Modelling and Proving Safety in Autonomous Cars Scenarios in HOL-CSP

Deliverable No. 2 — Final Report

Paolo Crisafulli     Safouan Taha     Burkhart Wolff

December 1, 2021

University Paris-Saclay
3 rue Joliot Curie
Bâtiment Breguet
91190 Gif-sur-Yvette

Laboratoire des Methodes Formelles (LMF)
1, Rue Raimond Castaing
91190 Gif-sur-Yvette
France
All rights reserved to SystemX and its SVR's project partners. You may not copy, modify, distribute, sell, or lease any part of codes described herein. Also, you may not use our source code unless you have our written permission or applicable law lets you do so.

Contributors. We would like to thank the following contributor of this work: Nicolas Méric.

Acknowledgments. This research work has been carried out in the framework of the SVR project (Robot Vehicle Scenarios and autonomous shuttles) at IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the French Program Investissements d’Avenir (PIA).
# Contents

1. **Introduction** ................................................................. 7

2. **The CSP-based Generic Autonomous Car Model** ..................... 11
   2.1 Fundamental Modeling Assumptions .................................. 11
   2.2 Actor States and its Extensions .................................... 12
   2.3 Global States: Scenes and “Open Scenes” .......................... 14
   2.4 Motions (Driving Strategies) ......................................... 14
   2.5 CSP Model I ............................................................... 15
     2.5.1 Preliminaries ...................................................... 15
     2.5.2 Events (Simplified) ............................................. 15
     2.5.3 The Behaviour of Actors as CSP-Processes ................. 15
     2.5.4 Maxwells Daemon (I) ........................................... 16
     2.5.5 Safe Scenarios .................................................. 18
     2.5.6 Scenario Refinement ........................................... 19

3. **From MOSAR-Ontologies to a Semantics in CSP** .................. 25
   3.1 More Examples: Expressing the MOSAR hierarchy of Actor States 25
     3.1.1 Examples .......................................................... 26
   3.2 Static Environment : The Street Topology .......................... 27
     3.2.1 Linear Lane Example .......................................... 27
   3.3 CSP Model II ............................................................ 27
     3.3.1 Generalized Events ............................................. 27
     3.3.2 Maxwells Daemon II ............................................ 27
     3.3.3 Behavior of Actors ............................................. 28
     3.3.4 Examples .......................................................... 28
     3.3.5 Scenario Examples ............................................. 28

4. **A Safety-Property in Autonomous Cars: RSS in a 2-cars-Scenario** 33
   4.1 Global Parameters of the Scenario-Class .......................... 33
   4.2 Relations between Driving Strategies .............................. 34

5. **A Safety-Property in Autonomous Cars: RSS in a N-cars-Scenario** 43
   5.1 Global Parameters of the Scenario-Class .......................... 43
   5.2 Relations between Driving Strategies .............................. 43

6. **An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario** 53
   6.1 Context : modeling N cars, one lane, one direction ............. 53
   6.2 Preliminaries .......................................................... 54
## Contents

6.3 Our motion vs RSS ........................................... 55
6.4 Our motion is safe .......................................... 55
   6.4.1 Safety distance proofs ................................. 58
   6.4.2 "No collision" proofs when \( b'_0 < b'_1 \) .......... 62

7 Conclusion ................................................. 69
   7.1 Summary ................................................ 69
      7.1.1 Outline of the Theory Development .......... 69
      7.1.2 General Results .................................. 69
      7.1.3 Abstract Test-Cases derived from the Proof-Structure .. 71
   7.2 Lessons Learnt .......................................... 73
   7.3 Future Directions ....................................... 74
Abstract

We present an approach to model scenarios of autonomous cars in HOL-CSP \[10\] and prove particular safety properties via interactive proofs in the Isabelle/HOL system (https://en.wikipedia.org/wiki/Isabelle_(proof_assistant)).

The basis of this work is an ontology for Autonomous Car Scenarios given in MOSAR (https://www.mosar.io) that describes a collection of actors (e.g. cars, trucks, bicycles), equipments (e.g. signals, vehicle lights, etc.), infrastructures (e.g. expressways, intersections, etc.) and their dynamic interactions throughout driving scenarios.

We represent the behaviour of actors and (rudimentarily) equipments as processes, i.e. infinite sets of traces denoting classes of scenarios. In particular, actors were represented as HOL-CSP processes. Due to the non-determinism and event-polymorphism of HOL-CSP, actor descriptions can be partially defined wrt. to data and arbitrarily "chaotic" in their behaviour. A translation scheme of MOSAR-ontologies into actor processes in HOL-CSP is sketched.

For a particular scenario described in \[9\] (two cars in a linear line, no backwards driving) we specialize our framework and demonstrate a machine-checked safety proof: If all the actors apply a particular driving strategy taking into account position, speed and acceleration as well as distance to the car in front, there will be no situation with a collision. This strategy — called Responsibility-Sensitive Safety — is formulated as a function and the resulting invariant formally proven in Isabelle/HOL, while overcoming a number of short-comings in both the original modeling and the original paper-and-pencil proof.
1 Introduction

This work stands in the context of the projet SVR (Robot Vehicle Scenarios and autonomous shuttles, https://www.irt-systemx.fr/en/projets/svr/) affiliated at the Institut de Recherche Technologique IRT SystemX (https://www.irt-systemx.fr, Palaiseau, France)

The overall goals of the SVR project are

1. to develop a “common frame of reference” between the branch autonomous bus systems and robot-taxis for both verification and validation of safe systems, where

2. this reference-frame comprises a common language and a library of scenarios serving to construct test-cases in virtual test environments.

The ISO standard SOTIF (ISO/PAS 21448) [1] introduces a classification of scenarios. The standard introduces a classification of scenarios in several categories called known safe, known unsafe, unknown safe and unknown unsafe, depending on whether the scenario is known during the design of the system or discovered during the test phase, and depending on whether the scenario does not destabilize the system or cause it to fail. In the context of the SVR project, it is also a question of defining an approach to list the cases covered and compliant (with success criteria) by the simulation, in order to be able to identify a safe perimeter of use (Operational Design Domain or ODD).

In the spirit of the SOTIF standard, this amounts to

[...] the development of an argument which will demonstrate on the one hand that all the scenarios unknown unsafe is small enough and on the other hand the set of known unsafe scenarios is taken into account through the improvements of SOTIF and therefore that the probability of encountering this type of scenario is low enough. [1, p. 8]

The objective of this sub-project, for which this document is the final deliverable, is to find behavioral models that are sufficiently flexible and open world to take into account known unsafe and unknown unsafe scenarios, but which are sufficiently well-founded in mathematical logic in order to allow for strong guarantees for (safety) properties for a large set of scenarios, i.e., properties of the form: for all scenarios in which cars respect a driving strategy X, there will be no collision. Note that this work is not about quantifying the probability of encountering unknown unsafe scenarios, which remains a question difficult if not impossible to define in the context of an endless number of possible scenarios. Rather, it will be possible to model classes of scenarios allowing simulation, generation of test cases, and / or proof of the inclusion of scenario-classes and relations between behaviours of actors.

The objectives of our sub-project in the SVR context are described as follows:

1. Design sufficiently generic models to represent classes of situations and relevant scenarios.
1 Introduction

2. Model domain concepts inspired by SOTIF (such as dangers and situations), ODD (optional), etc.

3. Model scenario classes and study the generation of test cases with coverage criteria to be established.

4. Prove that a class of scenarios is safe, which allows to establish the absence of unknown unsafe scenarios among such a class.

5. Illustrate what an "open world" model based on CSP would be, and its capacity to produce unknown unsafe scenarios.

This list of objectives boils down to two major targets.

The first target is a general modeling framework that allows for giving a common semantics for car scenario modeling languages such as used within the MOSAR platform\(^1\) or Foretellix's M-SDL\(^2\). The framework will necessarily be very general with respect to expressivity in order to capture both the desired variability of behaviour and data which is expressed in the terms unknown safe scenarios and unknown unsafe scenarios. A compilation of MOSAR or M-SDL into this common semantic framework should be possible. The framework should support refinement notions in order to structure scenario classes as well as a smooth transition to executable sub-models allowing simulation and test.

Second target of the Analysis: Formalising and proving the safety-property "no collision" for Responsibility Sensitive Safety (RSS), a particular driving strategy that controls acceleration, speed and distance to the car in front. This represents a concrete instance of the aforementioned general modeling framework which is designed to demonstrate the possibility to formally analyse safety properties by proof techniques inside the framework.

More concretely: we will formalize the concepts of S. Shaliv-Shwartz, S. Shammah, A. Shashua in "On a Formal Model of Safe and Scalable Self-driving Cars"s in the framework and prove formally the intended safety property ("no collision" in all situations).

1. We outline the RSS as presented in [9]:
   - A formal model and an analysis of the the collision danger
   - Formal definition of a behaviour (the "driving strategy") compatible with the law ("Duty of Care")
   - Paper-and-pencil proof that this behavior ensures global safety ("Utopia is possible")

2. we provide an instantiation of our general framework for actors as a CSP based model

3. prove formally the re-formulation of the problem as an invariant-preservation proof, and

\(^1\)https://www.mosar.io
\(^2\)https://www.foretellix.com/open-language/
4. prove finally the preservation of this invariant formally in Isabelle/HOL.

A well-known reference theory to study behavioral models is the theory ”Concurrent Sequential Processes” (CSP) originally proposed by Anthony Hoare in the late 70ies [4], but has since evolved substantially [2, 3, 8].

Recent work of the authors comprised a formalization of CSP inside Isabelle/HOL [7] and a formal development environment for modeling and theorem proving called HOL-CSP [10]. This environment can cope with events of arbitrary type \( \alpha \), i.e. infinite sets of events, in contrast to model-checking approaches limited to finite ones. This paves the way for models involving processes with rich data states, the handling of dense and real time as well as the handling of newtonian physics by explicit representation of acceleration and speed by vectors in \( \mathbb{R}^2 \) or \( \mathbb{R}^3 \). HOL-CSP and an extension module CSP-Ref-Tk (the ”CSP Refinement Toolkit”) has been archived in the Isabelle ”Archive of Formal Proofs” https://www.isa-afp.org in the entries https://www.isa-afp.org/entries/HOL-CSP.html and https://www.isa-afp.org/entries/CSP_RefTK.html and represent therefore the background of this work.

Finally, for this report, we use Isabelle/DOF, a Document Ontology Framework, which is able to enforce specific ontologies during document evolution by specific Isabelle IDE support. Thus the coherence between the formal and the informal parts of the content of this document can be mechanically checked.
2 The CSP-based Generic Autonomous Car Model

The setup of this Core-Theory results from importing the following background theories:

```isabelle
theory CSP_AutoCars
imports Isabelle_DOF.technical_report
  CSP_RefTK.Properties  CSP_RefTK.CSP_ext
  HOL.Real_Vector_Spaces HOL--Analysis.Product_Vector
```

Beyond technicalities for the underlying document preparation, this core theory is based on the following library components:

1. Vector-Spaces and analysis packages for modeling physical quantities, and
2. CSP and the CSP-Refinement Toolkit in order to model situations (snapshots), scenarios (a sequence of snapshots) and relations between classes of scenarios.

2.1 Fundamental Modeling Assumptions

We will use parametric polymorphism\(^1\) in order to model the class hierarchies on actors (resp. their states) and their extensibility. Since actor states will comprise physical quantities like position, speed, acceleration, the global world state based upon the states of all its actors (and environment components), the latter will also be polymorphic and extensible.

We chose to reuse the theory `HOL.Real_Vector_Spaces` from the Isabelle/HOL library in order to model the above mentioned quantities as vectors. This theory provides the type-class `real_normed_vector`, which provides the algebraic structure of vector-spaces for types such as `\(\mathbb{R}\)`, `\(\mathbb{R} \times \mathbb{R}\)`, `\(\mathbb{Q}\)^3`, or even machine type models such as, for example, the 2-dimensional `64\text{ word} \times 64\text{ word}` over 64-bit-vectors as occuring in concrete calculation in a programming language.

The only compromise that this choice implies is that `real_normed_vector` comes with a scalar multiplication\(^2\) `\(a \star \mathbb{R} \times\)` for which the `a` is required to be a real number `\(\mathbb{R}\)`. In order to model the physics of vector spaces in Newtonian physics, however, this choice is utmost nearly unavoidable: real-valued scaling factors appear naturally for vectors in physical computations such as `(1::'a) / 2 \star \mathbb{R} \; t^2 \; \star \mathbb{R} \; a` or even `sqrt 2 \; \star \mathbb{R} \; a` (as occurring naturally in norms of 2-dimensional vectors).

---

\(^1\)https://en.wikipedia.org/w/index.php?title=Parametric_polymorphism&oldid=1038893302
2.2 Actor States and its Extensions

We model the dynamic environment of an autonomous car ego as a system of parallel processes. These processes have a state — a position, a speed, a set of possible positive and negative accelerations, potentially a physical extension — modeling the physical quantities of these objects. Speed can only evolve via acceleration over time, and positions must be bound to a topology. However, positions, speeds and accelerations are parameterized in this model and can be (two-dimensional, three-dimensional) vectors in discrete or non-discrete space-times. The mother of all actor states is thus parameterized by vector-type-parameter \( \mathcal{v} \) which is constrained to the type-class \( \text{real_normed_vector} \).

\[
\text{record} \quad \mathcal{v} \bowtie \text{real_normed_vector} \quad \text{as} \quad \begin{align*}
\text{pos} \ &:: \ \mathcal{v} \quad \text{— current position} \\
\text{speed} \ &:: \ \mathcal{v} \quad \text{— current speed} \\
\text{acc} \ &:: \ \mathcal{v} \quad \text{— current acceleration}
\end{align*}
\]

Based on parametric polymorphism, it is possible to express some form of single-inheritance as required by our overall objective "common frame of reference" which should be extensible and therefore allow forms of "unknowns" in the models.

The trick is done by representing record types as cartesian products \( \tau_1 \times \ldots \times \tau_n \times \alpha \) where the \( \tau_i \) correspond to the types of the attributes and the type-variable \( \alpha \) stands for an "extension field" of the record. In this view, the "attributes" of a record become a projection function into the cartesian tuple.

For example, the record representation above for the most general actor state space can be seen as a product type \( \mathcal{v} \times \mathcal{v} \times \mathcal{v} \times \alpha \) with the projection functions:

1. \( \text{pos} \equiv \text{fst} \)
2. \( \text{speed} \equiv \text{fst o snd} \)
3. \( \text{acc} \equiv \text{fst o snd o snd} \)
4. and the implicit projection on the more-field: more $\equiv$ \textit{fst} o snd o snd o snd (where \textit{fst} and \textit{snd} are the usual projections into a binary cartesian product and \textit{o} is the function composition.)

Isabelle/HOL generates from a record declaration a special syntax for its support. Rather than tuple notations such as $(a,b,c,$more$)$, records can be denoted by $\langle pos = a, speed = b, acc = c, ... = more \rangle$, and the corresponding projection functions were defined to work directly on this notation. Isabelle also introduces a notation for record-types which were written $(v, 'a)$ \textit{as_scheme} for which the synonym $(v,'a)$ \textit{as_scheme} is generated.

Note that Isabelle also supports the equivalent notation: $\langle pos = a, speed = b, acc = c \rangle$ for $\langle pos = a, speed = b, acc = c, ... = () \rangle$. Here, the () is the "empty-product", i.e. the only element in the so-called \textit{unit}-type. This corresponds to the idea of a \textit{finalized} class in an class hierarchy, i.e. a class from which no further inheritance is possible (and which is therefore "known" in all respects).

Isabelle can prove theorems over states containing polymorphic variables, that is, that will hold for all type instances of this theorem. This implies, that theorems can be established on schematic actor states $(v,'a)$ \textit{as_scheme} that hold for all future extensions, thus holds over a certain form of "unknowns".

However, we have to be precise if we want to reason over a closed state-space (so: $(v$ as or equivalently: $(v$ as) or an open state-space: (so: $(v$,$'a)$ \textit{as_scheme} or equivalently: $(v$,$'a)$ \textit{as_scheme}). In order to bridge a little the gap between these two forms we introduce by convention the following \textit{type_synonym}'s for each actor-state class:

$$\text{type_synonym} \ (v$,$'a)$ \textit{as} $= (v$,$'a)$ \textit{as_scheme}$$

\textbf{Examples of Actor Instances:}

\begin{itemize}
  \item \textbf{term} pos :: $(v$:\textit{real}$_\text{normed}_\text{vector}$,$'a)$ \textit{as} $\Rightarrow (v$
  \item \textbf{term} $\langle pos = 14, speed = 2, acc = -1 \rangle :: \text{real as}$
  \begin{itemize}
    \item One-dimensional \textit{real} as.
    \item \textbf{term} $\langle pos = (14, 15), speed = (\sqrt{2}, 2.4),$
  \item \textbf{term} $\langle pos = (0, 0), ... = a \rangle :: (\text{real} \times \text{real}, 'a)$ \textit{as}$\_$.\]
  \item Two-dimensional, extensible actor state space of type $(\text{real} \times \text{real}, 'a)$ \textit{as}$\_$.\]
  \item We provide the first major extension of the actor state space: we characterize an actor by a set of possible accelerations it may choose from, and a physical extension as an object which becomes relevant in case of collisions:
  \begin{itemize}
    \item \textbf{record} $(v$:\textit{real}$_\text{normed}_\text{vector}$) \textit{as}$\_\text{range}$ $= (v$ as $+$
  \item \textit{acc_range} $:: (v$ set) $-$ The set of possible accelerations of an actor. Depending on the instance for the underlying vector-space, this can be an acceleration in front, towards a side, or braking.
    \item \textit{extension_field} $:: (v$ set) $-$ gives the expansion of the physical object in terms of a set of positions (relative vectors).
  \item Again, following our convention, we provide an abbreviation for the "open" version of this actor state space:
2. CSP-based Generic Autonomous Car Model

```
type_synonym ('v', 'a) as_range = ('v', 'a) as_range_scheme

This construction implies the need of a particular invariant for ('v, 'a) as_range: the
current acceleration must be one of the possible ones:
```

definition wd_as :: ('v::real_normed_vector, 'a) as_range ⇒ bool
  where wd_as σ ≡ acc σ ∈ acc_range σ
corollary ⟨wd_as (pos = (14::real), speed = 2, acc = −1), acc_range = {−1..1}, extension_field = {}⟩
  by (simp_all add: wd_as_def)
```

2.3 Global States: Scenes and "Open Scenes"

Scenes are global system states comprising actor states.
```
type_synonym id_actor = nat — an index for actors
type_synonym ('v scene = id_actor ⇒ 'v as
  — a scene where all objects are identified by their index
type_synonym ('v, 'a) scene_ = id_actor ⇒ ('v, 'a) as_
  — ... the "open" version
```

Note that the above model of an "Open Scene" is not that open after all: while it encom-
passes the possibility of completely new actors and their characteristics, it does not allow
open-ness wrt. equipments (e.g. signals, vehicle lights, etc.) and infrastructures. However,
we can apply the general extension techniques of data in Isabelle/HOL also for this objective:
```
record ('v, 'a)scene_ = global_state :: id_actor ⇒ ('v, 'a) as_range
  — a record with a single field named global_state
type_synonym ('v, 'a, 'β)scene_ = ('v, 'a, 'β)scene_ as
  — and its open version
```

Note that the demon-state is still extensible: we can add a new table that captures the
state of environment components such as traffic lights or fences/gates etc.
```
Let ω be a 'v scene. Then we provide a predicate no_collision ω which is true when all
actors have relative distances ε.
```
definition no_collision :: ('v::real_normed_vector) scene ⇒ id_actor set ⇒ 'v ⇒ bool
  — when actors should keep some ε as distance from each other
  where no_collision ω sid ε ≡ ∀ i∈sid. ∀ j∈sid. norm (pos(ω j)−pos(ω i)) ≤ norm ε
```

corollary no_collision_ε_0 : no_collision ω sid 0 = inj_on (λi. pos (ω i)) sid
  — when actors are points we can set ε to zero

unfolding inj_on_def no_collision_ε_0 by auto
```

2.4 Motions (Driving Strategies)

Let us define motions as descriptions of the moving behavior of the actors, or, in other
words, as a driving strategy giving next states after taking into account the capture of the
surrounding world.

14
2.5 CSP Model I

2.5.1 Preliminaries

declare cont_fun[simp] — missing in the general reasoning on continuity.

2.5.2 Events (Simplified)

datatype ’v events = sync (dt: real)(sc: ’v scene) — simulation step
  | up (’v scene) — update

abbreviation sync_pair ≡ λ(δt,ω). sync δt ω
— another way to denote the constructor events.sync

— proof tactic to break the cartesian product, not relevant
lemma sync_pair_simps[simp]:
  (Mprefix (range sync_pair) (P o inv sync_pair)) = Mprefix (range sync_pair) (λe sync. P (dt e sync, sc e sync))
by (smt (z3) case_prod_beta' comp_apply events.sel(1) events.sel(2) f_inv_into_f mprefix_eq prod.collapse)

2.5.3 The Behaviour of Actors as CSP-Processes

The processes generated by a set of actors and a motion function they share:
definition μ motion :: (id_actor set ⇒ ’v motion ⇒ (’v::real_normed_vector) events process)
  ([(_,_)]) where
μ motion sid motion ≡ μ X. sync_pair.? (δt,ω) → (up!ω'(∀ i∈sid. ω' i ∈ motion i δt ω)
  → X)

The "usual" recursive lemma, used for simplification and fixpoint reasoning:
lemma μ motion_rec: [sid·motion] = sync_pair.? (δt,ω)
2.5.4 Maxwells Daemon (I)

In classical physics, Maxwell’s demon is a concept around thought-experiments that postulate the existence of a kind of actor that “knows” the global state of all physical objects at a given point in time. Tongue-in-cheek, we will postulate this demon here as well (not diving in the discussion about the possibility of its existence in thermo-dynamics); one can see it as an “orchestrator” or “mediator” in our modeling framework.

In our framework, we assume a “Maxwell’s demon”:

1. that chooses non-deterministically some positive real sampling interval $\delta t$ strictly smaller that some bound $\Delta t$

2. that “knows” for given points in time the scene (global physical state) $\sigma$ of all actors.

Actors (and in future extensions of the model, similarly: infrastructure components):

1. receive the $\delta t$ from the demon,
Figure 2.2: The Behaviour of the Orchestrator (The "Maxwell Demon").
2 The CSP-based Generic Autonomous Car Model

2. send their physical state \( \Delta \sigma \) to him,
3. receive the scene (global state \( \sigma \)),
4. decide non-deterministically their possible accelerations according to their driving strategy which is sensitive to the road/environment (topology) and the scene \( \sigma \), and
5. are sensitive in their choice via their driving strategy to the scene \( \sigma \).

In this section, we present a somewhat simplified demon which makes strong assumptions on synchronisation over the global scene — basically 1) all actors learn about the global situation via a kind of broadcast and 2) the absence of infrastructure elements such as signals and communication between actors — in order to simplify the major proofs in the proof chapters over different versions of RSS driving strategies. A version of the demon overcoming these limitations is presented in the next chapter.

**definition** demon :: \( \text{real} \Rightarrow (\nu\text{::real_normed_vector}) \text{ scene } \Rightarrow (\nu \text{ events process}) \)

where \( \text{demon} \Delta t \equiv \mu X. (\lambda \omega. (\text{sync} \text{pair}!(\delta t, \omega_s)|(|\delta t \in \{0<..\Delta t\} \land \omega_s = \omega) \rightarrow (\text{up}?\omega' \rightarrow X \omega'))) \)

Here, \( \{0::a<..<\Delta t\} \) denotes the set containing the interval between 0 (exclusively) and a given clock-time \( \Delta t \).

Let’s do some formal verification: let’s prove the “usual” recursive unfolding lemma...

**lemma** demon_rec:
\( \text{demon} \Delta t \omega = (\text{sync} \text{pair}!(\delta t, \omega_s)|(|\delta t \in \{0<..\Delta t\} \land \omega_s = \omega) \rightarrow (\text{up}?\omega' \rightarrow \text{demon} \Delta t \omega')) \)

by (simp add : demon_def, subst fix_eq, simp)

Note that the \((\sqcap)\)-operator from CSP lets the demon choose an arbitrary time interval smaller than some bound \( \Delta t \) and send it to all actors, which respond with their actor state after that time interval.

2.5.5 Safe Scenarios

We define here a scenario as the constructor of a process, based on a set of actor ids, a motion (driving strategies of the actors), a maximum time step and an initial scene. The resulting process is the sequence of simulation steps (sync events) triggered by the demon.

**definition** scenario ::
\( \text{id}_\text{actor} \text{ set } \Rightarrow (\nu::\text{real_normed_vector}) \text{ motion } \Rightarrow \text{ real } \Rightarrow (\nu \text{ scene } \Rightarrow (\nu \text{ events process}) \)

where \( \text{scenario} \text{ sid motion} \Delta t \omega_0 \equiv (\text{demon} \Delta t \omega_0 | | \text{sid motion}) \setminus \{e. \neg \text{is sync e}\} \)

A safe (instantiated) scenario must refine \( \text{safe} \) (once instantiated with actor ids and an \( \varepsilon \) value).

**definition** safe :: \( \text{id}_\text{actor} \text{ set } \Rightarrow (\nu::\text{real_normed_vector}) \Rightarrow (\nu \text{ events process}) \)

where \( \text{safe} \text{ sid } \varepsilon \equiv \mu X. (\text{sync} \text{pair}!(\delta t, \omega_s)|(|\delta t > 0 \land \text{no collision} \omega_s \text{ sid } \varepsilon) \rightarrow X) \)

**definition** is_safe where \( \text{is_safe} \text{ sid motion} \Delta t \omega_0 \varepsilon \leftrightarrow \text{safe} \text{ sid } \varepsilon \leq \text{scenario sid motion} \Delta t \omega_0 \)

18
2.5.6 Scenario Refinement

monotony of $\text{is}_{\text{safety}}$: stricter driving strategies inherit safety from less rigorous ones ...

**Lemma** $\text{is}_{\text{safety}}_{\text{refine}}$:

**Assumes** $a : \lbrack \text{sid}\text{-motion}_2 \rbrack \leq \lbrack \text{sid}\text{-motion}_1 \rbrack$

and $b : \text{is}_{\text{safety}} \text{sid motion}_2 \Delta t \omega_0 \epsilon$

**Shows** $\text{is}_{\text{safety}} \text{sid motion}_1 \Delta t \omega_0 \epsilon$

**Proof** —

have $\langle \text{scenario sid motion}_2 \Delta t \omega_0 \leq \text{scenario sid motion}_1 \Delta t \omega_0 \rangle$

by $(\text{simp add: scenario_def a})$

then show $\langle \text{thesis} \rangle$

by $(\text{meson b is}_{\text{safety}}_{\text{def}} \text{order_trans})$

**Lemma** $\text{scenario_rec}$:

$\langle \text{scenario sid motion} \Delta t \omega = (\forall \omega \in \{ \text{sync} \delta t \omega | \delta t \in \{0 < .. \Delta t \}) \rightarrow (\forall \omega' \in \{ \omega', \forall i \in \text{sid}. \omega' i \in \text{motion} i (\text{dt e sync} \omega) \}. \text{scenario sid motion} \Delta t \omega' \rangle)$

apply $(\text{subt scenario_def, subt demon_rec, subt \text{motion}_\text{rec})}$

apply $(\text{simp add: read_def, subst gndet_mprefix_sync)}$

apply $(\text{simp add: image_mono})$

apply $(\text{subt \text{Mndet_Hide1, apply fastforce)}$

apply $(\text{subt \text{Mndet_Hide2, apply fastforce)}$

**Proof** (goal_cases)

**Case 1**

have $a : \langle \text{sync_pair'} \{ (\delta t, \omega) \}, 0 < \delta t \land \delta t \leq \Delta t \land \omega_s = \omega \rangle$

$= \{ \text{events.sync} \delta t \omega | \delta t. 0 < \delta t \land \delta t \leq \Delta t \}$ by $\text{auto}$

have $b : \langle (\text{inv events.up (events.up x)}) = x \text{ for } x :: \text{nat} \Rightarrow \forall \text{as} \rangle$

by $(\text{metis events.inject(2) f_inv_into_f rangell)}$

**Show** $\text{?case}$

apply $(\text{subt a})$

apply $(\text{intro mndet_eq all impl, auto, rule gndet_eq2, auto, subst b)}$

by $(\text{simp add: par_comm scenario_def)}$

**Qed**

Let $\omega_0$ be a scene with no collision, $\text{motion}$ a driving strategy always mapping to a scene with no collision, hence the corresponding scenario is safe

**Lemma** $\text{is}_{\text{safety}}_{\text{rule1}}$:

**Assumes** $a : \Delta t > 0$

and $b : \text{no}\text{collision} \omega_0 \text{sid} \epsilon$

and $c : \langle \omega \delta t. \text{no}\text{collision} \omega \text{sid} \epsilon \land \delta t \in \{0 < .. \Delta t \} \Rightarrow \text{let } A = \{ \omega'. \forall i \in \text{sid}. \omega' i \in \text{motion} i \delta t \omega \}

\text{in } A \neq \emptyset \land (\forall \omega' \in A. \text{no}\text{collision} \omega' \text{sid} \epsilon) \rangle$

**Shows** $\text{is}_{\text{safety}} \text{sid motion} \Delta t \omega_0 \epsilon$
2 The CSP-based Generic Autonomous Car Model

\[ \text{no}_{\text{collision}} \omega \text{ sid } \varepsilon \land \exists \delta t \in \{0<\ldots \Delta t\} \implies \text{let } A = \{\omega' . \forall i \in \text{sid. } \omega i \in \text{motion } i \exists \delta t \}
\]

\[ \exists \omega \text{ in } A \neq \{\} \land (\forall \omega' \in A. \text{no}_{\text{collision}} \omega' \text{ sid } \varepsilon) \text{ defines motion such that starting from a collision free scene, we move to another collision free scene} \]

**using** b **proof** (unfold is_{safe_def} safe_def, induct arbitrary: \( \omega_0 \) rule:fix_ind)

**case** 1

**then show** ?case **by** (simp add:monofun_def)

**next**

**case** 2

**then show** ?case **by simp**

**next**

**case** (3 x)

**have** d: \( \prod \in \text{sync pair} . (Collect ((<) 0) \times \{\omega_s. \text{no}_{\text{collision}} \omega_s \text{ sid } \varepsilon}) \) \( \rightarrow \) \( x \leq \text{\prod e sync} \in \text{events.sync } \delta \omega_0 | \delta t. 0 < \delta t \land \delta t \leq \Delta t \rightarrow x) \)

**apply** (rule Mndetprefix_FD_subset)

**using** 3 a c **by** auto

**have** \( \exists \delta t \in \{0<\ldots \Delta t\} \implies x \leq \text{gndet } \{\omega'. \forall i \in \text{sid. } \omega i \in \text{motion } i \delta t \omega_0\} \ (\text{scenario sid motion } \Delta t) \)

**for** \( \delta t \)

**apply** (rule mono_gndet_const)

**using** 3 c **by** auto

**with** 3 **show** ?case

**apply**(subst scenario_rec, auto)

**apply**(intro trans_FD[simplified failure_divergence_refine_def, OF d], rule mono_Mndetprefix_FD)

**by** auto

**qed**

— a second variant lemma, strictly equivalent to \text{is}_{safe_rule1}

**lemma** is_{safe_rule2}:

**assumes** a: \( \Delta t > 0 \)

**and** b: \( \text{no}_{\text{collision}} \omega_0 \text{ sid } \varepsilon \)

**and** c:\( \forall \omega \delta t. \text{no}_{\text{collision}} \omega \text{ sid } \varepsilon \land \delta t \in \{0<\ldots \Delta t\} \)

\[ \implies \forall i \in \text{sid. } \text{motion } i \delta t \omega \neq \{\} \]

**and** d:\( \forall \omega \delta t. \text{no}_{\text{collision}} \omega \text{ sid } \varepsilon \land \delta t \in \{0<\ldots \Delta t\} \)

\[ \implies \forall \omega' \in \{\omega'. \forall i \in \text{sid. } \omega i \in \text{motion } i \delta t \omega\}. \text{no}_{\text{collision}} \omega' \text{ sid } \varepsilon \]

**shows** is_{safe sid motion} \( \Delta t \omega_0 \varepsilon \)

**by** (intro is_{safe_rule1} a b, auto simp:c d motion_step)

**lemma** is_{safe_rule_inv}:

**fixes** \( P_{\text{inv}}:\ (t \Rightarrow (\forall \omega:\text{real_normed_vector}) \Rightarrow \text{nat set } \Rightarrow \text{bool}) \)

— let \( P_{\text{inv}} \) be a predicate

**assumes** a: \( \Delta t > 0 \)

**and** b: \( P_{\text{inv}} \omega_0 \text{ sid}\)

— assume \( P_{\text{inv}} \) is true for scene \( \omega_0 \)

**and** c:\( \forall \omega \delta t. P_{\text{inv}} \omega \text{ sid } \varepsilon \land \delta t \in \{0<\ldots \Delta t\} \implies \forall i \in \text{sid. } \text{motion } i \delta t \omega \neq \{\} \)

— given a non empty motion

**and** d:\( \forall \omega \delta t. P_{\text{inv}} \omega \text{ sid } \varepsilon \land \delta t \in \{0<\ldots \Delta t\} \)

\[ \implies \forall \omega' \in \{\omega'. \forall i \in \text{sid. } \omega i \in \text{motion } i \delta t \omega\}. P_{\text{inv}} \omega' \text{ sid} \)
— which preserves $P_{\text{inv}}$
and $\varepsilon / (\omega, P_{\text{inv}}, \omega \text{ sid} \Rightarrow \text{no\_collision} \omega \text{ sid} \varepsilon)$
— assume $P_{\text{inv}}$ implies no collision
shows $\text{is\_safe \_sid \_motion} \Delta t \omega_0 \varepsilon$
— then the motion is safe

using b

proof (unfold is\_safe\_def, induct arbitrary $\omega_0$ rule: fix_ind)

show adm ($\lambda a. \forall x. P_{\text{inv}} x \text{ sid} \Rightarrow P_{\text{inv}} x \text{ sid} \Rightarrow a \leq \text{scenario \_sid \_motion} \Delta t x$)

by (simp add: monofun_def)

next

fix $\omega_0$::nat $\Rightarrow \varepsilon$ as

assume $P_{\text{inv}} \omega_0 \text{ sid}$ and $P_{\text{inv}} \omega_0$ sid

then show $\bot \leq \text{scenario \_sid \_motion} \Delta t \omega_0$

by simp

next

fix $x$ fix $\omega_0$::nat $\Rightarrow \varepsilon$ as

assume hyp: ($\forall \omega_0. P_{\text{inv}} \omega_0 \text{ sid} \Rightarrow P_{\text{inv}} \omega_0 \text{ sid} \Rightarrow x \leq \text{scenario \_sid \_motion} \Delta t \omega_0$)

and base: $P_{\text{inv}} \omega_0$ sid

have $d$::$\exists \_\in \text{sync\_pair}$ (Collect (($<$) 0) $\times \{\omega, \text{ no\_collision} \omega \text{ sid} \varepsilon\}) $\Rightarrow$

$x \leq \exists e_{\text{sync}}\in\{\text{events\_sync} \delta t \omega_0 | \delta t. 0 < \delta t \land \delta t \leq \Delta t \} \Rightarrow x$

apply (rule Mndetprefix_FD_subset) using $a$[of $\omega_0$, OF base] by auto

have $e$:$\delta t \in \{0<..\Delta t\}$

$\Rightarrow x \leq \text{gndet} \{\omega'. \forall i \in \text{sid}. \omega' i \in \text{motion} i \delta t \omega_0\} (\text{scenario \_sid \_motion} \Delta t)$ $\Rightarrow$ for $\delta t$

apply (rule mono_gndet_const)

using base hyp c assms(4) apply auto by (rule motion_step, simp)

with $e$

show ($\Lambda x. \exists \_\in \text{sync\_pair} \{((\delta t, \omega, a). 0 < \delta t \land \text{no\_collision} \omega_0 \text{ sid} \varepsilon) \Rightarrow x\} x \leq \text{scenario \_sid \_motion} \Delta t \omega_0$)

apply (subst scenario_rec, auto)

apply (intro trans_FD [simplified failure_divergence_refine_def, OF d], rule mono_Mndetprefix_FD)

by auto

qed

— computing a motion from a set of accelerations following Newtonian kinetics: motion by acc

**type** synonym $\varepsilon$ motion\_acc $\Rightarrow \text{id\_actor} \Rightarrow \text{time} \Rightarrow \varepsilon$ scene $\Rightarrow \varepsilon$ set

— the set of possible accelerations for a given actor, $\delta t$ and initial scene

**definition** kinetics :: ($\varepsilon$::real\_normed\_vector) motion\_acc $\Rightarrow \varepsilon$ motion

where

kinetics motion\_acc $\equiv$

— defines a motion for a $\delta t$ such that $a$ (acceleration) remains constant during $\delta t$

$\lambda i \delta t \omega. \{ | \text{pos} = \text{pos} (\omega i) + (\delta t) \ast R \text{ speed} (\omega i) + ((\delta t^2)/2) \ast R a, \text{speed} = \text{speed} (\omega i) + (\delta t \ast R a), \text{acc} = a \} | a. a \in \text{motion\_acc} i \delta t \omega\}$

**lemma** kinetics\_empty: kinetics motion\_acc $i \delta t \omega = \{ \} \iff \text{motion\_acc} i \delta t \omega = \{ \}$

by (auto simp add: kinetics_def)
2 The CSP-based Generic Autonomous Car Model

**Lemma** \(\mu_{\text{kinetics\_refine}}\):

- **Assumes**:
  - \(a : \forall \delta t\, i.\ (i \in \text{sid} \land \text{accs}_2\, i \delta t\, \omega \neq \{\}) \rightarrow (\text{accs}_1\, i \delta t\, \omega \neq \{\})\)
  - \(b : \forall \delta t\, i.\ i \in \text{sid} \rightarrow \text{accs}_1\, i \delta t\, \omega \subseteq \text{accs}_2\, i \delta t\, \omega\)
- **Shows**:
  - \([\text{sid\_kinetics\,accs}_2] \subseteq [\text{sid\_kinetics\,accs}_1]\)
- Then the corresponding kinetics generate refining processes

**Proof**

- **Apply** \(\text{intros\ all\ imp\ }\mu_{\text{motion\_refine}}\)
- **Using** \(a\, b\) by (auto simp add: \text{kinetics\_def}) blast

---

**Theorem** \(\mu_{\text{same\_acc\_is\_safe}}\):

- **Fixes** \(\text{accs} : ('v::\text{real\_normed\_vector})\) \(\text{motion}\_\text{acc}\)
- **Assumes**:
  - \(a : \Delta t > 0\)
  - \(b : \text{no}\_\text{acc} \equiv \lambda i\, \delta t\, \omega.\ \{0\}\)
  - \(c : \text{no}\_\text{collision}\ \omega_0\ (0..<n) \in \langle \forall i<n.\ \text{speed}\ (\omega_0\, i) = V\rangle\)
- **Shows**:
  - \(\text{is}\_\text{safe}\ (0..<n)\ (\text{kinetics}\ \text{no}\_\text{acc})\ \Delta t\ \omega_0 \in\)

**Proof**

- **Define** \(P_{\text{inv}}\) : (\(\text{nat} \Rightarrow ('v::\text{real\_normed\_vector})\) \as \Rightarrow \text{nat\ set} \Rightarrow \text{bool}\)
- **Where** \(P_{\text{inv}} \equiv \lambda \omega\, \text{sid} .\ \text{no}\_\text{collision}\ \omega\ \text{sid} \in \langle \forall i\in\text{sid}.\ \text{(speed)}::'v\ as\Rightarrow 'v\ (\omega\, i) = V\rangle\)
- **Show** \(\text{??}\) **Thesis**

**Apply** \(\text{rule}\ \text{is}\_\text{safe}\_\text{rule}\_\text{inv}[\text{where}\ \ P_{\text{inv}}=P_{\text{inv}}],\ \text{simp\ all\ add\;assms\ }P_{\text{inv}}\_\text{def}\ \text{kinetics}\_\text{def}\)

**Using** \(c\) \(\text{no}\_\text{collision}\_\text{def}\) by (metis \(\text{add}\_\text{diff}\_\text{cancel}\_\text{right}\ \text{as}\ \text{select\_convs}(1))

**Qed**

---

**Theorem** \(\mu_{\text{same\_acc\_is\_safe}}\):

- **Fixes** \(\text{accs} : ('v::\text{real\_normed\_vector})\) \(\text{motion}\_\text{acc}\) and \(A : 'v\)
- **Assumes**:
  - \(a : \Delta t > 0\)
  - \(b : \langle \Delta t\, \omega.\ \exists a.\ \forall i.\ \text{same}\_\text{acc}\, i\ \delta t\, \omega = \{a\}\)
  - \(c : \text{no}\_\text{collision}\ \omega_0\ (0..<n) \in \langle \forall i<n.\ \text{speed}\ (\omega_0\, i) = V\rangle\)
- **Shows**:
  - \(\text{is}\_\text{safe}\ (0..<n)\ (\text{kinetics}\ \text{same}\_\text{acc})\ \Delta t\ \omega_0 \in\)

**Proof**

- **Define** \(P_{\text{inv}}\) : (\(\text{nat} \Rightarrow ('v::\text{real\_normed\_vector})\) \as \Rightarrow \text{nat\ set} \Rightarrow \text{bool}\)
- **Where** \(P_{\text{inv}} \equiv \lambda \omega\, \text{sid} .\ \text{no}\_\text{collision}\ \omega\ \text{sid} \in \langle \forall i\in\text{sid}.\ \text{(speed)}::'v\ as\Rightarrow 'v\ (\omega\, i) = V\rangle\)
- **Show** \(\text{??}\) **Thesis**

**Proof** \(\text{rule}\ \text{is}\_\text{safe}\_\text{rule}\_\text{inv}[\text{where}\ \ P_{\text{inv}}=P_{\text{inv}},\ \text{simp\ all\ add\;assms\ }P_{\text{inv}}\_\text{def},\ \text{goal\_cases}\]

**Case** \((1\ \omega\ \delta t)\)

- **Then show** \(\text{??}\) **Case** using \(c\) by (metis \(b\ \text{empty\_not\_insert}\ \text{kinetics\_empty})

**Next**

**Case** \((2\ \omega\ \delta t)\)

- **Then show** \(\text{??}\) **Case** unfolding \(\text{no}\_\text{collision}\_\text{def}\ \text{kinetics\_def}\)

**By** (smt \(\text{z3}\)) \(\text{add}\_\text{diff}\_\text{cancel}\_\text{right}\ \text{as}\ \text{select\_convs}(1)\) \(\text{as}\ \text{select\_convs}(2)\) atLeastLessThan_iff

**Qed**
2.5 CSP Model I

qed

end
3 From MOSAR-Ontologies to a Semantics in CSP

theory MOSAR_Encodings
imports CSP_AutoCars
begin

3.1 More Examples: Expressing the MOSAR hierarchy of Actor States

term acc_range (X::('v::real_normed_vector,'b) as_range_)

record ('v::real_normed_vector) Usual_Road_User_ = 'v as_range_ +
   speed_max :: real
   passengers :: nat

type_synonym ('v,'a) Usual_Road_User_ = ('v,'a) Usual_Road_User_ scheme

record ('v::real_normed_vector) PassengerCar_ = 'v Usual_Road_User_ +
   xxx :: string

type_synonym ('v,'a) PassengerCar_ = ('v,'a) PassengerCar_ scheme

record ('v::real_normed_vector) Bike_ = 'v Usual_Road_User_ +
   yyy :: string

type_synonym ('v,'a) Bike_ = ('v,'a) Bike_ scheme

record ('v::real_normed_vector) Truck_ = 'v Usual_Road_User_ +
   zzz :: string

type_synonym ('v,'a) Truck_ = ('v,'a) Truck_ scheme

The moving object types can be hierarchically extended — space for "unknown" features and its properties.
3 From MOSAR-Ontologies to a Semantics in CSP

3.1.1 Examples

One-dimensional \textit{int} as.

\textsf{term} person = ( \mid pos = 14, 
speed = 2, 
acc = 1, 
acc\_range = \{-2, -1, 0, \}, 
extension\_field = \{1, 0, -1\} \}

Two-dimensional \textit{(rat $\times$ rat)} as.

\textsf{term} (1.0, 1.0) :: ((rat $\times$ rat))

\textsf{term} racing\_car = ( \mid pos = (140.0, 210.0) :: rat$\times$rat, 
speed = (200.0, 0.0), 
acc = (10.0, 0.0), 
acc\_range = \{(10.0, 0.0), \text{ acceleration ahead} \} 
\{2.0, 3.0\}, \text{ acceleration to left} \) 
\{2.0, -3.0\}, \text{ acceleration to right} \) 
\{-40.0, 0.0\}, \text{ halfbreak} \) 
\{-80.0, 0.0\}, \text{ fullbreak} 
extension\_field = \{(x, y). x*x + y*y \leq 4.0\} \}

\textbf{definition} collision :: (\textit{\textit{v}} :: real\_normed\_vector, 'a) as\_range
\Rightarrow (\textit{\textit{v}}, 'a) as\_range
\Rightarrow bool (\textit{infix 70})
\textbf{where} emo\_1 emo\_2 \equiv (\forall e_1 \in (extension\_field emo\_1), 
\forall e_2 \in (extension\_field emo\_2).
\((\text{pos emo}_1 + e_1 \neq (\text{pos emo}_2) + e_2 ))

\textbf{definition} dynamics :: real
\Rightarrow ((\textit{\textit{v}} :: real\_normed\_vector), 'a) as\_range
\Rightarrow ('v, 'a) as\_range\_set
\textbf{where} dynamics \delta\, \sigma \equiv \{\sigma', \exists\, \text{acc} \in \text{acc}\_range\, \sigma.
\text{let speed}' = \text{speed} \, \sigma + (\delta\, \sigma \, \text{acc})
in \, \sigma' = \sigma(\text{speed} ::= \text{speed}', 
pos ::= pos\, \sigma + \)
(0.5 \, \delta\, \sigma \, \text{acc} \, \text{speed} + \text{speed}')\}

\textbf{definition} pos\_speed :: ('\textit{\textit{v}} :: real\_normed\_vector, 'a) Usual\_Road\_User_\sigma\_set
\textbf{where} pos\_speed \equiv \{\sigma. 0 \leq \text{norm(speed}\, \sigma) \wedge \text{norm(speed}\, \sigma) \leq \text{speed\_max}\, \sigma \}
3.2 Static Environment : The Street Topology

datatype lane_type = sidewalkL | boardwalkzoneL
| laneL1 | midlane12 | laneL2
| boardwalkzoneR | sidewalkR

type synonym \( \forall \) topology = lane_type \( \Rightarrow \) \( \forall \) set

3.2.1 Linear Lane Example

definition linear_lane = undefined (laneL1 := {0 .. 100:int})

3.3 CSP Model II

In this section, we present a more refined demon which makes weaker assumptions on syn-
chronisation over the global scene and allows communication of infrastructure elements such
as signals and communication between actors.

3.3.1 Generalized Events

Moving objects signal their status as moving object

datatype \((\forall,\alpha,\beta)\) events = up\(_{local}\) (id : id\(_{actor}\)

| get_status : \((\forall,\alpha)\) as\(\_range\) \(\langle\_,\_\rangle\)
| sync
| world_state : \((\forall,\alpha,\beta)\)scene\(\_o\)
| time_lapse (time : real)

term \(?\) (y,y) \(\to\) \(d\ x \to P = Q\)

definition status\(_c\) = case_prod up\(_\_local\) --- necessary for uncurried notation in read commands

term status\(_c\)? (ide,\sigma) \(\to\) \(X((ds : id\(_{actor}\) \(\to\) \((\forall,\alpha)\) as\(_\_range\) \(\langle\_\_\rangle\))

3.3.2 Maxwells Daemon II

In classical physics, Maxwell’s demon is a concept around thought-experiments that postulate
the existence of a kind of actor that "knows" the global state of all physical objects at a
given point in time. Tongue-in-cheek, we will postulate this demon here as well (not diving
in the discussion about the possibility of its existence in thermo-dynamics); one can see it
as an "orchestrator" or "mediator" in our modeling framework.

definition demon :: real \(\Rightarrow\) \((id\(_{actor}\) \(\Rightarrow\) \((\forall::\_real\_\_normed\_\_vector,\alpha)\) as\(_\_range\)\) \(\Rightarrow\) \((\forall,\alpha,\beta)\)

events process

where demon \(\Delta t\) \(\equiv\) \(\mu \ X. (\lambda ds. \cap \Delta t \in \{\_time_lapse x | x. 0 < x \wedge x \leq \Delta t\}

| (\mu X. (\lambda ds. status\(_c\)? (ide,\sigma) \(\to\) \(X (ds(ide := \sigma))))) ds
| \(\cap X ds\) --- inner loop for reads; not necessarily df

term \(\text{info} \sigma_G \to P\)
3 From MOSAR-Ontologies to a Semantics in CSP

3.3.3 Behavior of Actors

\[ \mu_{as} : (v, \alpha, \beta) \xrightarrow{\text{scene}} (v, \alpha) \xrightarrow{\text{as\_range\_set}} \text{nat} \xrightarrow{(v, \alpha)} \text{as\_range} \]

\[ \begin{align*}
\text{where } & \mu_{as} \xrightarrow{\text{driving\_strategy ide}} \\
& \equiv \mu \xrightarrow{\lambda \alpha. \Delta t \in \{\text{time\_lapse} x | 0 \leq x\}} \\
& \xrightarrow{\text{sync?}\sigma_G} \\
& \xrightarrow{(\alpha' \in (\text{up\_local\_ide}) \cap \text{dynamics}(\Delta t \alpha)} \\
& \xrightarrow{(X \text{get\_status as'})}} \\
\end{align*} \]

\[ \text{syntax } asbeh : (v, \alpha) \xrightarrow{\text{as\_set}} (v, \alpha) \xrightarrow{\text{as\_set}} \text{nat} \xrightarrow{(v \cdot \text{real\_normed\_vector}, \alpha, \beta)} \text{events process} \]

\[ \text{translations } [ A | A ]_n \xrightarrow{(\text{CONST } \mu_{as})} A \]

3.3.4 Examples

\[ \text{definition rear\_car} \equiv 0: \text{id\_actor} \]

\[ \text{definition front\_car} \equiv 1: \text{id\_actor} \]

\[ \text{definition mt \_x} \equiv \text{undefined} \]

\[ \text{definition racing\_car} \equiv \text{undefined} \]

\[ \text{definition on\_lane1} \equiv \text{undefined} \]

\[ \text{definition ego\_id} \equiv (0: \text{id\_actor}) \]

\[ \text{definition rear\_Car} \equiv [\text{racing\_car} | (\lambda \text{pos\_speed} \cap \text{on\_lane1})]_\text{rear\_car} \]

\[ \text{definition front\_Car} \equiv [\text{racing\_car} | (\lambda \text{pos\_speed} \cap \text{on\_lane1})]_\text{front\_car} \]

\[ \text{term } \text{dynamic\_system} = (\text{rear\_Car} [\text{range time\_lapse}] \text{front\_Car} [\text{range time\_lapse}] \text{demon 0.5 A}) \]

Obviously, this is not necessarily deadlock free. But since we are only interested in traces, I think this as acceptable. Anyway, the driving strategy "K" will always produce deadlocks, for example.

3.3.5 Scenario Examples

\[ \text{definition disk z} = \{(x, y) | x^2 + y^2 \leq z\} \]

\[ \text{term fun\_upd} \]

\[ \text{term } [\text{id} : \text{nat} := x] \]

\[ \text{term } \text{sync } \{\text{global\_state} = \text{mt} (\text{rear\_car} := \text{pos} = (0, 0, 0), \text{speed} = (33.3, 0), \text{acc} = (1, 0, 0), \text{acc\_range} = \{(1, 0, 0), (0, 1, 0), (0, -1, 0)\}) \]
\[ \begin{align*} \text{extension_field} &\equiv \text{disk} 4.0 \cup \\
\text{front} \text{car} &\equiv \{ \text{pos} = (25.0, -3.0) : \text{real} \times \text{real}, \\
\text{speed} &\equiv (33.3, 0.0), \quad -120 \text{ km/h} \\
\text{acc} &\equiv (0.0, 0.0), \\
\text{acc_range} &\equiv \{ (1.0, 0.0), (0.0, 1.0), (0.0, -1.0), \\
&\quad (10.0, 0.0), (-20.0, 0.0) \}, \\
\text{extension_field} &\equiv \text{disk} 4.0 \cup \}} \\
\text{time_lapse} 0.5, \quad -0.5 \text{ s later} \\
\text{sync} \{ \text{global_state} = \{ \text{mt: rear} \text{car} \equiv \{ \text{pos} = (16.7, 0.0) : \text{real} \times \text{real}, \\
\text{speed} &\equiv (33.3, 0.0), \quad -120 \text{ km/h} \\
\text{acc} &\equiv (0.0, 0.0), \\
\text{acc_range} &\equiv \{ (1.0, 0.0), (0.0, 1.0), (0.0, -1.0), \\
&\quad (10.0, 0.0), (-20.0, 0.0) \}, \\
\text{extension_field} &\equiv \text{disk} 4.0 \cup \}} \\
\text{front} \text{car} &\equiv \{ \text{pos} = (50.0, -2.0) : \text{real} \times \text{real}, \\
\text{speed} &\equiv (50.0, 0.0), \quad -120 \text{ km/h} \\
\text{acc} &\equiv (0.0, 1.0), \quad \text{move right} \\
\text{acc_range} &\equiv \{ (1.0, 0.0), (0.0, 1.0), (0.0, -1.0), \\
&\quad (10.0, 0.0), (-20.0, 0.0) \}, \\
\text{extension_field} &\equiv \text{disk} 4.0 \cup \}} \\
\text{time_lapse} 0.5, \quad -0.5 \text{ s later} \\
\text{sync} \{ \text{global_state} = \{ \text{mt: rear} \text{car} \equiv \{ \text{pos} = (33.3, 0.0) : \text{real} \times \text{real}, \\
\text{speed} &\equiv (33.3, 0.0), \quad -120 \text{ km/h} \\
\text{acc} &\equiv (0.0, 0.0), \\
\text{acc_range} &\equiv \{ (1.0, 0.0), (0.0, 1.0), (0.0, -1.0), \\
&\quad (10.0, 0.0), (-20.0, 0.0) \}, \\
\text{extension_field} &\equiv \text{disk} 4.0 \cup \}} \\
\text{front} \text{car} &\equiv \{ \text{pos} = (66.7, -2.0) : \text{real} \times \text{real}, \\
\text{speed} &\equiv (33.3, 0.0), \quad -120 \text{ km/h} \\
\text{acc} &\equiv (0.0, 1.0), \quad \text{move right} \\
\text{acc_range} &\equiv \{ (10.0, 0.0), (2.0, 3.0), (2.0, -3.0), \\
&\quad (10.0, 0.0), (-20.0, 0.0) \}, \\
\text{extension_field} &\equiv \text{disk} 4.0 \cup \}} \\
\text{time_lapse} 0.5, \quad -0.5 \text{ s later} \\
\text{sync} \{ \text{global_state} = \{ \text{mt: rear} \text{car} \equiv \{ \text{pos} = (50.0, 0.0) : \text{real} \times \text{real}, \\
\text{speed} &\equiv (33.3, 0.0), \quad -120 \text{ km/h} \\
\text{acc} &\equiv (0.0, 0.0), \\
\text{acc_range} &\equiv \{ (10.0, 0.0), (2.0, 3.0), (0.0, -3.0), \\
&\quad (10.0, 0.0), (-20.0, 0.0) \}, \\
\text{extension_field} &\equiv \text{disk} 4.0 \cup \}} \\
\text{front} \text{car} &\equiv \{ \text{pos} = (66.7, -2.0) : \text{real} \times \text{real}, \\
\text{speed} &\equiv (33.3, 0.0), \quad -120 \text{ km/h} \\
\text{acc} &\equiv (0.0, 1.0), \quad \text{move right} \\
\text{acc_range} &\equiv \{ (10.0, 0.0), (2.0, 3.0), (2.0, -3.0), \\
&\quad (10.0, 0.0), (-20.0, 0.0) \}, \\
\text{extension_field} &\equiv \text{disk} 4.0 \cup \}} \]
Extension field = disk 4.0 |]]

term map ev

[sync {global_state = (mt(rear_car := {pos = (0.0,0.0):real×real,
  speed = (33.3,0.0), — 120 km/h
  acc = (1.0,0.0),
  acc_range = {(1.0,0.0), (0.0,1.0), (0.0,-1.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |],
  front_car := {pos = (25.0,0.0):real×real,
  speed = (33.3,0.0), — 120 km/h
  acc = (0.0,0.0),
  acc_range = {(1.0,0.0), (0.0,1.0), (0.0,-1.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |)}]}

time_lapse 0.5, — 0.5 s later

sync {global_state = (mt(rear_car := {pos = (0.0,0.0):real×real,
  speed = (33.3,0.0), — 120 km/h
  acc = (0.0,0.0),
  acc_range = {(10.0,0.0), (2.0,3.0), (2.0,-3.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |],
  front_car := {pos = (50.0,0.0):real×real,
  speed = (50.0,0.0), — 120 km/h
  acc = (0.0,1.0), — move right
  acc_range = {(1.0,0.0), (0.0,1.0), (0.0,-1.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |)}]}

time_lapse 0.5, — 0.5 s later

sync {global_state = (mt(rear_car := {pos = (33.3,0.0):real×real,
  speed = (33.3,0.0), — 120 km/h
  acc = (0.0,0.0),
  acc_range = {(1.0,0.0), (0.0,1.0), (0.0,-1.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |],
  front_car := {pos = (66.7,0.0):real×real,
  speed = (33.3,0.0), — 120 km/h
  acc = (0.0,1.0), — move right
  acc_range = {(10.0,0.0), (2.0,3.0), (2.0,-3.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |)}]}

time_lapse 0.5, — 0.5 s later

sync {global_state = (mt(rear_car := {pos = (33.3,0.0):real×real,
  speed = (33.3,0.0), — 120 km/h
  acc = (0.0,0.0),
  pos = (0.0,0.0),
  acc = (0.0,0.0),
  acc_range = {(1.0,0.0), (0.0,1.0), (0.0,-1.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |),
  front_car := {pos = (0.0,0.0):real×real,
  speed = (0.0,0.0), — 120 km/h
  acc = (0.0,0.0),
  acc_range = {(10.0,0.0), (2.0,3.0), (2.0,-3.0),
    (-10.0,0.0), (-20.0,0.0)},
  extension_field = disk 4.0 |)}]}

3 From MOSAR-Ontologies to a Semantics in CSP

extension_field = disk 4.0 |]]
3.3 CSP Model II

\[ \text{acc\_range} = \{(10.0,0.0), (2.0,3.0), (2.0,-3.0), \]
\[ (-10.0,0.0), (-20.0,0.0)\}, \]
\[ \text{extension\_field} = \text{disk 4.0}\}, \]

\[ \text{front\_car} := \{ \text{pos} = (66.7,-2.0):\text{real}\times\text{real}, \]
\[ \text{speed} = (33.3,0.0), \quad \text{— 120 km/h} \]
\[ \text{acc} = (0.0,1.0), \quad \text{— move right} \]
\[ \text{acc\_range} = \{(10.0,0.0), (2.0,3.0), (2.0,-3.0), \]
\[ (-10.0,0.0), (-20.0,0.0)\}, \]
\[ \text{extension\_field} = \text{disk 4.0}\} \} \]

\[ \in \mathcal{T} \quad (\text{demon 0.5 Actors [S] front\_car}_p, [S] \text{ rear\_car}_p) \]

\text{end}
4 A Safety-Property in Autonomous Cars:

RSS in a 2-cars-Scenario

theory RSS_2cars
  imports CSP_AutoCars
begin

The instance of the framework for a 2 cars scenario: One-lane topology, topology is identical
to simple vector-space real, no backwards-driving. Simple demon.

This section discusses a 2-cars scenario where both cars respect the RSS strategy. We
refer to the definitions for RSS in [9].

abbreviation rear_id :: nat where rear_id ≡ 0
abbreviation front_id :: nat where front_id ≡ 1

4.1 Global Parameters of the Scenario-Class

locale RSS_2cars_samelane_samedirection = — 2 cars, one lane, one direction

fixes a_max_acc :: real and a_min_brake :: real and a_max_brake :: real and ϒ :: real
fixes d_real :: real scene ⇒ real — signed distance between the two vehicles
and d_safe :: real scene ⇒ real — safe distance between the two vehicles
fixes rss_motion :: real motion_acc and kinetics_RSS :: real motion_acc ⇒ real motion:

assumes a_0 : 0 < a_max_acc
and a_1 : 0 < a_min_brake
and a_2 : a_min_brake ≤ a_max_brake
and a_3 : ϒ > 0

— safe longitudinal distance as defined by RSS, in section 3.1

defines d_0 : d_safe σ_g ≡ max 0 (ϒ * speed (σ_g rear_id) + ((ϒ^2)/2) * a_max_acc +
  (speed (σ_g rear_id) + ϒ * a_max_acc)^2 / (2 * a_min_brake) −
  (speed (σ_g front_id)^2 / (2 * a_max_brake)));

— To improve: use current acc (σ_g rear_id) is more precise than a_max_acc

and d_1 : d_real σ_g ≡ pos(σ_g front_id) − pos(σ_g rear_id))

— signed distance between the two vehicles

assumes r_0 : rss_motion 0 δ t σ_g ≡ (.. if d_safe σ_g > d_real σ_g then −a_min_brake else a_max_acc)
  — possible accelerations for v_0 or c_r (rear car)
  — le pb est qu’ici on ne tient pas compte du délai de réaction : la vitesse connue au moment
de la décision d’ego est la vitesse à t−ϒ ??

and r_1 : rss_motion 1 δ t σ_g ≡ (−a_max_brake ..)
  — possible accelerations for v_1 or c_f (front car)
4 A Safety-Property in Autonomous Cars: RSS in a 2-cars-Scenario

— RSS kinetics are not backward

defines k0::kineticsRSS motion_acc ≡
λ i δt σ_g. { if speed (σ_g i) + (δt * a) > 0
then (pos (σ_g i) + (δt) * speed (σ_g i)) + ((δt^2)/2) * a,
speed = speed (σ_g i) + (δt * a),
acc = a )
else (pos (σ_g i) - (speed (σ_g i))^2 / (2 * a),
speed = 0,
acc = 0 )
— the −(speed (σ_g i))^2 / (2 * a) comes from the newtonian kinetics formula
with t_i=−v/a, i.e. the time when v = v_0 + a * t_i = 0 (the vehicle stops at t_i∈[t, t+δt])
| a. a ∈ motion_acc i δt σ_g |}

begin

4.2 Relations between Driving Strategies

lemma kineticsRSS_empty: (kineticsRSS motion_acc i δt σ_g = { } ↔ motion_acc i δt σ_g = { })
by (auto simp add: k0)

difference between the min braking distance of c_f and the max braking distance of c_r

definition d_min ≡ λ (σ_g::real scene). max 0 ((speed (σ_g rear_id))^2 / (2 * a_minbrake) − (speed (σ_g front_id))^2 / (2 * a_maxbrake))

declare minus_divide_le_eq[ simp] not_less[ simp]

theorem rss_is_safe_2cars:
assumes a: d_min σ_g0 < d_real σ_g0 ∧ 0 ≤ speed (σ_g0 0) ∧ 0 ≤ speed (σ_g0 1)
and b: 0 < Δt ∧ Δt < v — et ici, pourquoi inégalité stricte??
shows (is_safe {0, 1} (kineticsRSS rss_motion) Δt σ_g0)

proof —
define P_inv where P_inv = λ (σ_g::real scene). 0 ≤ speed (σ_g rear_id) ∧ 0 ≤ speed (σ_g front_id)
∧ d_min σ_g < d_real σ_g;
— invariant: the distance between the two cars is d_min

show ?thesis

proof(rule is_safe_rule_inv [where P_inv = λ σ_g sid. P_inv σ_g])
— use is_safe_rule_inv with P_inv defined above
show 0 < Δt: using b by simp

next

show (P_inv σ_g0)
using a by (auto simp add: P_inv_def d0)
next

fix σ_g δt

show λ P Inv σ_g. ∀ i∈{0, 1}. kineticsRSS rss_motion i δt σ_g ≠ {})
using 0 a1 a2 r0 r1 P_inv_def by (auto simp: kineticsRSS_empty)

next

34
4.2 Relations between Driving Strategies

\[ \text{fix } \sigma_g \Delta t \]
\[ \text{show } P_{\text{inv}} \sigma_g \wedge \Delta t \in \{0, \Delta t\} \implies \forall \sigma_g' \in \{\sigma_g, \forall i \in \{0, 1\}, \sigma_g' \in \text{kinetics}_{RSS} \text{ rss motion } i \Delta t \sigma_g \} \]

\[ \text{proof } \text{(auto simp: } P_{\text{inv}}\_\text{def}) \]
\[ \text{fix } \sigma_g' \]
\[ \text{assume } \sigma_g' (0::nat) \in \text{kinetics}_{RSS} \text{ rss motion } 0 \Delta t \sigma_g \text{ and } (0 < \Delta t) \]
\[ \text{then show } (\text{speed } (\sigma_g' \text{ rear}_i)) \text{ and } (\text{speed } (\sigma_g' \text{ front}_i)) \]
\[ \text{by } \text{(auto simp:k0 r0 mult.commute)} \]

\[ \text{next } \text{fix } \sigma_g' \]
\[ \text{assume } \sigma_g' (1::nat) \in \text{kinetics}_{RSS} \text{ rss motion } 1 \Delta t \sigma_g \text{ and } (0 < \Delta t) \]
\[ \text{then show } (\text{speed } (\sigma_g' \text{ front}_i)) \]
\[ \text{by } \text{(auto simp:k0 r1 mult.commute)} \]

\[ \text{next } \text{fix } \sigma_g' \]
\[ \text{assume } \text{as1: } 0 \leq \text{speed } (\sigma_g \text{ rear}_i) \quad \text{and as2: } 0 \leq \text{speed } (\sigma_g \text{ front}_i) \]
\[ \text{and as3: } d_{\text{min}} \sigma_g < d_{\text{real}} \sigma_g \quad \text{and as4: } 0 < \Delta t \]
\[ \text{and as5: } \Delta t \leq \Delta t \quad \text{and as6: } \sigma_g' (0::nat) \in \text{kinetics}_{RSS} \text{ rss motion } 0 \Delta t \sigma_g \]
\[ \text{and as7: } \sigma_g' \text{ front}_i \in \text{kinetics}_{RSS} \text{ rss motion } 1 \Delta t \sigma_g \]

\[ \text{let } ?a_0 = \text{acc}(\sigma_g' \text{ rear}_i) \text{ and } ?a_1 = \text{acc}(\sigma_g' \text{ front}_i) \]
\[ \text{and } ?b_0 = (\text{speed } (\sigma_g \text{ rear}_i))^2 / (2 \times a_{\text{minbrake}}) \text{ and } ?b_1 = (\text{speed } (\sigma_g \text{ front}_i))^2 / (2 \times a_{\text{minbrake}}) \]
\[ \text{and } ?b_0' = (\text{speed } (\sigma_g' \text{ rear}_i))^2 / (2 \times a_{\text{minbrake}}) \text{ and } ?b_1' = (\text{speed } (\sigma_g' \text{ front}_i))^2 / (2 \times a_{\text{minbrake}}) \]

— some simplifications factorized here

\[ \text{have simp0: } 0 \leq \text{speed } (\sigma_g' \text{ rear}_i) \text{ and simp1: } 0 \leq \text{speed } (\sigma_g' \text{ front}_i) \]
\[ \text{using as6 as7 by } \text{(auto simp:k0 r0 r1 mult.commute)} \]

\[ \text{have simp2: speed } (\sigma_g \text{ rear}_i) / 2 + ?a_0 \Delta t / 2 \geq 0 \text{ and simp3: speed } (\sigma_g \text{ front}_i) / 2 + ?a_1 \Delta t / 2 \geq 0 \]
\[ \text{using as6 as7 by } \text{(auto simp add:k0 r0 r1 as4 as2 as1 mult.commute)} \]

\[ \text{have simp4: speed } (\sigma_g \text{ rear}_i) + ?a_0 \Delta t / 2 \geq 0 \text{ and simp5: speed } (\sigma_g \text{ front}_i) + ?a_1 \Delta t / 2 \geq 0 \]
\[ \text{using as1 simp2 apply linarith} \]
\[ \text{using as2 simp3 by linarith} \]

\[ \text{have simp6: } \Delta t \times \text{speed } (\sigma_g \text{ rear}_i) + ?a_0 \Delta t / 2 \geq 0 \text{ and simp7: } \Delta t \times \text{speed } (\sigma_g \text{ front}_i) + ?a_1 \Delta t / 2 \geq 0 \]
\[ \text{using as4 simp4 simp5} \]
\[ \text{by } \text{(metis no_types, hide_lams distrib_left less_eq real_def mult.assoc mult.commute mult_nonneg_nonneg power2_eq_square times_divide_eq_right)} \]

35
have simp4p: speed (\(\sigma_g\) \(\text{rear}_{id}\)) + ?a0*\(\delta t\) / 2 \(\geq\) speed (\(\sigma_g\) \(\text{rear}_{id}\)) / 2;
and simp5p: speed (\(\sigma_g\) \(\text{front}_{id}\)) + ?a1*\(\delta t\) / 2 \(\geq\) speed (\(\sigma_g\) \(\text{front}_{id}\)) / 2;
using as1 simp2 apply linarith
using as2 simp3 by linarith

have simp6p: \(\delta t\) * speed (\(\sigma_g\) \(\text{rear}_{id}\)) + ?a0*\(\delta t^2\) / 2 \(\geq\) \(\delta t\) * speed (\(\sigma_g\) \(\text{rear}_{id}\)) / 2;
and simp7p: \(\delta t\) * speed (\(\sigma_g\) \(\text{front}_{id}\)) + ?a1*\(\delta t^2\) / 2 \(\geq\) \(\delta t\) * speed (\(\sigma_g\) \(\text{front}_{id}\)) / 2;
using as4 simp4p simp5p
by (metis distrib_left less_eq_real_def mult.left_commute mult.left_mono power2.eq_square times_divide_eq_right)

have simp8p: speed (\(\sigma_g\) \(\text{rear}_{id}\)) + a * \(\delta t\) \(\leq\) 0 \(\implies\) − (speed (\(\sigma_g\) \(\text{rear}_{id}\)))^2 / (a * 2) \(\leq\) (\(\delta t\) * speed (\(\sigma_g\) \(\text{rear}_{id}\))) / 2;
using as1 mult.left_mono [where \(b=\delta t\) and \(a=(−\text{speed (\(\sigma_g\) \(\text{rear}_{id}\)) / a})\) and \(c=(\text{speed (\(\sigma_g\) \(\text{rear}_{id}\))) / 2}\)]
apply (auto simp add: mult.commute power2.eq_square split:if_splits)
using as4 mult_pos_pos by force

have simp9p: speed (\(\sigma_g\) \(\text{front}_{id}\)) + a * \(\delta t\) \(\leq\) 0 \(\implies\) − (speed (\(\sigma_g\) \(\text{front}_{id}\)))^2 / (a * 2) \(\leq\) (\(\delta t\) * speed (\(\sigma_g\) \(\text{front}_{id}\))) / 2;
using as2 mult.left_mono [where \(b=\delta t\) and \(a=(−\text{speed (\(\sigma_g\) \(\text{front}_{id}\)) / a})\) and \(c=(\text{speed (\(\sigma_g\) \(\text{front}_{id}\))) / 2}\)]
apply (auto simp add: mult.commute power2.eq_square split:if_splits)
using as4 mult_pos_pos by force

— safety distance proofs
{ assume speed (\(\sigma_g\)' \(\text{front}_{id}\)) = 0
then have \(\max 0 \ ?b_1 \leq \text{pos (\(\sigma_g\)' \(\text{front}_{id}\))} − \text{pos (\(\sigma_g\) \(\text{front}_{id}\))}\)
using as7 as2 as4 apply (auto simp add:k0 r1 mult.commute divide_le_0_iff split:if_splits)

  apply (smt (z3) zero_less_mult_iff)
  using divide_left_mono [where c=(\(\text{speed (\(\sigma_g\) \(\text{front}_{id}\)))^2}\) and \(a=(a_{\text{max\_brake}} * 2)\)]
  by (smt (z3) divide_cancel_left minus_divide_right mult_minus_left zero_eq_power2 zero_le_mult_iff zero_le_power)
} note dmin_front_car_stopping= this

{ assume \(h_0\): speed (\(\sigma_g\)' \(\text{front}_{id}\)) > 0
d from as7 have \(h_1: = \ ?\tilde{a}_1) / a_{\text{max\_brake}} \leq 1\)
by (auto simp add:k0 r1) (smt (z3) a1 a2 divide_le_eq_1_neg minus_divide_right)
have h2: (speed (\(\sigma_g\) \(\text{front}_{id}\)) + \(\delta t\) * \(\tilde{a}_1\))^2 / (2 * \(a_{\text{max\_brake}}\))
= (speed (\(\sigma_g\) \(\text{front}_{id}\))^2 / (2 * \(a_{\text{max\_brake}}\)) + (?\tilde{a}_1 / a_{\text{max\_brake}}) * (\(\delta t\) * speed (\(\sigma_g\) \(\text{front}_{id}\)) + ?\tilde{a}_1 * \(\delta t^2\) / 2)),
by (simp add: power2.sum distrib_left mult.commute power2.eq_square add_divide_distrib mult.left_commute)
have 0 \(\leq\) pos (\(\sigma_g\)' \(\text{front}_{id}\)) − pos (\(\sigma_g\) \(\text{front}_{id}\))
using as7 h0 simp7 by (auto simp add:k0 r1 mult.commute split:if_splits)
then have \(\max 0( \ ?b_1 \leq \text{pos (\(\sigma_g\)' \(\text{front}_{id}\))} − \text{pos (\(\sigma_g\) \(\text{front}_{id}\))}\)
using as7 h0 mult.left_mono [OF h1 simp7]\ h2 a1 a2
4.2 Relations between Driving Strategies

by (auto simp add:k0 r1 mult.commute split:if_splits)

have front_car:max 0 (?b1 - ?b'1) ≤ \text{pos}(\sigma_g' \text{ front}_id) - \text{pos}(\sigma_g \text{ front}_id):
using \text{dmin_front_car_motion}\text{dmin_front_car_stopping simp1 by fastforce}

\{ assume h0:dsafe \sigma_g > d_{\text{real}} \sigma_g \text{ and h1:}\text{speed}(\sigma_g' \text{ rear}_id) > 0 \:
have h2:a \geq \frac{\sigma_g'}{\text{a}_{\text{minbrake}}} \Rightarrow a / \text{a}_{\text{minbrake}} \leq -1 \text{ for a }
by (metis a1 divide_eq_minus_1_iff divide_le_cancel less_le not_le)
have h4:pos(\sigma_g' \text{ rear}_id) - pos(\sigma_g \text{ rear}_id) \leq (\text{speed}(\sigma_g \text{ rear}_id))^2 / (2 \times \text{a}_{\text{minbrake}}):
using as6 a1 a2 h0 h1 apply(auto simp add:k0 r0 power2_sum add_divide_distrib distrib_left mult.commute split:if_splits)

(smt \text{verit} \text{mult.assoc mult.commute power2_eq_square)
then have (?b'0 - ?b_0) \leq \text{pos}(\sigma_g \text{ rear}_id) - \text{pos}(\sigma_g' \text{ rear}_id):
by force
\}

\} note \text{dmin_front_car_nonsafe_breaking=\text{this}}

\{ assume h0:dsafe \sigma_g > d_{\text{real}} \sigma_g \text{ and h1:}\text{speed}(\sigma_g' \text{ rear}_id) = 0:
have h2:a \leq \frac{\sigma_g' \text{ speed}(\sigma_g \text{ rear}_id)}{2} \leq \frac{\sigma_g' \text{ speed}(\sigma_g \text{ rear}_id)}{2}:
using as1 a5 b less_eq_real_def by force
have h3:pos(\sigma_g' \text{ rear}_id) - pos(\sigma_g \text{ rear}_id) \leq \sigma_g' \text{ speed}(\sigma_g \text{ rear}_id):
using as6 a1 a2 h0 h1 apply(auto simp add:k0 r0 power2_sum add_divide_distrib split:if_splits)

using mult_left_mono[OF h2] by (simp add: mult.commute)
then have (?b_0 - b_0) \leq \text{pos}(\sigma_g \text{ rear}_id) - \text{pos}(\sigma_g' \text{ rear}_id):
by argo
\}

\} note \text{rear_car_nonsafe_stopping=\text{this}}

\{ assume h0:dsafe \sigma_g > d_{\text{real}} \sigma_g \Rightarrow ?b'0 - ?b_0 \leq \text{pos}(\sigma_g \text{ rear}_id) - \text{pos}(\sigma_g' \text{ rear}_id):
using \text{dmin_front_car_nonsafe_breaking rear_car_nonsafe_stopping simp1 by fastforce}

\{ assume h0:dsafe \sigma_g \leq d_{\text{real}} \sigma_g \text{ and h1:}\text{speed}(\sigma_g' \text{ rear}_id) = 0:
have h2: (\sigma_g' \text{ speed}(\sigma_g \text{ rear}_id)) / 2 \leq \sigma_g' \text{ speed}(\sigma_g \text{ rear}_id):
using as1 as5 b less_eq_real_def by force
have h3:pos(\sigma_g' \text{ rear}_id) - pos(\sigma_g \text{ rear}_id) \leq \sigma_g' \text{ speed}(\sigma_g \text{ rear}_id):
using as6 a1 a3 a1 h0 h1 apply(auto simp add:dmin_def k0 r0 mult.commute not_le split:if_splits)

using order_trans[OF simp8 h2]
apply (smt \text{verit} \text{best} \text{mult.commute neg_divide_le_eq)
by (metis \text{add_cancel_right_right add_le_less_mono as4 mult_eq_0_iff mult_less_cancel_left_pos not_less)
have h4: (?b_0 \leq \sigma_g' \text{ speed}(\sigma_g \text{ rear}_id) + \sigma_g' \text{ speed}(\sigma_g \text{ rear}_id)) / (2 \times \text{a}_{\text{minbrake}}):
by (smt \text{verit} \text{ccfv_SIG a0 a1 a3 a1 frac_le mult_pos power mono zero_le_power2)
with h0 have h5: \sigma_g' \text{ speed}(\sigma_g \text{ rear}_id) + ?b_0 - ?b_1 < \text{pos}(\sigma_g \text{ front}_id) - \text{pos}(\sigma_g \text{ rear}_id):
by (smt \text{verit} \text{best} a0 a3 d0 d1 divide_pos_pos zero_less_mult_iff zero_less_power

37
4 A Safety-Property in Autonomous Cars: RSS in a 2-cars-Scenario

with \( h3 \) have \(?b_0 \leq ?b_1 < \text{pos}(\sigma_g \text{ front}_{id}) - \text{pos}(\sigma_g \text{ ' rear}_{id}) \):
  by fastforce
} note \text{dmin\_rear\_car\_safe\_stopping} = \text{this}

\{ assume \( h0 : d_{safe} \sigma_g \leq d_{real} \sigma_g \) and \( h1 : \text{speed}(\sigma_g \text{ ' rear}_{id}) > 0 \):
  have \( h2 : \delta t \times \text{speed}(\sigma_g \text{ rear}_{id}) \leq u \times \text{speed}(\sigma_g \text{ rear}_{id}) \):
    using as1 as4 as5 b mult_right_less_imp_less by force
  from as6 have \( h3 : ?a_0 \times \delta t^2 / 2 < \text{maxacc}\_speed^2 / 2 \):
    using a0 a1 a3 h0 apply (auto simp add: k0 r0 as4 split_if_splits)
  by (smt (verit) as4 as5 b eq_divide_imp \text{frac\_le} mult\_commute \text{pow2\_le\_imp\_le} zero_eq\_pow2 zero\_le\_multi\_iff)
  from as6 have \( h4 : ?b_0' \leq (\text{speed}(\sigma_g \text{ rear}_{id}) + u \times \text{maxacc})^2 / (2 \times \text{minbrake}) \):
    using a1 h0 apply (auto simp add: k0 r0)
  by (smt (verit, best) a0 a1 as4 as5 b divide\_le\_cancel mult\_mono\_power\_mono zero\_le\_multi\_iff)
  have \( h5 : ?b_1 \leq (\text{speed}(\sigma_g \text{ rear}_{id}) + u \times \text{maxacc})^2 / (2 \times \text{minbrake}) \):
    by (smt (verit) mult\_commute)
  with \( h0 \) have \( h6 : \delta t \times \text{speed}(\sigma_g \text{ rear}_{id}) + ?a_0 \times \delta t^2 / 2 + (\max ?b_0 ?b_0') - ?b_1 < \text{pos}(\sigma_g \text{ front}_{id}) - \text{pos}(\sigma_g \text{ ' rear}_{id}) \):
    by fastforce
} note \text{dmin\_rear\_car\_safe\_motion} = \text{this}

have \text{dmin\_rear\_car\_safe} : \( d_{safe} \sigma_g \leq d_{real} \sigma_g \rightarrow (\max ?b_0 ?b_0') - ?b_1 < \text{pos}(\sigma_g \text{ front}_{id}) - \text{pos}(\sigma_g \text{ ' rear}_{id}) \):
  by (metis as1 divide\_le\_cancel \text{less_eq \_real\_def} \text{max\_def} \text{mult\_2} \text{pow2\_less_eq\_zero\_iff}\text{power\_zero\_numeral dmin\_rear\_car\_safe\_motion dmin\_rear\_car\_safe\_stopping simp0 zero\_less\_double\_add\_iff zero\_less\_single\_add})

\{ assume \( h0 : d_{safe} \sigma_g \leq d_{real} \sigma_g \)
  then have \( 0 \leq \text{pos}(\sigma_g \text{ ' rear}_{id}) - \text{pos}(\sigma_g \text{ rear}_{id}) \):
    using \( h0 \) as1 as6 simp6 as4 apply (auto simp add: k0 r0 mult\_commute)
  by (metis add\_right\_neutral add\_mono\_thms\_linordered\_field(4) divide\_nonneg\_nonpos not\_le\_not\_numeral\_less\_zero zero\_le\_pow2 zero\_less\_multi\_iff)
} note \text{dmin\_rear\_car\_forward} = \text{this}

have \( \text{dmin} : ?b_0' > ?b_1' < d_{real} \sigma_g \):
  using as3 d1 dmin\_def \text{front\_car dmin\_rear\_car\_nonsafe dmin\_rear\_car\_safe} by fastforce
4.2 Relations between Driving Strategies

--- no collision's proofs

\[
\{ \text{assume } h0::?b_0 \leq ?b_1 \text{ and } h1::?b'_0 < ?b'_1 \} \]

--- ?b'_0 < ?b'_1, is added to profit from previous results of dmin

\[
\text{and h2::speed } (\sigma_g' \text{ } \text{rear}_{id}) = 0
\]

from h0 have h3::speed (\sigma_g' \text{ } \text{rear}_{id}) \leq speed (\sigma_g \text{ } \text{front}_{id})

using mult_right_mono \{ \text{OF } h0, \text{where } c=\langle(2 * a_{\text{min_brake}})\rangle, \text{simplified} \}

by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_eq_square real_add_le_0_iff

\text{assumption})

from h2 have h6::pos (\sigma_g' \text{ } \text{rear}_{id}) - pos (\sigma_g' \text{ } \text{rear}_{id}) \leq \delta t * speed (\sigma_g \text{ } \text{rear}_{id}) / 2

using as6 as1 apply(auto simp add:k0 r0 a1)

using simp8[simplified divide_divide_eq_left[+symmetric]]

apply (smt (verit, ccfv_threshold) mult_assoc mult_left_commute

mult_left_mono mult_minus_left neg_le_iff_le power2_eq_square real_add_le_0_iff)

by (metis add_cancel_left_right add_le_less_mono as4 mult_eq_0_iff

mult_less_cancel_left_pos not_less)

have h6::pos (\sigma_g' \text{ } \text{rear}_{id}) - pos (\sigma_g' \text{ } \text{front}_{id}) \leq pos (\sigma_g' \text{ } \text{front}_{id}) - pos (\sigma_g \text{ } \text{front}_{id})

using as7 h2 apply(auto simp add:k0 r1 a1)

by (smt (verit, best) as.select_convs(3) as4 field_sum_of_halves h3 h6

mult.commute mult_less_cancel_left_pos simp7p)

\}

note no_collision_if_slow_rear_car_stopping=--this

\{
\text{assume } h0::?b_0 \leq ?b_1 \text{ and } h1::?b'_0 < ?b'_1 \}

\text{and h2::speed } (\sigma_g' \text{ } \text{front}_{id}) > 0

from h0 have h3::speed (\sigma_g' \text{ } \text{rear}_{id}) \leq speed (\sigma_g \text{ } \text{front}_{id})

using mult_right_mono \{ \text{OF } h0, \text{where } c=\langle(2 * a_{\text{min_brake}})\rangle, \text{simplified} \}

by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_le_imp_le zero_le_power2

\text{assumption})

have h4::speed (\sigma_g' \text{ } \text{front}_{id}) > 0

by (metis a1 div_0 divide_nonneg_pos h1 mult.commute mult_2_right not_less

order_not_eq_order_implies_strict simp1 zero_le_power2

zero_less_double_add_iff_zero_less_single_add zero_power2)

from h1 h2 h3 h4 have h5::\text{speed } (\sigma_g \text{ } \text{rear}_{id}) + ?a_0 * \delta t < \text{speed } (\sigma_g \text{ } \text{front}_{id}) + ?a_1 * \delta t

using as6 as7 apply(auto simp add:k0 r0 r1)

by (smt (verit, best) a1 a2 frac_le mult.commute power_mono zero_le_power2

\text{assumption})

have h6::\delta t * speed (\sigma_g \text{ } \text{rear}_{id}) + \delta t^2 * ?a_0 < \delta t * speed (\sigma_g \text{ } \text{front}_{id}) + \delta t^2 * ?a_1

by (metis no_types, hide_lams) as4 distrib_left h5 mult.commute mult.left_commute

mult_less_cancel_left_pos power2_eq_square

with h3 have h7::\delta t * speed (\sigma_g \text{ } \text{rear}_{id}) + \delta t^2 * ?a_0 / 2 < \delta t * speed (\sigma_g \text{ } \text{front}_{id}) + \delta t^2 * ?a_1 / 2

by (smt (verit, ccfv_SIG) as4 field_sum_of_halves mult_less_cancel_left_pos

\text{assumption})

have pos (\sigma_g' \text{ } \text{rear}_{id}) - pos (\sigma_g' \text{ } \text{rear}_{id}) < pos (\sigma_g' \text{ } \text{front}_{id}) - pos (\sigma_g \text{ } \text{front}_{id})

using as6 as7 h2 h4 h7 by (auto simp add:k0 r0 r1 a1 dmin_def d1

\text{divide_nonneg_pos power2 sum not_le})

\}

note no_collision_if_slow_rear_car_nonstopping=--this

remaining case (REM):

1. non-safe mode dsafe > dreal: in the opposite case, we profit from previous results of dmin_rear_car_safe
2. faster rear car: but it becomes slower after breaking.

```plaintext
{ assume h0::?b1 < ?b0 and h1::d_safe σ_g > d реал σ_g and h2::?b′_0 < ?b′_1 and h3::speed (σ_g′ _front_id) = 0

then have False by (metis a1 divide_eq_0_iff le_divide_eq mult_eq_0_iff not_less power_zero_numeral zero_le_power2 zero_less_numeral)

} note REM_front_car_not_stopping

{ assume h0::?b1 < ?b0 and h1::d_safe σ_g > d реал σ_g and h2::speed (σ_g′ _front_id) > 0 and h3::speed(σ_g _front_id) - a_maxbrake * b ≤ 0

have h4::?b1 + pos (σ_g′ _front_id) - pos (σ_g _front_id)

using as7 h2 apply (auto simp add: k0 r1 mult.commute not_le_power2_sum add_divide_distrib split_if_splits)

by (smt (z3) as.select_conv(3) divide_minus_left h3 minusdivide_right mult_minus_left simp7p simp9)

then have 0 < pos (σ_g′ _front_id) - pos (σ_g _rear_id)

by (smt (verit, best) a1 as3 d1 d_min_def divide_nonneg_pos dmin_rear_car_nonsafe h1 zero_le_power2)

} note REM_front_car_may_stop

{ assume h0::?b1 < ?b0 and h1::d_safe σ_g > d реал σ_g and h2::speed (σ_g′ _front_id) > 0 and h3::speed(σ_g _front_id) - a_maxbrake * b > 0

have s1::a_maxbrake / a_minbrake ≥ 1 ∧ sqrt a_maxbrake ≥ sqrt a_minbrake ∧ sqrt a_minbrake > 0

using a1 a2 by simp

then have s2::a_maxbrake / a_minbrake ≥ sqrt a_maxbrake / sqrt a_minbrake

by (smt (verit, ccfv_threshold) divide_less_eq_1 real_div_sqrt real_sqrt_divide)

then have s3::a_maxbrake * sqrt a_minbrake ≥ a_minbrake * sqrt a_maxbrake

by (smt (verit, ccfv_SIG) a1 a2 divide_divide_times_eq divide_le_eq_1_pos divide_less_eq_1 mult_nonneg_nonneg real_sqrt_gt_0_iff)

have s4::sqrt a_maxbrake / sqrt a_minbrake ≥ (speed (σ_g _front_id)) / (speed (σ_g _rear_id))

using real_sqrt_le_mono OF order.strict_implies_order[OF h0]

apply(simp add:real_sqrt_divide real_sqrt_mult as1 as2 mult.commute)
apply(subst (asm) (1 2) divide_divide_eq_left|symmetric), subst(asm) divide_le_cancel, simp)

by (smt (verit, ccfv_threshold) as1 divide_less_eq_1 mult.commute pos_less_divide_eq s1 times_divide_eq_left)

have s5::speed(σ_g _rear_id) / a_minbrake ≥ speed(σ_g _front_id) / a_maxbrake

using order_trans[OF s4 s2] apply(subst (asm) pos_divide_le_eq)

apply (smt (verit, ccfv_threshold) a1 a2 as1 divide_divide_eq_0_iff divide_nonneg_pos h0 zero_le_power2 zero_power2)

apply(simp add: a1 le_divide_eq mult.commute)

by (smt (verit) a1 a2 divide_le_cancel nonzero_mult_div_cancel_left)

with h3 have s6::speed(σ_g _rear_id) - a_minbrake * b > 0

using h3 a1 a2 by (metis a1 diff_gt_0_iff_gt less_le_trans mult.commute pos_less_divide_eq)

```
4.2 Relations between Driving Strategies

have s7: (speed (σ_g front_id)) * sqrt a_minbrake \leq (speed (σ_g rear_id)) * sqrt a_maxbrake
using real_sqrt_le_mono[OF order.strict_implies_order[OF h0]]
apply(simp add:real_sqrt_divide real_sqrt_mult as1 as2 mult.commute)
apply(subst (asm) (1 2) divide_divide_eq_left[symmetric], subst(asm) divide_le_cancel, simp)
apply(subst (asm) pos_divide_le_eq)
using a1 a2 apply autol[1]
by (simp add: a1_le_divide_eq mult.commute)
have s8: (speed (σ_g front_id) - a_maxbrake * \delta t) / sqrt a_maxbrake
\leq (speed (σ_g rear_id) - a_minbrake * \delta t) / sqrt a_minbrake
apply(subst pos_divide_le_eq) using a1 a2 apply auto
apply(subst pos_le_divide_eq) apply (simp_all add: left_diff_distrib)
using s7 s3 by (metis as4 diff_mono mult.commute mult.le_cancel_left_pos)
with h3 have s9: (speed(σ_g front_id) - a_maxbrake * \delta t)^2 / (2 * a_maxbrake)
\leq (((speed(σ_g rear_id) - a_minbrake * \delta t)^2 / (2 * a_minbrake))
using abs_le_square_iff[where x = (speed (σ_g front_id) - a_maxbrake * \delta t) / sqrt a_maxbrake]
and y = (speed (σ_g rear_id) - a_minbrake * \delta t) / sqrt a_minbrake]
using s1 apply (auto simp add:power_divide)
by (smt (verit, best) divide_less_0_iff s1)
have s10: \delta t * speed(σ_g rear_id) / 2 \leq (\delta t * speed (σ_g rear_id) - a_minbrake * \delta t^2 / 2)
by (smt (verit, best) as4 field_sum_of_halves le_divide_eq power2_eq_square
real_divide_square_eq s6 zero_less_power)
have h4: \delta t * speed(σ_g front_id) - a_maxbrake * \delta t^2 / 2 \leq pos (σ_g' front_id) - pos (σ_g front_id):
using as7 h2
apply (auto simp add:k0 r1 mult.commute not_le power2_sum add_divide_distrib
split:if_splits)
by (metis comm_semiring_class.distrib mult_nonneg_nonneg real_0_le_add_iff
zero_le_power2)
have \delta t * speed(σ_g front_id) - a_maxbrake * \delta t^2 / 2 - b1 = -(speed(σ_g front_id) - a_maxbrake
* \delta t)/ (2 * a_maxbrake):
apply(simp add:power2_diff add_divide_distrib minus_add diff_add_eq_diff_diff_swap
diff_divide_distrib)
by (metis a1 a2 mult.assoc mult.left_commute nonzero_mult_div_cancel_left not_less
power2_eq_square)
with h4 have h5: pos (σ_g' front_id) - pos (σ_g front_id) - b1 \geq -(speed(σ_g front_id) -
a_maxbrake * \delta t^2 / (2 * a_maxbrake))
by linarith
have a \leq - a_minbrake \implies (\delta t * speed (σ_g rear_id) + a \leq \delta t^2 / 2)
\leq (\delta t * speed (σ_g rear_id) - a_minbrake * \delta t^2 / 2) for a
by (smt (verit) as4 eq_divide_imp field_sum_of_halves mult_minus_left
pos_minus_divide_le_eq zero_less_power2)
then have h6: pos (σ_g' rear_id) - pos (σ_g rear_id) \leq \delta t * speed(σ_g rear_id) - a_minbrake
* \delta t^2 / 2;
4 A Safety-Property in Autonomous Cars: RSS in a 2-cars-Scenario

using as6 h1 h2 a1
apply (auto simp add:k0 r0 d0 d1 mult.commute not_le power2_sum add_divide_distrib
min_def
split:if_splits)
using simp8 s10
by (smt (verit, ccfv_SIG divide_le_cancel eq_divide_imp mult.commute)

have h7: δt\cdot\text{speed}(\sigma_g \text{rear}_{id}) - a_{\text{minbrake}} \cdot \delta t^2/2 - ?b_0 = -(\text{speed}(\sigma_g \text{rear}_{id}) - a_{\text{minbrake}} \cdot \delta t^2)/(2\cdot a_{\text{minbrake}});
apply (simp add:power2_diff add_divide_distrib minus_add diff_add_eq_diff_diff_swap
diff_divide_distrib)
by (metis a1 less_eq_real_def mult.assoc mult.left_commute nonzero_mult_div_cancel_left
not_less power2_eq_square)

with h5 h6 s9 have \langle (\sigma_g \text{ }\text{front}_{id}) - \text{pos}(\sigma_g \text{ }\text{rear}_{id}) - ?b_1 \rangle \leq \langle \text{pos}(\sigma_g \text{ }\text{rear}_{id}) - \text{pos}(\sigma_g \text{ }\text{front}_{id}) - ?b_0 \rangle
by linarith
then have \langle 0 < \text{pos}(\sigma_g \text{ }\text{front}_{id}) - \text{pos}(\sigma_g \text{ }\text{rear}_{id}) \rangle
using as3 d1 dmin
by auto
}
ote REM_front_car_nonstopping= this

have nocollision:: \langle 0 < d_{\text{real}} \sigma_g \rangle;
apply (cases (?b'_0 \geq \ ?b'_1))
using dmin apply simp
apply (cases (?b_0 \leq \ ?b_1))
using as3 d1 dmin_def no_collision_if_slow_rear_car_nonstopping
no_collision_if_slow_rear_car_stopping simp0
apply fastforce
apply (cases \langle d_{\text{safe}} \sigma_g > d_{\text{real}} \sigma_g \rangle; simp add:not_le)
using REM_front_car_may_stop REM_front_car_not_stopping d1
REM_front_car_nonstopping simp1
apply fastforce
using d1 front_car dmin_rear_car_safe by force

show \langle d_{\text{min}} \sigma_g' < d_{\text{real}} \sigma_g \rangle
using dmin nocollision by (simp add: dmin_def)
qed
next
fix \sigma_g
show \langle \pi^{-1} \sigma_g \Rightarrow \text{no_collision} \sigma_g \{0, 1\} \rangle
using unfolding dmin_def d1 \pi^{-1}_{\text{def}} no_collision_{\text{def}} by fastforce
qed

end

end
5 A Safety-Property in Autonomous Cars: RSS in a N-cars-Scenario

theory RSS_Ncars
imports CSP-AutoCars
begin

Instantiation of the Framework for Single-Lane scenarios with N-cars, that altogether use RSS.

declare minus_divide_le_eq[simp] not_less[simp]

5.1 Global Parameters of the Scenario-Class

locale RSS_Ncars_samelane_same_direction =

fixes a\_max\_accel :: real and a\_min\_brake :: real and a\_max\_brake :: real and \( \varphi \) :: real

fixes d\_real :: real scene \( \Rightarrow \) nat \( \Rightarrow \) real — actual distance between a vehicle and its front vehicle

and d\_rss :: real scene \( \Rightarrow \) nat \( \Rightarrow \) real — safe longitudinal distance as defined by RSS

and d\_min :: real scene \( \Rightarrow \) nat \( \Rightarrow \) real — safe longitudinal distance as defined by us ("RSS++")

fixes rss\_motion :: real motion\_acc

assumes a0 :: \( \langle 0 < a\_max\_accel \rangle \)
and a1 :: \( \langle 0 < a\_min\_brake \rangle \)
and a2 :: \( \langle a\_min\_brake \leq a\_max\_brake \rangle \)
and a3 :: \( \langle \varphi > 0 \rangle \)

defines \( d\_min \omega i \equiv \max 0 (\langle \text{speed} (\omega i) \rangle^2 / (2 * a\_min\_brake) - (\langle \text{speed} (\omega (i+1)) \rangle^2 / (2 * a\_max\_brake)) \)

and d0 :: \( d\_rss \omega i \equiv \max 0 (\langle \varphi * \text{speed} (\omega i) \rangle + ((\langle \varphi \rangle)^2 / 2) * a\_max\_accel + (\langle \text{speed} (\omega i) \rangle + \varphi * a\_max\_accel)^2 / (2 * a\_min\_brake) - (\langle \text{speed} (\omega (i+1)) \rangle)^2 / (2 * a\_max\_brake)) \)

and d1 :: \( d\_real \omega i \equiv \text{pos} (\omega (i+1)) - \text{pos} (\omega i) \)

defines r0 :: rss\_motion i \( \delta t \omega \equiv \) if \( d\_rss \omega i \leq d\_real \omega i \) then \( \{ -a\_max\_brake \ldots a\_max\_accel \} \)
else \( \{ -a\_max\_brake \ldots -a\_min\_brake \} \)

begin

5.2 Relations between Driving Strategies

— RSS kinetics are not backward

definition k0 :: kinetics\_RSS\_motion\_acc \equiv
5 A Safety-Property in Autonomous Cars: RSS in a N-cars-Scenario

\[
\lambda \omega \delta t. \{ \text{if speed} (\omega \, i) + (\delta t \cdot a) > 0 \text{ then } |pos = \text{pos} (\omega \, i) + (\delta t) \cdot \text{speed} (\omega \, i) + ((\delta t^2) / 2) \cdot a, } \\
\text{speed = speed} (\omega \, i) + (\delta t \cdot a), } \\
\text{acc = acc} \} \\
\text{else } |pos = \text{pos} (\omega \, i) - (\text{speed} (\omega \, i)^2 / (2 \cdot a), } \text{speed = 0, acc = 0} \}
\]

\text{lemma \ kinetics}_{\text{RSS}}_{\text{-empty}}: \text{\{kinetics}_{\text{RSS}} \text{ motion}_{\text{acc}} i \delta t \omega = \{\} \leftrightarrow \text{motion}_{\text{acc}} i \delta t \omega = \{\}}

\text{by (auto simp add: k0)}

\text{theorem \ rss\_is\_safe\_N\_cars:\ }
\text{fixes } N::\text{nat}\\
\text{assumes } a: \forall i \in \{0 .. N\}. d_{\text{min}} \omega_0 i < d_{\text{real}} \omega_0 i \wedge 0 \leq \text{speed} (\omega_0 i)\\
\text{and } b: 0 < \Delta t \land \Delta t < q;\\
\text{shows } \text{isafe}_{\text{f}e} \{0 .. N\} \text{ kineticss}_{\text{RSS}} \text{ motion}_{\text{RSS}} \text{ motion}_{\text{acc}} i \delta t \omega 0;\\
\text{proof --}
\text{define } P_{\text{inv}} \text{ where } P_{\text{inv}} \equiv \lambda (\omega::\text{real scene}). (\forall i \leq N. 0 \leq \text{speed} (\omega \, i) \wedge (\forall i < N. d_{\text{min}} \omega i < d_{\text{real}} \omega i));\\
\text{show \ \?\thesis}
\text{proof (rule isafe \_rule\_inv\[where \ P_{\text{inv}}=\lambda \omega \, \text{sid. } P_{\text{inv}} \omega\])}
\text{show } 0 < \Delta t \text{ using } b \text{ by simp next}
\text{show } (P_{\text{inv}} \omega_0)
\text{using } a \text{ by (auto simp add: } P_{\text{inv}}_{\text{def}} d0)\text{next}
\text{fix } \omega \delta t
\text{show } (P_{\text{inv}} \omega \wedge \delta t \in \{0 <.. \Delta t\} \implies \forall i \in \{0 .. N\}. \text{kinetics}_{\text{RSS}} \text{ motion}_{\text{RSS}} \text{ motion}_{\text{acc}} i \delta t \omega \neq \{\});
\text{using } a0 a1 a2 b0 b1 b2 \text{ by (auto simp:kinetics}_{\text{RSS}}_{\text{-empty}})\text{next}
\text{fix } \omega \delta t
\text{show } (P_{\text{inv}} \omega \wedge \delta t \in \{0 <.. \Delta t\} \implies \forall \omega' \in \omega'. \forall i \in \{0 .. N\}, \omega' i \in \text{kinetics}_{\text{RSS}} \text{ motion}_{\text{RSS}} \text{ motion}_{\text{acc}} i \delta t \omega). P_{\text{inv}} \omega';
\text{proof (auto simp: } P_{\text{inv}}_{\text{def}})\text{fix } \omega' i
\text{assume } \forall i \in \{0 .. N\}, \omega' i \in \text{kinetics}_{\text{RSS}} \text{ motion}_{\text{RSS}} \text{ motion}_{\text{acc}} i \delta t \omega \wedge i \leq N \wedge 0 < \delta t
\text{then show } 0 \leq \text{speed} (\omega' i);
\text{apply (auto simp:k0 r0 mult.commute split:if_splits)}
\text{by (metis as.select_convs (2) atLeastAtMost_iff less_eq_real_def not_less zero_le) next}
\text{fix } \omega' i
\text{assume } assm1: \forall i \leq N. 0 \leq \text{speed} (\omega \, i) \wedge \text{assm2: } \forall i < N. d_{\text{min}} \omega \, i < d_{\text{real}} \omega \, i \wedge \text{assm3: } 0 < \delta t
\text{and assm4: } \delta t \leq \Delta t \wedge \text{assm5: } \forall i \in \{0 .. N\}. \omega' i \in \text{kinetics}_{\text{RSS}} \text{ motion}_{\text{RSS}} \text{ motion}_{\text{acc}} i \delta t \omega)
\text{and assm6: } i < N;


5.2 Relations between Driving Strategies

have as1: \( 0 \leq \text{speed}(\omega i) \)
  using assm1[rule_format, of \( i \) assm6 by simp]
have as2: \( 0 \leq \text{speed}(\omega (i+1)) \)
  using assm1[rule_format, of \( i \) assm6 by simp]
note as3 = assm2[rule_format, of \( i \) OF assm6]
note as4 = assm3
note as5 = assm4
have as6: \( \omega' i \in \text{kinetics}_{RSS} \text{rss}_{motion} i \delta t \omega \)
  using assm5[rule_format, of \( i \) assm6 by simp]
have as7: \( \omega'(i+1) \in \text{kinetics}_{RSS} \text{rss}_{motion} (i+1) \delta t \omega \)
  using assm5[rule_format, of \( i \) assm6 by simp]

let \( a_r = \text{acc}(\omega i) \) and \( a_f = \text{acc}(\omega'(i+1)) \)
and \( v_r = \text{speed}(\omega i) \) and \( v_f = \text{speed}(\omega'(i+1)) \)
and \( v'_r = \text{speed}(\omega' i) \) and \( v'_f = \text{speed}(\omega' (i+1)) \)
and \( p_r = \text{pos}(\omega i) \) and \( p_f = \text{pos}(\omega'(i+1)) \)
and \( p'_r = \text{pos}(\omega' i) \) and \( p'_f = \text{pos}(\omega' (i+1)) \)
and \( b_r = ((\text{speed}(\omega i))^2 / (2*a_{\text{minbreak}})) \) and \( b_f = ((\text{speed}(\omega'(i+1)))^2 / (2*a_{\text{maxbreak}})) \)
and \( b'_r = ((\text{speed}(\omega' i))^2 / (2*a_{\text{minbreak}})) \) and \( b'_f = ((\text{speed}(\omega' (i+1)))^2 / (2*a_{\text{maxbreak}})) \)

— some simplifications factorized here
have simp0: \( 0 \leq v'_r \) and simp1: \( 0 \leq v'_f \)
  using as6 as7 by (auto simp:k0 r0 mult.commute split_if_splits)

have simp2: \( v_r / 2 + a_r * \delta t / 2 \geq 0 \) and simp3: \( v_f / 2 + a_f * \delta t / 2 \geq 0 \)
  using as1 as2 as6 as7 by (auto simp add:k0 r0 as4 mult.commute)

have simp4: \( v_r + a_r * \delta t / 2 \geq 0 \) and simp5: \( v_f + a_f * \delta t / 2 \geq 0 \)
  using as1 simp2 apply linarith
  using as2 simp3 by linarith

have simp6: \( \delta t * v_r + a_r * \delta t^2 / 2 \geq 0 \) and simp7: \( \delta t * v_f + a_f * \delta t^2 / 2 \geq 0 \)
  using as4 simp4 simp5
  by (metis (no_types, hide_lams) distr_left less_eq_real_def mult.assoc
      mult.commute mult_nonneg_nonneg power2_eq_square times_divide_eq_right)+

have simp4p: \( v_r + a_r * \delta t / 2 \geq v_r / 2 \)
  and simp5p: \( v_f + a_f * \delta t / 2 \geq v_f / 2 \)
  using as1 simp2 apply linarith
  using as2 simp3 by linarith

have simp6p: \( \delta t * v_r + a_r * \delta t^2 / 2 \geq \delta t * v_r / 2 \)
  and simp7p: \( \delta t * v_f + a_f * \delta t^2 / 2 \geq \delta t * v_f / 2 \)
  using as4 simp4p simp5p
  by (metis distr_left less_eq_real_def mult.left_commute mult_left_mono
      power2_eq_square times_divide_eq_right)+

45
5 A Safety-Property in Autonomous Cars: RSS in a N-cars-Scenario

have simp8:\( \forall \nu_r, a \ast \delta t \leq 0 \implies - (\nu_r)^2 / (a \ast 2) \leq (\delta t \ast \nu_r) / 2 \). for a
using as1 mult_left_mono [where \( b = \delta t \) and \( a = - (\nu_r) \) and \( c = (?\nu_r) \), OF as1]
apply (auto simp add: mult.commute power2_eq_square split:if_splits)
using as4 mult_pos_pos by force

have simp9:\( \forall \nu_r, a \ast \delta t \leq 0 \implies - (\nu_r)^2 / (a \ast 2) \leq (\delta t \ast \nu_r) / 2 \). for a
using as2 mult_left_mono [where \( b = \delta t \) and \( a = - (\nu_r) \) and \( c = (?\nu_r) \), OF as2]
apply (auto simp add: mult.commute power2_eq_square split:if_splits)
using as4 mult_pos_pos by force

— safety distance proofs
{ assume \( \forall \nu_r > 0 \)
from as7 have \( \forall \nu_f > 0 \)
by (auto simp add: k0 r0 mult.commute divide_le_0_iff split:if_splits) (smt (z3) a1 a2 divide_le_eq_1_neg minus_divide_right)

have h2:\( - (\nu_f + \delta t \ast a_f)^2 / (2 \ast a_{\max \text{brake}})^2 = (\nu_f)^2 / (2 \ast a_{\max \text{brake}})^2 + (a_f / a_{\max \text{brake}})^2 \ast (\delta t \ast \nu_f + 2 \ast a_f \ast \delta t^2 / 2) \)
by (simp add: mult.commute power2_eq_square power2_eq_square split:if_splits)

have front_car: \( \forall \nu_f > 0 \)
using dmin_front_car_motion dmin_front_car_stopping simp1 by fastforce

{ assume h0: \( d_{\text{rss}} \forall \omega > d_{\text{real}} \omega \) and \( \forall \nu_r > 0 \)
have h2:\( \forall a_r / a_{\min \text{brake}} \leq -1 \)
using as6 a1 a2 h0 h1 apply (auto simp add: mult.commute power2_eq_square split:if_splits)
by (metis a1 add.inverse inverse_le_divide_eq_1_pos minus_divide_left neg_le_iff_le)

have \( \forall \nu_r > 0 \)
using h0 h1 mult_left_mono [OF h2 simp6]
by (auto simp add: k0 r0 mult.commute split:if_splits)

(smt (verit) mult.assoc mult.commute power2_eq_square)
5.2 Relations between Driving Strategies

then have \(?b'_r - b_r \leq p_r - p'_r\)
by force
	note dmin_rear_car_nonsafe_breaking=this

{ assume h0:d_r,ω i > d_{real} ω i and h1:?v'_r = 0:
have h2:a ≤ -a_{minbrake} \Rightarrow a / a_{minbrake} ≤ -1 for a
by (metis a1 divide_eq_minus_1_iff divide_le_cancel_less_le not_le)

have h4:?p'_r - p_r ≤ (?v_r)^2 / (2*a_{minbrake})
using as6 a1 a2 h0 h1 apply(auto simp add:k0 r0 power2_sum add_divide_distrib
split_if_splits)
using mult_left_mono[OF h2] by (simp add: mult.commute)
then have \(- p_r ≤ b_r - p'_r\)
by argo

note dmin_rear_car_nonsafe_stopping=this

have dmin_rear_car_nonsafe:d_r,ω i > d_{real} ω i \implies \(?b'_r - b_r \leq p_r - p'_r\)
using dmin_rear_car_nonsafe_breaking dmin_rear_car_nonsafe_stopping simp0 by fastforce

{ assume h0:d_r,ω i ≤ d_{real} ω i and h1:?v'_r = 0:
have h2: (\delta * ?v_r) / 2 ≤ g*?v_r
using as1 as5 b less_eq_real_def by force
have h3:?p'_r - p_r ≤ g*?v_r
using as6 as1 a1 h0 h1 apply(auto simp add:d0 d_{min} k0 r0 mult.commute not_le
split_if_splits)
using order_trans[OF simp8 h2]
apply (smt (verit, best) mult.commute neg_divide_le_eq)
by (metis add_cancel_right_right add_le_less_mono as4
mult_eq_0_iff mult_less_cancel_left_pos not_less)

have h4: \(?b_r \leq (?v_r + g*a_{maxaccel})^2/(2*a_{minbrake})\)
by (smt (verit, ccfv_SIG) a0 a1 a3 as1 frac_le mult_pos_pos power_mono zero_le_power2
with h0 have h5: g*?v_r + \delta - b_r < ?p_f - p_r
by (smt (verit, best) a0 a3 d0 d1 divide_pos_pos zero_le_mult_iff zero_less_power)
with h3 have \(?b_r - \delta < ?p_f - p'_r\)
by fastforce

note dmin_rear_car_safe_stopping=this

{ assume h0:d_r,ω i ≤ d_{real} ω i and h1:?v'_r > 0:
have h2: \delta * ?v_r ≤ g*?v_r
using as1 as4 as5 b mult_right_less_imp_less by force
from as6 have h3: ?a * \delta^2 / 2 ≤ a_{maxaccel}\delta^2 / 2
using a0 a1 a3 h0 apply (auto simp add:k0 r0 as4 split_if_splits)
by (smt (verit) as4 as5 b eq_divide_imp frac_le mult.commute
power2_le_imp_le zero_eq_power2 zero_le_mult_iff)

from as6 have h4: \(?b_r^2 \leq (?v_r + g*a_{maxaccel})^2/(2*a_{minbrake})\)
using a1 h0 apply (auto simp add:k0 r0)
by (smt (verit, best) a0 a1 as5 b divide_le_cancel mult_mono power_mono
zero_le_mult_iff)

have h5: \(?b_r \leq (?v_r + g*a_{maxaccel})^2/(2*a_{minbrake})\)
by (smt (verit, ccfv_SIG) a0 a1 a3 as1 frac_le mult_pos_pos power_mono zero_le_power2)
5 A Safety-Property in Autonomous Cars: RSS in a N-cars-Scenario

from h2 h3 h4 h5 have ( δt*νr + a∗δt^2/2 + (max ?b_r ?b′_r) < a*νr + a_{maxaccel}*ν^2/2 + (?ν_r + a_{maxaccel})^2/(2*a_{minbrake}))
   by (smt (verit) mult.commute)
with h0 have h6:δt*νr + a*δt^2/2 + (max ?b_r ?b′_r) − ?b_f < ?p_f − ?p_r′
   by (smt (verit, ccfV_SIG) d0 d1 mult.commute times_divide_eq_right)
have (?p_r′ − ?p_f = δt*νr + a*δt^2/2)
using a6 as1 a1 h1 by (auto simp add:d0 d_{min_def} k0 r0 mult.commute not_le_split:if_splits)

with h6 have (max ?b_r ?b′_r) − ?b_f < ?p_f − ?p_r′
   by fastforce
} note dmin_rear_car_safe_motion=this

have dmin_rear_car_safe::d_{rss} ω i ≤ d_{real} ω i ⇒ (max ?b_r ?b′_r) − ?b_f < ?p_f − ?p_r′
   by (metis a1 divide_le_cancel less_eq_real_def max_def power2_less_eq_zero_iff
        power_zero_numerals dmin_rear_car_safe_motion dmin_rear_car_safe_stopping
        simp0 zero_less_double_add_iff_zero_less_single_add)

have dmin:?b′_r − ?b_f < d_{real} ω i
   using as3 d1 d_{min_def} front_car dmin_rear_car_nonsafe dmin_rear_car_safe by fastforce

— no collision’s proofs

{ assume h0:?b_r ≤ ?b_f and h1:?b′_r < ?b′_f
   — ?b′_f 0 < ?b′_f 1, is added to profit from previous results of dmin
   and h2:?ν_r′ = 0
from h0 have h3:ν_r ≤ ?ν_f
   using mult_right_mono[OF h0, where c=(2 * a_{minbrake})]; simplified]
by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_le_imp_le zero_le_power2)

from h2 have h6: ?p_r' − ?p_r ≤ δt * ν_r / 2
   using a6 as1 apply (auto simp add:k0 r0 a1)
   using simp8[simplified divide_divide_eq_left[symmetric]]
   apply (smt (verit, ccfV_threshold) mult.assoc mult.left_commute
           mult_left_mono mult_minus_left_neg_le_iff power2_eq_square real_add_le_0_iff)
   by (metis add_cancel_left_right add_le_less_mono as4 mult_eq_0_iff
           mult_less_cancel_left_pos not_less)

have h6: ?p_r' − ?p_r ≤ ?p_f − ?p′_f
   using a7 h1 h2 apply (auto simp add:k0 r0 a1)
   by (smt (verit, best) as.select_convs(3) as4 field_sum_of_halves h3 h6
       mult.commute mult_less_cancel_left_pos simp7p)
} note no_collision_if_slow_rear_car_stopping=this

{ assume h0:?b_r ≤ ?b_f and h1:?b′_r < ?b′_f
   and h2:?ν_r′ > 0
from h0 have h3:ν_r ≤ ?ν_f
   using mult_right_mono[OF h0, where c=(2 * a_{minbrake})]; simplified]
by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_le_imp_le zero_le_power2)

have h4:?ν_f > 0

48
5.2 Relations between Driving Strategies

by (metis a1 div_0 divide_nonneg_pos h1 mult.commute mult_2_right not_less order not_eq_order_implies_strict simp1 zero_le_power2 zero_less_double_add_iff_zero_less_single_add_zero_power2)
from h1 h2 h3 h4 have h5: ?v_r + ?a_r * ?t < ?v_f + ?a_f * ?t
  using as6 apply(auto simp add:k0 r0)
by (smt (verit, best) a1 a2 frac_le mult commute mult_2_right not_less order not_eq_order_implies_strict simp1 zero_le_power2 zero_less_double_add_iff_zero_less_single_add_zero_power2)
have h6: ?v_r + ?t^2 * ?a_r < ?v_f + ?t^2 * ?a_f
  using as4 distrib_left h5 mult commute mult left_commute power_mono zero_le_power2 mult_less_cancel_left_pos power2_eq_square
with h3 have h7: ?v_r + ?t^2 * ?a_r / 2 < ?v_f + ?t^2 * ?a_f / 2:
  by (smt (no_types, hide_lams) as4 distrib_left h5 mult commute mult left_commute mult_less_cancel_left_pos)

have \( \langle ?p'_r - ?p_r < ?p'_f - ?p_f \rangle \)
  using as7 h2 h4 h7
  by (auto simp add:k0 mult.commute mult_less_cancel_left_pos)

have h8: ?p'_r < ?p_r
  using as7 h2 h4 h7
  by (smt (verit, best) a1 as3 d1 divide_nonneg_pos power2_sum not_le)

remaining case (REM):

1. non-safe mode \( d_{safe} > d_{real} \): in the opposite case, we profit from previous results of \( d_{min\_rear\_car\_safe} \)

2. faster rear car : but it becomes slower after breaking

\[
\begin{align*}
\text{assume} & \ h0: ?b_f < ?b_r \quad \text{and} \quad h2: ?b'_r < ?b'_f \quad \\
\text{then have} & \ \text{False} \\
\text{by} & \ (\text{metis a1 divide_eq_0_iff le_divide_eq mult_eq_0_iff not_less power_zero_numeral zero_le_power2 zero_less_numeral})
\end{align*}
\]

\[
\begin{align*}
\text{assume} & \ h0: ?b_f < ?b_r \quad \text{and} \quad h1: d_{rss} \omega i > d_{real} \omega i \quad \text{and} \quad h2: ?v_f > 0 \\
\text{and} & \ h3: ?v_f - a_{\max\_brake} * ?t \leq 0 \\
\text{have} & \ h4: ?b_f \leq ?p'_f - ?p_f \\
\text{using} & \ as7 h2
\end{align*}
\]

\[
\begin{align*}
\text{by} & \ (\text{smt (z3) as.select_convs(3) divide_minus_left h3 minus_divide_right mult_minus_left simp7p simp9}) \\
\text{then have} & \ 0 < ?p'_f - ?p_f \\
\text{by} & \ (\text{smt (verit, best) a1 as3 d1 d_{min\_def} divide_nonneg_pos d_{min\_rear\_car\_nonsafe} h1 zero_le_power2})
\end{align*}
\]

\[
\begin{align*}
\text{assume} & \ h0: ?b_f < ?b_r \quad \text{and} \quad h1: d_{rss} \omega i > d_{real} \omega i \quad \text{and} \quad h2: ?v_f > 0 \\
\text{and} & \ h3: ?v_f - a_{\max\_brake} * ?t > 0 \\
\text{have} & \ s1: a_{\max\_brake} / a_{\min\_brake} \geq 1 \land \sqrt{a_{\max\_brake}} \geq \sqrt{a_{\min\_brake}} \land \sqrt{a_{\min\_brake}} > 0
\end{align*}
\]

\[
\begin{align*}
\text{using} & \ a1 a2 \text{ by simp}
\end{align*}
\]
then have s2: \( a_{\text{max\,brake}} / a_{\text{min\,brake}} \geq \sqrt{a_{\text{max\,brake}}} / \sqrt{a_{\text{min\,brake}}} \) 
by (smt (verit, ccfv_threshold) divide_less_eq_1 real_div_sqrt real_sqrt_divide)
then have s3: \( a_{\text{max\,brake}} \cdot \sqrt{a_{\text{min\,brake}}} \geq a_{\text{min\,brake}} \cdot \sqrt{a_{\text{max\,brake}}} \) 
by (smt (verit, ccfv_SIG) a1 a2 divide_divide_times_eq divide_le_eq_1_pos divide_less_eq_1 mult_nonneg_nonneg real_sqrt_gt_0_iff)

have s4: \( \sqrt{a_{\text{max\,brake}}} / \sqrt{a_{\text{min\,brake}}} \geq (?v_f) / (?v_c) \) 
using real_sqrt_le_mono OF order.strict_implies_order[OF h0]
apply(simp add:real_sqrt_divide real_sqrt_mult as1 a2 mult.commute)
apply(subst asm) (1 2) divide_divide_eq_left[symmetric], subst_asm) divide_le_cancel, simp
by (smt (verit, ccfv_threshold) a1 divide_less_eq_1 mult.commute
pos_divide_le_eq s1 times_divide_eq_left)
have s5: \(?v_f / a_{\text{min\,brake}} \geq (?v_f / a_{\text{max\,brake}}) \) 
using order_trans[OF s4 s2] apply(subst asm) pos_divide_le_eq
apply(smt (verit, ccfv_threshold) a1 a2 divide_eq_0_iff divide_nonneg_pos
h0 zero_le_power2 zero_power2)
apply(simp add:a1 le_divide_eq mult.commute)
by (smt (verit) a1 a2 divide_le_cancel nonzero_mult_div_cancel_left)
then have s6: \(?v_f - a_{\text{min\,brake}} \cdot \delta t > 0 \) 
using h3 a1 a2 by (metis a1 diff_gt_0_iff_gt less_trans mult.commute
pos_less_divide_eq)

have s7: \((?v_f) \cdot \sqrt{a_{\text{min\,brake}}} \leq (?v_f) \cdot \sqrt{a_{\text{max\,brake}}} \) 
using real_sqrt_le_mono OF order.strict_implies_order[OF h0]
apply(simp add:real_sqrt_divide real_sqrt_mult as1 a2 mult.commute)
apply(subst asm) (1 2) divide_divide_eq_left[symmetric], subst_asm) divide_le_cancel, simp
apply(subst asm) pos_divide_le_eq
using a1 a2 apply auto[1]
by (simp add:a1 le_divide_eq mult.commute)

have s8: \((?v_f - a_{\text{max\,brake}} \cdot \delta t) / \sqrt{a_{\text{max\,brake}}} \leq (?v_f - a_{\text{min\,brake}} \cdot \delta t) / \sqrt{a_{\text{min\,brake}}} \) 
apply(subst pos_divide_le_eq) using a1 a2 apply auto
apply(subst pos_divide_le_eq) apply(simp_all add:le_diff_distrib)
using s7 s3 by (metis as4 diff_mono mult.commute mult.left_commute
mult_le_cancel_left_pos)
with h3 have s9: \((?v_f - a_{\text{max\,brake}} \cdot \delta t)^2 / (2 \cdot a_{\text{max\,brake}}) \leq ((?v_f - a_{\text{min\,brake}} \cdot \delta t)^2 / (2 \cdot a_{\text{min\,brake}})) \) 
using abs_le_square_iff(\text{where } x=(?v_f - a_{\text{max\,brake}} \cdot \delta t) / \sqrt{a_{\text{max\,brake}}} \) and \( y = ((?v_f - a_{\text{min\,brake}} \cdot \delta t) / \sqrt{a_{\text{min\,brake}}}) \) 
using s1 apply(auto simp add:power_divide)
by (smt (verit, best) divide_less_0_iff s1)

have s10: \( \delta t \cdot ?v_f / 2 \leq (\delta t \cdot ?v_f - a_{\text{min\,brake}} \cdot \delta t^2 / 2) \) 
by (smt (verit, best) as4 field_sum_of_halves le_divide_eq power2_eq_square
real_divide_square_eq s6 zero_less_power)

have h4: \( \delta t \cdot ?v_f - a_{\text{min\,brake}} \cdot \delta t^2 / 2 \leq ?p_f - ?p_f \)
5.2 Relations between Driving Strategies

using as7 h2
apply (auto simp add:k0 r0 mult.commute not_le power2_sum add_divide_distrib
split:if_splits)
by (metis comm_semiring_class.distrib mult_nonneg_nonneg real_0_le_add_iff
zero_le_power2)+
have $\delta t \cdot \nu f - a_{\text{max brake}} \cdot \delta t^2 / 2 - b_f = - (\nu f - a_{\text{max brake}} \cdot \delta t^2 / (2 \cdot a_{\text{max brake}}))$
apply(simp add:power2_diff add_divide_distrib minus_add diff_add_eq_diff_diff_swap
diff_divide_distrib)
by (metis a1 a2 mult.assoc mult.left_commute nonzero_mult_div_cancel_left not_less
power2_eq_square)

with h4 have $\langle ?p' - ?p - ?b \rangle \geq - (\nu f - a_{\text{max brake}} \cdot \delta t^2 / (2 \cdot a_{\text{max brake}}))$
by linarith

have $a \leq - a_{\text{min brake}} \Rightarrow (\delta t \cdot \nu_r + a \cdot \delta t^2 / 2) \leq (\delta t \cdot \nu_r - a_{\text{min brake}} \cdot \delta t^2 / 2)$ for $a$
by (smt (verit as4 eq_divide_imp field_sum_of_halves mult_minus_left
pos_minus_divide_le_eq zero_less_power2)
then have $\delta t \cdot \nu_r - a_{\text{min brake}} \cdot \delta t^2 / 2$.

using as6 h1 h2 a1 apply (auto simp add:k0 r0 d0 d1 mult.commute not_le power2_sum
add_divide_distrib min_def split:if_splits)
using simp8 s10 by (smt (verit, ccfv_SIG) divide_le_cancel eq_divide_imp mult
.commutestart}}
next
fix \omega
have \( \langle P_{\text{inv}} \omega \implies (i < j \land j \leq N) \implies \text{pos} (\omega i) < \text{pos} (\omega j) \rangle \) for \( i j \)
unfolding \( d_{\text{min}} \_\text{def} \) \( d_{\text{inv}} \_\text{def} \)
apply(\text{induct } j, \text{simp})
by (\text{smt (z3) } \text{Suc\_leD Suc\_le\_lessD add\_commute less\_Suc\_eq plus\_1\_eq\_Suc})
then show \( \langle P_{\text{inv}} \omega \implies \text{no\_collision} \omega \{0..N\} 0 \rangle \)
unfolding \( \text{no\_collision} \_\epsilon \_0 \)
by (\text{smt (z3) } \text{atLeastAtMost\_iff inj\_on\_def less\_le not\_less})
qed
qed
6 An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario

theory RSS_plus
imports CSP−AutoCars
begin

6.1 Context: modeling N cars, one lane, one direction
locale same_lane_samedirection_Ncars =

fixes \(a_{\text{max accel}}; a_{\text{min brake}}; a_{\text{max brake}}; \mu; \varepsilon; \)::real

assumes \(a_0::0 < a_{\text{max accel}}\)

and \(a_1::a_{\text{min brake}} < 0\)

and \(a_2::a_{\text{max brake}} \leq a_{\text{min brake}}\)

and \(a_3::\mu > 0\)

and \(a_4::\varepsilon > 0\)

begin

definition \(d_{\text{real}}::\langle\text{real scene} \Rightarrow \text{nat} \Rightarrow \text{real}\rangle\)

where \(d_0::d_{\text{real}} \omega i \equiv \text{pos}(\omega (i+1)) − \text{pos}(\omega i)\)

definition \(d_{\text{rss}}::\langle\text{real scene} \Rightarrow \text{nat} \Rightarrow \text{real}\rangle\)

where \(d_1::d_{\text{rss}} \omega i \equiv \text{let} \ v_r = \text{speed}(\omega i) ; \ v_f = \text{speed}(\omega (i+1)) \ in\)

if \(v_r \geq 0\)

then max 0 ((\(\mu * v_r + (\mu^2/2)*a_{\text{max accel}} + (v_r + \mu*a_{\text{max accel}})^2/(2*−a_{\text{min brake}})\))

− \((v_f)^2/(2*−a_{\text{max brake}})\))

else undefined

Considering the next acceleration of the rear car where \(a_r < a_{\text{max accel}}\)

definition \(d_{\text{safe}}::\langle\text{real scene} \Rightarrow \text{nat} \Rightarrow \text{real} \Rightarrow \text{real}\rangle\)

where \(d_2::d_{\text{safe}} \omega i a_r \equiv \text{let} \ v_r = \text{speed}(\omega i) ; \ v_f = \text{speed}(\omega (i+1)) \ in\)

if \(v_r \geq 0\)

then max 0

((if \(v_r + \mu*a_r > 0\)

then \((\mu * v_r + (\mu^2/2)*a_r + (v_r + \mu*a_r)^2/(2*−a_{\text{min brake}}))\)

else \((v_r)^2/(2*−a_r)\))

− \((v_f)^2/(2*−a_{\text{max brake}})\))

else undefined

definition \(\text{drive}_{\text{rss}}::\langle\text{real} \Rightarrow \text{real motion}_{\text{accel}}\rangle\)

53
where $r_0$:

\[
\text{drive}_{rss} \epsilon' \ i \ \delta t \ \omega \ \equiv \ \text{if } d_{rss} \ \omega \ i + \epsilon' \ \leq \ d_{real} \ \omega \ i \ \\
\quad \text{then } \{ a_{maxbrake} \ldots a_{maxaccel} \} \\
\quad \text{else } \{ a_{maxbrake} \ldots a_{minbrake} \}
\]

\(\epsilon'\) is a margin we can add to drive safely.

\textbf{definition} drive_{safe} \(\epsilon'\) \(i\) \(\delta t\) \(\omega\) \(\equiv\) \(\{ a_{maxbrake} \ldots a_{maxaccel} \} \cap \{ a \ a > a_{minbrake} \rightarrow d_{safe} \ \omega \ i \ a + \epsilon' \ \leq \ d_{real} \ \omega \ i \})

Cars can not go backward! Overriding kinetics:

\textbf{definition} kinetics_{fw} \(\omega\) \(\rightarrow\) real motion_{acc}.

\textbf{lemma} kinetics_{fw-empty}:

\(\text{kinetics}_{fw} \ \text{motion}_{acc} \ i \ \delta t \ \omega \ = \ \{ \} \ \leftrightarrow \ \text{motion}_{acc} \ i \ \delta t \ \omega \ = \ \{ \} \)

\textbf{and} kinetics_{fw-forward}:

\(\omega' \ i \ \in\ \text{kinetics}_{fw} \ \text{motion}_{acc} \ i \ \delta t \ \omega \ \Longrightarrow \ \text{speed}(\omega' \ i) \ \geq \ 0\)

\textbf{using} \(k_0\) \textbf{by} \textbf{auto}

safety distance is monotonic w.r.t the next acceleration: when we accelerate, we increase the breaking distance.

\textbf{lemma} d_{safe-monoton}

\textbf{proof} (rule monol)

\textbf{fix} \(x\) \(::\) \text{real} \text{ and } \(y\)

\text{assume} \(h_0:: x \ \leq \ y\)

\text{have} \(h_1:: 0 \ < \ \text{speed}(\omega \ i) + g \ast x \ \Longrightarrow \ 0 \ < \ \text{speed}(\omega \ i) + g \ast y\)

\quad by (meson a3 add_le_cancel_left h0 less_eq_real_def less_le_trans mult_left_mono)

\text{have} \(h_2:: 0 \ < \ \text{speed}(\omega \ i) + g \ast x \ \Longrightarrow \ g \ast \text{speed}(\omega \ i) + g \ast x \ \leq \ 2 + (\text{speed}(\omega \ i) + g \ast x)^2 / (2 \ast a_{minbrake})\)

\quad by (smt (verit. best) a1 a3 divide_le_cancel h0 mult_eq_0_iff mult_less_cancel_left_pos power_mono zero_le_power2)

\text{(assume} \(hh_0:: \text{speed}(\omega \ i) \ \geq \ 0\)

\quad \text{and} \(hh_1:: 0 \ < \ \text{speed}(\omega \ i) + g \ast y\)

\quad \text{and} \(hh_2:: \text{speed}(\omega \ i) + g \ast x \ \leq \ 0\)

\text{have} \(hh_3:: (\text{speed}(\omega \ i) + g \ast y)^2 / (2 \ast a_{minbrake}) \ \geq \ 0\)

\quad \text{using} a1 divide_le_0_iff by fastforce

\text{have} \(hh_4:: \text{speed}(\omega \ i) / 2 \ \leq \ \text{speed}(\omega \ i) + (g/2) \ast y\)

\quad \text{using} hh1 by simp

\text{have} \(hh_5:: (\text{speed}(\omega \ i) / x \ \leq \ y)\)

6.2 Preliminaries

\textbf{declare} Let_def\[simp\]
6.3 Our motion vs RSS

lemma \(d_{safe} \cdot d_{rss} \cdot i = d_{safe} \cdot \omega \cdot i \cdot a_{max\_accel}\) using d1 d2 a0 a3 by (auto simp add: le\_less\_trans)

lemma drive\_safe \cdot drive\_rss: \('d_{rss} \cdot i \cdot \omega \subseteq drive\_safe \cdot i \cdot \omega\)
apply (cases \(d_{rss} \cdot i \cdot \omega \cdot i + \varepsilon' \cdot \le\_real\cdot \omega \cdot i\))
using a0 a1 a2 kinetics\_f\_\_empty r0 i1 apply auto[1]
using drive\_safe \cdot drive\_rss k0 apply auto
using atLeast\_At\_Most\_if by blast

corollary is\_safe \{0..N\} \((kine\_f_u \cdot drive\_safe \cdot \varepsilon')\) \(\Delta t \cdot \omega_0 \cdot \varepsilon\) \(\Rightarrow is\_safe \{0..N\} \((kine\_f_u \cdot drive\_rss \cdot \varepsilon')\) \(\Delta t \cdot \omega_0 \cdot \varepsilon\))
by (rule is\_safe\_refine, rule cars\_safe \cdot cars\_rss, assumption)

6.4 Our motion is safe

definition \(d_{min}::\text{(real scene} \Rightarrow \text{nat} \Rightarrow \text{real}\)
where \(d3::d_{min} \cdot \omega \cdot i \equiv \text{let} \ v_r = \text{speed}\cdot \omega \cdot i; v_f = \text{speed}\cdot \omega \cdot (i+1)\) in
\text{if} \ v_r \geq 0
\text{then} max 0 ((v_r)^2/(2*a_{min\_brake}) - (v_f)^2/(2*a_{max\_brake}))
6 An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario

lemma \( d_{\text{min-safe}} \cdot (\omega_i = d_{\text{safe}} \cdot \omega \cdot a_{\text{minbrake}}) \)

using a1 d3 d2 apply (auto simp add: power2_sum add_divide_distrib)
by (auto simp:power2_eq_square)

declare minus_divide_le_eq[simp] not_less[simp]

theorem rss_is_safe_N_cars:
assumes
\( a \cdot \forall i \in \{0..N\}, d_{\text{min}} \omega_0 \cdot i + \epsilon' \leq d_{\text{real}} \omega_0 \cdot i \land 0 \leq \text{speed}(\omega_0 \cdot i) \)
and \( b \cdot 0 < \Delta t \land \Delta t \leq \psi \)
and \( c \cdot \epsilon < \epsilon' \)
shows \( \text{is_safe} \cdot (0..N) \cdot (\text{kinetics} \cdot (\text{drive_safe} \cdot \epsilon')) \cdot \Delta t \cdot \omega_0 \cdot \epsilon \)

proof -
define \( P_{inv} \) where \( P_{inv} = \lambda \omega \cdot \text{real} \cdot \omega \cdot \text{sid} \cdot \text{P_{inv} \cdot \omega} \)

show \( \varepsilon < \varepsilon' \)

fix \( \omega \cdot \delta t \)

have \( \forall i \in \{0..N\}, \{a_{\text{maxbrake}} \cdot a_{\text{minbrake}}\} \subseteq \text{drive_safe} \cdot \epsilon' \cdot i \cdot \delta t \cdot \omega \)

using r1 a0 a1 a2 by auto

then show \( \text{P_{inv} \cdot \omega} \cdot \omega \cdot \delta t \in \{0\ldots\Delta t\} \implies \forall i \in \{0..N\}. \text{kinetics} \cdot (\text{drive_safe} \cdot \epsilon') \cdot i \cdot \delta t \cdot \omega \neq \{\} \)

using a2 kinetics\_fw\_empty by fastforce

next
fix \( \omega \cdot \delta t \)

show \( \forall i \in \{0..N\}. \omega \cdot i \in \text{kinetics} \cdot (\text{drive_safe} \cdot \epsilon') \cdot i \cdot \delta t \cdot \omega \)

\( \text{P_{inv} \cdot \omega} \cdot \omega' \cdot i \)

proof (auto simp:P_{inv}\_def)

fix \( \omega \cdot i \)

assume \( \forall i \in \{0..N\}. \omega' \cdot i \in \text{kinetics} \cdot (\text{drive_safe} \cdot \epsilon') \cdot i \cdot \delta t \cdot \omega \) and \( i \leq N \) and \( 0 < \delta t \)

then show \( 0 \leq \text{speed}(\omega' \cdot i) \)

apply (auto simp:k0 r1 mult.commute split\_if_splits)
by (metis as.select\_convs(2) atLeastAtMost\_iff less\_eq\_real\_def not\_less\_zero\_le)

next
fix \( \omega \cdot i \)

assume assm1: \( \forall i \leq N. 0 \leq \text{speed}(\omega \cdot i) \)
and assm2: \( \forall i \leq N, \text{d_{min}} \omega \cdot i + \epsilon' \leq \text{d_{real}} \omega \cdot i \)
and assm3: \( 0 < \delta t \)
and assm4: \( \delta t \leq \Delta t \)
and assm5: \( \forall i \in \{0..N\}. \omega' \cdot i \in \text{kinetics} \cdot (\text{drive_safe} \cdot \epsilon') \cdot i \cdot \delta t \cdot \omega \)
and assm6: \( i < N \)

56
6.4 Our motion is safe

have as1: \( 0 \leq \text{speed}(\omega) \)

using assms1 rule_format, of \( i \) assms6 by simp

have as2: \( 0 \leq \text{speed}(\omega (i+1)) \)

using assms1 rule_format, of \( i+1 \) assms6 by simp

note as3 = assms2 rule_format, of \( i \) OF assms6

note as4 = assms3

note as5 = assms4

have as6: \( \omega' \ i \in \text{kinetics}_{fw} \ (\text{drive}_{sa} f_{e} \ e') \ i \delta t \omega \)

using assms5 rule_format, of \( i \) assms6 by simp

have as7: \( \omega' (i+1) \in \text{kinetics}_{fw} \ (\text{drive}_{sa} f_{e} \ e') \ (i+1) \delta t \omega \)

using assms5 rule_format, of \( i+1 \) assms6 by simp

some simplifying rules factorized here

have simp0: \( 0 \leq \text{speed}(\omega') \) and simp1: \( 0 \leq \text{speed}(\omega' (i+1)) \)

using as6 as7 by (auto simp:k0 r1 mult.commute split:if_splits)

have simp2: \( \text{speed}(\omega) + \delta t \ast a > 0 \Longrightarrow \text{speed}(\omega) + \delta t \ast a > \text{speed}(\omega) / 2 \) and simp3: \( \text{speed}(\omega (i+1)) + \delta t \ast a > 0 \Longrightarrow \text{speed}(\omega (i+1)) + \delta t \ast a > \text{speed}(\omega (i+1)) / 2 \) for a \( \delta t \)

by simp+

— a lower bound of the traversed distance when non stopping

have simp4: \( \delta t > 0 \Longrightarrow \text{speed}(\omega) + \delta t \ast a > 0 \Longrightarrow \text{speed}(\omega) + \delta t \ast a > \text{speed}(\omega) / 2 \) and simp5: \( \delta t > 0 \Longrightarrow \text{speed}(\omega (i+1)) + \delta t \ast a > 0 \Longrightarrow \text{speed}(\omega (i+1)) + \delta t \ast a > \text{speed}(\omega (i+1)) / 2 \) for a \( \delta t \)

using mult_strict_left_mono[OF simp2, of a \( \delta t \)] mult_strict_left_mono[OF simp3, of \( \delta t \)] a \( \delta t \)

by simp_all add: distrib_left power2_eq_square

have simp4': \( \delta t > 0 \Longrightarrow \text{speed}(\omega) + \delta t \ast a > 0 \Longrightarrow \text{speed}(\omega) + \delta t \ast a > \text{speed}(\omega) / 2 \) and simp5': \( \delta t > 0 \Longrightarrow \text{speed}(\omega (i+1)) + \delta t \ast a > 0 \Longrightarrow \text{speed}(\omega (i+1)) + \delta t \ast a > \text{speed}(\omega (i+1)) / 2 \) for a \( \delta t \)

using as1 as2 simp4 simp5 zero_le_mult_iff by fastforce+

— an upper bound of the traversed distance when stopping

have simp6: \( \delta t > 0 \Longrightarrow \text{speed}(\omega) + \delta t \ast a \leq 0 \Longrightarrow \text{speed}(\omega) / (2 \ast a) \leq \text{speed}(\omega) / 2 \) and simp7: \( \delta t > 0 \Longrightarrow \text{speed}(\omega (i+1)) + \delta t \ast a \leq 0 \Longrightarrow \text{speed}(\omega (i+1)) / (2 \ast a) \leq \text{speed}(\omega (i+1)) / 2 \)

using as1 as2 mult_left_mono[where b=\( \delta t \) and a=\( \text{speed}(\omega) / a \) and c=\( \text{speed}(\omega) \), \( \text{OF as1} \) mult_left_mono[where b=\( \delta t \) and a=\( \text{speed}(\omega (i+1)) / a \) and c=\( \text{speed}(\omega (i+1)) \), \( \text{OF as2} \)

apply (auto simp add:mult.commute power2_eq_square split:if_splits)

using real_0_le_add_iff by fastforce+
6 An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario

— cars are going forward

have \( \text{front\_car\_forward}: 0 \leq \text{pos}(\omega^\prime (i)) - \text{pos}(\omega (i)) \)

using as1 simp4 as1 as4 a1 a2 apply (auto simp add:k0 r1 mult.commute

\text{divide\_nonneg\_nonpos field\_sum\_of\_halves split:if_splits}

apply (smt (verit, best) \text{divide\_eq\_0\_iff divide\_nonneg\_neg mult\_nonneg\_nonneg zero\_eq\_power2 zero\_le\_power})

by (metis \text{(no\_types, hide\_lams) mult\_commute not\_le simp4\_times\_divide\_eq\_right})+

have \( \text{front\_car\_forward}: 0 \leq \text{pos}(\omega^\prime (i+1)) - \text{pos}(\omega (i+1)) \)

using as7 simp5 as4 a1 apply (auto simp add:k0 r1 mult.commute divide\_nonneg\_nonpos field\_sum\_of\_halves split:if_splits)

apply (smt (verit, best) \text{divide\_eq\_0\_iff divide\_nonneg\_neg mult\_nonneg\_nonneg zero\_eq\_power2 zero\_le\_power})

by (metis \text{(no\_types, hide\_lams) mult\_commute not\_le simp5\_times\_divide\_eq\_right})+

### 6.1 Safety distance proofs

\[
\text{have h0: speed}(\omega^\prime (i+1)) > 0
\]

from as7 have h1: \(?a_1 / a_{\text{max\_brake}} \leq 1\)

using a1 by (auto simp add:k0 r1) (smt (z3) a1 a2 \text{divide\_le\_eq\_1\_neg minus\_divide\_right})

have \( (\text{speed}(\omega (i+1)) + \delta \ast ?a_1)^2 / (2 \ast a_{\text{max\_brake}}) = (\text{speed}(\omega (i+1)) + \delta^2 / 2 \ast ?a_1) \)

by (simp add: power2\_sum distr\_left mult\commute power2\_eq\_square add\_divide\_distrib

mult\_left\_commute)

then have \(?b_1 - \text{pos}(\omega^\prime (i+1)) - \text{pos}(\omega (i+1))\)

using h0 as7 mult\_left\_mono[OF h1 simp5] as4 by (auto simp add:k0 r1 mult\commute

split:if_splits)

\text{have front\_car\_motion: ?b_1 - ?b_1' \leq pos(\omega^\prime (i+1)) - pos(\omega (i+1))}

using \text{dmin\_front\_car\_motion} \text{dmin\_front\_car\_stopping simp1 by fastforce}

\[
\text{fix a}
\]

assume h0: \text{speed}(\omega^\prime i) = 0

assume h1: a \leq a_{\text{min\_brake}}\text{ and h2: pos(\omega^\prime i) - pos(\omega i) = (speed(\omega i))^2/(2*\_a)}

have h3: \( (\text{speed}(\omega i))^2 / (2*\_a) \leq (\text{speed}(\omega i))^2 / (2* a_{\text{min\_brake}})\)
6.4 Our motion is safe

```isar
using a1 a2 h1 by (smt (z3) frac_le zero_le_power2)
with h2 have \( \text{pos} (\omega' i) - \text{pos} (\omega i) \leq ?b_0 \)
by linarith
} note dmin_rear_car_stopping_after_breaking=this

{ fix a
assume h0:speed(\omega' i) = 0;
assume h1:a > a_{minbrake} and h2:d_{safe} \omega i a + \varepsilon' \leq d_{real} \omega i
and h3:pos (\omega' i) = pos (\omega i)
(\omega i) = (speed(\omega i))^2 / (2a - a) and h4: speed(\omega i) + \delta t \cdot a \leq 0
have h5:speed(\omega i) + \omega = a \leq 0
by (smt (verit, best) as1 asm2 b h4 mult_le_cancel_right mult_pos_pos)
have \[ \text{pos} (\omega' i) - \text{pos} (\omega i) - \beta_1 + \varepsilon' \leq \text{pos} (\omega (i+1)) - \text{pos} (\omega i) \]
using h2 h5 apply (auto simp add:k0 r1 d0 d2 as1 max_def split:if_splits)
apply blast apply blast
using a1 apply linarith+
by fastforce
show \( ?\)thesis
apply(cases a \leq a_{minbrake})
using as1 as3 dmin_rear_car_stopping_after_breaking h0 h3 d0 d3 apply fastforce
using dmin_rear_car_stopping_after_slowing h0 h1 h3 h4 by force
qed

{ assume h0:speed(\omega' i) > 0 and h1:?a_0 \leq a_{minbrake} — h0 is not necessary, it is induced by h1
have h2:?a_0 / a_{minbrake} \geq 1;
using a1 a2 h1 by auto
with h1 have \[ \text{pos} (\omega' i) - \text{pos} (\omega i) \leq ?b_0 - ?b_0' \]
using as6 apply(auto simp add:k0 r1 power2_sum add_divide_distrib distrib_left not_le_split:if_splits)
by (drule mult_left_mono) OF h2 simp4[OF as4], simplified, simp add:distrib_right distrib_left power2_eq_square mult.left_commute add_divide_distrib)+
} note dmin_rear_car_breaking_without_stopping=this

{ assume h0:speed(\omega' i) > 0 and h1:?a_0 > a_{minbrake} and h2:?a_0 \geq 0
have h3:speed(\omega' i) = speed(\omega i) + ?a_0 \cdot \delta t
and h4: speed(\omega i) + ?a_0 \cdot \delta t > 0
using h0 as6 by (auto simp:k0 r1 mult.commute)
then have h5:speed(\omega i) + ?a_0 \cdot \delta t > 0
by (smt (verit) a3 as1 h2 mult_nonneg_nonneg zero_less_mult_iff)
from h0 h1 have h6:d_{safe} \omega i ?a_0 + \varepsilon' \leq d_{real} \omega i
```

59
An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario

using as6 k0 r1 apply(simp split_if_splits) by auto

have h7: |δt| ≤ (δt^2 / 2) ?a0 + (speed(ω i) + δt ?a0)^2 / (2 * -a_minbrake) ≤ g * speed(ω i) + (g^2 / 2) ?a0 + (speed(ω i) + g * ?a0) / (2 * -a_minbrake) | (is (?A))

proof |

have h8: 0 < 1 + ?a0 / -a_minbrake |

using a1 a2 h1 by auto |

then have h9: 0 ≤ speed(ω i) + ?a0 * (g + δt) / 2 + (?a0 / -a_minbrake) * (speed(ω i) + ?a0 * (g + δt) / 2)

| using mult_pos_pos | OF h8, of | speed(ω i) + ?a0 * (g + δt) / 2, simplified distrib_right |

by (smt (z3) distrb_left field_sum_of_halves h4 h5)

have h10: | (g - δt) * (speed(ω i) + (g + δt) * ?a0 / 2) |

= g * speed(ω i) - δt * speed(ω i) + (g^2 / 2) * ?a0 - δt^2 / 2 * ?a0 |

apply (subst distrb_left, subst left_diff_distrib, simp) |

by (metis (no_types, lifting) mult_assoc mult.commute power2_eq_square right_diff_distrib square_diff_square_factored)

have h11: | (g - δt) * (?a0 / -a_minbrake) * (speed(ω i) + (g + δt) * ?a0 / 2) |

= (speed(ω i) + g * ?a0)^2 / (2 * -a_minbrake) - (speed(ω i) + δt * ?a0)^2 / (2 * -a_minbrake) |

apply (subst diff_divide_distrib [symmetric]) |

apply (simp add: power2_eq_square square_diff_square_factored) |

apply (subst divide_divide_eq_left, subst divide_cancel_right) |

using a1 by (simp add: distrib_right) |

have 0 ≤ (g * speed(ω i) - δt * speed(ω i)) + (g^2 / 2) * ?a0 - δt^2 / 2 * ?a0 + |

( (speed(ω i) + g * ?a0)^2 / (2 * -a_minbrake) - (speed(ω i) + δt * ?a0)^2 / (2 * -a_minbrake) ) |

apply (subst h10 [symmetric], subst h11 [symmetric]) |

using mult_nonneg_nonneg | OF _ h9, of | (g - δt) |

by (metis (no_types, hide_lams) assn4 b diff_ge_0_iff_ge distrib_left le_less_trans mult.assoc mult.commute not_less)

then show ?A |

by simp |

qed

with h3 h4 h5 have pos (ω’ i) - pos (ω i) + b’h0 - b1 + ϵ’ ≤ pos (ω (i+1)) - pos (ω i) |

using as6 as1 a1 h1 by (auto simp add: d0 d2 k0 r1 mult_commute not_le split_if_splits)

} note dmin_rear_car_accelerate = this

{ assume h0: speed(ω’ i) > 0, and h1: ?a0 > a_minbrake |

and h2: ?a0 < 0, and h3: speed(ω i) + ?a0 * g > 0 |

have h4: speed(ω’ i) = speed(ω i) + ?a0 * δt, and h5: speed(ω i) + ?a0 * δt > 0 |

using h0 as6 by (auto simp: k0 r1 mult_commute) |

from h0 h1 have h6: d_safe ω i ?a0 + ϵ’ ≤ d_real ω i |

using as6 k0 r1 apply (simp split_if_splits) by auto |

have h7: | δt * speed(ω i) + (δt^2 / 2) * ?a0 + (speed(ω i) + δt * ?a0)^2 / (2 * -a_minbrake) |

≤ g * speed(ω i) + (g^2 / 2) * ?a0 + (speed(ω i) + g * ?a0)^2 / (2 * -a_minbrake) | (is (?A))

proof |

have h8: 0 < 1 + ?a0 / -a_minbrake |

using a1 a2 h1 by auto |

then have h9: 0 ≤ speed(ω i) + ?a0 * (g + δt) / 2 + (?a0 / -a_minbrake) * (speed(ω i) +
6.4 Our motion is safe

\[ a_0 \cdot (\alpha + \delta t) / 2 \]

using `mult_pos_pos[OF h8, of speed(\omega i) + a_0 \cdot (\alpha + \delta t) / 2, simplified distrib_right]

by (smt (z3) distrib_left field_sum_of_halves h3 h5)

have h10: \((\omega - \delta t) \ast (\text{speed}(\omega i) + (\omega + \delta t) \ast a_0) / 2\)

\[ = (\omega \ast \text{speed}(\omega i) - \delta t \ast \text{speed}(\omega i)) + (\omega^2 / 2 \ast a_0 - \delta t^2 / 2 \ast a_0) \]

by (metis (no_types, lifting) mult_assoc mult.commute power2_eq_square right_diff_distrib square_diff_square_factored)

have h11: \((\omega - \delta t) \ast (a_0 - a_{\text{minbrake}}) \ast \text{speed}(\omega i) + (\omega + \delta t) \ast a_0) / 2\)

\[ = (\text{speed}(\omega i) + \omega \ast a_0)^2 / (2 \ast a_{\text{minbrake}}) - (\text{speed}(\omega i) + \delta t \ast a_0)^2 / (2 \ast a_{\text{minbrake}}) \]

by (smt simp add: distribute_mult_divide symmetric)

apply (simp add: power2_eq_square square_diff_square_factored)

apply (subt divide_cancel_right)

using `mult_nonneg_nonneg[OF _ h9, of \(\omega - \delta t\)]` by (metis (no_types, hide_funs) asm4 b diff_ge_0_iff_ge distrib_left)

then show ?A

by simp

qed

with h3 h4 h5 have \(\text{pos}(\omega i) - \text{pos}(\omega i) + b_0 - b_1 + \epsilon \leq \text{pos}(\omega (i+1)) - \text{pos}(\omega i)\)

using as6 as1 a1 h1 by (auto simp add: d0 d2 k0 r1 mult.commute not_le split_if_splits)

note `dmin_rear_car_safe_slowing_no_possibly_stopping = this`

— dmin_rear_car_safe_accelerating and dmin_rear_car_safe_slowing_no_possibly_stopping can be merged
An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario

6.4.2 "No collision" proofs when \( ?b’_0 < ?b’_1 \)

{ assume h0: \( ?b_0 \leq ?b_1 \) and h1: \( ?b’_0 < ?b’_1 \) and h2: \( \text{speed}(\omega’ i) = 0 \) from h0 have h3: \( \text{speed}(\omega i) \leq \text{speed}(\omega (i+1)) \) using mult_right_mono[OF h0, where \( \text{where} = (2 * - \text{a}_{\text{mazbrake}}) \), simplified] by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_le_imp_le zero_le_power2) from h2 have h4: \( \text{pos}(\omega’ i) - \text{pos}(\omega (i+1)) \leq \text{delta}(\omega’ i) / 2 \) using as6 as1 simp6[OF as4] by (auto simp add:k0 r1) have h5: \( \text{speed}(\omega’ (i+1)) > 0 \) using h1 h2 not_le simp1 by fastforce then have h6: \( \text{delta}(\omega (i+1)) / 2 < \text{pos}(\omega’ (i+1)) - \text{pos}(\omega (i+1)) \) using as7 h1 h2 simp5[OF as4] by (auto simp add:k0 r1 a1) have \( \text{pos}(\omega’ i) - \text{pos}(\omega (i+1)) \leq \text{pos}(\omega’ (i+1)) - \text{pos}(\omega (i+1)) \) using as4 h3 h4 h6 by (smt (verit, ccfvc_SIG) divide_le_cancel mult_less_cancel_left_pos) } note no_collision_if_slow_rear_car_stopping=this

{ assume h0: \( ?b_0 \leq ?b_1 \) and h1: \( ?b’_0 < ?b’_1 \) and h2: \( \text{speed}(\omega’ i) > 0 \) from h0 have h3: \( \text{speed}(\omega i) \leq \text{speed}(\omega (i+1)) \) using mult_right_mono[OF h0, where \( \text{where} = (2 * - \text{a}_{\text{mazbrake}}) \), simplified] by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_le_imp_le zero_le_power2) have h4: \( \text{speed}(\omega’ (i+1)) > 0 \) by (smt (z3) a1 a2 frac_le h1 simp1 zero_le_power2 zero_power2) have \( \text{speed}(\omega’ i) / (2 * - \text{a}_{\text{mazbrake}}) < \text{?b’}_1 \) by (smt (verit, ccfvc_SIG) a1 a2 frac_le h1 zero_le_power2) with h1 h2 h4 have h5: \( ?a_0 * \text{delta} < \text{speed}(\omega’ (i+1)) + ?a_1 * \text{delta} \) using as6 as7 apply(auto simp:k0 r1) by (smt (verit, ccfvc_SIG) a1 a2 divide_le_cancel mult.commute power2_le_imp_le_power_mono zero_less_numeral) have h6: \( \text{delta} * \text{speed}(\omega’ i) + \text{delta}^2 * ?a_0 < \text{delta} * \text{speed}(\omega (i+1)) + \text{delta}^2 * ?a_1 \) by (metis (no_types, hide_lams) as4 distr_left h5 mult.commute mult.left_commute)
mult_less_cancel_left_pos power2_eq_square)
with h3 have h7: Δt * speed(ω (i + 1)) + Δt^2 * ?a_0 / 2 < Δt * speed(ω (i + 1)) + Δt^2 * ?a_1 / 2
by (smt (verit, ccfv_SIG) as4 field_sum_of_halves mult_less_cancel_left_pos)
have ipos (ω' (i)) = ipos (ω (i)) + ipos (ω (i + 1))
using as6 as7 h2 h4 h7
by (auto simp add: k0 r1 a1 d3 d1 divide_nonneg_pos power2_sum not_le)
} note no_collision_if_front_car_may_stop = this

{ assume h1: ?b'_0 < ?b'_i and h2: speed(ω (i + 1)) + a_maxbrake * Δt ≤ 0
— h1 only avoids the case when the front car stops
have h3: speed(ω' (i + 1)) > 0
by (smt (verit) a1 a2 as1 divide_eq_0_iff frac_le h1 simp1
zero_le_power2 zero_power2 divide_nonneg_pos)+
have h4: ?b_1 ≤ pos (ω' (i + 1)) - pos (ω (i + 1))
using as7 h2 h3 a1 a2
apply (auto simp add: k0 r1 mult.commute not_le power2_sum add_divide_distrib
split:if_splits)
using simp[OF as4] simp[OF as4]
by (smt (verit, del_insts) divide_minus_left_minus_divide_le_eq
mult.commute mult_less_0_iff times_divide_eq_right)+
then have ω' ≤ pos (ω' (i + 1)) - pos (ω')
using a1 simp0 as1 as3 simplified real_sqrt_divide real_sqrt_mult
mult_less_cancel_left_pos power2_eq_square)
with h3 have h7: Δt * speed(ω (i + 1)) + Δt^2 * ?a_0 / 2 < Δt * speed(ω (i + 1)) + Δt^2 * ?a_1 / 2
by (smt (verit, ccfv_SIG) as4 field_sum_of_halves mult_less_cancel_left_pos)
have ipos (ω' (i)) = ipos (ω (i)) + ipos (ω (i + 1))
using as6 as7 h2 h4 h7
by (auto simp add: k0 r1 a1 d3 d1 divide_nonneg_pos power2_sum not_le)
} note no_collision_if_front_car_may_stop = this

{ assume h0: ?b_0 < ?b_0 and h1: ?a_0 ≤ a_minbrake / speed(ω i) = 0: and h2: speed(ω (i + 1)) + a_maxbrake * Δt > 0
have h3: speed(ω i) > 0
by (smt (verit, ccfv_threshold) a1 a2 as1 divide_eq_0_iff
divide_nonneg_pos h0 zero_le_power2 zero_power2)

have s1: -a_maxbrake / a_minbrake ≥ 1 ∧ sqrt(-a_maxbrake) ≥ sqrt(-a_minbrake) ∧ sqrt(1) = 1
using a1 a2 by simp
then have s2: a_maxbrake / a_minbrake ≥ sqrt(-a_maxbrake) / sqrt(-a_minbrake)
by (smt (z3) divide_le_eq_1_pos minus_divide divide real_div_sqrt real_sqrt_divide)
then have s3: a_maxbrake * sqrt(-a_minbrake) ≤ a_minbrake * sqrt(-a_maxbrake)
by (smt (verit, best) a1 le_divide_eq mult.commute mult_minus_right s1 times_divide_eq_right)

have s4: sqrt(-a_maxbrake) / sqrt(-a_minbrake) ≥ (speed(ω (i + 1)))/(speed(ω i))
using real_sqrt_le_mono OF order.strict_implies_order[OF h0],
simplified real_sqrt_divide real_sqrt_mult
mult.commute divide_divide_eq_left[symmetric] divide_le_cancel]
apply(simp add: a1 as2) apply(rule mult_imp_div_pos_le)
by (smt (verit) a1 divide_le_eq mult.commute s1 times_divide_eq_right)
An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario

have $s5$: $\text{speed}(\omega i)/a_{\text{minbrake}} \geq \text{speed}(\omega (i+1))/a_{\text{maxbrake}}$
using order_trans[OF $s4$ $s2$] apply(subst (asm) pos_divide_le_eq)
apply(simp add: $a_{\text{minbrake}}$ $\ast \delta t$)
using (smt (verit, ccfv_threshold) $a_1$ $a_2$ $as1$ divide_eq_0_iff
divide_nonneg_pos $h_0$ zero_le_power2 zero_power2)
apply (smt (verit, ccfv_SLG) $a_1$ $a_2$ mult.commute)
have $\text{speed}(\omega (i+1))/a_{\text{maxbrake}} > \delta t$
apply (rule mult_imp_less_div_pos)
using $a_1$ $a_2$ apply fastforce
by (metis diff_gt_0_iff_gt diff_minus_eq_add $h_2$ mult.commute mult_minus_right)
with $s5$ have $s6$: $\text{speed}(\omega i) + a_{\text{minbrake}} \ast \delta t > 0$
using $h_2$ $a_1$ $a_2$
by (smt (z3) le_divide_eq_mult divide_right mult_less_cancel_left)

have $s7$: $(\text{speed}(\omega (i+1)) \ast \sqrt{-a_{\text{maxbrake}}}) \leq (\text{speed}(\omega i) \ast \sqrt{-a_{\text{maxbrake}}})$
using $s4$ by (metis divide_eq_0_iff $h_3$ le_divide_eq_mult.commute $s1$ times_divide_eq_left)
have $s8$: $(\text{speed}(\omega (i+1)) + a_{\text{maxbrake}} \ast \delta t) / \sqrt{-a_{\text{maxbrake}}}$
\leq $(\text{speed}(\omega i) + a_{\text{minbrake}} \ast \delta t) / \sqrt{-a_{\text{minbrake}}})$
apply(subst pos_divide_le_eq) using $a_1$ $a_2$ apply auto
apply(subst pos_le_divide_eq) apply(simp_all add: power2_sum)
using $s7$ $s3$

by (smt (verit, del_insts) add_mono_thms_linordered_seming(1)
assm3 mult.commute mult.left.commute mult_le_divide_left)
with $h_3$ have $s9$: $(\text{speed}(\omega (i+1)) + a_{\text{maxbrake}} \ast \delta t) / (2 * a_{\text{maxbrake}})$
\leq $(\text{speed}(\omega i) + a_{\text{minbrake}} \ast \delta t) / (2 * a_{\text{minbrake}}))$
using abs_le_square_iff[where $x = (\text{speed}(\omega (i+1)) + a_{\text{maxbrake}} \ast \delta t) / \sqrt{-a_{\text{maxbrake}}})$
\and $y = (\text{speed}(\omega i) + a_{\text{minbrake}} \ast \delta t) / \sqrt{-a_{\text{minbrake}}})$
using $s1$ apply (auto simp add: power_divide)
using $h_2$ $s6$ by linarith

have $h4$: $\delta t \ast \text{speed}(\omega (i+1)) + a_{\text{maxbrake}} \ast \delta t^2 / 2 \leq \text{pos}(\omega' (i+1)) - \text{pos} (\omega (i+1))$
using $as7$ $h_2$ as3m
apply (auto simp add: $k_0$ $r_1$ mult.commute not_le_power2_sum add_divide_distrib
split:if_splits)
using $a_1$ by (smt (z3) as3m mult.commute mult_less_cancel_left_pos zero_less_mult_iff+)
have $\delta t \ast \text{speed}(\omega (i+1)) + a_{\text{maxbrake}} \ast \delta t^2 / (2 * a_{\text{maxbrake}})$
\\leq $- (\text{speed}(\omega (i+1)) + a_{\text{maxbrake}} \ast \delta t^2 / (2 * a_{\text{maxbrake}}))$
using $s1$ apply(auto simp add: power2_diff add_divide_distrib minus_add
diff_add_eq_diff_diff_swap diff_divide_distrib power2_sum)
by (simp add: power2_eq_square)
with $h_4$ have $h5$: $\text{pos}(\omega' (i+1)) - \text{pos}(\omega (i+1)) = ?b_1$
\\geq $- (\text{speed}(\omega (i+1)) + a_{\text{maxbrake}} \ast \delta t^2 / (2 * a_{\text{maxbrake}}))$
by linarith

have $h6$: $a \leq a_{\text{minbrake}} \implies (\delta t \ast \text{speed}(\omega i) + a \ast \delta t^2 / 2) \leq \delta t \ast \text{speed}(\omega i) + a_{\text{minbrake}} \ast \delta t^2 / 2$
for $a$
by (smt (verit) as4 $h_1$ eq_divide_imp field_sum_of_halves mult_minus_left
pos_divide_le_eq mult_less_cancel_left)

64
Our motion is safe

\[
\text{have } \text{speed}(\omega^i) + a\cdot\delta t \leq 0 \implies (\text{speed}(\omega^i))^2 / (2a - a) \leq \delta t \cdot \text{speed}(\omega^i) / 2 \text{ for } a
\]

using simp[\text{OF as4}] by (smt (z3) divide_minus_left minus_divide_right mult.commute)

then have h6': \text{speed}(\omega^i) + a\cdot\delta t \leq 0 \implies (\text{speed}(\omega^i))^2 / (2a - a) \leq \delta t \cdot \text{speed}(\omega^i) + a_{\min \text{brake}} \cdot \delta t^2 / 2 \text{ for } a

using simp[\text{OF as4}, of \text{a}_{\min \text{brake}}] s6

by (smt \text{verit} mult.commute times_divide_eq_right)

from h6' h6'' have h6: \text{pos}(\omega^i) - \text{pos}(\omega^i) \leq \delta t \cdot \text{speed}(\omega^i) + a_{\min \text{brake}} \cdot \delta t^2 / 2,

using as6 a1 h1 h3 apply (auto simp add: k0 r1 d0 mult.commute split:if_splits)

by (smt (z3) mult.commute)+

have h7: \delta t \cdot \text{speed}(\omega^i) + a_{\min \text{brake}} \cdot \delta t^2 / 2 - \rho b_0 = -(\text{speed}(\omega^i) + a_{\min \text{brake}} \cdot \delta t^2 / (2a - a_{\min \text{brake}}))

using s1 apply(auto simp add:pow2_diff add_divide_distrib minus_add diff_add_eq_diff_diff_swap diff_divide_distrib pow2_sum)

by (simp add: pow2_eq_square)

with h5 h6 s9 have 'pos(\omega^i) - \text{pos}(\omega^i) - \rho b_0 \leq \text{pos}(\omega^i) (i+1) - \text{pos}(\omega^i) (i+1) - \rho b_i'

by linarith

then have \omega' \leq \text{pos}(\omega^i) (i+1) - \text{pos}(\omega^i)

using as1 as3 d0 d3 by auto

\} note no_collision_if_fast_rear_car_breaking_or_stopping_and_front_car_may_not_stop = this — 2 cases

\{ assume h0: \rho b_0 < \rho b_0 and h1: ?a_0 > a_{\min \text{brake}} and h2: \text{speed}(\omega^i + 1) + a_{\max \text{brake}} \cdot \delta t > 0 and h3: \text{speed}(\omega^i) > 0

have h4: \text{speed}(\omega^i) > 0

by (smt (verit, ccfv_threshold) a1 a2 as1 divide_eq_0_iff divide_nonneg_pos h0 zero_le_power2 zero_power2)

have s1: \(-a_{\max \text{brake}} / a_{\min \text{brake}} \geq 1 \wedge \text{sqrt}(-a_{\max \text{brake}}) \geq \text{sqrt}(-a_{\min \text{brake}}) \wedge \text{sqrt}(-a_{\min \text{brake}}) > 0

using a1 a2 by simp

then have s2: \text{sqrt}(-a_{\max \text{brake}} / a_{\min \text{brake}}) \geq \text{sqrt}(-a_{\max \text{brake}}) / \text{sqrt}(-a_{\min \text{brake}})

by (smt (z3) divide_le_eq_1_pos minus_divide_divide real_div_sqrt real_sqrt_divide)

then have s3: \text{sqrt}(-a_{\min \text{brake}}) \leq a_{\min \text{brake}} \cdot \text{sqrt}(-a_{\max \text{brake}})

by (smt (verit, best) a1 le_divide_eq mult.commute mult_minus_right s1 times_divide_eq_right)

have s4: \text{sqrt}(-a_{\max \text{brake}}) / \text{sqrt}(-a_{\min \text{brake}}) \geq (\text{speed}(\omega^i + 1)) / (\text{speed}(\omega^i))

using real_sqrt_le_mono[OF order.strict_implies_order[OF h0], simplified real_sqrt_divide real_sqrt_mult mult.commute divide_divide_eq_left[symmetric] divide_le-cancel]

apply(simp add: as1 as2) apply(rule mult_imp_div_pos_le)

by (simp add: h3')

(smt (verit) as1 divide_le_eq le_divide_eq mult.commute s1 times_divide_eq_right)

have s5: \text{speed}(\omega^i) / a_{\min \text{brake}} \geq \text{speed}(\omega^i) (i+1) / a_{\max \text{brake}}

using order_trans[OF OF s4 s2] apply(subst (asm) pos_divide_le_eq)

apply (smt (verit, ccfv_threshold) a1 a2 as1 divide_eq_0_iff divide_nonneg_pos h0

65
6 An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario

zero_le_power2 zero_power2)
  apply (simp add: a1 a2 le_divide_eq mult.commute)
  by (smt (verit, ccfv_SIG) a1 a2 mult.commute)
have (speed(\omega (i+1)))/-a_{max_brake} > \delta t
  apply (rule mult_imp_less_div_pos)
  using a1 a2 apply fastforce
by (metis diff_gt_0_iff_gt diff_minus_eq_add h2 mult.commute mult_minus_right)
with s5 have s6: speed(\omega i) + a_{min_brake} * \delta t > 0:
  using h2 a1 a2
by (smt (z3) le_divide_eq minus_divide_right mult.commute mult_minus_right)
have s6': speed(\omega i) + \delta t * a_0 > speed(\omega i) + a_{min_brake} * \delta t:
  by (simp add: assm3 h1)

  have s7: (speed(\omega (i+1)))*sqrt (-a_{min_brake}) \leq (speed(\omega i))*sqrt (-a_{max_brake}):
  using s4 by (metis divide_le_eq h3 times_divide_eq_left)
  have s8: (speed(\omega (i+1)) + a_{max_brake} * \delta t)/sqrt(-a_{max_brake}) \leq (speed(\omega i) + a_{min_brake} * \delta t)/sqrt(-a_{min_brake})
    apply (subst pos_divide_le_eq) using a1 a2 apply auto
    apply (subst pos_le_divide_eq) apply (simp_all add distrib_right)
  using s7 s3
  by (smt (verit, del_insts) add_mono_thms_linordered_semiring(1) assm3 mult.commute
      mult.left_commute mult_le_cancel_left_pos)
  then have s9: (speed(\omega (i+1)) + a_{max_brake} * \delta t^2)/(2*a_{max_brake}) \leq ((speed(\omega i) + a_{min_brake} * \delta t)^2)/(2*a_{min_brake})
    using abs_le_square_iff
    where x:=(speed(\omega (i+1)) + a_{max_brake} * \delta t)/sqrt(-a_{max_brake})
  and y = ((speed(\omega i) + a_{min_brake} * \delta t)/sqrt(-a_{min_brake}))
  using s1 apply (auto simp add: power_divide)
  using h2 s6 by linarith

  have h4: \delta t * speed(\omega (i+1)) + a_{max_brake} * \delta t^2/2 \leq pos(\omega' (i+1)) - pos (\omega (i+1)):
    using as7 h2 assm3 apply (auto simp add: k0 r1 mult.commute not_le power2_sum
      add_divide_distrib split:if_splits)
    using a1 by (smt (z3) assm3 mult.commute mult_less_cancel_left_pos zero_less_mult_iff) +
      have \delta t * speed(\omega (i+1)) + a_{max_brake} * \delta t^2/2 - b_1 = -(speed(\omega (i+1)) + a_{max_brake} * \delta t)^2/(2*a_{max_brake})
        using s1 apply (auto simp add: power2_eq_square)
      by linarith
  have h7: (~ ?b'_0 \leq -(speed(\omega i) + a_{min_brake} * \delta t)^2/(2*a_{min_brake}))
    using as6 a1 h1 h2 h3 apply (auto simp add: k0 r1 split:if_splits)
    apply (rule divide_right_mono_neg)
    using s6 s6' apply force
    by linarith
  with h5 s9 have h8: (~ ?b'_0 \leq pos(\omega' (i+1)) - pos (\omega (i+1)) - ?b_1)
6.4 Our motion is safe

by linarith

have h9: pos (\omega' i) - pos (\omega i) + \?b' - \?b_0 + \varepsilon' \leq pos (\omega (i+1)) - pos (\omega i)
  using dmin_rear_car_safe_motion dmin_rear_car_stopping h1 simp0 by force

with h8 have \varepsilon' \leq pos (\omega' (i+1)) - pos (\omega' i)
  by linarith

} note no_collision_if_fast_rear_car_not_breaking_not_stopping_and_front_car_may_not_stop=

have nocollision: \varepsilon' \leq d_{\text{real}} (\omega' i)
  apply (cases (?b'0 \geq ?b'1))
  using dmin apply simp
  apply (cases (?b0 \leq ?b1))
    using as3 as1 d0 d3 simp0 no_collision_if_slow_rear_car_nonstopping
    apply (cases (?a_0 > a_{\text{minbrake}}))
    using d0 no_collision_if_fast_rear_car_breaking_or_stopping_and_front_car_may_not_stop
    no_collision_if_fast_rear_car_not_breaking_not_stopping_and_front_car_may_not_stop
    no_collision_if_front_car_may_stop_but_dont simp0 apply force
    using d0 no_collision_if_fast_rear_car_breaking_or_stopping_and_front_car_may_not_stop
    no_collision_if_front_car_may_stop_but_dont by force

show \(d_{\text{min}} (\omega' i + \varepsilon') \leq d_{\text{real}} (\omega' i)\)
  using dmin nocollision by (simp add: simp0 d3)
qed

next

fix \omega

have \(P_{inv} \omega \Longrightarrow (i < j \land j \leq N) \longrightarrow pos (\omega i) + \varepsilon' \leq pos (\omega j)\) for \(i j\)
  unfolding d3 d0 P_{inv_def}
  apply (induct j, simp)
  by (smt (z3) Suc_leD Suc_le_lessD a4 add.commute c less_Suc_eq plus_1_eq_Suc)
with a4 c show \(P_{inv} \omega \Longrightarrow no_{\text{collision}} \omega \{0..N\} \varepsilon\)
  unfolding no_{collision_def} apply auto
  by (smt (verit, ccfv_threshold) le_neq_implies_less not_le)
qed

ded
e

end

end
7 Conclusion

7.1 Summary

7.1.1 Outline of the Theory Development

This deliverable consists of an integrated document used for the generation of this pdf-document. Validation and generation require an Isabelle2021 system [https://isabelle.in.tum.de/](https://isabelle.in.tum.de/) (see also: [https://en.wikipedia.org/wiki/Isabelle_(proof_assistant)](https://en.wikipedia.org/wiki/Isabelle_(proof_assistant)) for an introduction). The Isabelle system is open-source. The theory graph of this integrated document is shown in Figure 7.1. The shown theories belong to the following categories:

1. Isabelle System (open source): Pure, Tools, HOL, HOL-Library, HOL-Analysis, HOLCF, AFP-Regular-Sets, AFP.Functional-Automata, HOL-Eisbach

2. Background (Paris-Saclay): HOL-CSP, CSP_RefTK.CSP_ext, CSP_RefTK.CSP_ext, Isabelle_DOF.technical_report


7.1.2 General Results

As major results, we’d like to emphasize:

- A common Modelling Framework giving Semantics to MOSAR and Foretellix’s M-SDL
- ... which can handle all known scenarios such as single-lane, cut, crossing, while maintaining openness for new actor-models
- precise meanings for, e.g., concepts such as driving strategies, jerk, comfort, and driving objectives, ...
- precise meanings for, e.g., concepts such as scenario-classes
- proofs on abstract relations between driving strategies (allowing arguments such as "drive_{safe} is stronger as motion_{_s_s} but offers more comfort and practicability")
- proofs on safety of three driving strategies
- by-products of proofs:
Figure 7.1: The Graph of Library-, Background-, and Foreground Theories of this Study
1. a precise understanding of the discretization effects of the continuous, cyber-
physical models as occurring in simulators and concrete discrete controllers,
2. offer completeness for partitionings of scenario-classes, and therefore more trust
than just simulations, and
3. proofs offered insight into behaviour and gave inspirations to unexpected im-
provements.

- HOL-CSP provided reasonable analytic power by semi-automated proof in order to
formally prove key safety properties for a relevant driving strategy intensively discussed
in the literature. The work comprises 400 loc for RSS, 450 loc for a generalisation
RSS-N and an substantial improvement, RSS+ with 570 loc.

- The proof work alone comprises about 3 man months of work; future automation may
bring down this effort considerably.

- the proofs give rise to a case-distinction structure that result in abstract test-cases
that can be used for system tests within a certification (see Table 7.2, Table 7.3 and
Table 7.4).

- the proofs reveal the completeness of the case-distinction underlying the abstract test
cases which couldn’t be established otherwise.

- a run to generate this document (which includes: type-checking, proof-checking, on-
tological consistency-checking, LaTeX generation and .pdf generation) takes 55 secs
elapsed time on a common machine (MacBook Pro, 2,9 Ghz, 6 Core).

7.1.3 Abstract Test-Cases derived from the Proof-Structure

Our proof is based on induction. A simulation step $\delta t$ defines the induction step. We can
produce test cases following the case-splits we used in our proof. Therefore, our test cases are
one step tests but we can randomly complete them or even combine them to obtain many-
steps test cases. We will first give test cases for the standard RSS driving strategy $\text{drive}_{\text{rss}}$,
and then we will consider test cases for our new driving strategy $\text{drive}_{\text{saf}}$. Requiring a
smaller safety distance the safety proof of this latter requires more distinctions and will give
more test cases.

Abstract Test-Cases for driving strategy $\text{drive}_{\text{rss}}$

Extracting the structure of case-distinctions from our proof, there are two testing objectives:

1. the minimal distance is preserved $d_{\text{min}} = b_r - b_f$ that is the difference between the
rear-car breaking distance $b_r = (\text{speed} (\omega i))^2 / ((2::'a) * a_{\text{minbrake}})$ and the front
car breaking distance $b_f = (\text{speed} (\omega (i + (1::'c))))^2 / ((2::'a) * a_{\text{maxbrake}})$. 

71
7 Conclusion

2. no collision between the two cars when the previous distance is negative (i.e. \( b_r < b_f \))

**Minimal distance test cases:** all cases are exclusive.

<table>
<thead>
<tr>
<th>No</th>
<th>Test Case description</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>r-car stops</td>
<td>( \text{speed}(\omega_i) = 0 )  ( d_{pss} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>2</td>
<td>r-car stops</td>
<td>( \text{speed}(\omega_i) = 0 )  ( d_{pss} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) &gt; 0 )</td>
</tr>
<tr>
<td>3</td>
<td>r-car doesn’t stop</td>
<td>( \text{speed}(\omega_i) &gt; 0 )  ( d_{pss} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>4</td>
<td>r-car doesn’t stop</td>
<td>( \text{speed}(\omega_i) &gt; 0 )  ( d_{pss} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) &gt; 0 )</td>
</tr>
<tr>
<td>5</td>
<td>r-car stops</td>
<td>( \text{speed}(\omega_i) = 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>6</td>
<td>r-car stops</td>
<td>( \text{speed}(\omega_i) = 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i &gt; d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>7</td>
<td>r-car stops</td>
<td>( \text{speed}(\omega_i) = 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) &gt; 0 )</td>
</tr>
<tr>
<td>8</td>
<td>r-car stops</td>
<td>( \text{speed}(\omega_i) = 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i &gt; d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>9</td>
<td>r-car doesn’t stop</td>
<td>( \text{speed}(\omega_i) &gt; 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>10</td>
<td>r-car doesn’t stop</td>
<td>( \text{speed}(\omega_i) &gt; 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>11</td>
<td>r-car doesn’t stop</td>
<td>( \text{speed}(\omega_i) &gt; 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) = 0 )</td>
</tr>
<tr>
<td>12</td>
<td>r-car doesn’t stop</td>
<td>( \text{speed}(\omega_i) &gt; 0 )  ( d_{pss} \omega_i &gt; d_{\text{real}} \omega_i ) ( d_{\text{min}} \omega_i \leq d_{\text{real}} \omega_i ) ( \text{speed}(\omega^{(i+1)}) &gt; 0 )</td>
</tr>
</tbody>
</table>

Table 7.1: Table with Abstract Test Cases for minimal distance of \( \text{drive}_{rss} \)

**No collision:** cases are not exclusive (see Table 7.2)
7.2 Lessons Learnt

Abstract Test-Cases for RSS++ version \textit{drive}_{sa,fe}

Following our proof, there are again two testing objectives: minimal distance and no-collision. In the following and to avoid a long enumeration, we do not specify the behavior of the front car for many abstract test cases. The reader can derive specific test cases by selecting any behavior for the front car (e.g. breaking, slowing, accelerating, stopping...)

\textbf{Minimal distance test cases:} (see Table 7.3).

\textbf{No collision:} (see Table 7.4)

7.2 Lessons Learnt

- The present study confirms that HOL-CSP is an adequate framework to give semantics to MOSAR- and similar models as used in the Automotive Car industry.

- More generally speaking, Isabelle/HOL-CSP — as being parametric for arbitrary HOL-types for events — is a suitable framework for modeling and analysing scenarios of cyber-physical systems involving both physical environments, communication, and discretisation problems of controllers.

- It provides distinctively more expressive power than conventional model-checking techniques such as, e.g., or PRISM [5], Hybrid CSP ([6], never implemented) or Hybrid Automata (numerous publications by Henzinger et al, but no usable tools).

Concerning the formal analysis of RSS, our study revealed a number of shortcomings in the existing description and analysis in [9].

We distinguish two lines of criticism: A \textit{local critique} concerning modeling details of RSS and its safety proof, and a \textit{global critique} on RSS concerning applicability and the possible generalizations that the authors claim.

Our analysis revealed the following local problems, mostly due to the implicit assumption that all relevant functions are linear between $t$ and $t + \varrho$:

- Case not treated in paper-and-pencil proof: if b-car behind, f-car in front, and if b is braking, and if f is accelerating, and if b-car behind and f-car in front at $t + \varrho$ there will be no collision. Counter-example: \textbf{Figure 7.2} (left).

- Case not treated in paper-and-pencil proof: if b-car behind, f-car in front, and if b is braking, and if f is braking, and if b-car behind and f-car in front at $t + \varrho$ there will be no collision. Counter-example: \textbf{Figure 7.2} (right).

- Case insufficiently treated: inflexion point (singularity) for $v_r = 0$, e.g. \textbf{Figure 7.3} (left).

- General solution: distinguishing $\varrho$ (the "reaction time" of the autonomous vehicle) from the sampling rate $\delta t$ which must be small compared to $\varrho$ in a simulator and infinitesimal in the proof (see \textbf{Figure 7.3} (right)).
7 Conclusion

(a) b-car braking, f-car accelerating.  

\[ b \text{-car braking, f\text{-car braking.} } \]

Figure 7.2: Collision scenarios overlooked.

Both phenomena — singularities as well as the need for an explicit sampling allowing for the construction of majorands and minorands — complicate the argument drastically and bring it to the limit of precision in usual paper-and-pencil proofs. However, we believe that the introduction of the sampling is inherently necessary for refinements towards concrete implementations.

Concerning the global points of critique:

- the distances in a realistic configuration of RSS (speeds 133 km/h, $v=0,1s$) results in nearly 60 m minimal distance ... (by courtesy of Paolo Crisafulli).

- given the large security distances, the argument of the RSS authors: "RSS can also handle curved street topologies and obstacles" is doubtful.

We believe that distances to side-walks need an own treatment in the driving strategy. More refined driving strategies will be out of reach of paper-and-pencil proofs and require a formal approach as the one discussed.

7.3 Future Directions

- Compilers from Foretellix’s M-SDL or MOSAR to our Framework
7.3 Future Directions

(a) hand-waving: singularities.  
(b) Solution: Explicit, infinitesimal sampling.

Figure 7.3: Phenomena overlooked.
7 Conclusion

- Simulators generated from sufficiently complete scenario instances
- Proofs on RSS are still one-dimensional (but an extension to 2-dimensional case is straight-forward similar to the original paper).
- More driving strategies taking into account jerk (and therefore comfort of the passengers)
- Going beyond safety properties towards lifeness properties (can the objective of the journey still be reached ?)
- Driving strategies in our model are still functions in the continuous real-time-space underlying Newtonian physics. Isabelle/HOL and HOL-CSP allow for modeling and refinement proofs towards concrete test-diver implementation in, for example, C, modeling concrete calculations on machine-types such as bit-vectors.
<table>
<thead>
<tr>
<th>No</th>
<th>Test Case description</th>
<th>Conds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>r-car stops</td>
<td>speed(\omega'(i) = 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>f-car stops</td>
<td>speed(\omega'(i+1) = 0)</td>
</tr>
<tr>
<td>2</td>
<td>r-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>f-car stops</td>
<td>speed(\omega'(i+1) = 0)</td>
</tr>
<tr>
<td>3</td>
<td>r-car stops</td>
<td>speed(\omega'(i) = 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>minimal distance</td>
<td>(d_{min} \omega i &lt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car very slow (could stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t \leq 0)</td>
</tr>
<tr>
<td>4</td>
<td>r-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car very slow (could stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t \leq 0)</td>
</tr>
<tr>
<td>5</td>
<td>r-car stops</td>
<td>speed(\omega'(i) = 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>minimal distance</td>
<td>(d_{min} \omega i &lt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car very slow (could stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t \leq 0)</td>
</tr>
<tr>
<td>6</td>
<td>r-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>non-minimal distance</td>
<td>(d_{min} \omega i &lt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car very slow (could stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t \leq 0)</td>
</tr>
<tr>
<td>7</td>
<td>r-car stops</td>
<td>speed(\omega'(i) = 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>f-car stops</td>
<td>speed(\omega'(i+1) = 0)</td>
</tr>
<tr>
<td>8</td>
<td>r-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>f-car stops</td>
<td>speed(\omega'(i+1) = 0)</td>
</tr>
<tr>
<td>9</td>
<td>r-car stops</td>
<td>speed(\omega'(i) = 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td>10</td>
<td>r-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>f-car stops</td>
<td>speed(\omega'(i+1) = 0)</td>
</tr>
<tr>
<td>11</td>
<td>r-car stops</td>
<td>speed(\omega'(i) = 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>minimal distance</td>
<td>(d_{min} \omega i &lt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car not very slow (could not stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t &gt; 0)</td>
</tr>
<tr>
<td>12</td>
<td>r-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>minimal distance</td>
<td>(d_{min} \omega i &lt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car not very slow (could not stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t &gt; 0)</td>
</tr>
<tr>
<td>13</td>
<td>r-car stops</td>
<td>speed(\omega'(i) = 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>minimal distance</td>
<td>(d_{min} \omega i &lt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car not very slow (could not stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t &gt; 0)</td>
</tr>
<tr>
<td>14</td>
<td>r-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car breaking faster</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>non-safe distance</td>
<td>(d_{RSS} \omega i &gt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>minimal distance</td>
<td>(d_{min} \omega i &lt; d_{real} \omega i)</td>
</tr>
<tr>
<td></td>
<td>f-car doesn't stop (breaking or accelerating)</td>
<td>speed(\omega'(i+1) &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>f-car not very slow (could not stop)</td>
<td>speed(\omega'(i+1) - a_{maxbrake} \delta t &gt; 0)</td>
</tr>
</tbody>
</table>

Table 7.2: Table with Abstract Test Cases for no-collision of drive_{RSS}

7.3 Future Directions
<table>
<thead>
<tr>
<th>No</th>
<th>Test Case description</th>
<th>Conds</th>
</tr>
</thead>
</table>
| 1  | r-car stops           | speed(\omega_i) = 0 \\
|    | safe distance         | \text{acc}(\omega_i) + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car stops           | speed(\omega'_i) = 0 |
| 2  | r-car stops           | speed(\omega_i') = 0 \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 |
| 3  | r-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 |
| 4  | r-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 \\
|    | f-car stops           | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 |
| 5  | r-car stops           | speed(\omega_i') = 0 \\
|    | r-car was breaking    | acc(\omega_i') \leq a_{\text{min}} \omega_i \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car stops           | speed(\omega_i') = 0 |
| 6  | r-car stops           | speed(\omega_i') = 0 \\
|    | r-car was breaking    | acc(\omega_i') \leq a_{\text{min}} \omega_i \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 |
| 7  | r-car stops           | speed(\omega_i') = 0 \\
|    | r-car was not breaking (slowing) | acc(\omega_i') > a_{\text{min}} \omega_i \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car stops           | speed(\omega_i') = 0 |
| 8  | r-car stops           | speed(\omega_i') = 0 \\
|    | r-car was not breaking but slowing | acc(\omega_i') > a_{\text{min}} \omega_i \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i \\
|    | f-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 |
| 9  | r-car stops           | speed(\omega_i') > 0 \\
|    | r-car was breaking    | acc(\omega_i') \leq a_{\text{min}} \omega_i \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i |
| 10 | r-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 \\
|    | r-car accelerates     | acc(\omega_i') \geq 0 \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i |
| 11 | r-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 \\
|    | r-car was not breaking but slowing | a_{\text{min}} \omega_i < acc(\omega_i') \leq 0 \\
|    | r-car would stop after response time | speed(\omega_i') + \tau_{\text{brake}} \omega_i \leq 0 \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i |
| 12 | r-car doesn’t stop (breaking or accelerating) | speed(\omega_i') > 0 \\
|    | r-car was not breaking but slowing | a_{\text{min}} \omega_i < acc(\omega_i') \leq 0 \\
|    | r-car would not stop after response time | speed(\omega_i') + \tau_{\text{brake}} \omega_i > 0 \\
|    | safe distance         | \text{acc}(\omega_i') + \epsilon' \leq d_{\text{real}} \omega_i |

Table 7.3: Table with Abstract Test Cases for minimal distance of \text{drive}_{\text{safe}} (RSS++)
### 7.3 Future Directions

Table 7.4: Table with Abstract Test Cases for no-collision of drive\textsubscript{safe} (RSS++)

<table>
<thead>
<tr>
<th>No</th>
<th>Test Case description</th>
<th>Conds</th>
</tr>
</thead>
</table>
| 1  | r-car stops           | speed(\omega' i) = 0  
|    | f-car breaking faster | |  
|    | f-car stops           | \|b_f < b_r\|  
|    |                       | speed(\omega'(i+1)) = 0 |
| 2  | r-car doesn’t stop (breaking or accelerating) | speed(\omega' i) > 0  
|    | f-car breaking faster | |  
|    | f-car stops           | \|b_f < b_r\|  
|    |                       | speed(\omega'(i+1)) = 0 |
| 3  | f-car breaking faster | \|b_f < b_r\|  
|    | safe distance         | \omega_i \leq d_{safe} \omega i  
|    | f-car doesn’t stop (breaking or accelerating) | speed(\omega'(i+1)) > 0  
|    | f-car very slow (could stop) | speed(\omega'(i+1)) + a_{max\_brake} \delta t \leq 0 |
| 4  | f-car breaking faster | \|b_f < b_r\|  
|    | non-safe distance     | \omega_i > d_{safe} \omega i  
|    | minimal distance      | \omega_i \leq d_{safe} \omega i  
|    | f-car doesn’t stop (breaking or accelerating) | speed(\omega'(i+1)) > 0  
|    | f-car very slow (could stop) | speed(\omega'(i+1)) - a_{max\_brake} \delta t \leq 0 |
| 5  | r-car stops           | speed(\omega'(i)) = 0  
|    | f-car breaking faster | |  
|    | f-car doesn’t stop    | \|b_f < b_r\|  
|    | f-car not very slow (could not stop) | speed(\omega'(i+1)) > 0  
|    |                       | speed(\omega'(i+1)) - a_{max\_brake} \delta t > 0 |
| 6  | r-car doesn’t stop    | speed(\omega'(i)) > 0  
|    | r-car was breaking    | acc(\omega' i) \leq a_{min\_brake}  
|    | f-car breaking faster | \|b_f < b_r\|  
|    | f-car doesn’t stop    | \|b_f < b_r\|  
|    | f-car not very slow (could not stop) | speed(\omega'(i+1)) > 0  
|    |                       | speed(\omega'(i+1)) - a_{max\_brake} \delta t > 0 |
| 7  | r-car doesn’t stop    | speed(\omega'(i)) > 0  
|    | r-car was not breaking (slowing or accelerating) | acc(\omega' i) > a_{min\_brake}  
|    | f-car breaking faster | \|b_f < b_r\|  
|    | f-car doesn’t stop    | speed(\omega'(i+1)) > 0  
|    | f-car not very slow (could not stop) | speed(\omega'(i+1)) - a_{max\_brake} \delta t > 0 |
Bibliography


