, \ldots, \alpha_{w,K_w}$, whose preconditions $A_{w,i}$ are weak (cf. (12)). The outcome is illustrated in Fig. 10.

The subscenario $\mathcal{T}_1$: We have to stop at a desired position y_{tgt} while driving in a single lane. A sensible program $\alpha_{1,1}$ that achieves it is to 1) first cruise with the initial velocity until braking is needed, and 2) then engage the maximum comfortable braking (i.e. at the rate $b_{\min}$) until the vehicle comes to a halt. Formally,

$$\alpha_{1,1} = \begin{pmatrix} a := 0; \\ \mathsf{dwhile}\,(\frac{v^2}{2b_{\min}} < y_{\text{tgt}} - y)\,\{\dot{y} = v, \dot{v} = a\}\,; \\ a := -b_{\min}; \\ \mathsf{dwhile}\,(v > 0)\,\{\dot{y} = v, \dot{v} = a\} \end{pmatrix}. \quad (13)$$

The switching point is where SV's position y is $y_{\text{tgt}} - \frac{v^2}{2b_{\min}}$. We came up with this condition by high-school maths; its correctness is confirmed later on Line 7.

We can also include other programs as proper responses $\alpha_{1,i}$—such as ones that brake more gently. We do not do so in this paper, since $\alpha_{1,1}$ in the above is the most powerful when it comes to goal achievement (namely, to stop at y_{tgt}).

The subscenario $\mathcal{T}_{11}$: The goal here is to change lanes, and it can be achieved by different longitudinal manoeuvre sequences: cruise; cruise and brake; accelerate; accelerate and cruise; etc. A general approach would be to include all these manoeuvre sequences as proper responses $\alpha_{11,1}, \ldots, \alpha_{11,K_{11}}$.

Among these possible proper responses, the "cruise-brake" one is the most relevant, given that our goal later is to stop at a given position. For simplicity, we only consider this proper response:

$$\alpha_{11,1} = \begin{pmatrix} l := 2.5; t := 0; a := 0; \\ \mathsf{dwhile}(F_{11,1} > 0 \wedge t < t_{\text{LC}})\,\{\dot{t} = 1, \dot{y} = v, \dot{v} = a\}\,; \\ a := -b_{\min}; \\ \mathsf{dwhile}(t < t_{\text{LC}})\,\{\dot{t} = 1, \dot{y} = v, \dot{v} = a\}\,; \\ l := 3 \end{pmatrix}. \quad (14)$$

Here, the change of lanes is indicated by the assignments $l := 2.5$ and $l := 3$. The constant t_{LC} stands for the maximum time needed for changing lanes; we use $t_{\text{LC}} = 3$ seconds as an estimate (see e.g. [37]). Note that assuming a larger t_{LC} means 1) SV occupies two lanes longer and 2) it takes longer to reach the destination lane, and thus makes analysis more conservative.

The switching point is harder to find here than for $\mathcal{T}_1$ in the above—we therefore leave it as a syntactic parameter $F_{11,1}$. It is instantiated later on Line 7.

The subscenario $\mathcal{T}_{12}$: By the same reasoning, we define

$$\alpha_{12,1} = \begin{pmatrix} l := 2.5; t := 0; a := 0; \\ \mathsf{dwhile}(F_{12,1} > 0 \wedge t < t_{\text{LC}})\,\{\dot{t} = 1, \dot{y} = v, \dot{v} = a\}\,; \\ a := -b_{\min}; \\ \mathsf{dwhile}(t < t_{\text{LC}})\,\{\dot{t} = 1, \dot{y} = v, \dot{v} = a\}\,; \\ l := 3 \end{pmatrix}.$$

Note that $F_{12,1}$ will be instantiated with a different expression from $F_{11,1}$, since they are constrained by different POVs (namely, POV1 as the immediate preceding vehicle for the former, and POV2 for the latter).

The subscenarios $\mathcal{T}_{111}, \mathcal{T}_{121}$: By the same reasoning as above, we define proper responses $\alpha_{111,1}, \alpha_{121,1}$ to be the same as (14), using different syntactic parameters such as $F_{111,1}$.

The subscenario $\mathcal{T}_{1111}$: The goal here is to prepare for merging between POV2 and POV1, by making enough distances in front (from POV2) and behind (from POV1) and matching the velocity with the preceding POV2, while driving in Lane 1. See Fig. 12. This may be achieved by various longitudinal manoeuvre sequences. We choose the following four, which we believe constitute a quite comprehensive list.

- ($\alpha_{1111,1}$: accel-brake) Accelerate, at the rate $a_{\max}$, to make enough distance behind (from POV1). Then brake in order to match the velocity with the preceding POV2.
- ($\alpha_{1111,2}$: accel-cruise-brake) Similar to accel-brake, but in case SV's velocity reaches the legal maximum during the acceleration manoeuvre, SV cruises until it has to brake.
- ($\alpha_{1111,3}$: accel) Accelerate only (at the rate $a_{\max}$). This is used when SV is initially slower than POV2.
- ($\alpha_{1111,4}$: brake) Brake only (at the maximum comfortable rate $b_{\min}$). This is used when SV is initially faster than POV2.

The subscenario $\mathcal{T}_{1211}$: The goal Goal_{1211} here is to prepare for merging behind POV1. For ease of logical reasoning later, we require that SV's velocity should be the legal minimum at the end ($v_{\min} = v$)—we did so already in (11). This requirement may delay the goal achievement (stopping at y_{tgt} in Lane 3) by travelling slowly, but it does not reduce the possibility of the goal achievement.

The goal Goal_{1211} may be achieved by various longitudinal manoeuvre sequences, but those which involve acceleration are obviously redundant. This leaves us with the following two proper responses.

- ($\alpha_{1211,1}$: brake-cruise) Brake until v is as small as $v_{\min}$, and then cruise at $v_{\min}$ for the time needed to make enough distance in front (from POV1).
- ($\alpha_{1211,2}$: brake) Brake only. This manoeuvre is used when SV is initially sufficiently behind POV1, in which case braking until $v_{\min} = v$ already makes enough distance from POV1.

Note again that there are other possible proper responses. The above list is nevertheless comprehensive enough and thus provide a useful RSS rule with a weak RSS condition.

Remark 4.7 (basic maneuvers): The proper responses in Example 4.6 are composed of several *basic manoeuvres*, namely

- to *cruise* ($a := 0; \mathsf{dwhile}(A)\{\dot{y} = v, \dot{v} = a\}$),
- to *brake* ($a := -b_{\min}; \mathsf{dwhile}(A)\{\dot{y} = v, \dot{v} = a\}$),

- to *accelerate* ($a := a_{\max}; \mathsf{dwhile}(A)\{\dot{y} = v, \dot{v} = a\}$),
- to *initiate lane change* (such as $l := 2.5$), and
- to *complete lane change* (such as $l := 3$).

Restriction to this limited vocabulary is not mandated by our framework. Still we find it useful because 1) the logical reasoning later on Line 7 can be modularised along *basic manoeuvres* (see Section IV-D), and 2) *basic manoeuvres* are easy to implement in a baseline controller (see Section VI-B).

D. Identifying Subscenario Preconditions (Line 7)

In this step, we identify *subscenario preconditions*—preconditions for subscenario proper responses $\alpha_{w,i}$ that we identified on Line 5. A subscenario precondition must guarantee, after the execution of the proper response $\alpha_{w,i}$ in question,

- not only the achievement of the subscenario goal Goal_w,
- but also the precondition of the next proper response $\alpha_{w,'i'}$ (where $w = w'j$ with some j).

The latter requirement is inductive: a subscenario precondition for $\alpha_{j_1 j_2 j_3}$ is constrained by one for $\alpha_{j_1 j_2}$, which is further constrained by one for α_{j_1}, etc. This forces us to identify subscenario preconditions *backwards*. Such backward reasoning is common in program verification; see e.g. [31].

Because of this backward reasoning, too, we identify subscenario preconditions for each *sequence* of subscenario proper responses, instead of for each subscenario proper response. This is made precise in the following definition.

Definition 4.8 (backward condition propagation): Let $\mathcal{S}$ be a scenario, $\mathcal{T}$ be a subscenario tree for $\mathcal{S}$, and $\alpha_{w,1}, \dots, \alpha_{w,K_w}$ be proper responses for each subscenario $\mathcal{T}_w$ in $\mathcal{T}$.

On Line 7 of Procedure 1, we identify an assignment $(A_{w,u})_{w,u}$. Specifically,

- to each node $w = j_1 j_2 \dots j_k$ of $\mathcal{T}$ and each sequence $u = i_1 i_2 \dots i_k$ of proper response indices (where $i_1 \in [1, K_{j_1}], i_2 \in [1, K_{j_1 j_2}], \dots, i_k \in [1, K_{j_1 \dots j_k}]$, cf. Notation 2.1),
- we assign a dFHL assertion $A_{w,u} = A_{j_1 j_2 \dots j_k, i_1 i_2 \dots i_k}$,

so that the assignment satisfies the following condition (15).

$$\begin{array}{l}\text{For each } k \in [0, N-1], j_{k+1}, i_{k+1}, w = j_1 \dots j_k,\\ u = i_1 \dots i_k, \text{ the dFHL quadruple}\\ \{A_{wj_{k+1}, ui_{k+1}}\}\alpha_{wj_{k+1}, i_{k+1}}\{\mathsf{Goal}_{wj_{k+1}} \wedge A_{w,u}\}\\ \qquad : \mathsf{Safe}_{wj_{k+1}} \wedge \mathsf{Env}_{wj_{k+1}}\\ \text{is valid.}\end{array} \tag{15}$$

Here we set, as a convention, $A_{\varepsilon,\varepsilon} = \mathsf{true}$ for $k = 0$. Note that the condition (15) is defined inductively on k; it therefore forces us to choose $A_{w,u}$ for shorter w, u first.

Note that the definition does not uniquely determine the assignment $(A_{w,u})_{w,u}$: the assignment is only constrained by the condition (15); there are generally many ways to satisfy (15). We aim at weaker preconditions—as is usual in program verification—so that the resulting RSS rule is applicable to wider situations.

Example 4.9: We continue Example 4.6 and identify subscenario preconditions $A_{1,1}, A_{11,11}, A_{12,11}, \dots$ in a backward manner. These preconditions and their relationships—such as the one required in (15)—are illustrated in Fig. 11.

The subscenario $\mathcal{T}_1$: We aim at $A_{1,1}$ that satisfies

$$\{A_{1,1}\}\ \alpha_{1,1}\ \{y = y_{\text{tgt}} \wedge v = 0\} : l = 3 \wedge \mathsf{Env} \tag{16}$$

where Env is from Example 3.2 and $\alpha_{1,1}$ is from (13) (see also Fig. 12). It turns out that

$$A_{1,1} = \left(\mathsf{Env} \wedge l = 3 \wedge v > 0 \wedge \tfrac{v^2}{2b_{\min}} \le y_{\text{tgt}} - y\right) \tag{17}$$

satisfies (16). A dFHL proof is in Fig. 13.

The subscenario $\mathcal{T}_{11}$: We aim at $A_{11,11}$ that satisfies

$$\{A_{11,11}\}\ \alpha_{11,1}\ \{l = 3 \wedge A_{1,1}\} : \mathsf{Safe}_{11} \wedge \mathsf{Env}_{11} \tag{23}$$

where Safe_{11} and Env_{11} are from Fig. 12 and $\alpha_{11,1}$ is from (14). Note that $A_{1,1}$ that we found in (17) is now part of the postcondition.

It turns out that the definition of Safe_{11}, Env and $\alpha_{11,1}$ simplifies the reasoning a lot: for example, the safety condition $y_2 - y \ge \mathsf{dRSS}(v_2, v)$ is obviously preserved in the course of the dynamics since we require $v \le v_2$ in Safe_{11}; moreover, $v \le v_2$ is preserved since $\alpha_{11,1}$ can brake or cruise but never accelerates. Not imposing legal minimum speed on SV (Example 3.2) simplifies the reasoning too.

In the end, we arrive at the following precondition, for which we can prove (23).

$$A_{11,11} = \begin{pmatrix}\mathsf{Env} \wedge l = 2 \wedge 0 < v \le v_2 \wedge\\ y_2 - y \ge \mathsf{dRSS}(v_2, v) \wedge \tfrac{v^2}{2b_{\min}} \le y_{\text{tgt}} - y\end{pmatrix},$$

Here we instantiate $F_{11,1}$ in (14) with $y_{\text{tgt}} - y - \frac{v^2}{2b_{\min}}$.

The key inequality here is $\frac{v^2}{2b_{\min}} \le y_{\text{tgt}} - y$, much like in (17). This is not surprising: ignoring lateral movements and the leading vehicle $\mathsf{POV}2$ (we can do so by the simplification discussed above), goal achievement depends solely on whether the braking is in time.

The subscenarios $\mathcal{T}_{12}, \mathcal{T}_{111}, \mathcal{T}_{121}$: Similarly to $\mathcal{T}_{11}$, we choose preconditions $A_{12,11}, A_{111,111}, A_{121,111}$, while instantiating syntactic parameters $F_{12,11}, F_{111,111}, F_{121,111}$ in the proper responses.

The subscenario $\mathcal{T}_{1111}$: We identified four proper responses $\alpha_{1111,1}, \dots, \alpha_{1111,4}$ in Example 4.6. Here we focus on $\alpha_{1111,2}$ (accel-cruise-brake)—it is the most complicated—and identify the corresponding precondition $A_{1111,1112}$. The reasoning below subsumes those for the other three proper responses.

The proper response $\alpha_{1111,2}$ accelerates until the legal maximum speed $v_{\max}$, cruises at $v_{\max}$ in order to increase the distance behind (from $\mathsf{POV}1$), and brakes to match its velocity with $\mathsf{POV}2$. There are two switching points.

- The one from acceleration to cruising—its timing is easily determined by $v < v_{\max}$ or not.
- The one from cruising to braking—its timing is decided so that, at the end of braking (when $v = v_2$), the distance behind (from $\mathsf{POV}1$) is precisely the required RSS safety distance $\mathsf{dRSS}(v, v_1)$.

These arguments can easily be translated to symbolic conditions, which are used to instantiate symbolic parameters in $\alpha_{1111,2}$. It is also easy to symbolically express the positions and velocities of SV and POVs at the end of the proper response.

$$\left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0\le v\le v_{\max}\\ \wedge\, y_{\mathrm{tgt}}-y-\frac{v^2}{2b_{\min}}\ge 0\end{array}\right\}\ \mathsf{dwhile}\,(\tfrac{v^2}{2b_{\min}}<y_{\mathrm{tgt}}-y)\ \dot y=v,\ \dot v=0\ \left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0\le v\le v_{\max}\\ \wedge\, y_{\mathrm{tgt}}-y-\frac{v^2}{2b_{\min}}=0\end{array}\right\}:\ \begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0<v\le v_{\max}\\ \wedge\, -b_{\min}\le a\le a_{\max}\\ \wedge\, y_{\mathrm{tgt}}-y-\frac{v^2}{2b_{\min}}\ge 0\end{array}\tag{18}$$

by (DWH) with $(e_{\mathsf{inv}}\sim 0)=(v>0)$, $e_{\mathsf{var}}=y_{\mathrm{tgt}}-y-\dfrac{v^2}{2b_{\min}}$, $e_{\mathsf{ter}}=-v$

$$\left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0<v\le v_{\max}\\ \wedge\,\frac{v^2}{2b_{\min}}\le y_{\mathrm{tgt}}-y\end{array}\right\}\ \mathsf{dwhile}\,(\tfrac{v^2}{2b_{\min}}<y_{\mathrm{tgt}}-y)\ \dot y=v,\ \dot v=0\ \left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0<v\le v_{\max}\\ \wedge\,\frac{v^2}{2b_{\min}}=y_{\mathrm{tgt}}-y\end{array}\right\}:\ \begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0\le v\le v_{\max}\\ \wedge\, -b_{\min}\le a\le a_{\max}\\ \wedge\, y\le y_{\mathrm{tgt}}\end{array}\tag{19}$$

by (LIMP) and (18)

$$\left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0<v\le v_{\max}\\ \wedge\,\frac{v^2}{2b_{\min}}=y_{\mathrm{tgt}}-y\end{array}\right\}\ \mathsf{dwhile}\,(v>0)\ \dot y=v,\ \dot v=-b_{\min}\ \left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge v=0\\ \wedge\,\frac{v^2}{2b_{\min}}=y_{\mathrm{tgt}}-y\end{array}\right\}:\ \begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0<v\le v_{\max}\\ \wedge\, -b_{\min}\le a\le a_{\max}\\ \wedge\,\frac{v^2}{2b_{\min}}=y_{\mathrm{tgt}}-y\end{array}\tag{20}$$

by (DWH) with $(e_{\mathsf{inv}}\sim 0)=(y_{\mathrm{tgt}}-y-\dfrac{v^2}{2b_{\min}}=0)$, $e_{\mathsf{var}}=v$, $e_{\mathsf{ter}}=-b_{\min}$

$$\left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0<v\le v_{\max}\\ \wedge\,\frac{v^2}{2b_{\min}}=y_{\mathrm{tgt}}-y\end{array}\right\}\ \mathsf{dwhile}\,(v>0)\ \dot y=v,\ \dot v=-b_{\min}\ \left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge v=0\\ \wedge\, y=y_{\mathrm{tgt}}\end{array}\right\}:\ \begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0\le v\le v_{\max}\\ \wedge\, -b_{\min}\le a\le a_{\max}\\ \wedge\, y\le y_{\mathrm{tgt}}\end{array}\tag{21}$$

by (LIMP) and (20)

$$\left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0<v\le v_{\max}\\ \wedge\,\frac{v^2}{2b_{\min}}\le y_{\mathrm{tgt}}-y\end{array}\right\}\ \begin{array}{l}\mathsf{dwhile}\,(\tfrac{v^2}{2b_{\min}}<y_{\mathrm{tgt}}-y)\ \{\,\dot y=v,\dot v=0\,\};\\ \mathsf{dwhile}\,(v>0)\ \{\,\dot y=v,\dot v=-b_{\min}\,\}\end{array}\ \left\{\begin{array}{l}\mathsf{Env}\wedge l=3\wedge v=0\\ \wedge\, y=y_{\mathrm{tgt}}\end{array}\right\}:\ \begin{array}{l}\mathsf{Env}\wedge l=3\wedge 0\le v\le v_{\max}\\ \wedge\, -b_{\min}\le a\le a_{\max}\\ \wedge\, y\le y_{\mathrm{tgt}}\end{array}\tag{22}$$

by (SEQ), (19) and (21)

Fig. 13. A dFHL proof for $\{A_{1,1}\}\alpha_{1,1}\left\{\mathsf{Goal}_1\right\}:\mathsf{Safe}_1\wedge\mathsf{Env}_1$, Example 4.9. Here we use the obvious constant substitution, replacing $\left(a:=0;\mathsf{dwhile}(A)\{\dot y=v,\dot v=a\}\right)$ with $(\mathsf{dwhile}(A\{\dot y=v,\dot v=0\})$, for example. The dynamics of POVs is not explicit here; see Remark 3.5. The derivation of (18) and (20) in fact requires a more general form of (DWh) than we presented in Fig. 7 (namely the "multiple-invariant multiple-variant" one in Fig. 19, Appendix A); see Remark 2.13.

Now, the precondition $A_{1111,1112}$ must be such that $\{A_{1111,1112}\}\ \ \alpha_{1111,2}\ \ \{\mathsf{Goal}_{1111}\wedge A_{111,111}\}:\mathsf{Safe}_{1111}\wedge\mathsf{Env}_{1111}$ is valid. In other words, we must address the following concerns.

- The subscenario goal $\mathsf{Goal}_{1111}=\big(y_2-y\ge\mathsf{dRSS}(v_2,v)\wedge y-y_1\ge\mathsf{dRSS}(v,v_1)\wedge v_2=v\big)$ (see Fig. 12) as part of the postcondition. The latter two conjuncts are trivially satisfied by the above design of the proper response; therefore $y_2-y\ge\mathsf{dRSS}(v_2,v)$ is a core part of the postcondition. Using the analytic solution of the proper response, the last postcondition is easily translated to a precondition on the initial positions, velocities, etc.
- The precondition $A_{111,111}$ of the next subscenario $\mathcal{T}_{111}$, as part of the postcondition. Much like for $A_{11,11}$ (discussed above), the key inequality in $A_{111,111}$ is again $\frac{v^2}{2b_{\min}}\le y_{\mathrm{tgt}}-y$—this is imposed ultimately to ensure that SV does not overshoot the stopping position y_{tgt}. The requirement of this inequality as a postcondition can easily be translated to a precondition, too.
- The condition $y_3-y\ge\mathsf{dRSS}(v_3,v)$ as part of the safety condition. Again, using the analytic solution of the proper response, it is easy to calculate a precondition that guarantees this safety condition. The reasoning here is much like for the original RSS proof [1] that the RSS safety distance is enough for collision avoidance (Example 1.3).

We define $A_{1111,1112}$ as the conjunction of the three preconditions that come from the above concerns. It requires enough distances from y_{tgt} and $\mathsf{POV}2$–3, all formulated symbolically in terms of the vehicles' initial positions and velocities.

The above calculation of a precondition $A_{1111,1112}$ is theoretically straightforward—an analysis of a quadratic dynamic system with some case distinctions. It is nevertheless laborious, with logical assertions easily blowing up to dozens of lines. We use Mathematica to manage the necessary symbolic manipulations, such as solving quadratic equations, substitution, and tracking case distinctions. See Section V for further discussion.

The subscenario $\mathcal{T}_{1211}$: We identified two proper responses $\alpha_{1211,1},\alpha_{1211,2}$ in Example 4.6. Preconditions $A_{1211,1111},A_{1211,1112}$ for those can be found much like in the above (for $\mathcal{T}_{1211}$)—it is much easier since $\mathcal{T}_{1211}$ does not have requirements at odds, such as $y_2-y\ge\mathsf{dRSS}(v_2,v)\wedge y-y_1\ge\mathsf{dRSS}(v,v_1)$.

In fact, the subscenario $\mathcal{T}_{1211}$ and the subsequent ones in the subscenario tree (namely $\mathcal{T}_{121},\mathcal{T}_{12},\mathcal{T}_1$) are so simple that we can automate the whole task of identification of subscenario proper responses (Line 5) and preconditions (Line 7). This partial automation will be presented in another venue.

E. Global Proper Response and Precondition (Line 8)

The goal of Procedure 1 is to find A (an RSS condition) and α (a proper response) such that $\{A\}\ \alpha\ \{\mathsf{Goal}\}:\mathsf{Safe}\wedge\mathsf{Env}$ is valid (Section III-C). In this last step of Procedure 1,

- to obtain α, we combine the proper responses $\alpha_{w,1},\dots,\alpha_{w,K_w}$ we have identified for different subscenarios $\mathcal{T}_w$, and
- compute a collective precondition A.

We do so using the dFHL rules—especially the (SEQ) and (CASE) rules, see Fig. 7 and (5).

Definition 4.10 (global proper response and precondition): Using the subscenario proper responses $(\alpha_{w,i})_{w,i}$ and subscenario preconditions $(A_{w,u})_{w,u}$ obtained on Line 5 and 7, we define the *global proper response* α and the *global precondition*

$$\{A_{j_1\dots j_{k+1},i_1\dots i_{k+1}}\}\ \alpha_{j_1\dots j_{k+1},i_{k+1}}\ \{\mathsf{Goal}_{j_1\dots j_{k+1}} \wedge A_{j_1\dots j_k,i_1\dots i_k}\}\ :\ \mathsf{Safe}_{j_1\dots j_{k+1}} \wedge \mathsf{Env}_{j_1\dots j_{k+1}} \quad \text{for each } j_1\dots j_{k+1}, i_1\dots i_{k+1} \quad \text{(By condition (15))} \tag{25}$$

$$\{A_{j_1\dots j_{k+1},i_1\dots i_{k+1}}\}\ \alpha_{j_1\dots j_{k+1},i_{k+1}}\ \{A_{j_1\dots j_k,i_1\dots i_k}\}\ :\ \mathsf{Safe} \wedge \mathsf{Env} \quad \text{for each } j_1\dots j_{k+1}, i_1\dots i_{k+1}$$
$$\text{(By (LIMP), (25), } \mathsf{Goal}_{j_1\dots j_{k+1}} \wedge A_{j_1\dots j_k,i_1\dots i_k} \Rightarrow A_{j_1\dots j_k,i_1\dots i_k}, \tag{26}$$
$$\mathsf{Safe}_{j_1\dots j_{k+1}} \wedge \mathsf{Env}_{j_1\dots j_{k+1}} \Rightarrow \mathsf{Safe}, \text{ and } \mathsf{Safe}_{j_1\dots j_{k+1}} \wedge \mathsf{Env}_{j_1\dots j_{k+1}} \Rightarrow \mathsf{Env}, \text{ see Definitions IV.1 and IV.4)}$$

$$\{A_{j_1,i_1}\}\ \alpha_{j_1,i_1}\ \{\mathsf{Goal}_{j_1}\}\ :\ \mathsf{Safe}_{j_1} \wedge \mathsf{Env}_{j_1} \quad \text{for each } j_1, i_1 \quad \text{((25) with } k=0\text{. Recall that } A_{\varepsilon,\varepsilon} = \mathsf{true}) \tag{27}$$

$$\{A_{j_1,i_1}\}\ \alpha_{j_1,i_1}\ \{\mathsf{Goal}\}\ :\ \mathsf{Safe}_{j_1} \wedge \mathsf{Env}_{j_1} \quad \text{for each } j_1, i_1$$
$$\text{(By (LIMP), (27), and } \mathsf{Safe}_{j_1} \wedge \mathsf{Env}_{j_1} \wedge \mathsf{Goal}_{j_1} \Rightarrow \mathsf{Goal}, \text{ see Definition IV.4)} \tag{28}$$

$$\{A_{j_1,i_1}\}\ \alpha_{j_1,i_1}\ \{\mathsf{Goal}\}\ :\ \mathsf{Safe} \wedge \mathsf{Env} \quad \text{for each } j_1, i_1$$
$$\text{(By (LIMP), (28), } \mathsf{Safe}_{j_1} \wedge \mathsf{Env}_{j_1} \Rightarrow \mathsf{Safe}, \text{ and } \mathsf{Safe}_{j_1} \wedge \mathsf{Env}_{j_1} \Rightarrow \mathsf{Env}, \text{ see Definitions IV.1 and IV.4)} \tag{29}$$

$$\{A_{j_1\dots j_{k+1},i_1\dots i_{k+1}}\}\ \alpha_{j_1\dots j_k,i_k};\cdots;\alpha_{j_1,i_1}\ \{\mathsf{Goal}\}\ :\ \mathsf{Safe} \wedge \mathsf{Env} \quad \text{for each } j_1\dots j_{k+1}, i_1\dots i_{k+1}$$
$$\text{(Repeated application of (SEQ) to (26) and (29))} \tag{30}$$

$$\{\textstyle\bigvee_{w,u} A_{w,u}\}\ \ \mathsf{case}\,(A_{w,u})_{w=j_1\dots j_k,u=i_1\dots i_k}\ \ \alpha_{j_1\dots j_k,i_k};\cdots;\alpha_{j_1,i_1}\ \ \{\mathsf{Goal}\}\ :\ \mathsf{Safe} \wedge \mathsf{Env} \quad \text{(By (CASE) and (30))} \tag{31}$$

Fig. 14. Correctness proof for Procedure 1 (Theorem 4.11).

A as follows.

$$\alpha = \mathsf{case}\ (A_{w,u})_{w=j_1\dots j_k,u=i_1\dots i_k}\ \alpha_{j_1\dots j_k,i_k};\alpha_{j_1\dots j_{k-1},i_{k-1}};\dots;\alpha_{j_1,i_1}, \qquad A = \textstyle\bigvee_{w=j_1\dots j_k,u=i_1\dots i_k} A_{j_1\dots j_k,i_1\dots i_k}. \tag{24}$$

Finally, A and α are returned as the outcome of Line 7.

In (24), the case distinction and the disjunction range over all w and u considered earlier. That is,

- every word $w = j_1 j_2 \dots j_k$ that designates a node of $\mathcal{T}$ (the node $\mathcal{T}_w$ need not be a leaf), and
- all index sequences $u = i_1 \dots i_k$ compatible with w (meaning $i_1 \in [1, K_{j_1}], \dots, i_k \in [1, K_{j_1\dots j_k}]$, as above, cf. Notation 2.1).

See Example 4.9 and Fig. 11 for an example.

The following is our main theorem; it states that the above outcome indeed achieves the specified goal while maintaining safety. Our framework—including the design of dFHL—has been carefully designed so that its proof is straightforward.

Theorem 4.11 (correctness of Procedure 1): In Procedure 1, the outcome (A, α) of Line 7 (Definition 4.8) is a goal-aware RSS rule for $\mathcal{S}$ (Definition 3.4), making the dFHL quadruple $\{A\}\ \alpha\ \{\mathsf{Goal}\} : \mathsf{Safe} \wedge \mathsf{Env}$ valid.

Proof: The proof is shown in Fig. 14. It builds upon the assumption (15) on the precondition $A_{w,u}$ that we identified for each subscenario proper response $\alpha_{w,i}$. It also relies crucially on Definition 4.1 and 4.4)—we require $\mathsf{Safe}_w \wedge \mathsf{Env}_w \Rightarrow \mathsf{Safe}$ for each subscenario $\mathcal{T}_w$ in $\mathcal{T}$ on Line 3. ■

We note that, in the last proof, the subgoals Goal_w with $|w| > 1$ play no role. They are useful, however, in designing subscenarios (especially choosing Safe_w and Env_w in Section IV-B2) and identifying proper responses (Section IV-C). In other words, the subgoals Goal_w play the role of glue in our compositional workflow.

The proof does not use the (ASSIGN), (WH), and (DWH) rules. They are used for establishing the assumption (15) for each subscenario proper response $\alpha_{w,i}$. See e.g. Fig. 13.

Example 4.12: Continuing Example 4.9, for the pull over scenario in Example 3.2, we obtain a global proper response

$$\alpha = \begin{pmatrix} \text{case} & (A_{1,1}) & \alpha_{1,1} \\ & (A_{11,11}) & \alpha_{11,1};\alpha_{1,1} \\ & (A_{12,11}) & \alpha_{12,1};\alpha_{1,1} \\ & \cdots & \\ & (A_{1111,1111}) & \alpha_{1111,1};\alpha_{111,1};\alpha_{11,1};\alpha_{1,1} \\ & (A_{1111,1112}) & \alpha_{1111,2};\alpha_{111,1};\alpha_{11,1};\alpha_{1,1} \\ & \cdots & \\ & (A_{1211,1112}) & \alpha_{1211,2};\alpha_{121,1};\alpha_{12,1};\alpha_{1,1} \end{pmatrix},$$

and a global precondition $A = A_{1,1} \vee \dots \vee A_{1211,1112}$. By Theorem 4.11, it is guaranteed that $\{A\}\ \alpha\ \{\mathsf{Goal}\} : \mathsf{Safe} \wedge \mathsf{Env}$ is valid. The resulting (A, α) are rather long—logically combining dozens of symbolic inequalities. This makes them hard to read for humans, but computers have little problem checking and executing them (Section VI).

Remark 4.13: In (24), we do not require w to designate a leaf node—this is strange if we think of α to lead from the *beginning* of the driving scenario in question to its goal. We include non-leaf nodes because of the use of the resulting RSS rule in the simplex architecture (cf. Section I-E and VI-B). In the simplex architecture, a global proper response (as BC's control) can be switched on and off, depending on whether the RSS conditions are true or not at each moment. It is then beneficial if we can start a global proper response from somewhere in its middle; this is what the program $\alpha_{j_1\dots j_k,i_k};\dots;\alpha_{j_1,i_1}$ stands for, when $j_1 \dots j_k$ designates a non-leaf node. It should be noted too that the formal notion of scenario (Definition 3.1) does not state where to start—it only specifies where to reach, and what conditions to maintain.

F. Another Example Scenario: Emergency Stop With Limited Visibility

We have given a detailed account of how to apply the workflow to the pull over scenario (Example 1.5). Here, we present how

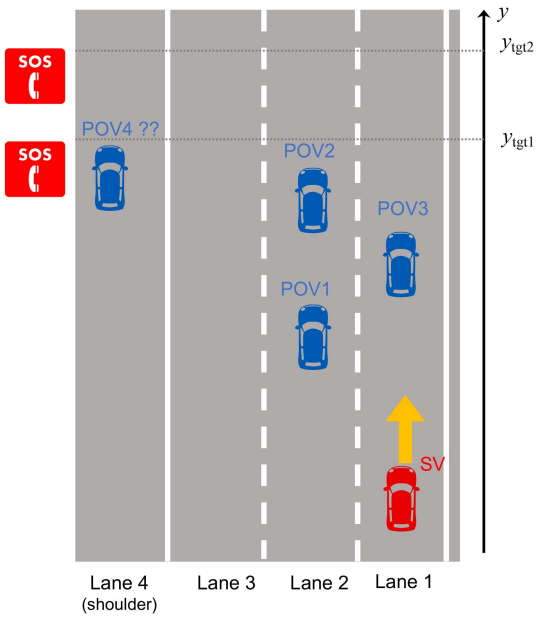


Fig. 15. The scenario in Section IV-F: emergency stop with limited visibility

the workflow applies to another scenario, in order to validate the applicability of the workflow.

The scenario is illustrated in Fig. 15. In this scenario there are 4 lanes: 3 driving lanes (Lane 1-3), and the hard shoulder (Lane 4). SV starts in Lane 1, there is a single POV in Lane 1 (ahead of SV) and two POVs in Lane 2, as in Example 1.5. SV is required to stop on the hard shoulder at one of two possible locations (y_{tgt1} or y_{tgt2}). Stopping at the first target location (y_{tgt1}) is preferred, but there may be a POV (POV4) parked there, in which case SV should stop at the second location (y_{tgt2}). Furthermore, SV only becomes aware of POV4 when it comes within sensing distance D of it, which we assume to be at least a fixed constant, say 50m.

The scenario differs from the previous one (Example 1.5) in, among others, 1) the number of lanes and 2) the dynamic character (the first location y_{tgt1} may be occupied). As we sketch below, our logical workflow is well applicable to this scenario.

To model the existence of POV4, we use a variable p whose value is 1 when POV4 exists, and 0 otherwise. In the case that $p = 1$, variable y_4 gives the position of POV4. Another variable d takes on the values 0 or 1 depending on whether SV has *d*etected POV4. The goal, safety and environment assertions are defined as follows:

$$\mathsf{Goal} = \begin{pmatrix} l = 4 \wedge v = 0 \wedge (y = y_{\text{tgt1}} \vee y = y_{\text{tgt2}}) \\ \wedge (p = 0 \Rightarrow y = y_{\text{tgt1}}) \end{pmatrix},$$

$$\mathsf{Safe} = \begin{pmatrix} \bigwedge_{i=1,2,3} (\mathsf{aheadSL}_i \Rightarrow y_i - y > \mathsf{dRSS}(v_i, v)) \\ \wedge (p = 1 \wedge \mathsf{aheadSL}_4) \Rightarrow y_4 - y > \mathsf{dRSS}(v_4, v) \\ \wedge (0 \le v \le v_{\max} \wedge -b_{\min} \le a \le a_{\max}) \end{pmatrix},$$

$$\mathsf{Env} = \begin{pmatrix} \bigwedge_{i=1,2,3} (v_{\min} \le v_i \le v_{\max} \wedge a_i = 0 \wedge l_i = i) \\ \wedge y_2 > y_1 \\ \wedge v_4 = 0 \wedge y_4 = y_{\text{tgt1}} \wedge D > 50 \end{pmatrix}.$$

Decomposition into subscenarios proceeds as in Section IV-B for Example 1.5. In total there are six subscenarios: 1) prepare to merge into Lane 2 (by adjusting speed and position); 2) merge into Lane 2; 3) merge into Lane 3; 4) prepare to merge into Lane 4, while waiting for $y_{\text{tgt1}} - y < D$; 5) merge into Lane 4; 6) stop at target location. The subscenario subgoals are similar to those in Section IV-B, except for $\mathsf{Goal}^{(4)} = (l = 3 \wedge y_{\text{tgt1}} - y = D)$. The subscenario tree branches at the level of subscenario 6 depending on which target location SV aims to stop at, and at the level of subscenario 1 depending on whether SV merges in front or behind POV1.

The subscenario proper responses were derived in much the same way as in Section IV-C for Example 1.5. Of note were those derived for subscenarios 4 and 5 which took the form:

$$\alpha_4 = \left(\mathsf{dwhile}\ (y_{tgt1} - y > D)\ \boxed{\beta_4} \right),$$

$$\alpha_5 = \left(d := p;\ \mathsf{if}\ (d = 1)\ \boxed{\beta_{5,1}}\ \mathsf{else}\ \boxed{\beta_{5,1}} \right).$$

SV waits to find out if there is a parked vehicle or not (β_4 is a hybrid program that stays in Lane 3), and then responds in one of two ways: if a POV is detected, then SV merges before y_{tgt2} ($\beta_{5,1}$), otherwise it merges before y_{tgt1} ($\beta_{5,0}$). From this, we derive preconditions and proper responses as in the pull over example. The preconditions and proper responses were successfully derived by following the workflow.

G. Discussions

We conclude with some discussions of our workflow.

1) Compositionality: We argue that our workflow (Procedure 1) is compositional.

Firstly, the design of proper responses is split up from the whole scenario to individual subscenarios, and it can be done independently for each subscenario (Line 5). We showed through our leading example that it can be done systematically, combining possible longitudinal and lateral movements, now that a goal and a safety condition are much simplified. It is also worth noting that many subscenarios are similar to each other, allowing one to reuse previous analysis.

In this paper, for simplicity, we focused on a limited number of subscenario proper responses that we see as more important than others (see Example 4.6). A viable alternative is to systematically list possible proper responses, even if some of them have limited applicability (i.e. strong preconditions). A well-developed software support and/or ample human resources would allow this brute-force approach. See also Section V.

Secondly, identification of subscenario preconditions (Line 7) is compositional, too—in the same sense as program verification in Floyd–Hoare logic is compositional. Unlike Line 5, identification of $A_{w,u}$ is not independent for different w, u—there is a backward interdependence in the form of (15). However, splitting up the task of precondition identification to simpler subscenarios certainly makes it easier, as we see in Example 4.9. There are a lot of duplicates, too, so that one can reuse previous reasoning.

2) Completeness: There are many "best-effort" elements in our workflow:

- On Line 5, the list of subscenario proper responses should better be more comprehensive, but there is no formal criterion on what is enough or what is the best.
- On Line 7, subscenario preconditions are only subject to (15) that can be satisfied even by false. It is only desired that they are weak.
- Moreover, on Line 2 and 3, there is no formal criterion what is a good subscenario decomposition. The conditions in Definitions 4.1 and 4.4 are only minimal sanity checks.

Consequently, the question "how useful is the obtained RSS rule (A, α)?," that is, "is A weak enough?," will always stand.

We argue, however, that this completeness issue should not block the use of our workflow.

- Firstly, it is not hard to come up with proper responses whose preconditions are fairly weak. This can be done by mimicking what human drivers would do, in which case the RSS-supervised ADS is at least as goal-achieving as human drivers.
- Secondly, we can always incrementally improve (A, α) by identifying more $\alpha_{w,i}$ and weaker $A_{w,u}$. Note that this process *monotonically weakens* the precondition A since it adds new disjuncts to A (see (24)). The process makes an RSS rule increasingly complete, without fallbacks.
- Thirdly, that (A, α) comes with a correctness guarantee (Theorem 4.11) means that they can be used for many years to come, as a solid basis of safe driving. The efforts for better (A, α) therefore pay off in the long run.

3) On Environmental Assumptions: In our leading example (the pull over scenario), we assumed constant speeds of the other vehicles (Env in Example 3.2). We note that

- while violation of this assumption may threaten goal achievement (namely stopping at y_{tgt} in Lane 3),
- it does *not* threaten collision avoidance,

because the scenario's safety condition requires the RSS safety distance (dRSS from Example 1.3) from every other vehicle (see (8)).

The above point has the following practical implication, in the expected use of (CA- and GA-)RSS rules in the simplex architecture (Section I-E; see also Section VI). In actual ADS, we expect another layer of the simplex architecture on top of the one based on our goal-aware RSS rules. The "collision avoiding" BC of this other simplex architecture monitors the RSS safety distance and brakes if necessary, thus ensuring collision avoidance at the possible sacrifice of goal achievement.

V. Software Support for Rule Derivation

We discuss software support for our workflow (Procedure 1). Note that, in this section, we focus on software for *deriving* goal-aware RSS rules. In contrast, software for *using* goal-aware RSS rules in the simplex architecture is heavily dependent on the choice of AC (Section I-E)—it is discussed separately in Section VI-B.

Our workflow (Procedure 1) involves two types of tasks:

- *human discovery* tasks, namely of subscenarios (Line 3), proper responses (Line 5), and preconditions (Line 7), and
- *logical reasoning* tasks in dFHL, such as ensuring the condition (15) and computing A, α (see (24)).

Much like other formal verification problems, we can imagine different ways to execute them.

- A *pen-and-paper* execution. This requires less preparation/infrastructure work, but is more error-prone.
- A *fully formalised* execution, much like in formal verification by theorem proving (see e.g. [26]).

A. Current Software Support With Mathematica

Our current execution scheme of the workflow (Procedure 1) is only partially formalised. Its software support principally uses *Mathematica notebooks* [38], an interactive environment in which users can mix

- symbol manipulations such as solving quadratic equations, substitution, and tracking case distinctions, exploiting advanced algorithms of Mathematica as a computer algebra system, and
- rich annotations for human readers, such as structured natural language descriptions, figures, and tables.

Our logical reasoning in the workflow is currently formalised as much as Mathematica can accommodate. Specifically,

- (*static reasoning is formalised*) all dFHL assertions are expressed formally in Mathematica (which is possible since each dFHL assertion is a predicate logic formula over reals), and their implications are formally checked using Mathematica functions such as `Simplify`; but
- (*dynamic reasoning is not formalised*) neither dFHL quadruples nor their derivation using the rules in Fig. 7 is formalised, since the language of Mathematica does not accommodate them.

Our current execution scheme therefore has room for improvement—formalisation of dynamic reasoning is certainly desirable. See Section V-C for its prospects.

Nevertheless, our current Mathematica-based and partially formalised execution scheme has the following distinctive advantages.

- (Well-documented informal reasoning) Dynamic reasoning for deriving dFHL quadruples is recorded in Mathematica notebooks in an informal yet trackable manner, with natural language explanations that explicate the dFHL rules used therein. Therefore these proofs can be efficiently checked by human reviewers, if not machine checkable.
- (Interaction for discovery) The interactive nature of Mathematica notebooks allows us to make trials and errors quickly for the human discovery part of our workflow (Procedure 1).
- (No static reasoning errors) A large part of mistakes in executing our workflow is in the treatment of vehicle dynamics expressed in the double integrator model. Formalisation of static reasoning in Mathematica purges these mistakes. Note that our vehicle dynamics (Remark 4.7) have closed-form solutions, which easily reduce dynamic reasoning to static one.

Given these advantages, we claim that our current execution scheme gives high confidence in the correctness of derived goal-aware RSS rules, and that the execution scheme is a viable option for practical use.

B. Estimated Workload for Rule Derivation

We estimate the workload as follows: an expert in our workflow and its software support would need several days to derive a GA-RSS rule, for a scenario of the complexity of Example 1.5. This was our experience when one of the current authors conducted the task. Getting acquainted with our workflow and its software support is not hard, either, especially for people with backgrounds in formal logic. We expect that the required training would take a couple of weeks.

Moreover, the compositional workflow and its implementation in Mathematica (featuring informal yet well-documented reasoning) allow efficient collaboration of multiple people. For example, we could parallelise the identification of subscenario proper responses $\alpha_{1111,1}$ and $\alpha_{1111,2}$—together with the identification of the preconditions $A_{1111,1111}$ and $A_{1111,1112}$, see Fig. 11—and distribute the task to different people. The same happened between the subscenarios $\mathcal{T}_{1111}$ and $\mathcal{T}_{1211}$ in Fig. 11. The communication cost between different workers was kept minimal, since they could communicate semi-formally via Mathematica notebooks.

Overall, while the workload of deriving GA-RSS rules is not very light (it is hardly a matter of minutes, for example), we claim that it is light enough to be useful, especially given that the derived GA-RSS rules can be used as a rigorous basis of safe ADS in many years to come.

C. Towards Full Formalisation

For fully formalised execution of our workflow (Procedure 1), natural tools to use are theorem provers for differential dynamics such as KeYmaera X [39]. Our logic dFHL is designed with its translation to differential dynamic logic dL [18] in mind, so the use of KeYmaera X (which is a theorem prover for dL) should not be hard.

We are currently working on systematic translation of dFHL to dL, and on some dedicated proof tactics in KeYmaera X. Our preliminary experience of manually formalising part of the reasoning for the pull over scenario (Example 4.9) is encouraging. Building a fully formalised infrastructure for the workflow will take considerable time and effort, though.

VI. Experiments and Evaluation

In Section IV we presented a general workflow to generate GA-RSS rules. The "implementation" of the workflow, specifically software support for its execution, was discussed in Section V.

In this section, we seek to quantitatively evaluate our workflow by conducting experiments on a specific *output* of the workflow. Concretely, we evaluate the GA-RSS rule set for the pull over scenario (Example 1.5)—obtained using the workflow in Section IV and finalised in Example 4.12—using it as the baseline controller (BC) in the simplex architecture (Section I-E).

The resulting RSS-supervised controller is denoted by AC+RSS^{GA}; we will compare its performance to other similar controllers (namely AC+RSS^{CA} and AC, introduced later).

The rest of the section is organised as follows. We pose several research questions in Section VI-A, based on which we designed our experiments. The implementation of AC+RSS^{GA} is introduced in Section VI-B, together with those of AC+RSS^{CA} and AC. Experiment settings and results are described in Section VI-C. Based on these results, in Section VI-D, we address the research questions that we posed earlier. Finally, in Section VI-E, we take a closer look at two notable scenario instances that further demonstrate the value of GA-RSS.

A. Research Questions

To fully evaluate the GA-RSS rule set for the pull over scenario (Example 4.12), we need to answer several research questions. Our claim is that the rule set we derived with our workflow is able to achieve a given goal (namely, pulling over) safely, which leads to our first research questions.

RQ 1: How does the GA-RSS-supervised controller perform in terms of safety?

RQ 2: How does the GA-RSS-supervised controller perform in terms of accomplishing its goal?

Another crucial point of the method, if we want it to be used in automated driving, is whether this controller can be used in practice. For example, are the RSS conditions not too complex to be computed repeatedly in a control loop at run-time? Are they not too restrictive to apply to many common driving situations?

RQ 3: Can the GA-RSS-supervised controller be useful in practice (e.g. in terms of computation speed and weakness of the RSS condition)?

While reaching the goal and maintaining safety are the two principal requirements of our controller, it is also desirable to test it for other metrics. Reaching the goal in good time and comfortably are desirable, even if these concerns are secondary to safety and goal achievement.

RQ 4: How does the GA-RSS-supervised controller perform in terms of other significant metrics (progress, comfort, etc.)?

Finally, the controller we build is based on the simplex architecture, and contains an advanced controller (AC), which may be unsafe, but is usually optimised for speed and comfort. Our simplex architecture should thus interrupt AC as rarely as possible.

RQ 5: How often is AC in control during execution?

B. Implementation of the Controllers

Our controller AC+RSS^{GA} that uses the GA-RSS rule set (Example 4.12) is based on the simplex architecture. As AC of the architecture (Section I-E), we used a prototype planner[4] based on the algorithm in [3]. AC is a sampling-based controller that, at each time step, generates a large number of candidate

[4]This is a research prototype that is provided by Mazda Motor Corporation. It is however unrelated to any of its products.

short-term paths and chooses the best in terms of a cost function. The cost function is a weighted sum of costs for multiple concerns; they are namely safety, progress, vehicle dynamics feasibility, traffic law compliance, and comfort.

In our GA-RSS-supervised controller AC+RSS^{GA}, the current implementation of BC is a "surrogate" one: instead of directly implementing the proper responses we identified in Section IV, we implemented them as an alternative cost function for the sampling-based controller used as AC. Specifically, this cost function favours the short-term path that is the closest to the desired proper response. The decision module (DM) checks all RSS conditions $A_{w,u}$ (computed on Line 7 of Procedure 1), and allows AC to remain in control as long as one of them remains valid. When the last one fails, DM switches control to BC which engages in the corresponding proper response $\alpha_{w,i}$ (identified on Line 5 of Procedure 1).

Remark 6.1 (prioritisation of GA-RSS rules): The above description of DM and BC is simplified: for enhanced progress, we additionally employ the *prioritisation* mechanism, explained below.

Specifically, some of our proper responses are designed with progress in mind (while still ensuring safety), while others reach the goal but do not make significant progress. In order to make our controller efficient, we separate rules into a high-priority and a low-priority group, and only use rules from the high-priority group whenever possible. If at any point during the execution, some high-priority rule can be engaged, then the low-priority set is discarded to only allow the use of high-priority rules.

We compare our controller AC+RSS^{GA} to two other controllers:

AC: AC alone—the sampling-based controller discussed above—without any BC.

$AC{+}RSS^{\mathrm{CA}}$: A collision-avoiding RSS-supervised controller. In this instance of the simplex architecture, BC implements the following CA-RSS rules. They are a straightforward adaptation of the classic CA-RSS rule in Example 1.3.

- SV must maintain the RSS safety distance from any POV 1) that SV shares a lane with and 2) that is in front of SV. The proper response is to brake until the RSS safety distance is restored.
- Additionally, when SV changes lanes, it must allow RSS safety distances both in front of it and behind it.[5] The proper response is to abort changing lanes if these RSS safety distances are not secured.

C. Experiment Settings and Results

We ran simulations with the three controllers above under different instances of the pull over scenario (Example 1.5). The scenario instances were generated by the following parameter values—we found them generate relevant scenario instances. Here the positions are in meters (m) and velocities are in $\mathrm{m/s}$:

$$v(0) \in \{10, 14\},\ y(0) = 0,\ v_i(0) \in \{10, 14\}\ (i = 1, 2, 3);$$

$$y_1(0) \in \{-10, -5, 0, 5, 10\};\ y_2(0) \in \{75, 80, 85, 90, 95\};$$

$$y_3(0) \in \{85, 90, 95, 100, 105\};\ y_{\mathrm{tgt}} \in \{140, 160, 180\},$$

We imposed the constraint $v_1(0) \leq v_2(0)$ to avoid collisions between POVs. For constants, we used the following values taken from our AC: $v_{\min} = 10$ m/s, $v_{\max} = 28$ m/s, $\rho = 0.3$ s, $a_{\max} = 0.98$ m/s^2, $b_{\max} = 8$ m/s^2, and $b_{\min} = 2.94$ m/s^2.

We nevertheless found that some scenario instances are clearly irrelevant (e.g. y_{tgt} is too close to stop at); we ruled out those instances as follows. While simulating, if none of the RSS conditions for our GA-RSS rule set are satisfied for the first 0.5 s, then we discarded the scenario instance. In total, we kept 2350 instances, that is 52% of the total number of instances.

The statistics of the simulation results are given in Table I. In the *goal* column, we count the number of instances that reach the goal. In the *collision* column, we count how many instances resulted in a collision. In the *RSS violation*, we count the number of instances where some RSS safety distance is violated; the average and maximal violation times. We also compute by how much the RSS safety distance was violated: it is computed as the maximal value along any execution, for any time t, and relevant POV i, of $1 - (y_i(t) - y(t))/\mathsf{dRSS}(v_i(t), v(t))$. In the *time* column, we record the average and maximal travel times. In the *jerk* column, we record the average and maximal amount of uncomfortable jerk accumulated along the trajectory. We only accumulate jerk over 0.5 m/s^3, as jerk below that threshold is considered comfortable [40]. In the *BC* time column, we quantify how long BC has been in control on average.

In Fig. 16 and Fig. 17, we give more details on the distributions of travel times and accumulated jerk. In Fig. 16, we show AC+RSS^{GA}'s total travel time compared to both AC+RSS^{CA} and AC. The size of a disc is proportional to the number of scenario instances with that travel time. A red disc indicates that AC+RSS^{CA} failed to achieve the goal in this case.

All these experiment results indicate comparative advantages and values of GA-RSS. We discuss them in detail below, along the research questions we posed in Section VI-A.

D. Discussion

Let us address the different research questions in light of the experimental results.

1) RQ 1: How does the GA-RSS-supervised controller perform in terms of safety?: All three controllers successfully avoided collisions. However, when it comes to maintaining RSS safety distances in order to prepare for sudden changes of behaviours of other cars, their performance varied a lot.

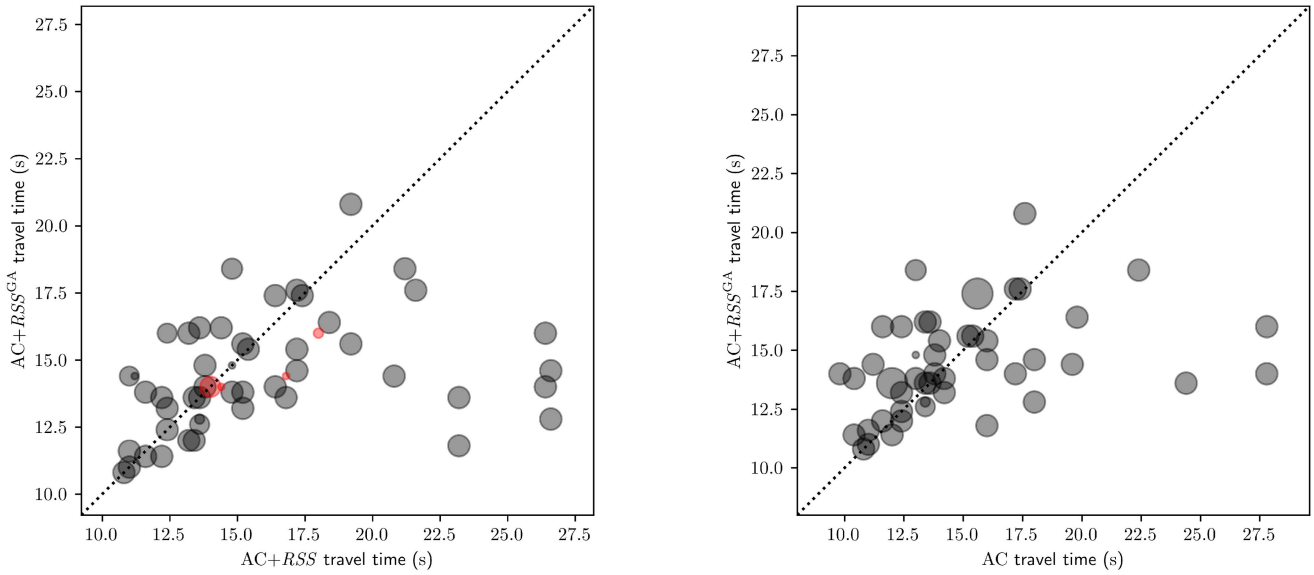
The worst performer in terms of RSS violation was AC, with a number of violations (12.8% of the scenarios) which tend to be longer (max. 0.8 s) and bigger (max. 82.16%). The last number is particularly alarming—it means the controller leaves only a fraction of the necessary safety distance. At the same time, the bad performance of AC was predicted, too—it has no safety mechanism that tries to ensure RSS safety distances.

Both AC+RSS^{CA} and AC+RSS^{GA} come equipped with BC and DM that implement RSS rules that guarantee RSS safety distances. We see, indeed, that the RSS violation was zero or nearly zero for these controllers. We attribute the rare RSS violations by AC+RSS^{GA} to the implementation details—especially

[5]This is how we formalise the RSS responsibility principle 2) "Don't cut in recklessly"—see Example 4.2.

TABLE I
EXPERIMENTAL RESULTS

	goal (%)	collision	RSS violation num. (%)	RSS violation avg. time	RSS violation max time	RSS violation max dist	time avg.	time max	jerk avg.	jerk max	BC time
AC	2350 (100%)	0	300 (12.8%)	0.09 s	0.8 s	82.16%	14.97 s	27.8 s	0.45 m/s^2	2.65 m/s^2	N/A
AC+RSS^{CA}	2285 (97.2%)	0	0 (0%)	N/A	N/A	N/A	16.23 s	26.6 s	0.36 m/s^2	0.88 m/s^2	9.8%
AC+RSS^{GA}	2350 (100%)	0	15 (0.6%)	0.00 s	0.3 s	5.16%	14.47 s	20.7 s	0.80 m/s^2	4.12 m/s^2	34.6%



(a) AC+RSS^{CA} vs AC+RSS^{GA}. (b) AC vs AC+RSS^{GA}.

Fig. 16. Comparison for progress. A red disc indicates that AC+RSS^{CA} failed to achieve the goal in this scenario instance.

to the current "surrogate implementation" of proper responses (cf. Section VI-B). In any case, the degree of RSS violation by AC+RSS^{GA} was small (max. 5.16% of the required distance), the level of error that can be easily caused by other uncertainties such as sensor inaccuracies. We do not expect serious safety concerns from these small RSS violations.

To conclude, we observed that AC+RSS^{GA} successfully ensured safety, by not only avoiding collisions but also respecting RSS safety distances (modulo minor exceptions that we attribute to our surrogate implementation of BC). This is much like AC+RSS^{CA} and unlike AC.

2) RQ 2: How does the GA-RSS-supervised controller perform in terms of accomplishing its goal?: Both AC and AC+RSS^{GA} managed to achieve the goal 100% of the time, as expected. AC+RSS^{CA} did not perform as well, and only achieved the goal in 97.3% of scenario instances. See Section VI-E1 for an example scenario instance where AC+RSS^{CA} does not achieve the goal. In situations where achieving the objective is of high importance (e.g. exiting a highway) or cannot be delayed (e.g. an automated emergency stop), this 2.7% difference is significant.

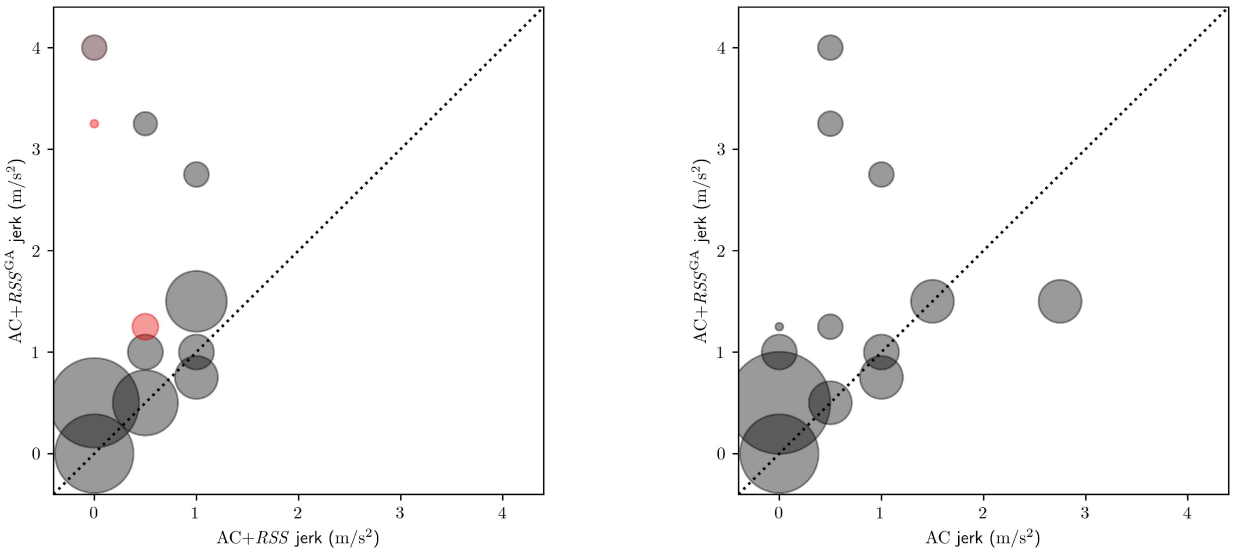
To conclude, AC+RSS^{GA} managed to achieve the goal (while maintaining safety). AC also managed to achieve the goal (but at the expense of safety), while for AC+RSS^{CA}, safety comes at the expense of goal achievement.

3) RQ 3: Can the GA-RSS-supervised controller be useful in practice?: This question can be split into two concerns.

- (The strength of RSS conditions) Are the RSS conditions weak enough, so that they are true in many driving situations? If yes, it means the GA-RSS rule set (Example 4.12) is widely applicable.
- (The computation cost) Is the computational time manageable? Is it small enough to be computable at each control loop?

The first point is hard to analyze quantitatively. Our rule set was applicable to only 52% of the total number of scenario instances—as we discussed in Section VI-C—but this does not mean that our rule set has overly restrictive RSS conditions. We expect that the remaining 48% are such that no possible control can safely achieve the goal there (e.g. y_{tgt} is too close to stop at). To show that this expectation of ours is indeed the case, we need to show the behavior of an ideal controller for each scenario instance, which is hard and is left as future work.

As a more practical consequence of the above consideration, we will pursue an automated search-based method for identifying proper responses—using e.g. evolutionary computation techniques similar to [4]—so that it either 1) shows probable adequacy of an existing current rule set (in case it does not find a new proper response) or 2) adds a new proper response to the rule set (in case it does).



(a) AC+RSS^{CA} vs AC+RSS^{GA}. (b) AC vs AC+RSS^{GA}.

Fig. 17. Comparison for comfort. A red disc indicates that AC+RSS^{CA} failed to achieve the goal in this scenario instance.

In any case, after the compositional derivation described in Section IV and our manual inspection of experiment results (which allowed us to identify the notable instances shown in Section VI-E), we are pretty confident that the rule set in Example 4.12 is as extensive and widely applicable as it can be. Let us nevertheless emphasize that adding a newly discovered proper response is easy and done modularly (cf. Fig. 11).

For the second point, recall that the GA-RSS rule set is derived in advance; therefore, the runtime task is merely to check the truth of RSS conditions (the truth of the preconditions $A_{w,u}$ to be precise, see Example 4.12). We can expect that the computational cost for doing so is light.

Indeed, the typical execution time for one of the more complicated preconditions on commodity hardware (2.9 GHz quad-core Intel Core i7) was 19.83 μs, averaged over 10^6 computations. This means that the computational cost is manageable. It will remain so, even in the future where we have hundreds or thousands of RSS conditions.

To conclude, the GA-RSS rule set in Example 4.12 can indeed be used in practice, both from the point of view of applicability and computation cost.

4) RQ 4: How does the GA-RSS-supervised controller perform in terms of other significant metrics (progress, comfort, etc.)?: We measured progress and comfort. Here, we expect AC to be the best in both these metrics, AC+RSS^{CA} to perform slightly worse (because it is more constrained by its RSS conditions), and AC+RSS^{GA} to perform poorer still (because its RSS conditions are stronger than those of AC+RSS^{CA}).

In terms of progress, AC+RSS^{CA} does not reach the goal as fast as AC on average (16.23 s and 14.97 s respectively). However, contrary to expectations, AC+RSS^{GA} performs comparably to AC on average (14.47 s), and better than AC+RSS^{CA}. To compare AC+RSS^{GA} to other controllers, let us look more closely at Fig. 16. We can see that, in most instances, the controllers behave similarly, with the distribution of scenario instances concentrated around the diagonal. There are however some outliers on the bottom right of the figures, where AC+RSS^{GA} performs better than the other controllers. See Section VI-E2 for an example of such an outlier.

Moreover, when comparing the maximal travel times of the controllers, AC+RSS^{GA} performed much better (20.7 s) than both AC and AC+RSS^{CA} (27.8 s and 26.6 s respectively). This can be explained by the existence of scenario instances in which AC+RSS^{GA} accelerates to overtake POV1 (because it knows it is safe to do so), while the other controllers do not. In emergency situations (e.g. health emergency), this can be a significant time gain.

In terms of comfort, as expected, AC+RSS^{GA} does not perform as well as AC or AC+RSS^{CA}. On average, it accumulates $0.80\,m/s^2$ of jerk, against $0.45\,m/s^2$ for AC and $0.36\,m/s^2$ for AC+RSS^{CA}. Maximal accumulated jerk follows the same pattern. In Fig. 17, we give more details about the comparison of accumulated jerk. We can see that AC+RSS^{GA} consistently performs worse than both AC and AC+RSS^{CA} (with nearly all scenario instances above the diagonal), and sometimes much worse (outliers at the top-left of Fig. 17a).

This can be explained by the fact that the GA-RSS rule set takes control more often, and the proper response can be harsh, e.g. accelerating quickly to overtake POV1.

To conclude, as expected, AC+RSS^{GA} performs poorly in terms of comfort. Surprisingly however, it performs comparably to or better than the other controllers in terms of progress.

5) RQ 5: How often is AC in control during execution?: We expect AC+RSS^{GA} to take control more often than AC+RSS^{CA}, since its RSS conditions are more strict. And indeed, the GA-RSS-supervised controller was more intrusive, taking control

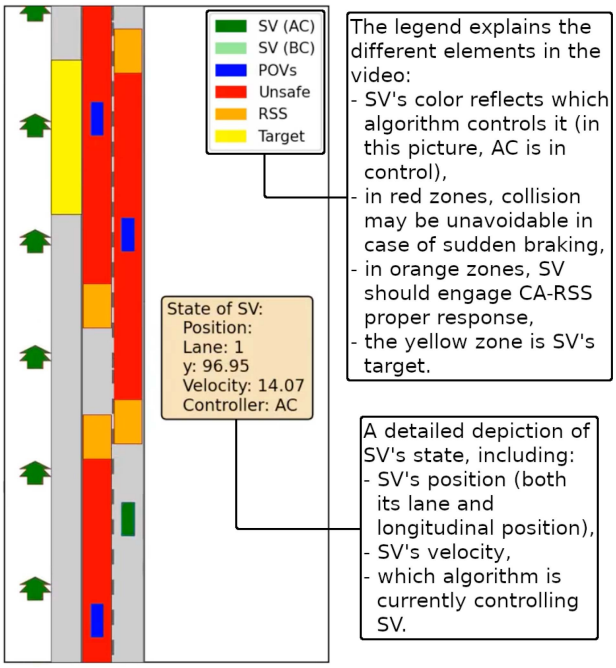


Fig. 18. A sample screenshot from a notable scenario instance

for 34.6% of the journey, on average, compared to only 9.8% for AC+RSS^{CA}.

We would say that the level of intrusiveness (34.6%) by AC+RSS^{GA} is acceptable, especially because the goal it achieves is an imminent one (such as pull over or taking a highway ramp). We can also say that the level of intrusiveness is rather low, which is enabled by the use of GA-RSS rules in the simplex architecture (Section I-E)—the control is given back to AC whenever it can.

E. Notable Scenario Instances

To better understand some of the results, we analyse in detail two notable scenario instances in which the GA-RSS-supervised controller AC+RSS^{GA} improves upon the behaviour of AC and AC+RSS^{CA}.

Video animations of these scenario instances are provided on the web; see Fig. 18 for a sample screenshot. We present this screenshot to explain the videos, and the reader should look at the videos rather than the screenshot for more information.

In these videos the zones for which dRSS $(v_f, v_r, 0)$ would be violated are coloured in red. Similarly, the orange zones indicate a violation of dRSS(v_f, v_r, ρ). (Recall that the three-argument version of dRSS is from Example 2.15.) Entering the red zone is unsafe, as collisions may no longer be avoidable. When the SV is in an orange zone, it must engage in the CA-RSS proper response within time ρ, or risk entering the red zone. SV's colour reflects which controller is presently active: dark green for AC and light green for BC.

1) Preventing Overshoot: In this scenario, POV1 starts slightly behind SV. All other cars and the target are rather close, and all cars travel rather fast. The concrete scenario instance parameters are as follows:

$$y_1(0) = -5,\ y_2(0) = 75,\ y_3(0) = 85,\ y_{\text{tgt}} = 140,$$
$$v(0) = 14,\ v_i(0) = 14.$$

The observed behaviours are as follows:

- AC: SV merges in front of POV1 when it is unsafe to do so. It manages to accomplish the goal in 9.8s, but violates the RSS safety distance by 82% with respect to POV1, for 0.8s. In the video,[6] we can see that, when changing lanes, SV crosses over POV1's red zone.
- AC+RSS^{CA}: The CA-RSS-supervised controller repeatedly interrupts AC as it is attempting to merge in front of POV1, because the distance in front of POV1 is less than the RSS safety distance. Eventually AC is forced to abandon merging into lane 2, and this results in SV failing to accomplish the goal. The RSS minimum safety distance is never violated. In the video[7] we see that AC tries to overtake POV1, but is repeatedly blocked by BC, which prevents SV from entering the red zone. BC then immediately returns control to AC to make the same action again.
- AC+RSS^{GA}: For the GA-RSS-supervised controller, none of the RSS conditions for any of the rules which merge in front of POV1 were satisfied. This resulted in a proper response for merging behind POV1 to be engaged. SV brakes so as to merge behind POV1 and successfully stops at the target in 14s without ever violating the RSS safety distance. In the video,[8] at first our AC+RSS^{GA} exhibits the same behaviour as AC+RSS^{CA}. However, when SV has no choice but to brake in order to safely reach the goal, then BC takes control to slow down and merge behind POV1.

In this scenario, AC accomplishes the goal at the expense of safety: an RSS safety distance violation of 82% would surely lead to POV1 engaging in dangerous evasive actions, which may in turn lead to a loss of control and possibly a collision.

Both AC+RSS^{CA} and AC+RSS^{GA} prevent SV from violating the RSS safety distances, but in different ways. AC+RSS^{CA} corrects AC's behaviour but only takes safety into account, which leads to failing the goal. AC+RSS^{GA} also corrects AC's behaviour by taking the goal into account as well, and therefore manages to accomplish the goal while maintaining safety.

The accumulated uncomfortable jerk was 1.17 m/s^2, more than double that of AC+RSS^{CA} (0.47 m/s^2) or AC (0.50 m/s^2).

In conclusion, in this scenario instance, AC+RSS^{GA} was the only controller to safely achieve the goal: AC achieved the goal, but violated the RSS safety distance when cutting in front of POV1, and AC+RSS^{CA} overshot the goal.

2) Bold but Safe: In this scenario, POV1 is in front of SV, but SV is faster. At first glance, this does not seem like a situation where merging in front of POV1 can be done safely.

[6]https://bit.ly/3r3IvRW
[7]https://bit.ly/3FqRYIp
[8]https://bit.ly/3qdCcMl

$$\begin{array}{rll}
\mathsf{inv}_1\colon & A \Rightarrow e_{\mathsf{inv},1} \sim_1 0 & \bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} \geq 0 \wedge \bigwedge_{i=1}^{m} e_{\mathsf{inv},\mathsf{i}} \sim_i 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{inv},1} \simeq_1 0 \\
\cdots & & \\
\mathsf{inv}_m\colon & A \Rightarrow e_{\mathsf{inv},\mathsf{m}} \sim_m 0 & \bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} \geq 0 \wedge \bigwedge_{i=1}^{m} e_{\mathsf{inv},\mathsf{i}} \sim_i 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{inv},\mathsf{m}} \simeq_m 0 \\
\mathsf{var}_1\colon & A \Rightarrow e_{\mathsf{var},1} \geq 0 & \bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} \geq 0 \wedge \bigwedge_{i=1}^{m} e_{\mathsf{inv},\mathsf{i}} \sim_i 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{var},1} \leq e_{\mathsf{ter},1} \\
\mathsf{ter}_1\colon & A \Rightarrow e_{\mathsf{ter},1} < 0 & \bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} \geq 0 \wedge \bigwedge_{i=1}^{m} e_{\mathsf{inv},\mathsf{i}} \sim_i 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{ter},1} \leq 0 \\
\cdots & & \\
\mathsf{var}_n\colon & A \Rightarrow e_{\mathsf{var},\mathsf{n}} \geq 0 & \bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} \geq 0 \wedge \bigwedge_{i=1}^{m} e_{\mathsf{inv},\mathsf{i}} \sim_i 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{var},\mathsf{n}} \leq e_{\mathsf{ter},\mathsf{n}} \\
\mathsf{ter}_n\colon & A \Rightarrow e_{\mathsf{ter},\mathsf{n}} < 0 & \bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} \geq 0 \wedge \bigwedge_{i=1}^{m} e_{\mathsf{inv},\mathsf{i}} \sim_i 0 \Rightarrow \mathcal{L}_{\dot{\mathbf{x}}=\mathbf{f}}\, e_{\mathsf{ter},\mathsf{n}} \leq 0
\end{array}$$

$$\{A\}\ \mathsf{dwhile}\,(\textstyle\bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} > 0)\,\dot{\mathbf{x}} = \mathbf{f}\ \left\{\bigvee_{i=1}^{n}(e_{\mathsf{var},\mathsf{i}} = 0 \wedge \bigwedge_{j\neq i} e_{\mathsf{var},\mathsf{j}} \geq 0)\right\} :\ \textstyle\bigwedge_{i=1}^{m} e_{\mathsf{inv},\mathsf{j}} \sim_j 0 \wedge \bigwedge_{i=1}^{n} e_{\mathsf{var},\mathsf{i}} \geq 0 \qquad (\mathrm{DW_H})$$

Fig. 19. A more general dwhile rule. It accommodates any number of invariants $e_{\mathsf{inv},1}$, ..., $e_{\mathsf{inv},\mathsf{m}}$ and variants $e_{\mathsf{var},1}$, ..., $e_{\mathsf{var},\mathsf{n}}$.

The concrete scenario instance parameters are as follows:

$$y_1(0) = 10,\ y_2(0) = 75,\ y_3(0) = 90,\ y_{\mathrm{tgt}} = 160,$$
$$v(0) = 14,\ v_i(0) = 10.$$

The observed behaviours are as follows:

- AC: As can be seen in the video,[9] SV merged behind POV1, and successfully stopped at the target area, taking a total time of 15.9s. The RSS safety distance is violated for 0.6s, by a maximum of 25.6%.
- AC+RSS^{CA}: The behaviour of AC+RSS^{CA} similar to that of AC, taking a total time of 23.3 s (see the video),[10] however the RSS safety distance is not violated.
- AC+RSS^{GA}: SV accelerated so as to overtake POV1 (knowing it is safe to do so), merging in front of it, and then stopped in the target area. The total time taken was 11.8 s. The RSS safety distance was respected. As can be seen in the video,[11] at first SV's path to merging in front of POV1 seems totally blocked by overlapping red zones. However, by accelerating (to put itself in a favourable position between POV1 and POV2) then braking (thus reducing the sizes of the red zones), it manages to open a window through which it can merge in front of POV1.

In this case we observe that AC+RSS^{GA} is able to engage in bold behaviour, overtaking POV1, in a situation where AC and AC+RSS^{CA} simply merge behind POV1. AC+RSS^{GA} is able to engage in such bold behaviour to improve progress because a mathematical proof exists that it will be able to respect safety distances while also achieving the goal.

The discomfort level was roughly the same for all controllers: 0.88 m/s^2 for AC, 0.82 m/s^2 for AC+RSS^{CA}, and 0.88 m/s^2 for AC+RSS^{GA}.

To conclude, AC+RSS^{GA} performed better than the other controllers in this scenario instance. Indeed, all three controllers managed to reach the goal, but AC+RSS^{GA} performed better than both AC and AC+RSS^{CA} in terms of progress.

[9]https://bit.ly/33qJy6w
[10]https://bit.ly/3zKCuNR
[11]https://bit.ly/31Ib7Ye

VII. Conclusion

In this paper, we proposed a *goal-aware* extension of responsibility-sensitive safety (RSS), so that RSS rules ensure not only collision-avoidance but also achievement of goals such as pulling over at a desired position.

Derivation of goal-aware RSS rules involves complex planning that ranges over multiple manoeuvres. Our approach is to deal with such complex reasoning with *program logic*, specifically a program logic dFHL that we introduce as an extension of classic Floyd–Hoare logic.

We presented a dFHL-based compositional workflow for deriving goal-aware RSS rules, in which one can systematically 1) split a driving scenario into smaller subscenarios, 2) design proper responses for those subscenarios, and 3) compute the preconditions for those proper responses. Our current software support by Mathematica is only partially formal, yet provides enough automation and traceability to be practical. We are also working on a fully formalized implementation.

We conducted experiments in which RSS rules were used in the simplex architecture. Our comprehensive experiments showed the value of goal-aware RSS rules in 1) statistics (they can realize both goal achievement and collision-avoidance) and 2) notable scenarios (they can realize unexpected bold behaviours whose safety is nevertheless guaranteed).

Appendix A

A1 A Formal Proof of the One-Way Traffic Scenario

We want to prove the Hoare triple

$$\{A\}\ \alpha\ \{B\} : S$$

as defined in (4) is valid. Remember that the different dFHL assertions are defined as

$$A = (v_r \geq 0 \wedge v_f \geq 0 \wedge y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho))\,,$$
$$B = (v_r = 0 \wedge v_f = 0)\,,$$
$$S = (y_r < y_f)\,,$$

and α is defined in Fig. 6.

In order to do this, we need to be able to define $\mathsf{dRSS}(v_f, v_r, \rho)$ as a term of our syntax, so we make the following addition to our setting (see Remark 2.4): if e, e' are

$$\dfrac{\dfrac{\text{(Appendix A1)}}{\{A\}\ \alpha_1\ \{C\} : S} \quad \dfrac{\text{(Appendix A2)}}{\{C\}\ \alpha_2\ \{D\} : S_{\mathsf{inv}}}\ (\textsc{Limp}) \quad \dfrac{\dfrac{\text{(Appendix A3)}}{\{D_\top\}\ \alpha_3\ \{B\} : S_{\mathsf{inv}}} \quad \dfrac{\dfrac{\text{(Appendix A4)}}{\{D_\bot\}\ \alpha_{4,7}\ \{B\} : S_{\mathsf{inv}}}\ (\textsc{Limp})}{\{D \wedge v_f \neq 0\}\ \alpha_{4,7}\ \{B\} : S_{\mathsf{inv}}}\ (\textsc{If})}{\{D\}\ \alpha_{3,7}\ \{B\} : S_{\mathsf{inv}}}\ (\textsc{Limp})}{\{A\}\ \alpha\ \{B\} : S \text{ as in (4)}}\ (\textsc{Seq}_2)$$

Fig. 20. Overall proof structure. Here, α denotes the program in Fig. 6, α_i's are the program fragments on Line i of Fig. 6, and $\alpha_{i,j}$'s are the program fragments on Lines i–j of Fig. 6. ($\textsc{Seq}_2$) denotes two applications of the ($\textsc{Seq}$) rule. The rest of the proof is detailed in the corresponding sections.

terms, then $\max(e, e')$ is a term, and its partial derivatives are

$$\frac{\partial}{\partial x}(\max(e, e')) = \begin{cases} \frac{\partial e}{\partial x} \text{ if } e \geq e' \\ \frac{\partial e'}{\partial x} \text{ otherwise,} \end{cases}$$

which is also a term. Note that, if $\mathbf{f}$ and $\mathbf{g}$ are locally Lipschitz continuous, then so is $\dot{\mathbf{x}} = \max(\mathbf{f}, \mathbf{g})$ (where $\max$ on a list is defined as the pointwise $\max$).

Remark A.1: Since $\mathsf{dRSS}(v_f, v_r, \rho)$ is equal to $\max(0, \mathsf{dRSS}_{\pm}(v_f, v_r, \rho))$ (see Example 2.15), one may think of replacing $y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho)$ with the equivalent formula

$$y_f - y_r > 0 \wedge y_f - y_r > \mathsf{dRSS}_{\pm}(v_f, v_r, \rho),$$

as this would spare us the need to add $\max$ to our setting. However, these assertions are not preserved by the dynamics, so we do need to introduce $\max$ into our setting for the proof to go through.

Because, in this scenario, we do not know in which order different events happen (e.g. which car stops first), we will need a ($\textsc{DWh}$) rule that uses several variants. We thus need a more general ($\textsc{DWh}$) rule than the one in Fig. 7, and use the one in Fig. 19 (see Remark 2.13).

The overall proof structure is illustrated in Fig. 20, and is simply repeated application of the ($\textsc{Seq}$) rule (as dictated by the structure of α), composed with ($\textsc{Limp}$) rules to strengthen S to

$$S_{\mathsf{inv}} = (y_f - y_r > \mathsf{dRSS}\,(v_f, v_r, \rho - t))\,.$$

The assertions C, D, $D_\top$, and $D_\bot$ will be defined later in the corresponding sections.

1) Step 1: Line 1 of Fig. 6: We define C as follows:

$$C = \begin{pmatrix} v_r \geq 0 \wedge v_f \geq 0 \wedge t = 0 \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r) \end{pmatrix}.$$

Since α_1 is $t := 0$, we can show the validity of $\{A\}t := 0\{C\} : A \vee C$ directly by ($\textsc{Assign}$). We can then show the validity desired Hoare quadruple by ($\textsc{Limp}$), since $A \vee C$ (which is equivalent to A) implies S.

2) Step 2: Line 2 of Fig. 6: This part of the proof is detailed in Example 2.15. We define D as

$$D = \begin{pmatrix} ((v_f \geq 0 \wedge t = \rho) \vee (v_f = 0 \wedge t \leq \rho)) \wedge \\ v_r \geq 0 \wedge y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

(Note that, in Example 2.15, α_2 is called α', C is called A', and D is called B'.)

The only point that was left implicit in Example 2.15 was the computation of the Lie derivative of $e_{\mathsf{inv},2}$. We compute the Lie derivative as follows (remember that δ's are snippets from (3)):

$$\begin{aligned} \mathcal{L}_{\delta_f, \delta_r^1} e_{\mathsf{inv},2} &= v_f \frac{\partial e_{\mathsf{inv},2}}{\partial y_f} - b_{\max} \frac{\partial e_{\mathsf{inv},2}}{\partial v_f} + v_r \frac{\partial e_{\mathsf{inv},2}}{\partial y_r} \\ &\quad + a_{\max} \frac{\partial e_{\mathsf{inv},2}}{\partial v_r} \quad + 1 \frac{\partial e_{\mathsf{inv},2}}{\partial t} \\ &= v_f - b_{\max} \begin{cases} \frac{v_f}{b_{\max}} \\ 0 \end{cases} - v_r \\ &\quad + a_{\max} \begin{cases} -(\rho - t) - \frac{v_r + a_{\max}(\rho - t)}{b_{\min}} \\ 0 \end{cases} \\ &\quad + 1 \begin{cases} v_r + a_{\max}(\rho - t) + \frac{a_{\max}(v_r + a_{\max}(\rho - t))}{b_{\min}} \\ 0 \end{cases} \\ &= \begin{cases} 0 & \text{if } \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) \geq 0 \\ v_f - v_r \text{ otherwise.} \end{cases} \end{aligned}$$

With Example 2.15, this concludes this part of the proof.

3) Step 3: Line 3 of Fig. 6: We define $D_\top$ as

$$D_\top = \begin{pmatrix} v_f = 0 \wedge t \leq \rho \wedge v_r \geq 0 \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

Note that $D_\top$ is equivalent to $D \wedge v_f = 0$, which is needed for the (IF) rule in Fig. 20 to be applicable. We denote by α_3' and α_3'' the two sub-programs of α_3, as in

$$\alpha_3 = (\alpha_3'; \alpha_3'').$$

To prove the desired Hoare quadruple, we thus use the ($\textsc{Seq}$) and ($\textsc{Limp}$) rules as follows:

$$\dfrac{\vdots}{\{D_\top\}\alpha_3'\{E\} : S_{\mathsf{inv}}} \quad \dfrac{\vdots}{\{E\}\alpha_3''\{B'\} : S'}\ (\textsc{Limp})}{\{D_\top\}\alpha_3\{B\} : S_{\mathsf{inv}}}\ (\textsc{Seq})$$

Here, B' and S' are assertions that will be defined later, and E is defined as

$$E = \begin{pmatrix} v_f = 0 \wedge t = \rho \wedge v_r \geq 0 \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

Both branches of the proof above can be proved by an application of ($\textsc{DWh}$) followed by ($\textsc{Limp}$).

For α_3', we apply ($\textsc{DWh}$) with the following invariants, variants, and terminators:

- $e_{\mathsf{inv},1} = (v_r \geq 0)$,
- $e_{\mathsf{inv},2} = (v_f = 0)$,
- $e_{\mathsf{inv},3} = (y_f - y_r - \mathsf{dRSS}(v_f, v_r, \rho - t) > 0)$,

- $e_{\mathsf{var},1} = \rho - t, \quad e_{\mathsf{ter},1} = -1.$

Again, the only non-obvious point is the $e_{\mathsf{inv},3}$ is preserved by the dynamics, for which we compute

$$\mathcal{L}_{\delta_r^1} e_{\mathsf{inv},3} = \begin{cases} 0 & \text{if } \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) \geq 0 \\ -v_r & \text{otherwise.} \end{cases}$$

We can show that this quantity is always non-negative:

$$\begin{aligned} & \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) < 0 \\ \iff & v_r(\rho - t) + \frac{a_{\max}(\rho - t)^2}{2} \\ & \quad + \frac{(v_r + a_{\max}(\rho - t))^2}{2b_{\min}} - \frac{v_f^2}{2b_{\max}} < 0 \\ \implies & v_r(\rho - t) + \frac{a_{\max}(\rho - t)^2}{2} \\ & \quad + \frac{(v_r + a_{\max}(\rho - t))^2}{2b_{\min}} < 0 \qquad (i) \\ \implies & v_r(\rho - t) + \frac{(v_r + a_{\max}(\rho - t))^2}{2b_{\min}} < 0 \qquad (ii) \\ \implies & v_r(\rho - t) + \frac{v_r^2}{2b_{\min}} < 0 \qquad (iii) \\ \implies & v_r < 0 \qquad (iv) \end{aligned}$$

Here, (i) holds because $v_f = 0$ (by $e_{\mathsf{inv},2}$) and $b_{\max} 0$; (ii) because $t \leq \rho$ (by $e_{\mathsf{var},1}$) and $a_{\max} > 0$; (iii) because $v_r \geq 0$, $t \leq \rho$ (by $e_{\mathsf{inv},1}$ and $e_{\mathsf{var},1}$), $a_{\max} > 0$, and $b_{\min} > 0$; and (iv) because $v_r \geq 0$ (by $e_{\mathsf{inv},1}$) and $b_{\min} > 0$.

This proves the validity of

$$\{D_\top\}\, \alpha_3'\, \{E\} : \begin{pmatrix} v_r \geq 0 \wedge v_f = 0 \wedge t \leq \rho \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

The proof of validity of $\{D_\top\}\alpha_3'\{E\} : S_{\mathsf{inv}}$ follows by (LIMP), since the safety condition above implies S_{inv}.

For α_3'', we apply (DWH) with the following invariants, variants, and terminators:

- $e_{\mathsf{inv},1} = (v_f = 0),$
- $e_{\mathsf{inv},2} = (\rho - t = 0),$
- $e_{\mathsf{inv},3} = (y_f - y_r - \mathsf{dRSS}(v_f, v_r, \rho - t) > 0),$
- $e_{\mathsf{var},1} = v_r, \quad e_{\mathsf{ter},1} = -b_{\min}.$

We can compute the Lie derivative for $e_{\mathsf{inv},3}$:

$$\mathcal{L}_{\delta_r^2} e_{\mathsf{inv},3} = \begin{cases} (a_{\max} + b_{\min})\rho & \text{if } \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) \geq 0 \\ -v_r & \text{otherwise.} \end{cases}$$

The first term above is positive because $a_{\max}$, $b_{\min}$, and ρ all are. The proof that the second term is non-negative follows the same pattern as for α_3'.

This proves the validity of

$$\{E\}\, \alpha_3''\, \{B'\} : S',$$

where

$$B' = \begin{pmatrix} v_r = 0 \wedge v_f = 0 \wedge t = \rho \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix},$$

$$S' = \begin{pmatrix} v_r \geq 0 \wedge v_f = 0 \wedge t = \rho \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}. \qquad (32)$$

The proof of validity of $\{E\}\, \alpha_3''\, \{B\} : S_{\mathsf{inv}}$ follows by (LIMP), since B' implies B and S' implies S_{inv}. This concludes the proof of validity of $\{D_\top\}\, \alpha_3\, \{B\} : S_{\mathsf{inv}}$.

4) Step 4: Lines 4–7 of Fig. 6: We define $D_\perp$ as

$$D_\perp = \begin{pmatrix} v_f \geq 0 \wedge t = \rho \wedge v_r \geq 0 \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

Note that $D \wedge v_f \neq 0$ does imply $D_\perp$, which is necessary to apply the (LIMP) rule.

We can decompose $\alpha_{4,7}$ as

$$\alpha_{4,7} = (\alpha_5;\ \mathsf{if}\ (v_f = 0)\alpha_6\ \mathsf{else}\ \alpha_7),$$

where the α_i's correspond to the program fragments on Line i of Fig. 6.

To prove the desired Hoare quadruple, we use (SEQ) as follows (where $\alpha_{6,7}$ denotes the program fragment on Lines 6–7 of Fig. 6):

$$\frac{\begin{matrix}\vdots \\ \{D_b\}\alpha_5\{F\}:S_{\mathrm{inv}}\end{matrix} \quad \begin{matrix}\vdots \\ \{F\}\alpha_{6,7}\{B\}:S_{\mathrm{inv}}\end{matrix}}{\{D_b\}\alpha_{4,7}\{B\} : S_{\mathrm{inv}}}(\mathrm{Seq})$$

Here, F is defined as

$$F = \begin{pmatrix} ((v_f = 0 \wedge v_r \geq 0) \vee (v_f \geq 0 \wedge v_r = 0)) \wedge \\ t = \rho \wedge y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

For α_5, we use the (DWH) rule with the following invariants, variants, and terminators:

- $e_{\mathsf{inv},1} = (\rho - t = 0),$
- $e_{\mathsf{inv},2} = (y_f - y_r - \mathsf{dRSS}(v_f, v_r, \rho - t) > 0),$
- $e_{\mathsf{var},1} = v_f, \quad e_{\mathsf{ter},1} = b_{\max},$
- $e_{\mathsf{var},2} = v_r, \quad e_{\mathsf{ter},2} = b_{\min}.$

Once again, we compute the Lie derivative for $e_{\mathsf{inv},2}$:

$$\mathcal{L}_{\delta_f, \delta_r^2} e_{\mathsf{inv},3} = \begin{cases} (a_{\max} + b_{\min})\rho & \text{if } \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) \geq 0 \\ v_f - v_r & \text{otherwise.} \end{cases}$$

The top term above is positive because $a_{\max}$, $b_{\min}$, and ρ are, and the proof that the bottom one is non-negative is the same as in Example 2.15. This proves the validity of

$$\{D_\perp\}\, \alpha_5\, \{F\} : \begin{pmatrix} v_r \geq 0 \wedge v_f \geq 0 \wedge t = \rho \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}.$$

The validity of $\{D_\perp\}\alpha_5\{F\} : S_{\mathsf{inv}}$ follows directly by (LIMP), since the safety condition above implies S_{inv}.

Since $\alpha_{6,7}$ is an if construct, the proof structure is as follows:

$$\frac{\begin{matrix}\vdots \\ \{F_\top\}\ \alpha_6\ \{B\}:S_{\mathrm{inv}}\end{matrix} \quad \begin{matrix}\vdots \\ \frac{\{F_1\}\ \alpha_7\ \{B'\}:S'}{\{F\wedge v_f\neq 0\}\ \alpha_7\ \{B\}:S_{\mathrm{inv}}}\end{matrix}(LImp)}{\{F\}\ \alpha_{6,7}\ \{B\} : S_{\mathrm{inv}}}(If)$$

Here, $F_\top$ and $F_\perp$ are defined as follows:

$$F_\top = \begin{pmatrix} v_f = 0 \wedge v_r \geq 0 \wedge t = \rho \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix},$$

$$F_\perp = \begin{pmatrix} v_f \geq 0 \wedge v_r = 0 \wedge t = \rho \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix},$$

and B' and S' were defined in (32).

Note that $F_\top$ is equivalent to $F \wedge v_f = 0$, which allows application of (IF). Similarly, $F \wedge v_f \neq 0$ implies $F_\perp$, B' implies B, and S' implies S_{inv}, which allows us to apply (LIMP).

For α_6, we use (DWH) with the following invariants, variants, and terminators:

- $e_{\mathsf{inv},1} = (\rho - t = 0)$,
- $e_{\mathsf{inv},2} = (v_f = 0)$,
- $e_{\mathsf{inv},3} = (y_f - y_r - \mathsf{dRSS}(v_f, v_r, \rho - t) > 0)$,
- $e_{\mathsf{var},1} = v_r, \quad e_{\mathsf{ter},1} = -b_{\min}$.

If we compute the Lie derivative of $e_{\mathsf{inv},3}$, we get

$$\mathcal{L}_{\delta_r^1} e_{\mathsf{inv},3} = \begin{cases} 0 & \text{if } \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) \geq 0 \\ -v_r & \text{otherwise,} \end{cases}$$

and the proof that this is non-negative follows the same pattern as that in Section A3. This proves that

$$\{F_\top\}\, \alpha_6\, \{B\} : \begin{pmatrix} v_r \geq 0 \wedge v_f = 0 \wedge t = \rho \wedge \\ y_f - y_r > \mathsf{dRSS}(v_f, v_r, \rho - t) \end{pmatrix}$$

is valid. By (LIMP), we get that $\{F_\top\}\alpha_6\{B\} : S_{\mathsf{inv}}$, since the safety condition above implies S_{inv}.

For α_7, we use the (DWH) rule with the following invariants, variants, and terminators:

- $e_{\mathsf{inv},1} = (v_r = 0)$,
- $e_{\mathsf{inv},2} = (\rho - t = 0)$,
- $e_{\mathsf{inv},3} = (y_f - y_r - \mathsf{dRSS}(v_f, v_r, \rho - t) > 0)$,
- $e_{\mathsf{var},1} = v_f, \quad e_{\mathsf{ter},1} = -b_{\max}$.

Again, let us compute the Lie derivative of $e_{\mathsf{inv},3}$:

$$\mathcal{L}_{\delta_f} e_{\mathsf{inv},3} = \begin{cases} 0 & \text{if } \mathsf{dRSS}_{\pm}(v_f, v_r, \rho - t) \geq 0 \\ v_f & \text{otherwise.} \end{cases}$$

The term above is non-negative by $e_{\mathsf{var},1}$, which proves the validity of

$$\{F_\top\}\, \alpha_6\, \{B'\} : S',$$

where B' and S' were defined in (32). This concludes the proof of validity of $\{D_\perp\}\alpha_{4,7}\{B\} : S_{\mathsf{inv}}$, and finally that of $\{A\}\ \alpha\ \{B\} : S$.

REFERENCES

[1] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *CoRR preprint*, 2017. [Online]. Available: http://arxiv.org/abs/1708.06374

[2] I. Hasuo, "Responsibility-sensitive safety: An introduction with an eye to logical foundation and formalization," *CoRR*, 2022. [Online]. Available: https://arxiv.org/abs/2206.03418

[3] M. McNaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, "Motion planning for autonomous driving with a conformal spatiotemporal lattice," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 4889–4895.

[4] Y. Luo *et al.*, "Targeting requirements violations of autonomous driving systems by dynamic evolutionary search," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2021, pp. 279–291.

[5] E. I. Liu, C. Pek, and M. Althoff, "Provably-safe cooperative driving via invariably safe sets," in *Proc. IEEE Intell. Veh. Symp.*, 2020, pp. 516–523.

[6] C. Pek and M. Althoff, "Efficient computation of invariably safe states for motion planning of self-driving vehicles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3523–3530.

[7] A. Shashua, S. Shalev-Shwartz, and S. Shammah, "Implementing the RSS model on NHTSA pre-crash scenarios," *Mobileye*, 2018. [Online]. Available: https://static.mobileye.com/website/corporate/rss/rss_on_nhtsa.pdf

[8] J. Silberling, P. Wells, A. Acharya, J. Kelly, and J. Lenkeit, "Development and application of a collision avoidance capability metric," in *Proc. WCX SAE World Congr. Experience*, SAE International, Apr. 2020, pp. 1–13. [Online]. Available: https://doi.org/10.4271/2020-01-1207

[9] L. Wang, C. F. Lopez, and C. Stiller, "Realistic single-shot and long-term collision risk for a human-style safer driving," in *Proc. IEEE Intell. Veh. Symp.*, 2020, pp. 2073–2080.

[10] N. Altekar *et al.*, "Driving safety performance assessment metrics for ADS-equipped vehicles," *SAE Int. J. Adv. Curr. Pract. Mobility*, vol. 2, no. 5, pp. 2881–2899, Apr. 2020. [Online]. Available: https://doi.org/10.4271/2020-01-1206

[11] H. Zhao *et al.*, "Safety score: A quantitative approach to guiding safety-aware autonomous vehicle computing system design," in *Proc. IEEE Intell. Veh. Symp.*, Las Vegas, NV, USA, 2020, pp. 1479–1485.

[12] B. Weng, S. J. Rao, E. Deosthale, S. Schnelle, and F. Barickman, "Model predictive instantaneous safety metric for evaluation of automated driving systems," in *IEEE Intell. Veh. Symp.*, Las Vegas, NV, USA, 2020, pp. 1899–1906.

[13] *UL4600: Standard for Evaluation of Autonomous Products*, preprint, 1st ed. Northbrook, IL, USA: Underwriters Laboratories, Apr. 2020.

[14] F. Oboril and K.-U. Scholl, "Risk-aware safety layer for AV behavior planning," in *IEEE Intell. Veh. Symp.*, Las Vegas, NV, USA, 2020, pp. 1922–1928.

[15] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Commun. ACM*, vol. 18, no. 8, pp. 453–457, Aug. 1975. [Online]. Available: https://doi.org/10.1145/360933.360975

[16] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, pp. 576–580, 1969.

[17] F. S. de Boer, U. Hannemann, and W. P. de Roever, "Hoare-style compositional proof systems for reactive shared variable concurency," in *Proc. Found. Softw. Technol. Theor. Comput. Sci.,* in *Proc. 17th Conf., Proc., Ser. Lecture Notes Comput. Sci.*, S. Ramesh and G. Sivakumar, Eds., Springer, Kharagpur, India, vol. 1346, 1997, pp. 267–283. [Online]. Available: https://doi.org/10.1007/BFb0058036

[18] A. Platzer, *Logical Foundations of Cyber-Physical Systems*. Berlin, Germany: Springer, 2018.

[19] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *Proc. IEEE Int. Real-Time Syst. Symp.*, 2007, pp. 400–412.

[20] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe online control system upgrades," in *Proc. IEEE Amer. Control Conf.*, vol. 6, 1998, pp. 3504–3508.

[21] R. de Iaco, S. L. Smith, and K. Czarnecki, "Safe swerve maneuvers for autonomous driving," in *Proc. IEEE Intell. Veh. Symp.*, 2020, pp. 1941–1948.

[22] B. Gaßmann *et al.*, "Towards standardization of AV safety: C library for responsibility sensitive safety," in *Proc. IEEE Intell. Veh. Symp.*, Paris, France, 2019, pp. 2265–2271. [Online]. Available: https://doi.org/10.1109/IVS.2019.8813885

[23] N. Roohi, R. Kaur, J. Weimer, O. Sokolsky, and I. Lee, "Self-driving vehicle verification towards a benchmark," *CoRR preprint*, 2018. [Online]. Available: http://arxiv.org/abs/1806.08810

[24] A. Baheri, S. Nageshrao, I. Kolmanovsky, A. Girard, E. Tseng, and D. Filev, "Deep reinforcement learning with enhanced safety for autonomous highway driving," in *Proc. IEEE Intell. Veh. Symp.*, Las Vegas, NV, USA, 2020, pp. 1550–1555.

[25] A. Rizaldi, F. Immler, B. Schürmann, and M. Althoff, "A formally verified motion planner for autonomous vehicles," in *Proc. Autom. Technol. Verification Anal. - 16th Int. Symp.*, Oct. 2018, *Proc., Ser. Lecture Notes Comput. Sci.*, S. K. Lahiri and C. Wang, Eds., Springer, Los Angeles, CA, USA, vol. 11138, 2018, pp. 75–90. [Online]. Available: https://doi.org/10.1007/978-3-030-01090-4_5

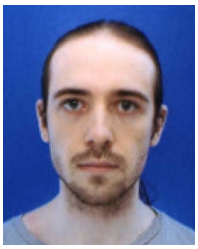
[26] T. Nipkow, L. C. Paulson, and M. Wenzel, "Isabelle/HOL - A proof assistant for higher-order logic," in *Lecture Notes in Computer Science*, vol. 2283, Berlin, Germany: Springer, 2002, pp. 67–104.

[27] R. K. Salay, M. Czarnecki S. Elli, I. J. Alvarez, S. Sedwards, and J. Weast, "PURSS: Towards perceptual uncertainty aware responsibility sensitive safety with ML," in *Proc. Workshop Artif. Intell. Safety*, *Co-Located 34th AAAI Conf. Artif. Intell., SafeAI, AAAI, ser. CEUR Workshop Proc.*, H. Espinoza, J. X. C. Hernández-Orallo Chen, S. S. ÓhÉigeartaigh, X. Huang, M. Castillo-Effen, R. Mallah, and J. McDermid, Eds., New York City, NY, USA, vol. 2560, 2020, pp. 91–95. [Online]. Available: http://ceur-ws.org/Vol-2560/paper34.pdf

[28] T. Kobayashi, R. Salay, I. Hasuo, K. Czarnecki, F. Ishikawa, and S. Katsumata, "Robustifying controller specifications of cyber-physical systems against perceptual uncertainty," in *Proc. NASA Formal Methods - 13th Int. Symp., NFM 2021, Virtual Event*, *Proc., Ser. Lecture Notes Comput. Sci.*, A. Dutle, M. M. Moscato, L. Titolo, C. A. Muñoz, and I. Perez, Eds., Springer, vol. 12673, 2021, pp. 198–213. [Online]. Available: https://doi.org/10.1007/978-3-030-76384-8_13

[29] M. Angus, K. Czarnecki, and R. Salay, "Efficacy of pixel-level OOD detection for semantic segmentation," *CoRR preprint*, 2019. [Online]. Available: http://arxiv.org/abs/1911.02897
[30] J. Chow *et al.*, "Certified control: A new safety architecture for autonomous vehicles," pp. 1–11, 2020. [Online]. Available: https://groups.csail.mit.edu/sdg/pubs/2020/certified-control.pdf
[31] G. Winskel, *The Formal Semantics of Programming Languages*. Hoboken, NJ, USA: MIT Press, 1993.
[32] M. Huisman and B. Jacobs, "Java program verification via a hoare logic with abrupt termination," in *Proc. Fundam. Approaches Softw. Eng., 3rd Int. Conf., FASE*2000, ETAPS, *Proc., Ser. Lecture Notes Comput. Sci.*, T. S.E. Maibaum, Ed., Springer, Berlin, Germany, vol. 1783, 2000, pp. 284–303. [Online]. Available: https://doi.org/10.1007/3-540-46428-X_20
[33] R. W. Floyd, "Assigning meanings to programs," in *Program Verification*. Berlin, Germany: Springer, 1993, pp. 65–81.
[34] H. Khalil, *Nonlinear Systems*. Hoboken, NJ, USA: Prentice Hall, 1996.
[35] A. Trautman, "Remarks on the history of the notion of Lie differentiation," in *Variations, Geometry and Physics: In Honour of Demeter Krupka's Sixty-Fifth Birthday*, O. Krupková and D. J. Saunders, Eds. NY, USA: Nova Science, 2008, pp. 297–302.
[36] C. Schmidt, F. Oechsle, and W. Branz, "Research on trajectory planning in emergency situations with multiple objects," in *Proc. IEEE Intell. Transp. Syst. Conf.*, 2006, pp. 988–992.
[37] H. Ataelmanan, O. C. Puan, and S. A. Hassan, "Examination of lane changing duration time on expressway," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 1144, no. 1, May 2021, Art. no. 012078. [Online]. Available: https://doi.org/10.1088/1757-899x/1144/1/012078
[38] Wolfram Research, Inc., "Wolfram Research, Inc., Version 12.3.1," Champaign, IL, 2021. [Online]. Available: https://www.wolfram.com/mathematica
[39] S. Mitsch and A. Platzer, "The keymaera X proof IDE - concepts on usability in hybrid systems theorem proving," in *Proc. 3rd Workshop Formal Integr. Develop. Environment, F-IDE, FM 2016*, Ser. EPTCS, C. Dubois, P. Masci, and D. Méry, Eds., Limassol, Cyprus, vol. 240, Nov. 2016, pp. 67–81. [Online]. Available: https://doi.org/10.4204/EPTCS.240.5
[40] K. Czarnecki, "Automated driving system (ADS) high-level quality requirements analysis-driving behavior comfort," preprint, Jul. 2018.

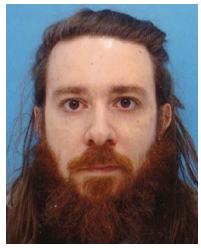
Ichiro Hasuo received the Ph.D. degree (*cum laude*) in computer science from Radboud University Nijmegen, Nijmegen, The Netherlands, in 2008. He is currently a Professor with the National Institute of Informatics (NII), Tokyo, Japan. He is also the Research Director of the JST ERATO Metamathematics for Systems Design Project, and the Director of Research Center for Mathematical Trust in Software and Systems with NII. His research interests include mathematical (logical, algebraic, and categorical) structures in software science, abstraction and generalization of deductive and automata-theoretic techniques in formal verification, integration of formal methods and testing, and their application to cyber-physical systems and systems with statistical machine learning components.



Clovis Eberhart received the Ph.D. degree in mathematics and computer science from Université Savoie Mont Blanc, Chambéry, France, in 2018. He is currently a Project Researcher with the JST ERATO Metamathematics for Systems Design Project, National Institute of Informatics, Tokyo, Japan. He is also a member of the Japanese-French Laboratory for Informatics. His research interests include semantics of programming languages, logic and category theory in computer science, and their applications to verification.

James Haydon received the Ph.D. degree in mathematics from the University of Oxford, Oxford, U.K., in 2014. He is currently a Project Technical Specialist with the JST ERATO Metamathematics for Systems Design Project.

His research interests include semantics of programming languages, functional programming, logic, and categories in computer science.



Jérémy Dubut received the Ph.D. degree in mathematics and computer science from Université Paris-Saclay, Gif-sur-Yvette, France, in 2017. He is currently a Project Assistant Professor with the JST ERATO Metamathematics for Systems Design Project. He is also a member of the Japanese-French Laboratory for Informatics. His research interests include category theory, algebraic topology, and formalised mathematics.

Rose Bohrer is currently an Assistant Professor with the Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA, USA. Her research interests include formal methods and programming language foundations for cyber-physical systems, including interactive theorem proving for hybrid systems with applications to driving.

Tsutomu Kobayashi received the Ph.D. degree from The University of Tokyo, Tokyo, Japan, in 2017. He is currently a Researcher with the JST ERATO Metamathematics for Systems Design Project, National Institute of Informatics, Tokyo, Japan.

His research interests include formal modeling and verification of software systems, theorem proving methods, and software testing.

Sasinee Pruekprasert received the Ph.D. degree in engineering from Osaka University, Toyonaka, Japan, in 2016. She is currently a Project Assistant Professor with the JST ERATO Metamathematics for Systems Design Project. Her research interests include supervisory control of discrete event systems, abstraction-based controller design, and decision-making of autonomous vehicles.

Xiao-Yi Zhang is currently a Project Assistant Professor with the National Institute of Informatics, Japan. His main research interests include software testing, software fault localisation, and hazard analysis for cyber-physical systems.

Erik André Pallas is currently working toward the master's degree on the deductive verification of safety rules for traffic scenarios in autonomous driving in the Elite Graduate Program Software Engineering with the University of Augsburg, Augsburg, Germany, Technical University of Munich, Munich, Germany, and Ludwig Maximilian University of Munich, Munich, Germany.

His research interests include formal methods for modeling and verification of software systems.

Akihisa Yamada received the Ph.D. degree in information science from Nagoya University, Nagoya, Japan, in 2014. He is currently a Senior Researcher with the National Institute of Advanced Industrial Science and Technology, Japan.

His research interests include term rewriting, termination and complexity analysis, and interactive theorem proving.

Kohei Suenaga is currently an Associate Professor with the Graduate School of Informatics, Kyoto University, Kyoto, Japan. His research interests include formal verification of software and hybrid systems and verification and testing of/for machine learning systems.

Fuyuki Ishikawa is currently an Associate Professor with Information Systems Architecture Science Research Division and a Deputy Director with GRACE Center, National Institute of Informatics, Tokyo, Japan. His research interests include dependability of advanced software systems, including testing and verification techniques for autonomous driving systems and machine learning-based systems.

Kenji Kamijo received the B.S. degree in electrical and electronic engineering from the Faculty of Engineering, Tokyo Institute of Technology, Tokyo, Japan, in 1991. He is currently an Assistant Manager with Integrated Control System Development Division, Mazda Motor Corporation, Hiroshima, Japan.

Yoshiyuki Shinya received the M.E. degree in electronics engineering from the University of Osaka, Suita, Japan, in 1984, and the M.B.A. degree from the University of Kobe, Kobe, Japan, in 2008. He is currently a Senior Principal Engineer with Integrated Control System Development Division, Mazda Motor Corporation, Hiroshima, Japan.

In 1984, he joined Mazda, where he has been engaged in research on engine control systems and computer aided control system design.

Takamasa Suetomi received the M.E. degree in mechanical engineering from The University of Tokyo, Tokyo, Japan, in 1987. He is currently a Senior Principal Engineer with Integrated Control System Development Division, Mazda Motor Corporation, Hiroshima, Japan.

In 1987, he joined Mazda, where he has been engaged in research on man-machine systems, driving simulators, advanced driver assistant systems and battery electric-drive systems. He is currently responsible for the development technology for vehicle control models.