



**HAL**  
open science

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

Paolo Crisafulli, Safouan Taha, Burkhart Wolff

► **To cite this version:**

Paolo Crisafulli, Safouan Taha, Burkhart Wolff. Modelling and Proving Safety in Autonomous Cars Scenarios in HOL-CSP. [Research Report] 1, University Paris-Saclay; IRT SystemX, Palaiseau. 2021, pp.81. hal-03429597v2

**HAL Id: hal-03429597**

**<https://hal.inria.fr/hal-03429597v2>**

Submitted on 1 Dec 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# 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.

This document has been generated by Isabelle/DOF version Unreleased/Isabelle2021.

**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

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>The CSP-based Generic Autonomous Car Model</b>	<b>11</b>
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
<b>3</b>	<b>From MOSAR-Ontologies to a Semantics in CSP</b>	<b>25</b>
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
<b>4</b>	<b>A Safety-Property in Autonomous Cars: RSS in a 2-cars-Scenario</b>	<b>33</b>
4.1	Global Parameters of the Scenario-Class . . . . .	33
4.2	Relations between Driving Strategies . . . . .	34
<b>5</b>	<b>A Safety-Property in Autonomopus Cars: RSS in a N-cars-Scenario</b>	<b>43</b>
5.1	Global Parameters of the Scenario-Class . . . . .	43
5.2	Relations between Driving Strategies . . . . .	43
<b>6</b>	<b>An improved RSS-alike driving strategy: RSS-plus in a N-cars-Scenario</b>	<b>53</b>
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
<b>7</b>	<b>Conclusion</b> . . . . .	<b>69</b>
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\\_\(proof\\_assistant\)](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.

## *Contents*

# 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 <sup>1</sup> or Foretellix's M-SDL <sup>2</sup>. 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" 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 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

---

<sup>1</sup><https://www.mosar.io>

<sup>2</sup><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](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:

```
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*<sup>1</sup> 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 word*  $\times$  *64 word* over 64-bit-vectors as occurring in concrete calculation in a programming language.

The only compromise that this choice implies is that *real\_normed\_vector* comes with a scalar multiplication<sup>2</sup>  $a *_R x$  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  $(1::'a) / 2 *_R t^2 *_R a$  or even  $\text{sqrt } 2 *_R a$  (as occurring naturally in norms of 2-dimensional vectors).

---

<sup>1</sup>[https://en.wikipedia.org/w/index.php?title=Parametric\\_polymorphism&oldid=1038893302](https://en.wikipedia.org/w/index.php?title=Parametric_polymorphism&oldid=1038893302)

<sup>2</sup>[https://en.wikipedia.org/w/index.php?title=Scalar\\_multiplication&oldid=1039985947](https://en.wikipedia.org/w/index.php?title=Scalar_multiplication&oldid=1039985947)

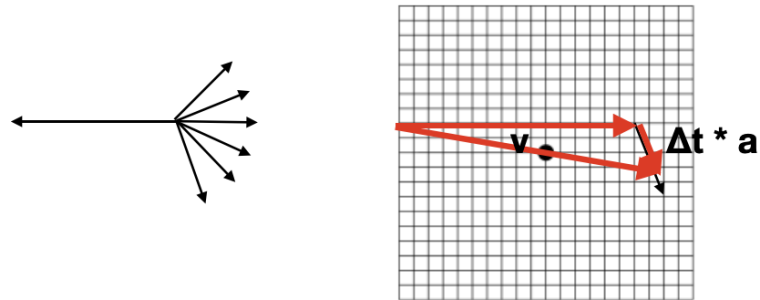


Figure 2.1: Generic Vector-Spaces in the Autonomous Car Modeling Framework.

## 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  $'v$  which is constrained to the type-class *real\_normed\_vector*.

*record* ( $'v::real\_normed\_vector$ ) *as* = — for actor state

<i>pos</i> :: $'v$	— current position
<i>speed</i> :: $'v$	— current speed
<i>acc</i> :: $'v$	— current acceleration

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  $\tau_1 \times \dots \times \tau_n \times \alpha$  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  $'v \times 'v \times 'v \times \alpha$  with the projection functions:

1.  $pos \equiv fst$
2.  $speed \equiv fst \ o \ snd$
3.  $acc \equiv fst \ o \ snd \ o \ snd$

4. and the implicit projection on the *more*-field:  $more \equiv fst \circ snd \circ snd \circ snd$  (where *fst* and *snd* are the usual projections into a binary cartesian product and *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  $(\lfloor pos = a, speed = b, acc = c, \dots = more \rfloor)$ , and the corresponding projection functions were defined to work directly on this notation. Isabelle also introduces a notation for record-types which were written  $(\prime v, \prime \alpha) as\_scheme$  for which the synonym  $(\prime v, \prime a) as\_scheme$  is generated.

Note that Isabelle also supports the equivalent notation:  $(\lfloor pos = a, speed = b, acc = c \rfloor)$  for  $(\lfloor pos = a, speed = b, acc = c, \dots = () \rfloor)$ . Here, the  $()$  is the "empty-product", i. e. the only element in the so-called *unit*-type. This corresponds to the idea of a *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  $(\prime v, \prime a) as\_scheme$  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:  $\prime v as$  or equivalently:  $\prime v as$ ) or an open state-space: (so:  $(\prime v, \prime \alpha) as\_scheme$  or equivalently:  $(\prime v, \prime \alpha) as\_scheme$ ). In order to bridge a little the gap between these two forms we introduce by convention the following *type\_synonym*'s for each actor-state class:

*type\_synonym*  $(\prime v, \prime \alpha) as\_ = (\prime v, \prime \alpha) as\_scheme$

#### Examples of Actor Instances:

*term*  $(pos :: (\prime v :: real\_normed\_vector, \prime \alpha) as\_ \Rightarrow \prime v)$

*term*  $(\lfloor pos = 14, speed = 2, acc = -1 \rfloor :: real as)$

— One-dimensional *real as*.

*term*  $(\lfloor pos = (14, 15), speed = (sqrt\ 2, 2.4), acc = (0, 0), \dots = a \rfloor :: (real \times real, \prime \alpha) as\_)$

— Two-dimensional, extensible actor state space of type  $(real \times real, \prime \alpha) as\_$ .

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:

*record*  $(\prime v :: real\_normed\_vector) as_{range} = (\prime v as) +$

*acc\_range*  $:: (\prime v set)$  — 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.

*extension\_field*  $:: (\prime v set)$  — gives the expansion of the physical object in terms of a set of positions (relative vectors).

Again, following our convention, we provide an abbreviation for the "open" version of this actor state space:

## 2 The 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**  $wd_{as} :: ((v::real\_normed\_vector, a) as_{range\_} \Rightarrow bool)$

**where**  $(wd_{as} \sigma \equiv acc \sigma \in acc\_range \sigma)$

**corollary**  $\langle wd_{as} (\mid pos = (14::real), speed = 2,$   
 $acc = -1, acc\_range = \{-1..1\}, extension\_field = \{\}) \rangle$

**and**  $\langle \neg wd_{as} (\mid pos = (14::real), speed = 2,$   
 $acc = -1, acc\_range = \{0..1\}, extension\_field = \{\}) \rangle$

**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**  $v scene$  =  $(id_{actor} \Rightarrow v as)$

— a scene where all objects are identified by their index

**type\_synonym**  $(v, a) scene\_$  =  $(id_{actor} \Rightarrow (v, a) as\_)$

— ... the "open" version

Note that the above model of an "Open Scene" is not that open after all: while it encompasses 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**  $(v, a) scene_o = global\_state :: id_{actor} \Rightarrow (v, a) as_{range\_}$   
 — a record with a single field named *global\_state*

**type\_synonym**  $(v, a, \beta) scene_o\_ = (v, a, \beta) scene_o\_scheme$   
 — 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  $\omega$  be a  $v scene$ . Then we provide a predicate  $no_{collision} \omega$  which is true when all actors have relative distances  $\varepsilon$ .

**definition**  $no_{collision} :: ((v::real\_normed\_vector) scene \Rightarrow id_{actor} set \Rightarrow v \Rightarrow bool)$

— when actors should keep some  $\varepsilon$  as distance from each other

**where**  $\langle no_{collision} \omega sid \varepsilon \equiv \forall i \in sid. \forall j \in sid. norm (pos(\omega j) - pos(\omega i)) \leq norm \varepsilon \longrightarrow i = j \rangle$

**corollary**  $no_{collision\_}\varepsilon\_0: \langle no_{collision} \omega sid 0 = inj\_on (\lambda i. pos (\omega i)) sid \rangle$

— when actors are points we can set  $\varepsilon$  to zero

**unfolding**  $inj\_on\_def no_{collision\_} def by auto$

## 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.

*type\_synonym* *time* = *real*

*type\_synonym*  $\langle 'v \text{ motion} = \langle id_{actor} \Rightarrow time \Rightarrow 'v \text{ scene} \Rightarrow 'v \text{ as set} \rangle$

— function returning the new set of states for a given actor, given an actor id, a  $\delta t$  and an initial scene.

*type\_synonym*  $\langle ('v, 'a) \text{ motion\_} = \langle id_{actor} \Rightarrow time \Rightarrow ('v, 'a) \text{ scene\_} \Rightarrow ('v, 'a) \text{ as\_ set} \rangle$

— ... and the "open" version.

*lemma* *motion\_step*:

**assumes**  $\langle \forall i \in sid. \text{motion } i \ \delta t \ \omega \neq \{\} \rangle$

— assume motion returns no empty sets of actor states on *sid*

**shows**  $\langle \exists \omega'. \forall i \in sid. \omega' \ i \in \text{motion } i \ \delta t \ \omega \rangle$

— then there is some scene  $\omega'$  corresponding to the result (image set) of a motion

**by**  $\langle \text{exl } \langle \lambda i. (\text{SOME } s. s \in \text{motion } i \ \delta t \ \omega) \rangle \rangle$

— this  $\lambda$  returns some actor state for every actor id: it is a scene.

— hence, we have built a scene, showing the lemma: *qed*

(*simp* *add*: *assms* *some\_in\_eq*)

## 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*  $\langle 'v \text{ events} = \text{sync } (dt: \text{real})(sc: \langle 'v \text{ scene} \rangle) \text{ — simulation step}$

| *up*  $\langle 'v \text{ scene} \rangle \text{ — update}$

*abbreviation*  $\langle \text{sync}_{pair} \equiv \lambda(\delta t, \omega). \text{sync } \delta t \ \omega \rangle$

— another way to denote the constructor *events.sync*

— proof tactic to break the cartesian product, not relevant

*lemma* *sync\_pair\_simp*[*simp*]:

$\langle \text{Mprefix } (\text{range } \text{sync}_{pair}) \ (P \circ \text{inv } \text{sync}_{pair}) = \text{Mprefix } (\text{range } \text{sync}_{pair}) \ (\lambda e_{\text{sync}}. P \ (dt \ e_{\text{sync}}, \text{sc } e_{\text{sync}})) \rangle$

**by**  $\langle \text{smt } (z3) \ \text{case\_prod\_beta}' \ \text{comp\_apply } \text{events.sel}(1) \ \text{events.sel}(2) \ \text{f\_inv\_into\_f } \text{mprefix\_eq } \text{prod.collapse} \rangle$

### 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*  $\mu_{\text{motion}} :: \langle id_{actor} \ \text{set} \Rightarrow 'v \ \text{motion} \Rightarrow ('v::\text{real\_normed\_vector}) \ \text{events } \text{process} \rangle$   
 $\langle \llbracket \_ \rrbracket : \_ \rrbracket \rangle$

**where**  $\langle \mu_{\text{motion}} \ \text{sid } \text{motion} \equiv \mu \ X. \ \text{sync}_{pair}?( \delta t, \omega) \rightarrow (\text{up!}\omega' | (\forall i \in \text{sid}. \omega' \ i \in \text{motion } i \ \delta t \ \omega) \rightarrow X) \rangle$

The "usual" recursive lemma, used for simplification and fixpoint reasoning:

*lemma*  $\mu_{\text{motion\_rec}}: \langle \llbracket \text{sid}:\text{motion} \rrbracket = \text{sync}_{pair}?( \delta t, \omega) \rangle$



## 2 The CSP-based Generic Autonomous Car Model

```

→ (up!ω' | (∀ i ∈ sid. ω' i ∈ motion i δt ω)
→ [[sid:motion]])
by (simp add: μmotion_def, subst fix_eq, simp)

```

This lemma allows to reduce CSP refinement proofs to "denotational" ones: if motion 1 is included in motion 2, then CSP1 refines CSP2

*lemma*  $\mu_{motion\_refine}$ :

```

assumes a: (∀ δt ω i. (i ∈ sid ∧ motion2 i δt ω ≠ {})) → (motion1 i δt ω ≠ {})
and b: (∀ δt ω i. i ∈ sid → motion1 i δt ω ⊆ motion2 i δt ω)
shows ⟨[[sid:motion2]] ≤ [[sid:motion1]]⟩
proof(subst μmotion_def, induct rule: fix_ind)
  show ⟨adm (λa. a ≤ [[sid:motion1]])⟩ by (simp_all add: monofunl)
next
  show ⟨⊥ ≤ [[sid:motion1]]⟩ by simp
next
  case induct_case:(3 x)
  have c:( (Πe ∈ up ' {ω'. ∀ i ∈ sid. ω' i ∈ motion2 i δt ω} → x)
    ≤ (Πe ∈ up ' {ω'. ∀ i ∈ sid. ω' i ∈ motion1 i δt ω} → x) for δt ω
  apply (cases (∃ i ∈ sid. motion2 i δt ω = {}))
  apply (metis (no_types, lifting) Collect_empty_eq bot.extremum_uniquel empty_iff eq_iff)
  apply(rule Mndetprefix_FD_subset) apply (auto simp:a) apply(exl (λi. SOME s. s ∈ motion1 i
δt ω))
  apply (simp add: a some_in_eq)
  using b by blast
show ?case
  apply(subst μmotion_rec, simp add:c read_def)
  apply(rule mono_mprefix_FD)
  using order_trans[OF c] by (meson induct_case.hyps mono_Mndetprefix_FD)
qed

```

### 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 than 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,

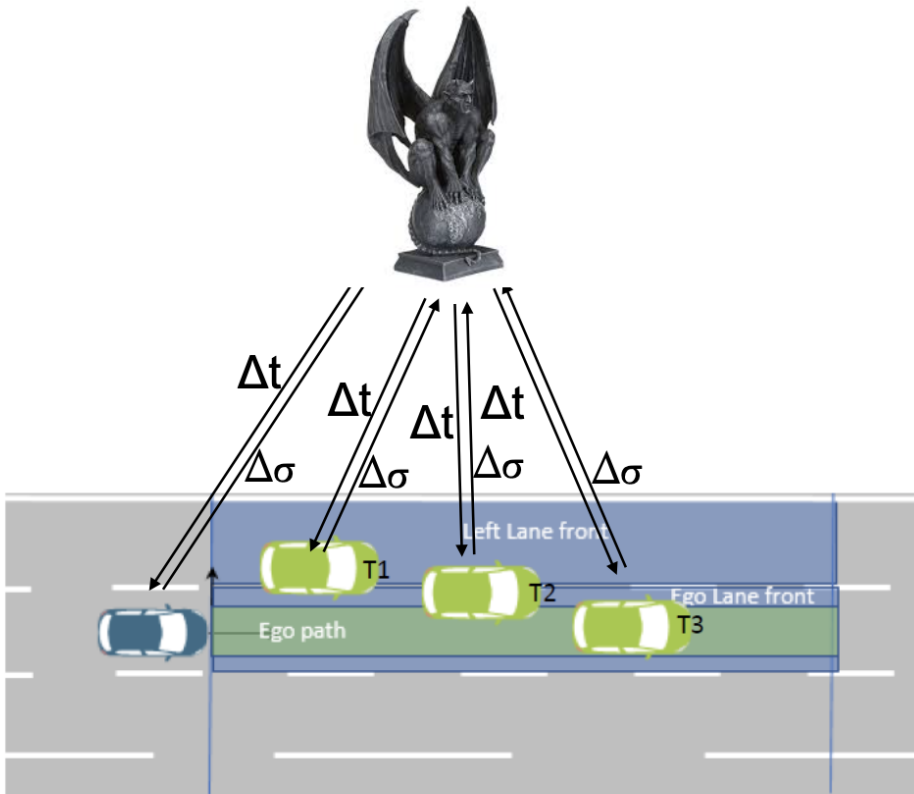


Figure 2.2: The Behaviour of the Orchestrator (The "Maxwell Demon").

## 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* :: (*real*  $\Rightarrow$  (*v*::*real\_normed\_vector*) *scene*  $\Rightarrow$  '*v* *events process*)  
**where** (*demon*  $\Delta t \equiv \mu X. (\lambda \omega. (sync_{pair}!(\delta t, \omega_s)|(\delta t \in \{0 < .. \Delta t\} \wedge \omega_s = \omega) \rightarrow (up?\omega' \rightarrow X \omega'))$ ))

Here,  $\{0 < .. \Delta t\}$  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*:  
(*demon*  $\Delta t \omega = (sync_{pair}!(\delta t, \omega_s)|(\delta t \in \{0 < .. \Delta t\} \wedge \omega_s = \omega) \rightarrow (up?\omega' \rightarrow demon \Delta t \omega'))$ )  
**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* ::  
(*id\_actor set*  $\Rightarrow$  (*v*::*real\_normed\_vector*) *motion*  $\Rightarrow$  *real*  $\Rightarrow$  '*v* *scene*  $\Rightarrow$  '*v* *events process*)  
**where** (*scenario* *sid motion*  $\Delta t \omega_0 \equiv (demon \Delta t \omega_0 \parallel \llbracket sid:motion \rrbracket \setminus \{e. \neg is\_sync\ e\})$ )

A safe (instantiated) scenario must refine *safe* (once instantiated with actor ids and an  $\varepsilon$  value).

**definition** *safe* :: (*id\_actor set*  $\Rightarrow$  (*v*::*real\_normed\_vector*)  $\Rightarrow$  '*v* *events process*)  
**where** (*safe* *sid*  $\varepsilon \equiv \mu X. (sync_{pair}!(\delta t, \omega_s)|(\delta t > 0 \wedge no_{collision} \omega_s \ sid \ \varepsilon) \rightarrow X)$ )

**definition** *is\_safe* **where** (*is\_safe* *sid motion*  $\Delta t \omega_0 \ \varepsilon \leftrightarrow safe \ sid \ \varepsilon \leq scenario \ sid \ motion \ \Delta t \ \omega_0$ )

## 2.5.6 Scenario Refinement

monotony of  $is_{safe}$ : stricter driving strategies inherit safety from less rigorous ones ...

**lemma**  $is_{safe\_refine}$ :

**assumes**  $a: \llbracket sid:motion_2 \rrbracket \leq \llbracket sid:motion_1 \rrbracket$

**and**  $b: (is_{safe} \ sid \ motion_2 \ \Delta t \ \omega_0 \ \varepsilon)$

**shows**  $(is_{safe} \ sid \ motion_1 \ \Delta t \ \omega_0 \ \varepsilon)$

**proof** –

**have**  $(scenario \ sid \ motion_2 \ \Delta t \ \omega_0 \leq scenario \ sid \ motion_1 \ \Delta t \ \omega_0)$

**by**  $(simp \ add:scenario\_def \ a)$

**then show**  $?thesis$

**by**  $(meson \ b \ is_{safe\_def} \ order\_trans)$

**qed**

**lemma**  $scenario\_rec$ :

$(scenario \ sid \ motion \ \Delta t \ \omega = (\prod_{e_{sync} \in \{sync \ \delta t \ \omega \mid \delta t. \ \delta t \in \{0 < .. \Delta t\}\}}$   
 $\rightarrow (\prod_{\omega' \in \{\omega'. \ \forall i \in sid. \ \omega' \ i \in motion \ i \ (dt \ e_{sync}) \ \omega\}}.$   
 $scenario \ sid \ motion \ \Delta t \ \omega'))$

**apply**  $(subst \ scenario\_def, \ subst \ demon\_rec, \ subst \ \mu_{motion\_rec})$

**apply**  $(simp \ add:read\_def, \ subst \ gndet\_mprefix\_sync) \ apply \ (simp \ add: \ image\_mono)$

**apply**  $(subst \ sync\_commute, \ subst \ gndet\_mprefix\_sync) \ apply \ (simp \ add: \ image\_mono)$

**apply**  $(subst \ Mndet\_Hide1) \ apply \ fastforce$

**apply**  $(subst \ Mndet\_Hide2) \ apply \ fastforce$

**proof**  $(goal\_cases)$

**case** 1

**have**  $a: (sync\_pair \ ' \ {(\delta t, \ \omega_s). \ 0 < \delta t \wedge \delta t \leq \Delta t \wedge \omega_s = \omega}$   
 $= \{events.sync \ \delta t \ \omega \mid \delta t. \ 0 < \delta t \wedge \delta t \leq \Delta t\}) \ by \ auto$

**have**  $b: ((inv \ events.up \ (events.up \ x)) = x) \ for \ x::(nat \Rightarrow \ 'v \ as)$

**by**  $(metis \ events.inject(2) \ f\_inv\_into\_f \ rangel)$

**show**  $?case$

**apply**  $(subst \ a)$

**apply**  $(intro \ mndet\_eq \ all \ impl, \ auto, \ rule \ gndet\_eq2, \ auto, \ subst \ b)$

**by**  $(simp \ add: \ par\_comm \ scenario\_def)$

**qed**

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

**lemma**  $is_{safe\_rule1}$ :

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

**and**  $b: (no\_collision \ \omega_0 \ sid \ \varepsilon)$

**and**  $c: (\bigwedge \omega \ \delta t. \ no\_collision \ \omega \ sid \ \varepsilon \wedge \delta t \in \{0 < .. \Delta t\}$

$\implies let \ A = \{\omega'. \ \forall i \in sid. \ \omega' \ i \in motion \ i \ \delta t \ \omega\}$

$in \ A \neq \{\} \wedge (\forall \omega' \in A. \ no\_collision \ \omega' \ sid \ \varepsilon)$

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

## 2 The CSP-based Generic Autonomous Car Model

$no_{collision} \ ?\omega \ sid \ \varepsilon \ \wedge \ ?\delta t \in \{0 < .. \Delta t\} \implies \text{let } A = \{\omega'. \forall i \in sid. \omega' \ i \in \text{motion } i \ ?\delta t \ ?\omega\} \text{ in } A \neq \{\} \wedge (\forall \omega' \in A. no_{collision} \ \omega' \ sid \ \varepsilon)$  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:(\Pi \_ \in sync_pair ' (Collect ((<) 0) \times \{\omega_s, no_{collision} \omega_s \ sid \ \varepsilon\}) \to x \le
    \Pi e_{sync} \in \{events.sync \ \delta t \ \omega_0 \ | \ \delta t. 0 < \delta t \ \wedge \ \delta t \le \Delta t\} \to x)
  apply(rule Mndetprefix_FD_subset)
  using 3 a c by auto
  have \delta t \in \{0 < .. \Delta t\} \implies x \le gndet \{\omega'. \forall i \in sid. \omega' \ i \in \text{motion } i \ \delta t \ \omega_0\} (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 *is\_safe\_rule1*

**lemma** *is\_safe\_rule2*:

```

assumes a:(\Delta t > 0)
  and b:(no_{collision} \omega_0 \ sid \ \varepsilon)
  and c:(\bigwedge \omega \ \delta t. no_{collision} \ \omega \ sid \ \varepsilon \ \wedge \ \delta t \in \{0 < .. \Delta t\}
    \implies \forall i \in sid. \text{motion } i \ \delta t \ \omega \neq \{\})
  and d:(\bigwedge \omega \ \delta t. no_{collision} \ \omega \ sid \ \varepsilon \ \wedge \ \delta t \in \{0 < .. \Delta t\}
    \implies \forall \omega' \in \{\omega'. \forall i \in sid. \omega' \ i \in \text{motion } i \ \delta t \ \omega\}. no_{collision} \ \omega' \ 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_inv::((nat \implies ('v::real_normed_vector) as) \implies nat set \implies bool)
— let P_inv be a predicate
assumes a:(\Delta t > 0)
  and b:(P_inv \omega_0 \ sid)
  — assume P_inv is true for scene \omega_0
  and c:(\bigwedge \omega \ \delta t. P_inv \ \omega \ sid \ \wedge \ \delta t \in \{0 < .. \Delta t\} \implies \forall i \in sid. \text{motion } i \ \delta t \ \omega \neq \{\})
  — given a non empty motion
  and d:(\bigwedge \omega \ \delta t. P_inv \ \omega \ sid \ \wedge \ \delta t \in \{0 < .. \Delta t\}
    \implies \forall \omega' \in \{\omega'. \forall i \in sid. \omega' \ i \in \text{motion } i \ \delta t \ \omega\}. P_inv \ \omega' \ sid)

```

```

— which preserves  $P_{inv}$ 
and  $e: (\wedge \omega. P_{inv} \omega sid \implies no_{collision} \omega sid \varepsilon)$ 
— assume  $P_{inv}$  implies no collision
shows  $\langle is_{safe} sid motion \Delta t \omega_0 \varepsilon \rangle$ 
— then the motion is safe
using  $b$ 
proof( $unfold is_{safe\_def} safe\_def, induct arbitrary:\omega_0 rule:fix\_ind$ )
  show  $adm (\lambda a. \forall x. P_{inv} x sid \longrightarrow P_{inv} x sid \longrightarrow a \leq scenario sid motion \Delta t x)$ 
  by ( $simp add: monofun\_def$ )
next
  fix  $\omega_0::nat \Rightarrow 'v as$ 
  assume  $P_{inv} \omega_0 sid$  and  $P_{inv} \omega_0 sid$ 
  then show  $\perp \leq scenario sid motion \Delta t \omega_0$ 
  by  $simp$ 
next
  fix  $x$  fix  $\omega_0::nat \Rightarrow 'v as$ 
  assume  $hyp: (\wedge \omega_0. P_{inv} \omega_0 sid \implies P_{inv} \omega_0 sid \implies x \leq scenario sid motion \Delta t \omega_0)$ 
  and  $base: P_{inv} \omega_0 sid$ 
  have  $d: (\sqcap \_ \in sync\_pair' (Collect ((< 0) \times \{\omega_s. no_{collision} \omega_s sid \varepsilon\}) \rightarrow$ 
     $x \leq \sqcap e_{sync} \{events.sync \delta t \omega_0 \mid \delta t. 0 < \delta t \wedge \delta t \leq \Delta t\} \rightarrow x)$ 
  apply( $rule Mndetprefix\_FD\_subset$ ) using  $a e[of \omega_0, OF base]$  by  $auto$ 

  have  $e: \delta t \in \{0 <.. \Delta t\}$ 
   $\implies x \leq gndet \{\omega'. \forall i \in sid. \omega' i \in motion i \delta t \omega_0\} (scenario sid motion \Delta t)$  for  $\delta t$ 
  apply( $rule mono\_gndet\_const$ )
  using  $base hyp c assms(4)$  apply  $auto$  by ( $rule motion\_step, simp$ )

with  $e$ 
  show  $(\wedge x. \sqcap \_ \in sync\_pair' \{(\delta t, \omega_s). 0 < \delta t \wedge no_{collision} \omega_s sid \varepsilon\} \rightarrow x) \cdot x \leq scenario sid motion$ 
 $\Delta t \omega_0$ 
  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  $'v motion_{acc} = \langle id_{actor} \Rightarrow time \Rightarrow 'v scene \Rightarrow 'v set \rangle$ 
— the set of possible accelerations for a given actor,  $\delta t$  and initial scene
definition  $kinetics :: \langle ('v::real\_normed\_vector) motion_{acc} \Rightarrow 'v motion \rangle$ 
where  $\langle kinetics motion_{acc} \equiv$ 
— defines a motion for a  $\delta t$  such that  $a$  (acceleration) remains constant during  $\delta t$ 
 $\lambda i \delta t \omega. \{ (\mid pos = pos (\omega i) + (\delta t) *_R speed (\omega i) + ((\delta t^2)/2) *_R a,$ 
 $speed = speed (\omega i) + (\delta t *_R a),$ 
 $acc = a \mid)$ 
 $\mid a. a \in motion_{acc} i \delta t \omega \}$ 

lemma  $kinetics\_empty: \langle kinetics motion_{acc} i \delta t \omega = \{\} \leftrightarrow motion_{acc} i \delta t \omega = \{\} \rangle$ 
by ( $auto simp add: kinetics\_def$ )

```

## 2 The CSP-based Generic Autonomous Car Model

**lemma**  $\mu_{kinetics\_refine}$ :

**assumes**  $a: (\forall \delta t \omega i. (i \in sid \wedge accs_2 i \delta t \omega \neq \{\}) \longrightarrow (accs_1 i \delta t \omega \neq \{\}))$   
**and**  $b: (\forall \delta t \omega i. i \in sid \longrightarrow accs_1 i \delta t \omega \subseteq accs_2 i \delta t \omega)$   
 — assume  $motion_{acc_1} \subseteq motion_{acc_2}$   
**shows**  $\llbracket sid:kinetics accs_2 \rrbracket \leq \llbracket sid:kinetics accs_1 \rrbracket$   
 — then the corresponding kinetics generate refining processes  
**apply**(intro all1 impl  $\mu_{motion\_refine}$ )  
**using**  $a$   $b$  **by** (auto simp add:kinetics\_def) blast

— obvious case: no move or same move no collision

**declare** inj\_on\_def[simp] One\_nat\_def[simp del]

**theorem** no\_acc\_is\_safe:

**fixes**  $no_{acc}::('v::real_normed_vector) motion_{acc}$   
**assumes**  $a:(\Delta t > 0)$   
**and**  $b:(no_{acc} \equiv \lambda i \delta t \omega. \{0\})$   
**and**  $c:(no_{collision} \omega_0 \{0..<n\} \varepsilon \wedge (\forall i < n. speed (\omega_0 i) = V))$   
**shows**  $(is_{safe} \{0..<n\} (kinetics no_{acc}) \Delta t \omega_0 \varepsilon)$

**proof** —

**define**  $P_{inv}::(nat \Rightarrow ('v::real_normed_vector) as) \Rightarrow nat set \Rightarrow bool$   
**where**  $\langle P_{inv} \equiv \lambda \omega sid. no_{collision} \omega sid \varepsilon \wedge (\forall i \in sid. (speed::'v as \Rightarrow 'v) (\omega i) = V) \rangle$   
**show** ?thesis  
**apply**(rule is\_safe\_rule\_inv[where  $P_{inv}=P_{inv}$ ], simp\_all add:assms  $P_{inv\_def}$  kinetics\_def)  
**using**  $c$  no\_collision\_def **by** (metis add\_diff\_cancel\_right as.select\_convs(1))

qed

**theorem** same\_acc\_is\_safe:

**fixes**  $same_{acc}::('v::real_normed_vector) motion_{acc}$  **and**  $A::'v$   
**assumes**  $a:(\Delta t > 0)$   
**and**  $b:(\bigwedge \delta t \omega. \exists a. \forall i. same_{acc} i \delta t \omega = \{a\})$   
**and**  $c:(no_{collision} \omega_0 \{0..<n\} \varepsilon \wedge (\forall i < n. speed (\omega_0 i) = V))$   
**shows**  $(is_{safe} \{0..<n\} (kinetics same_{acc}) \Delta t \omega_0 \varepsilon)$

**proof** —

**define**  $P_{inv}::(nat \Rightarrow ('v::real_normed_vector) as) \Rightarrow nat set \Rightarrow bool$   
**where**  $\langle P_{inv} \equiv \lambda \omega sid. no_{collision} \omega sid \varepsilon \wedge (\forall i j. i \in sid \wedge j \in sid \longrightarrow (speed (\omega i) = speed (\omega j))) \rangle$

**show** ?thesis

**proof**(rule is\_safe\_rule\_inv[where  $P_{inv}=P_{inv}$ ], simp\_all add:assms  $P_{inv\_def}$ , goal\_cases)

**case** (1  $\omega \delta t$ )

**then show** ?case **using**  $c$  **by** (metis b empty\_not\_insert kinetics\_empty)

**next**

**case** (2  $\omega \delta t$ )

**then show** ?case

**unfolding** no\_collision\_def kinetics\_def

**by** (smt (z3) add\_diff\_cancel\_right as.select\_convs(1) as.select\_convs(2) atLeastLessThan\_iff

$b$  linorder\_not\_less mem\_Collect\_eq not\_less\_zero singletonD)

qed

*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_σ = 'v as_range +
  speed_max :: real
  passengers :: nat
```

```
type_synonym ('v,'a) Usual_Road_User_σ_ = ('v,'a) Usual_Road_User_σ_scheme
```

```
record ('v::real_normed_vector) PassengerCar_σ = 'v Usual_Road_User_σ +
  xxx :: string
```

```
type_synonym ('v,'a) PassengerCar_σ_ = ('v,'a) PassengerCar_σ_scheme
```

```
record ('v::real_normed_vector) Bike_σ = 'v Usual_Road_User_σ +
  yyy :: string
```

```
type_synonym ('v,'a) Bike_σ_ = ('v,'a) Bike_σ_scheme
```

```
record ('v::real_normed_vector) Truck_σ = 'v Usual_Road_User_σ +
  zzz :: string
```

```
type_synonym ('v,'a) Truck_σ_ = ('v,'a) Truck_σ_scheme
```

The moving object types can be hierarchically extended — space for "unknown" features and its properties.

### 3.1.1 Examples

One-dimensional *int* as.

```
term( person = (| pos = 14,
                speed = 2,
                acc = -1,
                acc_range = {-2,-1,0,1},
                extension_field = {1, 0, -1} |))
```

Two-dimensional (*rat* × *rat*) as.

```
term( (1.0, 1.0) :: ((rat × rat)))
```

```
term( racing_car = (| pos = (140.0,210.0)::rat×rat,
                    speed = (200.0,0.0),
                    acc = (10.0,0.0),
                    acc_range = {(10.0,0.0), — acceleration ahead
                                (2.0,3.0), — acceleration to left
                                (2.0,-3.0), — acceleration to right
                                (-40.0,0.0), — halfbreak
                                (-80.0,0.0)}, — fullbreak
                    extension_field = {(x,y). x*x + y*y ≤ 4.0} |))
    — kis: disk with 4 units diameter.
```

```
definition collision :: ('v::real_normed_vector,'a) as_range_
    ⇒ ('v,'a) as_range_
    ⇒ bool (infix 70)
where emo1 emo2 ≡ (∀ e1 ∈ (extension_field emo1).
    ∀ e2 ∈ (extension_field emo2).
    ((pos emo1) + e1 ≠ (pos emo2) + e2))
```

```
definition dynamics :: real
    ⇒ (('v::real_normed_vector), 'a) as_range_
    ⇒ ('v,'a) as_range_ set
where dynamics δt σ ≡ {σ' . ∃ acc ∈ acc_range σ.
    let speed' = speed σ + (δt *R acc)
    in σ' = σ(|speed := speed',
    pos := pos σ +
    (0.5 *R δt *R (speed σ + speed'))))
}
```

```
definition pos_speed :: ('v::real_normed_vector,'a) Usual_Road_User_σ_ set
where pos_speed ≡ {σ. 0 ≤ norm(speed σ) ∧ norm(speed σ) ≤ speed_max σ }
```

## 3.2 Static Environment : The Street Topology

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

```
type_synonym 'v topology = lane_type  $\Rightarrow$  'v set
```

### 3.2.1 Linear Lane Example

```
definition linear_lane  $\equiv$  undefined (laneL1 := {0 .. 100::int})
```

## 3.3 CSP Model II

In this section, we present a more refined demon which makes weaker assumptions on synchronisation 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 ('v,' $\alpha$ ,' $\beta$ ) events = uplocal (id : id_actor)
  (get_status : ('v,' $\alpha$ ) as_range_) ( $\langle$ _,  $\_$  $\rangle$ )
  | sync (world_state : ('v,' $\alpha$ ,' $\beta$ )scene_o_)
  | time_lapse (time : real)
```

```
term (d?(y,y)  $\rightarrow$  c!x  $\rightarrow$  P = Q)
```

*definition* (*status*<sub>c</sub> = case\_prod *up*<sub>local</sub>) — necessary for uncurried notation in read commands

```
term (statusc?(ide, $\sigma$ )  $\rightarrow$  X((ds::id_actor  $\rightarrow$  ('v,'a) as_range_) (ide  $\mapsto$   $\sigma$ )))
```

### 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  $\Rightarrow$  ('v::real\_normed\_vector,' $\alpha$ ) as\_range\_)  $\Rightarrow$  ('v,' $\alpha$ ,' $\beta$ ) events process

```
where (demon  $\Delta t$   $\equiv$   $\mu$  X. ( $\lambda$  ds.  $\Pi$   $\Delta t \in$  {time_lapse x | x. 0 < x  $\wedge$  x  $\leq$   $\Delta t$ }
   $\rightarrow$  ( $\mu$  X. ( $\lambda$  ds. statusc?(ide, $\sigma$ )  $\rightarrow$  (X (ds(ide :=  $\sigma$ )))))) ds
   $\Pi$  X ds — inner loop for reads; not necessarily df
```

```
term (info? $\sigma_G$   $\rightarrow$  P)
```

### 3.3.3 Behavior of Actors

**definition**  $\mu_{as} :: ((v, \alpha, \beta)scene_{o_} \Rightarrow (v, \alpha) as_{range\_set}) \Rightarrow nat \Rightarrow (v, \alpha) as_{range\_}$   
 $\Rightarrow (v::real\_normed\_vector, \alpha, \beta) events\ process$   
**where**  $\langle \mu_{as} driving\_strategy\ ide$   
 $\equiv \mu X. (\lambda as. \square \Delta t \in \{time\_lapse\ x \mid x. 0 \leq x\}$   
 $\rightarrow sync?\sigma_G$   
 $\rightarrow (\prod as' \in (up_{local}\ ide) ((dynamics(time\ \Delta t)as) \cap driving\_strategy\ \sigma_G)$   
 $\rightarrow (X (get\_status\ as'))))$

**syntax**  $asbeh :: (v, a) as_ \Rightarrow (v, a) as\_set \Rightarrow nat$   
 $\Rightarrow (v::real\_normed\_vector, a, b) events\ process\ (\llbracket (\_) | (\_) \rrbracket_)$

**translations**  $\llbracket as \mid A \rrbracket_n \equiv (CONST\ \mu_{as})\ A\ n\ as$

### 3.3.4 Examples

**definition**  $rear\_car \equiv 0::id_{actor}$   
**definition**  $front\_car \equiv 1::id_{actor}$   
**definition**  $mt\ x \equiv undefined$

**definition**  $\langle racing\_car = undefined$   
**definition**  $\langle on\_lane1 = undefined$

**definition**  $ego\_id = (0::id_{actor})$

**definition**  $\langle rear\_Car = \llbracket racing\_car \mid (\lambda_. pos\_speed \cap on\_lane1) \rrbracket_{rear\_car}$   
**definition**  $\langle front\_Car = \llbracket racing\_car \mid (\lambda_. pos\_speed \cap on\_lane1) \rrbracket_{front\_car}$

**term**  $\langle dynamic\_system = (rear\_Car \llbracket range\ time\_lapse \rrbracket front\_Car \llbracket range\ time\_lapse \rrbracket demon\ 0.5$   
 $A)$

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

**definition**  $disk\ z = \{(x,y). x*x + y*y \leq z\}$   
**term**  $fun\_upd$   
**term**  $I\ (i0::nat := x)$

**term**  
 $[sync\ (\llbracket global\_state = (mt(rear\_car := (\llbracket pos = (0.0,0.0) \rrbracket::real \times real,$   
 $speed = (33.3,0.0), \quad - 120\ km/h$   
 $acc = (1.0,0.0),$   
 $acc\_range = \{(1.0,0.0), (0.0,1.0), (0.0,-1.0),$

```

        (-10.0,0.0), (-20.0,0.0)},
        extension_field = disk 4.0 |),
    front_car := (|pos      = (25.0,-3.0)::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      = (16.7,0.0)::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,-2.0)::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,-2.0)::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      = (50.0,0.0)::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), (0.0,-3.0),
            (-10.0,0.0), (-20.0,0.0)},
        extension_field = disk 4.0 |),
    front_car := (|pos      = (66.7,-2.0)::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)},

```

### 3 From MOSAR-Ontologies to a Semantics in CSP

```

                                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,-3.0)::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      = (16.7,0.0)::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,-2.0)::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,-2.0)::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),

```

```

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      = (66.7,-2.0)::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 ))

```

)]

∈  $\mathcal{T}$  (demon 0.5 Actors  $\llbracket S \rrbracket$  front\_car\_P  $\llbracket S \rrbracket$  rear\_car\_P)

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_maxaccel ::real and a_minbrake ::real and a_maxbrake ::real and ρ::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 ⇒ real motion)
```

```
assumes a0:(0 < a_maxaccel)
and a1:(0 < a_minbrake)
and a2:(a_minbrake ≤ a_maxbrake)
and a3:(ρ > 0)
```

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

```
defines d0:(d_safe σ_g ≡ max 0 (ρ * speed (σ_g rear_id) + ((ρ2)/2) * a_maxaccel +
(speed (σ_g rear_id) + ρ * a_maxaccel)2 / (2 * a_minbrake) -
(speed (σ_g front_id))2 / (2 * a_maxbrake)))
```

— To improve: use current *acc* ( $\sigma_g \text{ rear}_{id}$ ) is more precise than  $a_{maxaccel}$

```
and d1:(d_real σ_g ≡ pos(σ_g front_id) - pos(σ_g rear_id))
```

— signed distance between the two vehicles

```
assumes r0:(rss_motion 0 δt σ_g ≡ {.. if d_safe σ_g > d_real σ_g then -a_minbrake else a_maxaccel})
```

— 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  $t-\rho$  ??

```
and r1:(rss_motion 1 δt σ_g ≡ {-a_maxbrake ..})
```

— 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**  $k0: \langle kinetics_{RSS} motion_{acc} \equiv$

$\lambda i \delta t \sigma_g. \{$  if  $speed(\sigma_g i) + (\delta t * a) > 0$   
then  $(\mid pos = pos(\sigma_g i) + (\delta t) * speed(\sigma_g i) + ((\delta t^2)/2) * a,$   
 $speed = speed(\sigma_g i) + (\delta t * a),$   
 $acc = a \mid)$   
else  $(\mid pos = pos(\sigma_g i) - (speed(\sigma_g i))^2 / (2 * a),$   
 $speed = 0,$   
 $acc = 0 \mid)$

— the  $-(speed(\sigma_g i))^2 / (2 * a)$  comes from the newtonian kinetics formula  
with  $t_1 = -v/a$ , i.e. the time when  $v = v_0 + a * t_1 = 0$  (the vehicle stops at  $t_1 \in [t, t + \delta t]$ )  
 $\mid a. a \in motion_{acc} i \delta t \sigma_g \}$

**begin**

### 4.2 Relations between Driving Strategies

**lemma**  $kinetics_{RSS\_empty}: \langle kinetics_{RSS} motion_{acc} i \delta t \sigma_g = \{\} \leftrightarrow motion_{acc} i \delta t \sigma_g = \{\} \rangle$   
**by** (auto simp add: k0)

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

**definition**  $\langle d_{min} \equiv \lambda (\sigma_g :: real\ scene). max\ 0\ ((speed(\sigma_g\ rear_{id}))^2 / (2 * a_{minbrake}) - (speed(\sigma_g\ front_{id}))^2 / (2 * a_{maxbrake})) \rangle$

**declare**  $minus\_divide\_le\_eq[simp]$   $not\_less[simp]$

**theorem**  $rss\_is\_safe\_2cars:$

**assumes**  $a: d_{min} \sigma_{g0} < d_{real} \sigma_{g0} \wedge 0 \leq speed(\sigma_{g0} 0) \wedge 0 \leq speed(\sigma_{g0} 1)$

— pquoi une inégalité stricte?? Est-ce par définition de  $is_{safe}$

**and**  $b: \langle 0 < \Delta t \wedge \Delta t < \varrho \rangle$  — et ici, pourquoi inégalité stricte??

**shows**  $\langle is_{safe} \{0, 1\} (kinetics_{RSS} rss_{motion}) \Delta t \sigma_{g0} 0 \rangle$

**proof** —

**define**  $P_{inv}$  **where**  $\langle P_{inv} \equiv \lambda (\sigma_g :: real\ scene). 0 \leq speed(\sigma_g\ rear_{id}) \wedge 0 \leq speed(\sigma_g\ front_{id})$

$\wedge d_{min} \sigma_g < d_{real} \sigma_g \rangle$

— invariant: the distance between the two cars is  $d_{min}$

**show** ?thesis

**proof**(rule  $is_{safe\_rule\_inv}$ [**where**  $P_{inv} = \langle \lambda \sigma_g\ sid. P_{inv} \sigma_g \rangle$ ])

— use  $is_{safe\_rule\_inv}$  with  $P_{inv}$  defined above

**show**  $\langle 0 < \Delta t \rangle$  **using**  $b$  **by** simp

**next**

**show**  $\langle P_{inv} \sigma_{g0} \rangle$

**using**  $a$  **by** (auto simp add:  $P_{inv\_def}$  d0)

**next**

**fix**  $\sigma_g \delta t$

**show**  $\langle P_{inv} \sigma_g \wedge \delta t \in \{0 .. \Delta t\} \implies \forall i \in \{0, 1\}. kinetics_{RSS} rss_{motion} i \delta t \sigma_g \neq \{\} \rangle$

**using** a0 a1 a2 r0 r1  $P_{inv\_def}$  **by** (auto simp:  $kinetics_{RSS\_empty}$ )

**next**

## 4.2 Relations between Driving Strategies

```

fix  $\sigma_g \delta t$ 
show  $\langle P_{inv} \sigma_g \wedge \delta t \in \{0 <.. \Delta t\} \implies \forall \sigma_g' \in \{\sigma_g'. \forall i \in \{0, 1\}. \sigma_g' i \in kinetics_{RSS} rss_{motion} i \delta t \sigma_g\}. P_{inv} \sigma_g' \rangle$ 
proof(auto simp:Pinv_def)
  fix  $\sigma_g'$ 
  assume  $\langle \sigma_g' (0::nat) \in kinetics_{RSS} rss_{motion} 0 \delta t \sigma_g \rangle$  and  $\langle 0 < \delta t \rangle$ 
  then show  $\langle 0 \leq speed(\sigma_g' rear_{id}) \rangle$ 
    by(auto simp:k0 r0 mult.commute)
  next
  fix  $\sigma_g'$ 
  assume  $\langle \sigma_g' (1::nat) \in kinetics_{RSS} rss_{motion} 1 \delta t \sigma_g \rangle$  and  $\langle 0 < \delta t \rangle$ 
  then show  $\langle 0 \leq speed(\sigma_g' front_{id}) \rangle$ 
    by(auto simp:k0 r1 mult.commute)
  next
  fix  $\sigma_g'$ 
  assume as1: $\langle 0 \leq speed(\sigma_g rear_{id}) \rangle$ 
    and as2: $\langle 0 \leq speed(\sigma_g front_{id}) \rangle$ 
    and as3: $\langle d_{min} \sigma_g < d_{real} \sigma_g \rangle$ 
    and as4: $\langle 0 < \delta t \rangle$ 
    and as5: $\langle \delta t \leq \Delta t \rangle$ 
    and as6: $\langle \sigma_g' (0::nat) \in kinetics_{RSS} rss_{motion} 0 \delta t \sigma_g \rangle$ 
    and as7: $\langle \sigma_g' front_{id} \in kinetics_{RSS} rss_{motion} 1 \delta t \sigma_g \rangle$ 

  let ?a0 =  $\langle acc(\sigma_g' rear_{id}) \rangle$  and ?a1 =  $\langle acc(\sigma_g' front_{id}) \rangle$ 
    and ?b0 =  $\langle (speed(\sigma_g rear_{id}))^2 / (2 * a_{minbrake}) \rangle$  and ?b1 =  $\langle (speed(\sigma_g front_{id}))^2 / (2 * a_{maxbrake}) \rangle$ 
    and ?b'0 =  $\langle (speed(\sigma_g' rear_{id}))^2 / (2 * a_{minbrake}) \rangle$  and ?b'1 =  $\langle (speed(\sigma_g' front_{id}))^2 / (2 * a_{maxbrake}) \rangle$ 

  — some simplifications factorized here
  have simp0: $\langle 0 \leq speed(\sigma_g' rear_{id}) \rangle$  and simp1: $\langle 0 \leq speed(\sigma_g' front_{id}) \rangle$ 
    using as6 as7 by (auto simp:k0 r0 r1 mult.commute)

  have simp2: $\langle speed(\sigma_g rear_{id}) / 2 + ?a_0 * \delta t / 2 \geq 0 \rangle$  and simp3: $\langle speed(\sigma_g front_{id}) / 2 + ?a_1 * \delta t / 2 \geq 0 \rangle$ 
    using as6 as7 by (auto simp add:k0 r0 r1 as4 as2 as1 mult.commute)

  have simp4: $\langle speed(\sigma_g rear_{id}) + ?a_0 * \delta t / 2 \geq 0 \rangle$  and simp5: $\langle speed(\sigma_g front_{id}) + ?a_1 * \delta t / 2 \geq 0 \rangle$ 
    using as1 simp2 apply linarith
    using as2 simp3 by linarith

  have simp6: $\langle \delta t * speed(\sigma_g rear_{id}) + ?a_0 * \delta t^2 / 2 \geq 0 \rangle$  and simp7: $\langle \delta t * speed(\sigma_g front_{id}) + ?a_1 * \delta t^2 / 2 \geq 0 \rangle$ 
    using as4 simp4 simp5
    by (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)+

```

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

```

have simp4p:(speed (σg rearid) + ?a0*δt / 2 ≥ speed (σg rearid) / 2)
and simp5p:(speed (σg frontid) + ?a1*δt / 2 ≥ speed (σg frontid) / 2)
using as1 simp2 apply linarith
using as2 simp3 by linarith

have simp6p:(δt*speed (σg rearid) + ?a0*δt2 / 2 ≥ δt*speed (σg rearid) / 2)
and simp7p:(δt*speed (σg frontid) + ?a1*δt2 / 2 ≥ δt*speed (σg frontid) / 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 simp8:(speed (σg rearid) + a * δt ≤ 0 ⇒ - (speed (σg rearid))2 / (a * 2) ≤ (δt*speed
(σg rearid) / 2) for a
using as1 mult_left_mono[where b=(δt) and a=(-speed (σg rearid) / a) and c=(speed (σg
rearid)), OF_ as1]
apply (auto simp add:mult.commute power2_eq_square split:if_splits)
using as4 mult_pos_pos by force

have simp9:(speed (σg frontid) + a * δt ≤ 0 ⇒ - (speed (σg frontid))2 / (a * 2) ≤ (δt*speed
(σg frontid) / 2) for a
using as2 mult_left_mono[where b=(δt) and a=(-speed (σg frontid) / a) and c=(speed (σg
frontid)), 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 (speed (σg' frontid) = 0)
  then have (max 0 ?b1 ≤ pos (σg' frontid) - pos (σg frontid))
  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=((speed (σg frontid))2) and a=((amaxbrake * 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 h0:(speed (σg' frontid) > 0)
  from as7 have h1:(- ?a1 / amaxbrake ≤ 1)
  by (auto simp add:k0 r1) (smt (z3) a1 a2 divide_le_eq_1_neg minus_divide_right)
  have h2:(speed (σg frontid) + δt * ?a1)2 / (2*amaxbrake)
    = (speed (σg frontid))2 / (2*amaxbrake) + (?a1/amaxbrake) * (δt * speed (σg
frontid) + ?a1 * δt2 / 2)
  by (simp add: power2_sum distrib_left mult.commute power2_eq_square
        add_divide_distrib mult.left_commute)
  have (0 ≤ pos (σg' frontid) - pos (σg frontid))
  using as7 h0 simp7 by (auto simp add:k0 r1 mult.commute split:if_splits)
  then have (max 0 (?b1 - ?b'1) ≤ pos (σg' frontid) - pos (σg frontid))
  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)
  } note dmin_front_car_motion=this

have front_car:(max 0 (?b1 - ?b'1) ≤ pos (σg' frontid) - pos (σg frontid))
  using dmin_front_car_motion dmin_front_car_stopping simp1 by fastforce

{ assume h0:(dsafe σg > dreal σg) and h1:(speed (σg' rearid) > 0)
  have h2:(?a0 / aminbrake ≤ -1)
    using as6 a1 a2 h0 h1 apply(auto simp add:k0 r0 split:if_splits)
    by (metis a1 add.inverse_inverse le_divide_eq_1_pos minus_divide_left neg_le_iff_le)
  have (pos (σg' rearid) - pos (σg rearid) ≤ (speed (σg rearid)2 / (2*aminbrake) - (speed
(σg' rearid)2 / (2*aminbrake)))
    using as6 a1 a2 h0 h1 mult_left_mono[OF h2 simp6]
    by (auto simp add:k0 r0 power2_sum add_divide_distrib distrib_left mult.commute
split:if_splits)
    (smt (verit) mult.assoc mult.commute power2_eq_square)
  then have (?b'0 - ?b0 ≤ pos (σg rearid) - pos (σg' rearid))
    by force
  } note dmin_rear_car_nonsafe_breaking=this

{ assume h0:(dsafe σg > dreal σg) and h1:(speed (σg' rearid) = 0)
  have h2:(a ≤ -aminbrake ⇒ a / aminbrake ≤ -1) for a
    by (metis a1 divide_eq_minus_1_iff divide_le_cancel less_le not_le)
  have h4:(pos (σg' rearid) - pos (σg rearid) ≤ (speed (σg rearid)2 / (2*aminbrake)))
    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 (¬ ?b0 ≤ pos (σg rearid) - pos (σg' rearid))
    by argo
  } note rear_car_nonsafe_stopping=this

have dmin_rear_car_nonsafe:(dsafe σg > dreal σg ⇒ ?b'0 - ?b0 ≤ pos (σg rearid) - pos
(σg' rearid))
  using dmin_rear_car_nonsafe_breaking rear_car_nonsafe_stopping simp0 by fastforce

{ assume h0:(dsafe σg ≤ dreal σg) and h1:(speed (σg' rearid) = 0)
  have h2:(δt * speed (σg rearid) / 2 ≤ ρ*speed(σg rearid))
    using as1 as5 b less_eq_real_def by force
  have h3:(pos (σg' rearid) - pos (σg rearid) ≤ ρ*speed(σg rearid))
    using as6 as1 a1 h0 h1 apply(auto simp add:d0 dmin_def 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:(?b0 ≤ (speed(σg rearid) + ρ*amaxaccel)2 / (2*aminbrake))
    by (smt (verit, ccfv_SIG) a0 a1 a3 as1 frac_le mult_pos_pos power_mono zero_le_power2)
  with h0 have h5:(ρ*speed(σg rearid) + ?b0 - ?b1 < pos (σg frontid) - pos (σg rearid))
    by (smt (verit, best) a0 a3 d0 d1 divide_pos_pos zero_less_mult_iff zero_less_power)

```

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

```

with h3 have (?b0 - ?b1 < pos (σg frontid) - pos (σg' rearid))
  by fastforce
} note dmin_rear_car_safe_stopping=this

{ assume h0:(dsafe σg ≤ dreal σg) and h1:(speed (σg' rearid) > 0)
  have h2:(δt*speed(σg rearid) ≤ ρ*speed(σg rearid))
    using as1 as4 as5 b mult_right_less_imp_less by force
  from as6 have h3:(?a0*δt2/2 < amaxaccel*ρ2/2)
    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'0 ≤ (speed(σg rearid) + ρ*amaxaccel)2/(2*aminbrake))
    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_mult_iff)
  have h5:(?b0 ≤ (speed(σg rearid) + ρ*amaxaccel)2/(2*aminbrake))
    by (smt (verit, ccfv_SIG) a0 a1 a3 as1 frac_le mult_pos_pos power_mono zero_le_power2)
  from h2 h3 h4 h5
  have ( δt*speed(σg rearid) + ?a0*δt2/2 + (max ?b0 ?b'0)
        < ρ*speed(σg rearid) + amaxaccel*ρ2/2 + (speed(σg rearid) +
        ρ*amaxaccel)2/(2*aminbrake))
    by (smt (verit) mult.commute)
  with h0 have h6:(δt*speed(σg rearid) + ?a0*δt2/2 + (max ?b0 ?b'0) - ?b1 < pos (σg
  frontid) - pos (σg rearid))
    by (smt (verit, ccfv_SIG) d0 d1 mult.commute times_divide_eq_right)
  have (pos (σg' rearid) - pos (σg rearid) = δt*speed(σg rearid) + ?a0*δt2/2)
  using as6 as1 a1 h1 by (auto simp add:d0 dmin_def k0 r0 mult.commute not_le split:if_splits)

  with h6 have ((max ?b0 ?b'0) - ?b1 < pos (σg frontid) - pos (σg' rearid))
    by fastforce
} note dmin_rear_car_safe_motion=this

have dmin_rear_car_safe:(dsafe σg ≤ dreal σg ⇒ (max ?b0 ?b'0) - ?b1 < pos (σg frontid)
- pos (σg' rearid))
  by (metis a1 divide_le_cancel less_eq_real_def max_def mult_2 power2_less_eq_zero_iff
    power_zero_numerical dmin_rear_car_safe_motion dmin_rear_car_safe_stopping
    simp0
    zero_less_double_add_iff_zero_less_single_add)

{ assume h0:(dsafe σg ≤ dreal σg)
  then have (0 ≤ pos (σg' rearid) - pos (σg rearid))
    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_numerical_less_zero zero_le_power2 zero_less_mult_iff)
} note dmin_rear_car_forward=this

have dmin:(?b'0 - ?b'1 < dreal σg')
  using as3 d1 dmin_def front_car dmin_rear_car_nonsafe dmin_rear_car_safe by fastforce

```

## 4.2 Relations between Driving Strategies

— no collision's proofs

```

{ assume h0:(?b0 ≤ ?b1) and h1:(?b'0 < ?b'1)
  — ?b'0 < ?b'1, is added to profit from previous results of dmin
  and h2:(speed (σg' rearid) = 0)
  from h0 have h3:(speed (σg rearid) ≤ speed (σg frontid))
    using mult_right_mono[OF h0, where c=((2 * aminbrake)), simplified]
  by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_le_imp_le zero_le_power2)

  from h2 have h6:(pos (σg' rearid) - pos (σg rearid) ≤ δt * speed (σg rearid) / 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 (σg' rearid) - pos (σg rearid) ≤ pos (σg' frontid) - pos (σg frontid))
    using as7 h1 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

{ assume h0:(?b0 ≤ ?b1) and h1:(?b'0 < ?b'1)
  and h2:(speed (σg' rearid) > 0)
  from h0 have h3:(speed (σg rearid) ≤ speed (σg frontid))
    using mult_right_mono[OF h0, where c=((2 * aminbrake)), simplified]
  by (smt (z3) a1 a2 as2 less_divide_eq mult_left_mono power2_le_imp_le zero_le_power2)

  have h4:(speed (σg' frontid) > 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:(speed (σg rearid) + ?a0 * δt < speed (σg frontid) + ?a1 * δt)
    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)
  have h6:(δt * speed (σg rearid) + δt2 * ?a0 < δt * speed (σg frontid) + δt2 * ?a1)
    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:(δt * speed (σg rearid) + δt2 * ?a0 / 2 < δt * speed (σg frontid) + δt2 *
    ?a1 / 2)
    by (smt (verit, ccfv_SIG) as4 field_sum_of_halves mult_less_cancel_left_pos)
  have (pos (σg' rearid) - pos (σg rearid) < pos (σg' frontid) - pos (σg frontid))
    using as6 as7 h2 h4 h7 by (auto simp add:k0 r0 r1 a1 dmin_def d1
      divide_nonneg_pos power2_sum not_le)
} note no_collision_if_slow_rear_car_nonstopping=this

```

remaining case (REM):

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



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

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

```

{ assume h0:(?b1 < ?b0) and h1:(dsafe σg > dreal σg) and h2:(?b'0 < ?b'1)
  and h3:(speed (σg' frontid) = 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=this

{ assume h0:(?b1 < ?b0) and h1:(dsafe σg > dreal σg) and h2:(speed (σg' frontid) > 0)
  and h3:(speed(σg frontid) - amaxbrake *δt ≤ 0)
  have h4:(?b1 ≤ pos (σg' frontid) - pos (σg frontid))
  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_convs(3) divide_minus_left h3 minus_divide_right
    mult_minus_left simp7p simp9)
  then have (0 < pos (σg' frontid) - pos (σg' rearid))
  by (smt (verit, best) a1 as3 d1 dmin_def divide_nonneg_pos
    dmin_rear_car_nonsafe h1 zero_le_power2)
} note REM_front_car_may_stop=this

{ assume h0:(?b1 < ?b0) and h1:(dsafe σg > dreal σg) and h2:(speed (σg' frontid) > 0)
  and h3:(speed(σg frontid) - amaxbrake *δt > 0)

  have s1:(amaxbrake / aminbrake ≥ 1 ∧ sqrt amaxbrake ≥ sqrt aminbrake ∧ sqrt aminbrake
    > 0)
  using a1 a2 by simp
  then have s2:(amaxbrake / aminbrake ≥ sqrt amaxbrake / sqrt aminbrake)
  by (smt (verit, ccfv_threshold) divide_less_eq_1 real_div_sqrt real_sqrt_divide)
  then have s3:(amaxbrake * sqrt aminbrake ≥ aminbrake * sqrt amaxbrake)
  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 amaxbrake / sqrt aminbrake ≥ (speed (σg frontid) / (speed (σg rearid)))
  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_divide_le_eq
    s1 times_divide_eq_left)
  have s5:(speed(σg rearid) / aminbrake ≥ speed(σg frontid) / amaxbrake)
  using order_trans[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
    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(σg rearid) - aminbrake*δt > 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 (σg frontid)) * sqrt aminbrake ≤ (speed (σg rearid)) * sqrt amaxbrake)
  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 auto[1]
  by (simp add: a1 le_divide_eq mult.commute)
have s8:(( speed (σg frontid) - amaxbrake*δt) / sqrt amaxbrake
  ≤ (speed (σg rearid) - aminbrake *δt) / sqrt aminbrake)
  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.left_commute
mult_le_cancel_left_pos)
  with h3 have s9:((speed(σg frontid) - amaxbrake *δt)2/(2*amaxbrake)
  ≤ ((speed(σg rearid) - aminbrake *δt)2/(2*aminbrake)))
  using abs_le_square_iff[where x = ((speed (σg frontid) - amaxbrake*δt) / sqrt amaxbrake)
  and y = ((speed (σg rearid) - aminbrake *δt) / sqrt aminbrake)]
  using s1 apply (auto simp add:power_divide)
  by (smt (verit, best) divide_less_0_iff s1)

have s10:(δt * speed (σg rearid) / 2 ≤ (δt * speed (σg rearid) - aminbrake * δt2 / 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:(δt*speed(σg frontid) - amaxbrake *δt2/2 ≤ pos (σg' frontid) - pos (σg frontid))
  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 (δt*speed(σg frontid) - amaxbrake *δt2/2 - ?b1 = -(speed(σg frontid) - amaxbrake
*δt)2/(2*amaxbrake))
  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 (σg' frontid) - pos (σg frontid) - ?b1 ≥ - (speed(σg frontid) -
amaxbrake *δt)2/(2*amaxbrake))
  by linarith

have (a ≤ - aminbrake ⇒ (δt * speed (σg rearid) + a * δt2 / 2)
  ≤ (δt * speed (σg rearid) - aminbrake * δt2 / 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 (σg' rearid) - pos (σg rearid) ≤ δt*speed(σg rearid) - aminbrake
*δt2/2)

```

#### 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*speed(σg rearid) - aminbrake *δt2/2 - ?b0 = -(speed(σg rearid) - aminbrake
*δt)2/(2*aminbrake))
    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 (pos (σg' rearid) - pos (σg rearid) - ?b0 ≤ pos (σg' frontid) - pos
(σg frontid) - ?b1)
    by linarith
    then have (0 < pos (σg' frontid) - pos (σg' rearid))
    using as3 d1 dmin_def by auto
    } note REM_front_car_nonstopping=this

    have nocollision:(0 < dreal σg')
    apply (cases (?b'0 ≥ ?b'1))
    using dmin apply simp
    apply(cases (?b0 ≤ ?b1))
    using as3 d1 dmin_def no_collision_if_slow_rear_car_nonstopping
        no_collision_if_slow_rear_car_stopping simp0
    apply fastforce
    apply(cases (dsafe σg > dreal σg'), 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 (dmin σg' < dreal σg')
    using dmin nocollision by (simp add: dmin_def)
    qed
next
    fix σg
    show (Pinv σg ⇒ nocollision σg {0, 1} 0)
    unfolding dmin_def d1 Pinv_def nocollision_def by fastforce
    qed
qed

end

end

```

## 5 A Safety-Property in Autonomopus 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  $a_{maxaccel}::real$  and  $a_{minbrake}::real$  and  $a_{maxbrake}::real$  and  $\varrho::real$ 
  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  $a0:(0 < a_{maxaccel})$ 
    and  $a1:(0 < a_{minbrake})$ 
    and  $a2:(a_{minbrake} \leq a_{maxbrake})$ 
    and  $a3:(\varrho > 0)$ 
```

```
  defines  $d_{min} \ \omega \ i \equiv \max\ 0\ ((speed\ (\omega\ i))^2 / (2 * a_{minbrake}) - (speed\ (\omega\ (i+1)))^2 / (2 * a_{maxbrake}))$ 
    and  $d0:(d_{rss} \ \omega \ i \equiv \max\ 0\ (\varrho * speed\ (\omega\ i) + ((\varrho^2)/2) * a_{maxaccel} + (speed\ (\omega\ i) + \varrho * a_{maxaccel})^2 / (2 * a_{minbrake}) - (speed\ (\omega\ (i+1)))^2 / (2 * a_{maxbrake}))$ 
    and  $d1:(d_{real} \ \omega \ i \equiv pos(\omega\ (i+1)) - pos(\omega\ i))$ 
```

```
  defines  $r0:(rss_{motion} \ i \ \delta t \ \omega \equiv \text{if } d_{rss} \ \omega \ i \leq d_{real} \ \omega \ i$ 
    then  $\{-a_{maxbrake} .. a_{maxaccel}\}$ 
    else  $\{-a_{maxbrake} .. -a_{minbrake}\}$ )
```

```
begin
```

### 5.2 Relations between Driving Strategies

— RSS kinetics are not backward

```
definition  $k0:(kinetics_{RSS} \ motion_{acc} \equiv$ 
```

## 5 A Safety-Property in Autonomopus Cars: RSS in a N-cars-Scenario

$$\lambda i \delta t \omega. \{ \text{if speed } (\omega i) + (\delta t * a) > 0 \\ \text{then } (\mid \text{ pos} = \text{pos } (\omega i) + (\delta t) * \text{speed } (\omega i) + ((\delta t^2)/2) * a, \\ \text{speed} = \text{speed } (\omega i) + (\delta t * a), \\ \text{acc} = a \mid) \\ \text{else } (\mid \text{ pos} = \text{pos } (\omega i) - (\text{speed } (\omega i))^2 / (2 * a), \text{speed} = 0, \text{acc} = 0 \mid) \\ \mid a. a \in \text{motion}_{acc} i \delta t \omega \}$$

**lemma**  $\text{kinetics}_{RSS\_empty}$ :  $\langle \text{kinetics}_{RSS} \text{ motion}_{acc} i \delta t \omega = \{ \} \leftrightarrow \text{motion}_{acc} i \delta t \omega = \{ \} \rangle$   
**by** (auto simp add: k0)

**theorem**  $\text{rss\_is\_safe\_N\_cars}$ :

**fixes**  $N::\text{nat}$

**assumes**  $a: (\forall i \in \{0..N\}. d_{min} \omega_0 i < d_{real} \omega_0 i \wedge 0 \leq \text{speed } (\omega_0 i))$

**and**  $b: (0 < \Delta t \wedge \Delta t < \rho)$

**shows**  $\langle \text{is}_{safe} \{0..N\} (\text{kinetics}_{RSS} \text{ rss}_{motion}) \Delta t \omega_0 0 \rangle$

**proof** –

**define**  $P_{inv}$  **where**  $\langle P_{inv} \equiv \lambda (\omega::\text{real scene}). (\forall i \leq N. 0 \leq \text{speed } (\omega i)) \wedge (\forall i < N. d_{min} \omega i < d_{real} \omega i) \rangle$

**show** ?thesis

**proof**(rule  $\text{is}_{safe\_rule\_inv}$ [**where**  $P_{inv} = \langle \lambda \omega \text{ sid}. P_{inv} \omega \rangle$ ])

**show**  $\langle 0 < \Delta t \rangle$  **using**  $b$  **by** simp

**next**

**show**  $\langle P_{inv} \omega_0 \rangle$

**using**  $a$  **by** (auto simp add:  $P_{inv\_def} d0$ )

**next**

**fix**  $\omega \delta t$

**show**  $\langle P_{inv} \omega \wedge \delta t \in \{0.. \Delta t\} \implies \forall i \in \{0..N\}. \text{kinetics}_{RSS} \text{ rss}_{motion} i \delta t \omega \neq \{ \} \rangle$

**using**  $a0 a1 a2 r0 P_{inv\_def}$  **by** (auto simp:  $\text{kinetics}_{RSS\_empty}$ )

**next**

**fix**  $\omega \delta t$

**show**  $\langle P_{inv} \omega \wedge \delta t \in \{0.. \Delta t\} \implies \forall \omega' \in \{ \omega' . \forall i \in \{0..N\}. \omega' i \in \text{kinetics}_{RSS} \text{ rss}_{motion} i \delta t \omega \}. P_{inv} \omega' \rangle$

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

**fix**  $\omega' i$

**assume**  $\langle \forall i \in \{0..N\}. \omega' i \in \text{kinetics}_{RSS} \text{ rss}_{motion} i \delta t \omega \rangle$  **and**  $\langle i \leq N \rangle$  **and**  $\langle 0 < \delta t \rangle$

**then show**  $\langle 0 \leq \text{speed } (\omega' i) \rangle$

**apply** (auto simp:  $k0 r0 \text{mult.commute split:if_splits}$ )

**by** (metis  $\text{as.select_convs}(2) \text{atLeastAtMost\_iff less\_eq\_real\_def not\_less\_zero\_le}$ )

**next**

**fix**  $\omega' i$

**assume**  $\text{assm1}: \langle \forall i \leq N. 0 \leq \text{speed } (\omega i) \rangle$

**and**  $\text{assm2}: \langle \forall i < N. d_{min} \omega i < d_{real} \omega i \rangle$

**and**  $\text{assm3}: \langle 0 < \delta t \rangle$

**and**  $\text{assm4}: \langle \delta t \leq \Delta t \rangle$

**and**  $\text{assm5}: \langle \forall i \in \{0..N\}. \omega' i \in \text{kinetics}_{RSS} \text{ rss}_{motion} i \delta t \omega \rangle$

**and**  $\text{assm6}: \langle i < N \rangle$

## 5.2 Relations between Driving Strategies

```

have as1:(0 ≤ speed (ω i))
  using as1[rule_format, of i] as1 by simp
have as2:(0 ≤ speed (ω (i+1)))
  using as1[rule_format, of (i+1)] as1 by simp
note as3 = as2[rule_format, of i, OF as1]
note as4 = as3
note as5 = as4
have as6:(ω' i ∈ kineticsRSS rssmotion i δt ω)
  using as5[rule_format, of i] as5 by simp
have as7:(ω' (i + 1) ∈ kineticsRSS rssmotion (i + 1) δt ω)
  using as5[rule_format, of (i+1)] as5 by simp

let ?ar = ⟨acc (ω' i)⟩ and ?af = ⟨acc (ω' (i+1))⟩
and ?vr = ⟨speed (ω i)⟩ and ?vf = ⟨speed (ω (i+1))⟩
and ?v'r = ⟨speed (ω' i)⟩ and ?v'f = ⟨speed (ω' (i+1))⟩
and ?pr = ⟨pos (ω i)⟩ and ?pf = ⟨pos (ω (i+1))⟩
and ?p'r = ⟨pos (ω' i)⟩ and ?p'f = ⟨pos (ω' (i+1))⟩
and ?br = ⟨(speed (ω i))2 / (2*aminbrake)⟩ and ?bf = ⟨(speed (ω (i+1)))2 / (2*amaxbrake)⟩
and ?b'r = ⟨(speed (ω' i))2 / (2*aminbrake)⟩ and ?b'f = ⟨(speed (ω' (i+1)))2 / (2*amaxbrake)⟩

— some simplifications factorized here
have simp0:(0 ≤ ?v'r) and simp1:(0 ≤ ?v'f)
  using as6 as7 by (auto simp:k0 r0 mult.commute split:if_splits)

have simp2:(?vr / 2 + ?ar*δt / 2 ≥ 0) and simp3:(?vf / 2 + ?af*δt / 2 ≥ 0)
  using as1 as2 as6 as7 by (auto simp add:k0 r0 as4 mult.commute)

have simp4:(?vr + ?ar*δt / 2 ≥ 0) and simp5:(?vf + ?af*δt / 2 ≥ 0)
  using as1 simp2 apply linarith
  using as2 simp3 by linarith

have simp6:(δt*?vr + ?ar*δt2 / 2 ≥ 0) and simp7:(δt*?vf + ?af*δt2 / 2 ≥ 0)
  using as4 simp4 simp5
  by (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)+

have simp4p:(?vr + ?ar*δt / 2 ≥ ?vr / 2)
  and simp5p:(?vf + ?af*δt / 2 ≥ ?vf / 2)
  using as1 simp2 apply linarith
  using as2 simp3 by linarith

have simp6p:(δt*?vr + ?ar*δt2 / 2 ≥ δt*?vr / 2)
  and simp7p:(δt*?vf + ?af*δt2 / 2 ≥ δt*?vf / 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)+

```

## 5 A Safety-Property in Autonomopus Cars: RSS in a N-cars-Scenario

```

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

have simp9:( $?v_f + a * \delta t \leq 0 \implies - (?v_f)^2 / (a * 2) \leq (\delta t * ?v_f) / 2$ ) for a
  using as2 mult_left_mono[where b= $\delta t$  and a= $(-?v_f / a)$  and c= $(?v_f)$ , 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 ( $?v'_f = 0$ )
  then have ( $\max 0 ?b_f \leq ?p'_f - ?p_f$ )
    using as7 as2 as4 apply (auto simp add:k0 r0 mult.commute divide_le_0_iff split:if_splits)

    using divide_left_mono[where c= $(?v_f)^2$  and a= $(a_{maxbrake} * 2)$ ]
    by (smt (z3) zero_less_mult_iff 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 h0:( $?v'_f > 0$ )
  from as7 have h1:( $- ?a_f / a_{maxbrake} \leq 1$ )
    by (auto simp add:k0 r0 split:if_splits) (smt (z3) a1 a2 divide_le_eq_1_neg minus_divide_right)+
  have h2:( $(?v_f + \delta t * ?a_f)^2 / (2 * a_{maxbrake}) = (?v_f)^2 / (2 * a_{maxbrake}) + (?a_f / a_{maxbrake}) * (\delta t * ?v_f + ?a_f * \delta t^2 / 2)$ )
    by (simp add: power2_sum distrib_left mult.commute power2_eq_square add_divide_distrib mult.left_commute)
  have ( $0 \leq ?p'_f - ?p_f$ )
    using as7 h0 simp7 by (auto simp add:k0 r0 mult.commute split:if_splits)
  then have ( $\max 0 (?b_f - ?b'_f) \leq ?p'_f - ?p_f$ )
    using as7 h0 mult_left_mono[OF h1 simp7] h2 a1 a2
    by (auto simp add:k0 r0 mult.commute split:if_splits)
  } note dmin_front_car_motion=this

have front_car:( $\max 0 (?b_f - ?b'_f) \leq ?p'_f - ?p_f$ )
  using dmin_front_car_motion dmin_front_car_stopping simp1 by fastforce

{ assume h0:( $d_{rss} \omega i > d_{real} \omega i$ ) and h1:( $?v'_r > 0$ )
  have h2:( $?a_r / a_{minbrake} \leq -1$ )
    using as6 a1 a2 h0 h1 apply(auto simp add:k0 r0 split:if_splits)
    by (metis a1 add.inverse_inverse le_divide_eq_1_pos minus_divide_left neg_le_iff_le)
  have ( $?p'_r - ?p_r \leq (?v_r)^2 / (2 * a_{minbrake}) - (?v'_r)^2 / (2 * a_{minbrake})$ )
    using as6 a1 a2 h0 h1 mult_left_mono[OF h2 simp6]
    by (auto simp add:k0 r0 power2_sum add_divide_distrib distrib_left 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 ≤ ?p_r - ?p'_r)
  by force
} note dmin_rear_car_nonsafe_breaking=this

{ assume h0:(d_rss ω i > d_real ω i) and h1:(?v'_r = 0)
  have h2:(a ≤ -a_minbrake ⇒ 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 (¬ ?b_r ≤ ?p_r - ?p'_r)
    by argo
} note dmin_rear_car_nonsafe_stopping=this

have dmin_rear_car_nonsafe:(d_rss ω i > d_real ω i ⇒ ?b'_r - ?b_r ≤ ?p_r - ?p'_r)
  using dmin_rear_car_nonsafe_breaking dmin_rear_car_nonsafe_stopping simp0 by fastforce

{ assume h0:(d_rss ω i ≤ d_real ω i) and h1:(?v'_r = 0)
  have h2: ((δt * ?v_r) / 2 ≤ ρ*?v_r)
    using as1 as5 b less_eq_real_def by force
  have h3:(?p'_r - ?p_r ≤ ρ*?v_r)
    using as6 as1 a1 h0 h1 apply(auto simp add:d0 d_min_def 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 ≤ (?v_r + ρ*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:(ρ*?v_r + ?b_r - ?b_f < ?p_f - ?p_r)
    by (smt (verit, best) a0 a3 d0 d1 divide_pos_pos zero_less_mult_iff zero_less_power)
  with h3 have (?b_r - ?b_f < ?p_f - ?p'_r)
    by fastforce
} note dmin_rear_car_safe_stopping=this

{ assume h0:(d_rss ω i ≤ d_real ω i) and h1:(?v'_r > 0)
  have h2:(δt*?v_r ≤ ρ*?v_r)
    using as1 as4 as5 b mult_right_less_imp_less by force
  from as6 have h3:(?a_r*δt^2/2 < a_maxaccel*ρ^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 ≤ (?v_r + ρ*a_maxaccel)^2 / (2*a_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_mult_iff)
  have h5: (?b_r ≤ (?v_r + ρ*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 Autonomopus Cars: RSS in a N-cars-Scenario

```

from h2 h3 h4 h5 have (  $\delta t * v_r + a_r * \delta t^2 / 2 + (\max ?b_r ?b'_r)$ 
  <  $\varrho * v_r + a_{maxaccel} * \varrho^2 / 2 + (v_r + \varrho * a_{maxaccel})^2 / (2 * a_{minbrake})$ 
  by (smt (verit) mult.commute)
with h0 have h6:  $(\delta t * v_r + a_r * \delta t^2 / 2 + (\max ?b_r ?b'_r) - ?b_f < ?p_f - ?p_r)$ 
  by (smt (verit, ccfv_SIG) d0 d1 mult.commute times_divide_eq_right)
have (  $?p'_r - ?p_r = \delta t * v_r + a_r * \delta t^2 / 2$ 
using as6 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:  $(d_{rss} \omega i \leq d_{real} \omega i \implies (\max ?b_r ?b'_r) - ?b_f < ?p_f - ?p'_r)$ 
  by (metis a1 divide_le_cancel less_eq_real_def max_def mult_2 power2_less_eq_zero_iff
    power_zero_numeral dmin_rear_car_safe_motion dmin_rear_car_safe_stopping
    simp0 zero_less_double_add_iff_zero_less_single_add)

have dmin:  $(?b'_r - ?b'_f < d_{real} \omega' i)$ 
  using as3 d1 d_min_def front_car dmin_rear_car_nonsafe dmin_rear_car_safe by fastforce

— no collision's proofs

{ assume h0:  $(?b_r \leq ?b_f)$  and h1:  $(?b'_r < ?b'_f)$ 
  —  $?b'_0 < ?b'_1$ , is added to profit from previous results of dmin
  and h2:  $(?v'_r = 0)$ 
  from h0 have h3:  $(?v_r \leq ?v_f)$ 
    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:  $(?p'_r - ?p_r \leq \delta t * v_r / 2)$ 
    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:  $(?p'_r - ?p_r \leq ?p'_f - ?p_f)$ 
    using as7 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:  $(?b_r \leq ?b_f)$  and h1:  $(?b'_r < ?b'_f)$ 
  and h2:  $(?v'_r > 0)$ 
  from h0 have h3:  $(?v_r \leq ?v_f)$ 
    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:  $(?v'_f > 0)$ 

```

## 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:(?vr + ?ar * δt < ?vf + ?af * δt)
  using as6 as7 apply(auto simp add:k0 r0)
  by (smt (verit, best) a1 a2 frac_le mult.commute power_mono zero_le_power2)
have h6:(δt * ?vr + δt2 * ?ar < δt * ?vf + δt2 * ?af)
  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:(δt * ?vr + δt2 * ?ar / 2 < δt * ?vf + δt2 * ?af / 2)
  by (smt (verit, ccfv_SIG) as4 field_sum_of_halves mult_less_cancel_left_pos)
have (?p'r - ?pr < ?p'f - ?pf)
  using as6 as7 h2 h4 h7
  by (auto simp add:k0 r0 a1 d_min_def d1 divide_nonneg_pos power2_sum not_le)
} note no_collision_if_slow_rear_car_nonstopping=this

```

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

```

{ assume h0:(?bf < ?br) and h2:(?b'r < ?b'f)
  and h3:(?v'f = 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=this

{ assume h0:(?bf < ?br) and h1:(drss ω i > dreal ω i) and h2:(?v'f > 0)
  and h3:(?vf - amaxbrake * δt ≤ 0)
have h4:(?bf ≤ ?p'f - ?pf)
  using as7 h2
apply (auto simp add:k0 mult.commute not_le power2_sum add_divide_distrib split:if_splits)

  by (smt (z3) as.select_convs(3) divide_minus_left h3 minus_divide_right mult_minus_left
simp7p simp9)
  then have (0 < ?p'f - ?p'r)
  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=this

{ assume h0:(?bf < ?br) and h1:(drss ω i > dreal ω i) and h2:(?v'f > 0)
  and h3:(?vf - amaxbrake * δt > 0)

have s1:(amaxbrake / aminbrake ≥ 1 ∧ sqrt amaxbrake ≥ sqrt aminbrake ∧ sqrt aminbrake
> 0)
  using a1 a2 by simp

```

## 5 A Safety-Property in Autonomopus Cars: RSS in a N-cars-Scenario

```

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 ≥ (?v_f) / (?v_r))
  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_divide_le_eq s1 times_divide_eq_left)
have s5:(?v_r / a_minbrake ≥ ?v_f / a_maxbrake)
  using order_trans[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 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_r - a_minbrake*δt > 0)
  using h3 a1 a2 by (metis a1 diff_gt_0_iff_gt less_le_trans mult.commute
pos_less_divide_eq)

have s7:(?v_f) * sqrt a_minbrake ≤ (?v_r) * 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 auto[1]
  by (simp add: a1 le_divide_eq mult.commute)
have s8:( (?v_f - a_maxbrake*δt) / sqrt a_maxbrake
  ≤ (?v_r - a_minbrake *δt) / 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.left_commute
mult_le_cancel_left_pos)
with h3 have s9:( (?v_f - a_maxbrake *δt)^2/(2*a_maxbrake)
  ≤ ((?v_r - a_minbrake *δt)^2/(2*a_minbrake))
  using abs_le_square_iff[where x=((?v_f - a_maxbrake*δt) / sqrt a_maxbrake)
and y = ((?v_r - a_minbrake *δt) / sqrt a_minbrake)]
  using s1 apply (auto simp add:power_divide)
  by (smt (verit, best) divide_less_0_iff s1)

have s10:(δt * ?v_r / 2 ≤ (δt * ?v_r - a_minbrake * δt^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:(δt*?v_f - a_maxbrake *δt^2/2 ≤ ?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  $\langle \delta t * ?v_f - a_{maxbrake} * \delta t^2 / 2 - ?b_f = - (?v_f - a_{maxbrake} * \delta t)^2 / (2 * a_{maxbrake}) \rangle$ 
  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:  $\langle ?p'_f - ?p_f - ?b_f \geq - (?v_f - a_{maxbrake} * \delta t)^2 / (2 * a_{maxbrake}) \rangle$ 
  by linarith

  have  $\langle a \leq - a_{minbrake} \implies (\delta t * ?v_r + a * \delta t^2 / 2) \leq (\delta t * ?v_r - a_{minbrake} * \delta t^2 / 2) \rangle$  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:  $\langle ?p'_r - ?p_r \leq \delta t * ?v_r - a_{minbrake} * \delta t^2 / 2 \rangle$ 
  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, ccv, SIG) divide_le_cancel eq_divide_imp mult.commute)+

  have h7:  $\langle \delta t * ?v_r - a_{minbrake} * \delta t^2 / 2 - ?b_r = - (?v_r - a_{minbrake} * \delta t)^2 / (2 * a_{minbrake}) \rangle$ 
  apply (simp add:power2_diff add_divide_distrib minus_add 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  $\langle ?p'_r - ?p_r - ?b_r \leq ?p'_f - ?p_f - ?b_f \rangle$ 
  by linarith
  then have  $\langle 0 < ?p'_f - ?p'_r \rangle$ 
  using as3 d1 d_min_def by auto
} note REM_front_car_nonstopping=this

have nocollision:  $\langle 0 < d_{real} \omega' i \rangle$ 
  apply (cases  $\langle ?b'_r \geq ?b'_f \rangle$ )
  using dmin apply simp
  apply (cases  $\langle ?b_r \leq ?b_f \rangle$ )
  using as3 d1 d_min_def no_collision_if_slow_rear_car_nonstopping
no_collision_if_slow_rear_car_stopping simp0
  apply fastforce
  apply (cases  $\langle d_{rss} \omega i > d_{real} \omega i \rangle$ , 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  $\langle d_{min} \omega' i < d_{real} \omega' i \rangle$ 
  using dmin nocollision by (simp add: d_min_def)
qed

```

## 5 A Safety-Property in Autonomopus Cars: RSS in a N-cars-Scenario

```
next
  fix  $\omega$ 
  have  $\langle P_{inv} \omega \implies (i < j \wedge j \leq N) \longrightarrow pos(\omega i) < pos(\omega j) \rangle$  for  $i j$ 
    unfolding  $d_{min\_def}$   $d1$   $P_{inv\_def}$ 
    apply(induct  $j$ , simp)
    by (smt (z3) Suc_leD Suc_le_lessD add commute less_Suc_eq plus_1_eq_Suc)
  then show  $\langle P_{inv} \omega \implies no_{collision} \omega \{0..N\} 0 \rangle$ 
    unfolding  $no_{collision\_0}$ 
    by (smt (z3) atLeastAtMost_iff inj_on_def less_le not_less)
  qed
qed

end

end
```

## 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 samelane_samedirection_Ncars =
```

```
  fixes  $a_{maxaccel}::real$  and  $a_{minbrake}::real$  and  $a_{maxbrake}::real$  and  $\rho::real$  and  $\varepsilon::real$ 
```

```
  assumes  $a0:(0 < a_{maxaccel})$ 
    and  $a1:(a_{minbrake} < 0)$ 
    and  $a2:(a_{maxbrake} \leq a_{minbrake})$ 
    and  $a3:(\rho > 0)$ 
    and  $a4:(\varepsilon > 0)$ 
```

```
begin
```

```
definition  $d_{real}::(real\ scene \Rightarrow nat \Rightarrow real)$ 
  where  $d0:(d_{real}\ \omega\ i \equiv pos(\omega\ (i+1)) - pos(\omega\ i))$ 
```

```
definition  $d_{rss}::(real\ scene \Rightarrow nat \Rightarrow real)$ 
  where  $d1:(d_{rss}\ \omega\ i \equiv let\ v_r = speed(\omega\ i) ; v_f = speed(\omega\ (i+1))\ in$ 
    if  $v_r \geq 0$ 
    then  $max\ 0\ ((\rho*v_r + (\rho^2/2)*a_{maxaccel} + (v_r + \rho*a_{maxaccel})^2/(2*-a_{minbrake}))$ 
    -  $(v_f)^2/(2*-a_{maxbrake}))$ 
    else undefined)
```

Considering the next acceleration of the rear car where  $a_r < a_{maxaccel}$

```
definition  $d_{safe}::(real\ scene \Rightarrow nat \Rightarrow real \Rightarrow real)$ 
  where  $d2:(d_{safe}\ \omega\ i\ a_r \equiv let\ v_r = speed(\omega\ i) ; v_f = speed(\omega\ (i+1))\ in$ 
    if  $v_r \geq 0$ 
    then  $max\ 0$ 
      ((if  $v_r + \rho*a_r > 0$ 
        then  $(\rho*v_r + (\rho^2/2)*a_r + (v_r + \rho*a_r)^2/(2*-a_{minbrake}))$ 
        else  $(v_r)^2/(2*-a_r)$ )
      -  $(v_f)^2/(2*-a_{maxbrake}))$ 
    else undefined)
```

```
definition  $drive_{rss}::(real \Rightarrow real\ motion_{acc})$ 
```



```

using hh2 hh0 by (smt (verit) a3 divide_le_0_iff neg_divide_le_eq)
have (← ((speed(ω i))2 / (2 * x)) ≤ ρ * speed(ω i) + ρ2 * y / 2 - (speed(ω i) + ρ * y)2 / (2
* aminbrake))
  using mult_mono[OF hh5 hh4, simplified] hh3
  by (smt (verit, del_insts) a3 distrib_left divide_minus_left hh0 mult.assoc power2_eq_square
times_divide_eq_right)
} note h3=this
{ assume hh0:(speed(ω i) ≥ 0)
  and hh1:(speed(ω i) + ρ*y ≤ 0)
  and hh2:(speed(ω i) + ρ*x ≤ 0)
  have hh3:(y ≤ 0)
  using hh0 hh1 a3
  by (metis add.commute le_add_same_cancel1 not_less order_trans zero_less_mult_iff)
  have (← ((speed(ω i))2 / (2 * x)) ≤ - ((speed(ω i))2 / (2 * y)))
  by (smt (verit, best) a3 divide_eq_0_iff frac_le hh0 h0 hh1 minus_divide_right zero_eq_power2
zero_le_mult_iff zero_le_power)
} note h4=this
show (dsafe ω i x ≤ dsafe ω i y)
  unfolding max_def d2 using h0 h1 h2 h3 h4 by auto
qed

```

### 6.3 Our motion vs RSS

```

lemma dsafe_drss: (drss ω i = dsafe ω i amaxaccel)
  using d1 d2 a0 a3 by (auto simp add: le_less_trans)

```

```

lemma drivesafe_driverss: (driverss ε' i δt ω ⊆ drivesafe ε' i δt ω)
  apply(cases (drss ω i + ε' ≤ dreal ω i))
  using a0 a1 a2 r0 r1 apply (auto simp:dsafe_drss)
  by (meson add_mono_thms_linordered_semiring(3) dsafe_mono monoD order.trans)

```

```

lemma carssafe_carsrss:
  (⟦sid:kineticsfw (drivesafe ε')⟧ ≤ ⟦sid:kineticsfw (driverss ε')⟧)
  apply(intro all impl μmotion_refine)
  using a0 a1 a2 kineticsfw_empty r0 apply auto[1]
  using drivesafe_driverss k0 apply auto
  using atLeastAtMost_iff by blast+

```

```

corollary issafe {0..N} (kineticsfw (drivesafe ε')) Δt ω0 ε ⇒ issafe {0..N} (kineticsfw
(driverss ε')) Δt ω0 ε)
  by (rule issafe_refine, rule carssafe_carsrss, assumption)

```

### 6.4 Our motion is safe

```

definition dmin:: (real scene ⇒ nat ⇒ real)
  where d3:(dmin ω i ≡ let vr = speed(ω i) ; vf = speed(ω (i+1)) in
    if vr ≥ 0
      then max 0 ((vr)2/(2*-aminbrake) - (vf)2/(2*-amaxbrake))

```



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

else undefined)

```
lemma  $d_{min\_d\_safe}$ :  $\langle d_{min} \omega i = d_{safe} \omega i a_{minbrake} \rangle$ 
  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:
```

```
  fixes  $N::nat$ 
```

```
  assumes a:  $\langle \forall i \in \{0..N\}. d_{min} \omega_0 i + \varepsilon' \leq d_{real} \omega_0 i \wedge 0 \leq speed(\omega_0 i) \rangle$ 
```

```
    and b:  $\langle 0 < \Delta t \wedge \Delta t \leq \varrho \rangle$ 
```

```
    and c:  $\langle \varepsilon < \varepsilon' \rangle$ 
```

```
  shows  $\langle is_{safe} \{0..N\} (kinetics_{fw} (drive_{safe} \varepsilon')) \Delta t \omega_0 \varepsilon \rangle$ 
```

```
proof -
```

```
  define  $P_{inv}$  where  $\langle P_{inv} \equiv \lambda (\omega::real\ scene). (\forall i \leq N. 0 \leq speed(\omega i)) \wedge (\forall i < N. d_{min} \omega i + \varepsilon' \leq d_{real} \omega i) \rangle$ 
```

```
  show ?thesis
```

```
proof (rule is_safe_rule_inv[where  $P_{inv} = (\lambda \omega\ sid. P_{inv} \omega)$ ])
```

```
  show  $\langle 0 < \Delta t \rangle$  using b by simp
```

```
next
```

```
  show  $\langle P_{inv} \omega_0 \rangle$ 
```

```
  using a c by (auto simp add:  $P_{inv\_def}$  d0)
```

```
next
```

```
  fix  $\omega \delta t$ 
```

```
  have  $\langle \forall i \in \{0..N\}. \{a_{maxbrake}..a_{minbrake}\} \subseteq drive_{safe} \varepsilon' i \delta t \omega \rangle$ 
```

```
  using r1 a0 a1 a2 by auto
```

```
  then show  $\langle P_{inv} \omega \wedge \delta t \in \{0.. \Delta t\} \implies \forall i \in \{0..N\}. kinetics_{fw} (drive_{safe} \varepsilon') i \delta t \omega \neq \{\} \rangle$ 
```

```
  using a2 kinetics_fw_empty by fastforce
```

```
next
```

```
  fix  $\omega \delta t$ 
```

```
  show  $\langle P_{inv} \omega \wedge \delta t \in \{0.. \Delta t\} \implies \forall \omega' \in \{\omega'. \forall i \in \{0..N\}. \omega' i \in kinetics_{fw} (drive_{safe} \varepsilon') i \delta t \omega\}. P_{inv} \omega' \rangle$ 
```

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

```
  fix  $\omega' i$ 
```

```
  assume  $\langle \forall i \in \{0..N\}. \omega' i \in kinetics_{fw} (drive_{safe} \varepsilon') i \delta t \omega \rangle$  and  $\langle i \leq N \rangle$  and  $\langle 0 < \delta t \rangle$ 
```

```
  then show  $\langle 0 \leq speed(\omega' i) \rangle$ 
```

```
  apply (auto simp: k0 r1 mult.commute split: if_splits)
```

```
  by (metis as.select_convs(2) atLeastAtMost_iff less_eq_real_def not_less_le)
```

```
next
```

```
  fix  $\omega' i$ 
```

```
  assume  $assm1: \langle \forall i \leq N. 0 \leq speed(\omega i) \rangle$ 
```

```
  and  $assm2: \langle \forall i < N. d_{min} \omega i + \varepsilon' \leq d_{real} \omega i \rangle$ 
```

```
  and  $assm3: \langle 0 < \delta t \rangle$ 
```

```
  and  $assm4: \langle \delta t \leq \Delta t \rangle$ 
```

```
  and  $assm5: \langle \forall i \in \{0..N\}. \omega' i \in kinetics_{fw} (drive_{safe} \varepsilon') i \delta t \omega \rangle$ 
```

```
  and  $assm6: \langle i < N \rangle$ 
```

```

have as1:(0 ≤ speed(ω i))
  using assm1[rule_format, of i] assm6 by simp
have as2:(0 ≤ speed(ω (i+1)))
  using assm1[rule_format, of (i+1)] assm6 by simp
note as3 = assm2[rule_format, of i, OF assm6]
note as4 = assm3
note as5 = assm4
have as6:(ω' i ∈ kineticsfw (drivesafe ε') i δt ω)
  using assm5[rule_format, of i] assm6 by simp
have as7:(ω' (i+1) ∈ kineticsfw (drivesafe ε') (i+1) δt ω)
  using assm5[rule_format, of (i+1)] assm6 by simp

```

```

let ?a0 = ⟨acc(ω' i)⟩ and ?a1 = ⟨acc(ω' (i+1))⟩
and ?b0 = ⟨(speed(ω i))2 / (2*-aminbrake)⟩ and ?b1 = ⟨(speed(ω (i+1)))2 / (2*-amaxbrake)⟩

and ?b'0 = ⟨(speed(ω' i))2 / (2*-aminbrake)⟩ and ?b'1 = ⟨(speed(ω' (i+1)))2 / (2*-amaxbrake)⟩

```

some simplifying rules factorized here

```

have simp0:(0 ≤ speed(ω' i)) and simp1:(0 ≤ speed(ω' (i+1)))
  using as6 as7 by (auto simp:k0 r1 mult.commute split:if_splits)

```

```

have simp2:(speed(ω i) + δt*a > 0 ⇒ speed(ω i) + δt*a/2 > speed(ω i)/2) and simp3:(speed(ω (i+1)) + δt*a > 0 ⇒ speed(ω (i+1)) + δt*a/2 > speed(ω (i+1))/2) for a δt
  by simp+

```

— a lower bound of the traversed distance when non stopping

```

have simp4:(δt > 0 ⇒ speed(ω i) + δt*a > 0 ⇒ δt*speed(ω i) + (δt2/2)*a > δt*speed(ω i)/2) and simp5:(δt > 0 ⇒ speed(ω (i+1)) + δt*a > 0 ⇒ δt*speed(ω (i+1)) + (δt2/2)*a > δt*speed(ω (i+1))/2) for a δt
  using mult_strict_left_mono[OF simp2, of δt a δt] mult_strict_left_mono[OF simp3, of δt a δt]
  by (simp_all add: distrib_left power2_eq_square)

```

```

have simp4':(δt > 0 ⇒ speed(ω i) + δt*a > 0 ⇒ δt*speed(ω i) + (δt2/2)*a ≥ 0) and simp5':(δt > 0 ⇒ speed(ω (i+1)) + δt*a > 0 ⇒ δt*speed(ω (i+1)) + (δt2/2)*a ≥ 0) for a δt
  using as1 as2 simp4 simp5 zero_le_mult_iff by fastforce+

```

— an upper bound of the traversed distance when stopping

```

have simp6:(δt > 0 ⇒ speed(ω i) + δt*a ≤ 0 ⇒ -(speed(ω i))2/(2*a) ≤ δt*speed(ω i)/2) and simp7:(δt > 0 ⇒ speed(ω (i+1)) + δt*a ≤ 0 ⇒ -(speed(ω (i+1)))2/(2*a) ≤ δt*speed(ω (i+1))/2) for a δt
  using as1 as2 mult_left_mono[where b=(δt) and a=(-speed(ω i) / a) and c=(speed(ω i)), OF _ as1] mult_left_mono[where b=(δt) and a=(-speed(ω (i+1)) / a) and c=(speed(ω (i+1))), 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 rear_car_forward:  $\langle 0 \leq \text{pos}(\omega' i) - \text{pos}(\omega i) \rangle$ 
  using as6 simp4 as1 as4 a1 a2 apply (auto simp add:k0 r1 mult.commute divide_nonneg_nonpos field_sum_of_halves split:if_splits)
  apply (smt (verit, best) divide_eq_0_iff divide_nonneg_neg mult_nonneg_nonneg zero_eq_power2 zero_le_power)
  by (metis (no_types, hide_lams) mult.commute not_le simp4' times_divide_eq_right)+

have front_car_forward:  $\langle 0 \leq \text{pos}(\omega' (i+1)) - \text{pos}(\omega (i+1)) \rangle$ 
  using as7 simp5 as2 as4 a1 apply (auto simp add:k0 r1 mult.commute divide_nonneg_nonpos field_sum_of_halves split:if_splits)
  apply (smt (verit, best) divide_eq_0_iff divide_nonneg_neg mult_nonneg_nonneg zero_eq_power2 zero_le_power)
  by (metis (no_types, hide_lams) mult.commute not_le simp5' times_divide_eq_right)+

```

### 6.4.1 Safety distance proofs

```

{ assume h0:  $\text{speed}(\omega' (i+1)) = 0$ 
  then have  $\langle ?b_1 \leq \text{pos}(\omega' (i+1)) - \text{pos}(\omega (i+1)) \rangle$ 
    using a1 a2 as2 as7 as4 apply (auto simp add:k0 r1 split:if_splits)
    apply (subst times_divide_eq_right[symmetric], rule mult_left_mono[of 1 _  $\langle (\text{speed}(\omega (i+1)))^2 \rangle$ , simplified], simp)
    apply (cases  $\text{speed}(\omega (i+1)) = 0$ , simp)
    apply (subst times_divide_eq_right[symmetric], rule mult_left_mono[of 1 _  $\langle (\text{speed}(\omega (i+1)))^2 \rangle$ , simplified])
    by (metis add_0_right add_mono_thms_linordered_field(5) divide_less_eq_1 le_divide_eq_1 mult_eq_0_iff not_le zero_less_mult_iff)
  } note dmin_front_car_stopping=this

{ assume h0:  $\text{speed}(\omega' (i+1)) > 0$ 
  from as7 have h1:  $\langle ?a_1 / a_{\text{maxbrake}} \leq 1 \rangle$ 
  using a1 by (auto simp add:k0 r1) (smt (z3) a1 a2 divide_le_eq_1_neg minus_divide_right)
  have  $\langle (\text{speed}(\omega (i+1)) + \delta t * ?a_1)^2 / (2 * a_{\text{maxbrake}}) = (\text{speed}(\omega (i+1)))^2 / (2 * a_{\text{maxbrake}}) + (?a_1 / a_{\text{maxbrake}}) * (\delta t * \text{speed}(\omega (i+1)) + (\delta t^2 / 2) * ?a_1) \rangle$ 
    by (simp add: power2_sum distrib_left mult.commute power2_eq_square add_divide_distrib mult.left_commute)
  then have  $\langle ?b_1 - ?b'_1 \leq (\text{pos}(\omega' (i+1)) - \text{pos}(\omega (i+1))) \rangle$ 
    using h0 as7 mult_left_mono[OF h1 simp5'] as4 by (auto simp add:k0 r1 mult.commute split:if_splits)
  } note dmin_front_car_motion=this

have front_car:  $\langle ?b_1 - ?b'_1 \leq \text{pos}(\omega' (i+1)) - \text{pos}(\omega (i+1)) \rangle$ 
  using dmin_front_car_motion dmin_front_car_stopping simp1 by fastforce

{ fix a
  assume h0:  $\text{speed}(\omega' i) = 0$ 
  assume h1:  $\langle a \leq a_{\text{minbrake}} \rangle$  and h2:  $\langle \text{pos}(\omega' i) - \text{pos}(\omega i) = (\text{speed}(\omega i))^2 / (2 * -a) \rangle$ 
  have h3:  $\langle (\text{speed}(\omega i))^2 / (2 * -a) \leq (\text{speed}(\omega i))^2 / (2 * -a_{\text{minbrake}}) \rangle$ 

```

```

    using a1 a2 h1 by (smt (z3) frac_le zero_le_power2)
with h2 have ⟨pos (ω' i) - pos (ω i) ≤ ?b₀⟩
  by linarith
} note dmin_rear_car_stopping_after_breaking=this

{ fix a
  assume h0:(speed(ω' i) = 0)
  assume h1:(a > a_minbrake) and h2:(d_safe ω i a + ε' ≤ d_real ω i) and h3:(pos (ω' i) - pos
(ω i) = (speed(ω i)²/(2*a)) and h4:(speed(ω i) + δt*a ≤ 0)
  have h5:(speed(ω i) + ρ*a ≤ 0)
    by (smt (verit, best) as1 assm3 assm4 b h4 mult_le_cancel_right mult_pos_pos)
  have ⟨pos(ω' i) - pos(ω i) - ?b₁ + ε' ≤ pos(ω (i+1)) - pos(ω i)⟩
    using h2 h5 apply (auto simp add:k0 r1 d0 d2 as1 max_def split:if_splits)
    using c h0 h3 by force+
} note dmin_rear_car_stopping_after_slowing=this

have dmin_rear_car_stopping:(speed(ω' i) = 0 ⇒ - ?b₁ + ε' ≤ pos(ω (i+1)) - pos(ω' i))
proof -
  assume h0:(speed(ω' i) = 0)
  then obtain a where h1:(a ≤ a_minbrake ∨ d_safe ω i a + ε' ≤ d_real ω i) and h3:(pos (ω'
i) - pos (ω i) = (speed(ω i)²/(2*a)) and h4:(speed(ω i) + δt*a ≤ 0)
    using as6 apply(auto simp add:k0 r1 a1 split:if_splits)
    apply blast apply blast
    using a1 apply linarith+
    by fastforce
  show ?thesis
    apply(cases ⟨a ≤ 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(ω' i) > 0) and h1:(?a₀ ≤ a_minbrake) — h0 is not necessary, it is induced
by h1
  have h2:(?a₀/a_minbrake ≥ 1)
    using a1 a2 h1 by auto
  with h1 have ⟨pos (ω' i) - pos (ω i) ≤ ?b₀ - ?b'₀⟩
    using as6 a1 a2 h1 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(ω' i) > 0) and h1:(?a₀ > a_minbrake) and h2:(?a₀ ≥ 0)
  have h3:(speed(ω' i) = speed(ω i) + ?a₀*δt) and h4:(speed(ω i) + ?a₀*δt > 0)
    using h0 as6 by (auto simp:k0 r1 mult.commute)
  then have h5:(speed(ω i) + ?a₀*ρ > 0)
    by (smt (verit) a3 as1 h2 mult_nonneg_nonneg zero_less_mult_iff)
  from h0 h1 have h6:(d_safe ω i ?a₀ + ε' ≤ d_real ω i)

```

## 6 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:( $\delta t * \text{speed}(\omega i) + (\delta t^2 / 2) * ?a_0 + (\text{speed}(\omega i) + \delta t * ?a_0)^2 / (2 * -a_{\text{minbrake}})$ 
 $\leq \varrho * \text{speed}(\omega i) + (\varrho^2 / 2) * ?a_0 + (\text{speed}(\omega i) + \varrho * ?a_0)^2 / (2 * -a_{\text{minbrake}})$ ) (is (?A))
proof-
  have h8:( $0 < 1 + ?a_0 / -a_{\text{minbrake}}$ )
  using a1 a2 h1 by auto
  then have h9:( $0 \leq \text{speed}(\omega i) + ?a_0 * (\varrho + \delta t) / 2 + (?a_0 / -a_{\text{minbrake}}) * (\text{speed}(\omega i) +$ 
 $?a_0 * (\varrho + \delta t) / 2)$ )
  using mult_pos_pos[OF h8, of ( $\text{speed}(\omega i) + ?a_0 * (\varrho + \delta t) / 2$ ), simplified distrib_right]
  by (smt (z3) distrib_left field_sum_of_halves h4 h5)
  have h10:( $(\varrho - \delta t) * (\text{speed}(\omega i) + (\varrho + \delta t) * ?a_0 / 2)$ 
 $= (\varrho * \text{speed}(\omega i) - \delta t * \text{speed}(\omega i)) + (\varrho^2 / 2 * ?a_0 - \delta t^2 / 2 * ?a_0)$ )
  apply(subst distrib_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:( $(\varrho - \delta t) * (?a_0 / -a_{\text{minbrake}}) * (\text{speed}(\omega i) + (\varrho + \delta t) * ?a_0 / 2)$ 
 $= (\text{speed}(\omega i) + \varrho * ?a_0)^2 / (2 * -a_{\text{minbrake}}) - (\text{speed}(\omega i) + \delta t * ?a_0)^2 /$ 
 $(2 * -a_{\text{minbrake}})$ )
  apply(subst diff_divide_distrib[symmetric])
  apply(simp add:power2_eq_square square_diff_square_factored)
  apply(subst divide_divide_eq_left[symmetric],
subst (7) mult.commute, subst left_diff_distrib, subst divide_cancel_right)
  using a1 by (simp add: distrib_right)
  have ( $0 \leq (\varrho * \text{speed}(\omega i) - \delta t * \text{speed}(\omega i)) +$ 
 $(\varrho^2 / 2 * ?a_0 - \delta t^2 / 2 * ?a_0) +$ 
 $((\text{speed}(\omega i) + \varrho * ?a_0)^2 / (2 * -a_{\text{minbrake}}) - (\text{speed}(\omega i) + \delta t * ?a_0)^2 / (2 * -a_{\text{minbrake}}))$ )
  apply(subst h10[symmetric], subst h11[symmetric])
  using mult_nonneg_nonneg[OF h9, of ( $\varrho - \delta t$ )]
  by (metis (no_types, hide_lams) assm4 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 ( $\text{pos}(\omega' i) - \text{pos}(\omega i) + ?b'_0 - ?b_1 + \varepsilon' \leq \text{pos}(\omega(i+1)) - \text{pos}(\omega i)$ )
  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_accelerating=this

{ assume h0:( $\text{speed}(\omega' i) > 0$ ) and h1:( $?a_0 > a_{\text{minbrake}}$ )
and h2:( $?a_0 < 0$ ) and h3:( $\text{speed}(\omega i) + ?a_0 * \varrho > 0$ )
have h4:( $\text{speed}(\omega' i) = \text{speed}(\omega i) + ?a_0 * \delta t$ ) and h5:( $\text{speed}(\omega i) + ?a_0 * \delta t > 0$ )
  using h0 as6 by (auto simp:k0 r1 mult.commute)
from h0 h1 have h6:( $d_{\text{safe}} \omega i ?a_0 + \varepsilon' \leq d_{\text{real}} \omega i$ )
  using as6 k0 r1 apply(simp split:if_splits) by auto
have h7:( $\delta t * \text{speed}(\omega i) + (\delta t^2 / 2) * ?a_0 + (\text{speed}(\omega i) + \delta t * ?a_0)^2 / (2 * -a_{\text{minbrake}})$ 
 $\leq \varrho * \text{speed}(\omega i) + (\varrho^2 / 2) * ?a_0 + (\text{speed}(\omega i) + \varrho * ?a_0)^2 / (2 * -a_{\text{minbrake}})$ ) (is (?A))
proof-
  have h8:( $0 < 1 + ?a_0 / -a_{\text{minbrake}}$ )
  using a1 a2 h1 by auto
  then have h9:( $0 \leq \text{speed}(\omega i) + ?a_0 * (\varrho + \delta t) / 2 + (?a_0 / -a_{\text{minbrake}}) * (\text{speed}(\omega i) +$ 

```

```

?a0*(ρ+δt)/2)
  using mult_pos_pos[OF h8, of (speed(ω i) + ?a0*(ρ+δt)/2), simplified distrib_right]
  by (smt (z3) distrib_left field_sum_of_halves h3 h5)
have h10: ( (ρ - δt)*(speed(ω i) + (ρ+δt)*?a0/2)
  = (ρ*speed(ω i) - δt*speed(ω i)) + (ρ2/2*?a0 - δt2/2*?a0)
  apply(subst distrib_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: ( (ρ - δt)*(?a0/-aminbrake)*(speed(ω i) + (ρ+δt)*?a0/2)
  = (speed(ω i) + ρ*?a0)2 / (2*-aminbrake) - (speed(ω i) + δt*?a0)2 /
(2*-aminbrake)
  apply(subst diff_divide_distrib[symmetric])
  apply(simp add:power2_eq_square square_diff_square_factored )
  apply(subst divide_divide_eq_left[symmetric], subst (7) mult.commute,
    subst left_diff_distrib, subst divide_cancel_right)
  using a1 by (simp add: distrib_right)
have (0 ≤ (ρ*speed(ω i) - δt*speed(ω i)) +
  (ρ2/2*?a0 - δt2/2*?a0) +
  ((speed(ω i) + ρ*?a0)2/(2*-aminbrake) - (speed(ω i) + δt*?a0)2/(2*-aminbrake)))
  apply(subst h10[symmetric], subst h11[symmetric])
  using mult_nonneg_nonneg[OF _ h9, of (ρ - δt)]
  by (metis (no_types, hide_lams) assm4 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'0 - ?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_safe_slowing_no_possibly_stopping=this

```

— `dmin_rear_car_safe_accelerating` and `dmin_rear_car_safe_slowing_no_possibly_stopping` can be merged

```

{ assume h0:(speed(ω' i) > 0) and h1:(?a0 > aminbrake)
  and h2:(?a0 < 0) and h3:(speed(ω i) + ?a0*ρ ≤ 0)
  from h0 h1 have h4:(dsafe ω i ?a0 + ε' ≤ dreal ω i)
  using as6 k0 r1 apply(simp split:if_splits) by auto
  have h5:(speed(ω i) + ?a0*δt > 0) and h6:(speed(ω' i) = speed(ω i) + ?a0*δt)
  using h0 as6 as1 as4 by (auto simp:k0 r1 mult.commute)
  have ( δt*speed(ω i) + (δt2/2)*?a0 + (speed(ω i) + δt*?a0)2 / (2*-aminbrake)
    ≤ (speed(ω i))2 / (2*-?a0) ) (is (?A))
  proof -
    from h2 have h7:( δt*speed(ω i) + (δt2/2)*?a0 + (speed(ω i) + δt*?a0)2 / (2*-?a0)
      = (speed(ω i))2 / (2*-?a0)
    apply (simp add:power2_sum add_divide_distrib)
    by (smt (z3) eq_divide_imp mult.assoc mult.commute power2_eq_square)
  show ?A
  using h1 h2 h7 h5 h6
  by (smt (z3) frac_le zero_le_power2)

```

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

```

qed
with h2 h3 have (pos (ω' i) - pos (ω i) + ?b'_0 - ?b_1 + 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_safe_slowing_possibly_stopping=this

have dmin_rear_car_safe_motion:(speed(ω' i) > 0 ⇒ ?a_0 > a_minbrake ⇒
  ?b'_0 - ?b_1 + ε' ≤ pos (ω (i+1)) - pos (ω' i))
using dmin_rear_car_safe_accelerating dmin_rear_car_safe_slowing_no_possibly_stopping

  dmin_rear_car_safe_slowing_possibly_stopping by force

have dmin:(?b'_0 - ?b'_1 + ε' ≤ d_real ω' i)
  apply(cases (speed(ω' i) = 0))
  using d0 dmin_rear_car_stopping front_car apply simp
  apply(cases (?a_0 ≤ a_minbrake))
  using as1 as3 d0 d3 dmin_rear_car_breaking_without_stopping
  dmin_rear_car_safe_motion front_car simp0 by auto

```

### 6.4.2 "No collision" proofs when $?b'_0 < ?b'_1$

```

{ assume h0:(?b_0 ≤ ?b_1) and h1:(?b'_0 < ?b'_1) and h2:(speed(ω' i) = 0)
  from h0 have h3:(speed(ω i) ≤ speed(ω (i+1)))
    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 h4:(pos(ω' i) - pos(ω i) ≤ δt*speed(ω i) / 2)
    using as6 as1 simp6[OF as4] by(auto simp add:k0 r1)
  have h5:(speed(ω' (i+1)) > 0)
    using h1 h2 not_le simp1 by fastforce
  then have h6:(δt*speed(ω (i+1)) / 2 < pos (ω' (i+1)) - pos (ω (i+1)))
    using as7 h1 h2 simp5[OF as4] by (auto simp add:k0 r1 a1)
  have (pos (ω' i) - pos (ω i) ≤ pos (ω' (i+1)) - pos (ω (i+1)))
    using as4 h3 h4 h6 by (smt (verit, ccfv_SIG) divide_le_cancel mult_less_cancel_left_pos)
} note no_collision_if_slow_rear_car_stopping=this

{ assume h0:(?b_0 ≤ ?b_1) and h1:(?b'_0 < ?b'_1) and h2:(speed(ω' i) > 0)
  from h0 have h3:(speed(ω i) ≤ speed(ω (i+1)))
    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:(speed(ω' (i+1)) > 0)
    by (smt (z3) a1 a2 frac_le h1 simp1 zero_le_power2 zero_power2)
  have ((speed(ω' i))^2/(2*-a_maxbrake) < ?b'_1)
    by (smt (verit, ccfv_SIG) a1 a2 frac_le h1 zero_le_power2)
  with h1 h2 h4 have h5:(speed(ω i) + ?a_0*δt < speed(ω (i+1)) + ?a_1*δt)
    using as6 as7 apply(auto simp:k0 r1)
    by (smt (verit, ccfv_SIG) a1 a2 divide_le_cancel mult.commute
      power2_le_imp_le power_mono zero_less_numeral)+
  have h6:(δt*speed(ω i) + δt^2*?a_0 < δt*speed(ω (i+1)) + δt^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: (δt*speed(ω i) + δt²*?a₀/2 < δt*speed(ω (i+1)) + δt²*?a₁/2)
  by (smt (verit, ccfv_SIG) as4 field_sum_of_halves mult_less_cancel_left_pos)
have (pos (ω' i) - pos (ω i) < pos (ω' (i+1)) - pos (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_slow_rear_car_nonstopping=this

{ assume h1:(?b'₀ < ?b'₁) 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:(?b₁ ≤ 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 simp5[OF as4] simp7[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 (ε' ≤ pos (ω' (i+1)) - pos (ω' i))
  using a1 simp0 as1 as3[simplified d0 d3] dmin_rear_car_breaking_without_stopping
      dmin_rear_car_safe_motion dmin_rear_car_stopping
  by (smt (z3) divide_le_0_iff zero_less_power)
} note no_collision_if_front_car_may_stop_but_dont=this

{ assume h0:(?b₁ < ?b₀) and h1:(?a₀ ≤ 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(-a_minbrake) > 0)
      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: 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)

```



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

```

have s5:(speed(ω i)/-a_minbrake ≥ speed(ω (i+1))/-a_maxbrake)
  using order_trans[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 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(ω (i+1))/-a_maxbrake > δt)
  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(ω i) + a_minbrake*δt > 0)
  using h2 a1 a2
  by (smt (z3) le_divide_eq minus_divide_right mult.commute mult_minus_right)

have s7:(speed(ω (i+1))*sqrt(-a_minbrake) ≤ (speed(ω i))*sqrt(-a_maxbrake))
  using s4 by (metis divide_le_eq h3 le_divide_eq mult.commute s1 times_divide_eq_left)
  have s8:(speed(ω (i+1)) + a_maxbrake*δt)/sqrt(-a_maxbrake) ≤ (speed(ω i) +
a_minbrake*δt)/sqrt(-a_minbrake))
  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)
with h3 have s9:( (speed(ω (i+1)) + a_maxbrake*δt)2/(2*-a_maxbrake)
  ≤ ((speed(ω i) + a_minbrake *δt)2/(2*-a_minbrake)))
  using abs_le_square_iff[where x=(speed(ω (i+1)) + a_maxbrake*δt)/sqrt(-a_maxbrake)]
    and y = ((speed(ω i) + a_minbrake*δt)/sqrt(-a_minbrake))]
  using s1 apply (auto simp add:power_divide)
  using h2 s6 by linarith

have h4:(δt*speed(ω (i+1)) + a_maxbrake*δt2/2 ≤ pos(ω' (i+1)) -pos(ω (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 (δt*speed(ω (i+1)) + a_maxbrake *δt2/2 -?b1 = -(speed(ω (i+1)) + a_maxbrake
*δt)2/(2*-a_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 h4 have h5:( pos(ω' (i+1)) - pos(ω (i+1)) - ?b1
  ≥ -(speed(ω (i+1)) + a_maxbrake*δt)2/(2*-a_maxbrake))
  by linarith

have h6':(a ≤ a_minbrake ⇒ (δt*speed(ω i) + a*δt2 / 2) ≤ δt*speed(ω i) + a_minbrake*δt2
/ 2) for a
  by (smt (verit) as4 h1 eq_divide_imp field_sum_of_halves mult_minus_left
pos_minus_divide_le_eq zero_less_power2)

```

```

have (speed (ω i) + a*δt ≤ 0 ⇒ (speed(ω i))2/(2*-a) ≤ δt*speed(ω i)/2) for a
  using simp6[OF as4] by (smt (z3) divide_minus_left minus_divide_right mult.commute)
  then have h6':(speed (ω i) + a*δt ≤ 0 ⇒ (speed(ω i))2/(2*-a) ≤ δt*speed(ω i) +
a_minbrake*δt2 / 2) for a
  using simp4[OF as4, of a_minbrake] s6
  by (smt (verit) mult.commute times_divide_eq_right)
from h6' h6'' have h6:(pos(ω' i) - pos(ω i) ≤ δt*speed(ω i) + a_minbrake*δt2/2)
  using as6 a1 h1 h3 apply (auto simp add:k0 r1 d0 mult.commute split:if_splits)
  by (smt (z3) mult.commute)+

  have h7: (δt*speed(ω i) + a_minbrake*δt2/2 - ?b0 = -(speed(ω i) + a_minbrake
*δt)2/(2*-a_minbrake))
  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 h5 h6 s9 have (pos(ω' i) - pos(ω i) - ?b0 ≤ pos(ω' (i+1)) - pos(ω (i+1)) - ?b1)

  by linarith
  then have (ε' ≤ pos (ω' (i+1)) - pos (ω' 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:(?b1 < ?b0) and h1:(?a0 > a_minbrake) and h2:(speed(ω (i+1)) + a_maxbrake *δt
> 0) and h3:(speed(ω' i) > 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(-a_minbrake) > 0)
    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: 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:(speed(ω i)/-a_minbrake ≥ speed(ω (i+1))/-a_maxbrake)
    using order_trans[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

```

## 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(ω (i+1))/-a_maxbrake > δt)
  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(ω i) + a_minbrake*δt > 0)
  using h2 a1 a2
  by (smt (z3) le_divide_eq minus_divide_right mult.commute mult_minus_right)
have s6':(speed(ω i) + δt*a_0 > speed(ω i) + a_minbrake*δt)
  by (simp add: assm3 h1)

have s7:((speed(ω (i+1))*sqrt(-a_minbrake) ≤ (speed(ω i))*sqrt(-a_maxbrake))
  using s4 by (metis divide_le_eq h3' le_divide_eq mult.commute s1 times_divide_eq_left)
  have s8:((speed(ω (i+1)) + a_maxbrake*δt)/sqrt(-a_maxbrake) ≤ (speed(ω i) +
a_minbrake*δt)/sqrt(-a_minbrake))
  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(ω (i+1)) + a_maxbrake*δt)^2/(2*-a_maxbrake) ≤ ((speed(ω i) +
a_minbrake *δt)^2/(2*-a_minbrake)))
  using abs_le_square_iff[where x=((speed(ω (i+1)) + a_maxbrake*δt)/sqrt(-a_maxbrake))
and y = ((speed(ω i) + a_minbrake*δt)/sqrt(-a_minbrake))]
  using s1 apply (auto simp add:power_divide)
  using h2 s6 by linarith

have h4:(δt*speed(ω (i+1)) + a_maxbrake*δt^2/2 ≤ pos(ω' (i+1)) - pos(ω (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 (δt*speed(ω (i+1)) + a_maxbrake *δt^2/2 -?b_1 = -(speed(ω (i+1)) + a_maxbrake
*δt)^2/(2*-a_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 h4 have h5:(pos(ω' (i+1)) - pos(ω (i+1)) - ?b_1 ≥ -(speed(ω (i+1)) +
a_maxbrake*δt)^2/(2*-a_maxbrake))
  by linarith

have h7:(- ?b'_0 ≤ -(speed(ω i) + a_minbrake *δt)^2/(2*-a_minbrake))
  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 ≤ pos(ω' (i+1)) - pos(ω (i+1)) - ?b_1)

```

```

by linarith

have h9:(pos (ω' i) - pos (ω i) + ?b'_0 - ?b_1 + ε' ≤ pos (ω (i+1)) - pos (ω i))
  using dmin_rear_car_safe_motion dmin_rear_car_stopping h1 simp0 by force

with h8 have (ε' ≤ pos (ω' (i+1)) - pos (ω' i))
  by linarith
} note no_collision_if_fast_rear_car_not_breaking_not_stopping_and_front_car_may_not_stop=this

have nocollision:(ε' ≤ d_real ω' i)
  apply (cases ( ?b'_0 ≥ ?b'_1 ))
  using dmin apply simp
  apply(cases ( ?b_0 ≤ ?b_1 ))
    using as3 as1 d0 d3 simp0 no_collision_if_slow_rear_car_nonstopping
no_collision_if_slow_rear_car_stopping apply force
  apply(cases ( ?a_0 > a_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_min ω' i + ε' ≤ d_real ω' i)
  using dmin nocollision by (simp add:simp0 d3)
qed
next
fix ω
have (P_inv ω ⇒ (i < j ∧ j ≤ N) → pos (ω i) + ε' ≤ pos (ω 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 ω ⇒ no_collision ω {0..N} ε)
  unfolding no_collision_def apply auto
  by (smt (verit, ccfv_threshold) le_neq_implies_less not_le)
qed
qed
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/> (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*, *CSP\_RefTK.CSP\_ext*, *CSP\_RefTK.CSP\_ext*, *Isabelle\_DOF.technical\_report*,
3. Foreground (Sub-Contract SystemX-SVR): *Frontmatter*, *Introduction*, *CSP\_AV.CSP-AutoCars*, *CSP\_AV.MOSAR-Encodings*, *CSP\_AV.RSS\_2cars*, *CSP\_AV.RSS\_Ncars*, *CSP\_AV.RSS\_plus*, *Conclusion*.

### 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<sub>saf</sub>* is stronger as *motion<sub>rss</sub>* but offers more comfort and practicability")
- proofs on **safety** of three driving strategies
- by-products of proofs :

## 7 Conclusion

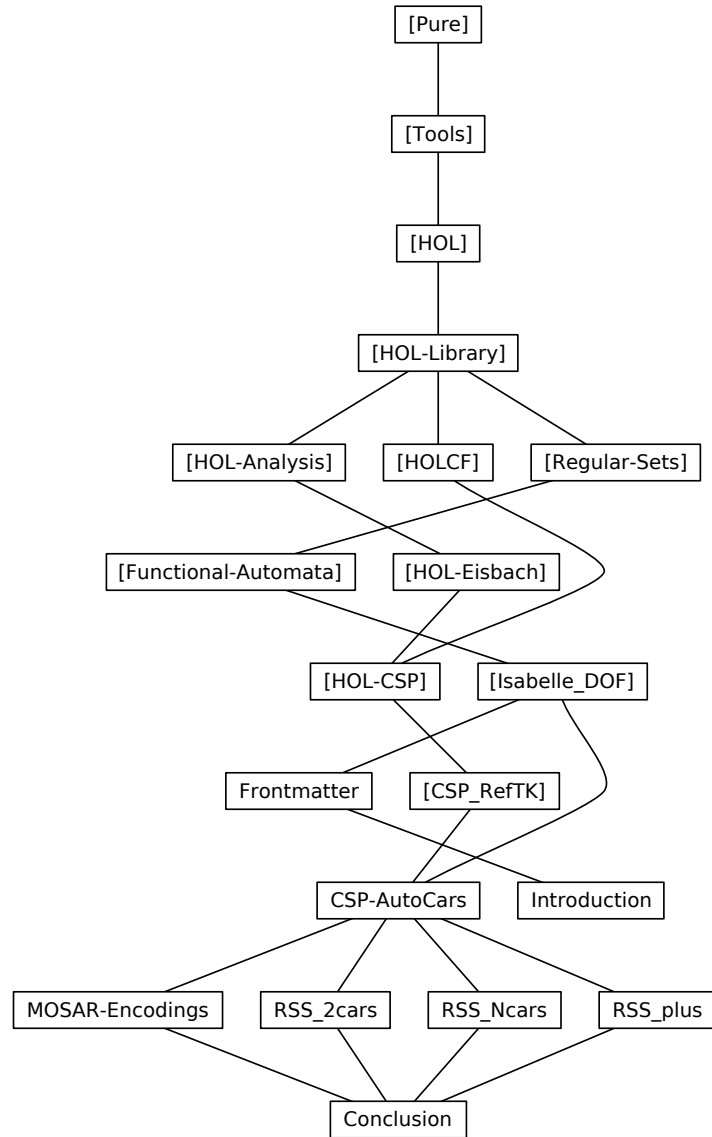


Figure 7.1: The Graph of Library-, Background-, and Foreground Theories of this Study

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 improvements.
- 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, ontological 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  $drive_{rss}$ , and then we will consider test cases for our new driving strategy  $drive_{safe}$ . 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 $drive_{rss}$

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

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



## 7 Conclusion

- 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.

No	Test Case description	Conds
1	r-car stops safe distance f-car stops	$speed(\omega' i) = 0$ $d_{r_{ss}} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
2	r-car stops safe distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) = 0$ $d_{r_{ss}} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
3	r-car doesn't stop (breaking or accelerating) safe distance f-car stops	$speed(\omega' i) > 0$ $d_{r_{ss}} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
4	r-car doesn't stop (breaking or accelerating) safe distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) > 0$ $d_{r_{ss}} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
5	r-car stops non-safe distance minimal distance f-car stops	$speed(\omega' i) = 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
6	r-car stops non-safe distance non-minimal distance f-car stops	$speed(\omega' i) = 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
7	r-car stops non-safe distance minimal distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) = 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
8	r-car stops non-safe distance non-minimal distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) = 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
9	r-car doesn't stop (breaking or accelerating) non-safe distance minimal distance f-car stops	$speed(\omega' i) > 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
10	r-car doesn't stop (breaking or accelerating) non-safe distance non-minimal distance f-car stops	$speed(\omega' i) > 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
11	r-car doesn't stop (breaking or accelerating) non-safe distance minimal distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) > 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
12	r-car doesn't stop (breaking or accelerating) non-safe distance non-minimal distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) > 0$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) > 0$

Table 7.1: Table with Abstract Test Cases for minimal distance of  $drive_{r_{ss}}$

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

### Abstract Test-Cases for RSS++ version $drive_{safe}$

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...)

**Minimal distance test cases:** (see Table 7.3.).

**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 *local critique* concerning modeling details of RSS and its safety proof, and a *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: 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: Figure 7.2 (right).
- Case insufficiently treated: inflexion point (singularity) for  $v_r = 0$ , e.g. 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 Figure 7.3 (right)).

## 7 Conclusion

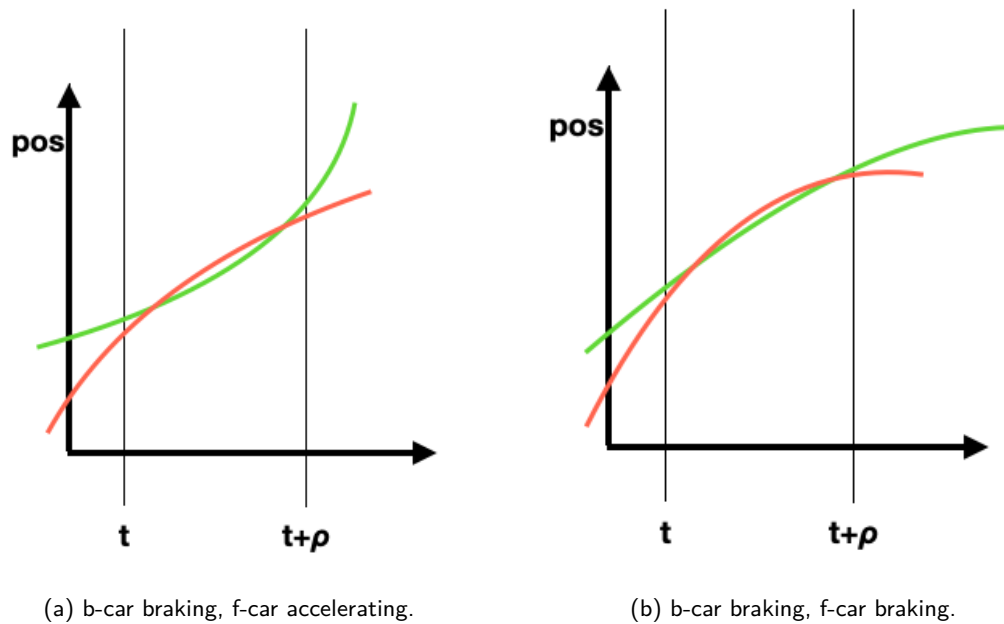


Figure 7.2: Collision scenarios overlooked.

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.

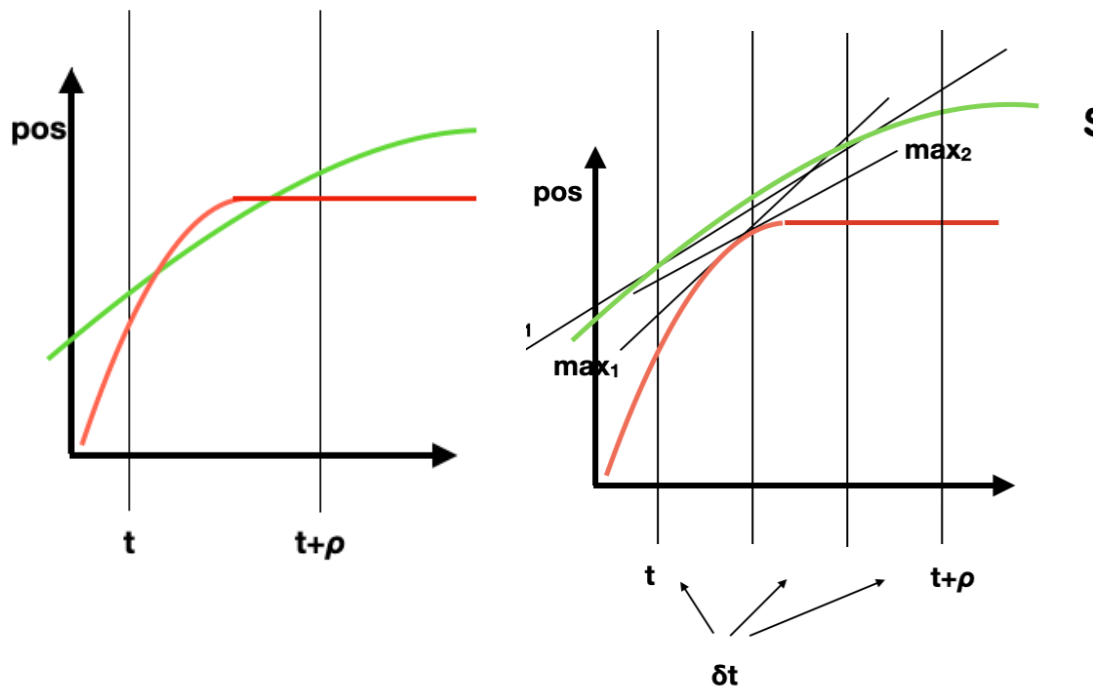
Concerning the global points of critique:

- the distances in a realistic configuration of RSS (speeds 133 km/h,  $\rho=0,1s$ ) 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



(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 liveness 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.

No	Test Case description	Conds
1	r-car stops f-car breaking faster f-car stops	$speed(\omega' i) = 0$ $\langle b_f < b_r \rangle$ $speed(\omega' (i+1)) = 0$
2	r-car doesn't stop (breaking or accelerating) f-car breaking faster f-car stops	$speed(\omega' i) > 0$ $\langle b_f < b_r \rangle$ $speed(\omega' (i+1)) = 0$
3	r-car stops f-car breaking faster non-safe distance minimal distance f-car doesn't stop (breaking or accelerating) f-car very slow (could stop)	$speed(\omega' i) = 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t \leq 0$
4	r-car doesn't stop (breaking or accelerating) f-car breaking faster non-safe distance minimal distance f-car doesn't stop (breaking or accelerating) f-car very slow (could stop)	$speed(\omega' i) > 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t \leq 0$
5	r-car stops f-car breaking faster non-safe distance non-minimal distance f-car doesn't stop (breaking or accelerating) f-car very slow (could stop)	$speed(\omega' i) = 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t \leq 0$
6	r-car doesn't stop (breaking or accelerating) f-car breaking faster non-safe distance non-minimal distance f-car doesn't stop (breaking or accelerating) f-car very slow (could stop)	$speed(\omega' i) > 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t \leq 0$
Is 7	r-car stops r-car breaking faster f-car stops	$speed(\omega' i) = 0$ $\langle b_f \geq b_r \rangle$ $speed(\omega' (i+1)) = 0$
8	r-car doesn't stop (breaking or accelerating) r-car breaking faster f-car stops	$speed(\omega' i) > 0$ $\langle b_f \geq b_r \rangle$ $speed(\omega' (i+1)) = 0$
9	r-car stops r-car breaking faster f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) = 0$ $\langle b_f \geq b_r \rangle$ $speed(\omega' (i+1)) > 0$
10	r-car doesn't stop (breaking or accelerating) r-car breaking faster f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) > 0$ $\langle b_f \geq b_r \rangle$ $speed(\omega' (i+1)) > 0$
11	r-car stops f-car breaking faster non-safe distance minimal distance f-car doesn't stop (breaking or accelerating) f-car not very slow (could not stop)	$speed(\omega' i) = 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t > 0$
12	r-car doesn't stop (breaking or accelerating) f-car breaking faster non-safe distance minimal distance f-car doesn't stop (breaking or accelerating) f-car not very slow (could not stop)	$speed(\omega' i) > 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t > 0$
13	r-car stops f-car breaking faster non-safe distance non-minimal distance f-car doesn't stop (breaking or accelerating) f-car not very slow (could not stop)	$speed(\omega' i) = 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t > 0$
14	r-car doesn't stop (breaking or accelerating) f-car breaking faster non-safe distance non-minimal distance f-car doesn't stop (breaking or accelerating) f-car not very slow (could not stop)	$speed(\omega' i) > 0$ $\langle b_f < b_r \rangle$ $d_{r_{ss}} \omega i > d_{real} \omega i$ $d_{min} \omega i > d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t > 0$

Table 7.2: Table with Abstract Test Cases for no-collision of  $drive_{r_{ss}}$

## 7 Conclusion

No	Test Case description	Conds
1	r-car stops safe distance f-car stops	$speed(\omega' i) = 0$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
2	r-car stops safe distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) = 0$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
3	r-car doesn't stop (breaking or accelerating) safe distance f-car stops	$speed(\omega' i) > 0$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
4	r-car doesn't stop (breaking or accelerating) safe distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) > 0$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
5	r-car stops r-car was breaking safe distance f-car stops	$speed(\omega' i) = 0$ $acc(\omega' i) \leq a_{minbrake}$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
6	r-car stops r-car was breaking safe distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) = 0$ $acc(\omega' i) \leq a_{minbrake}$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
7	r-car stops r-car was not breaking (slowing) safe distance f-car stops	$speed(\omega' i) = 0$ $acc(\omega' i) > a_{minbrake}$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) = 0$
8	r-car stops r-car was not breaking but slowing safe distance f-car doesn't stop (breaking or accelerating)	$speed(\omega' i) = 0$ $acc(\omega' i) > a_{minbrake}$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$
9	r-car doesn't stop r-car was breaking safe distance	$speed(\omega' i) > 0$ $acc(\omega' i) \leq a_{minbrake}$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$
10	r-car doesn't stop (breaking or accelerating) r-car accelerates safe distance	$speed(\omega' i) > 0$ $acc(\omega' i) \geq 0$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$
11	r-car doesn't stop r-car was not breaking but slowing r-car would stop after response time safe distance	$speed(\omega' i) > 0$ $a_{minbrake} < acc(\omega' i) \leq 0$ $speed(\omega i) + ?a_0 * \rho \leq 0$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$
12	r-car doesn't stop r-car was not breaking but slowing r-car would not stop after response time safe distance	$speed(\omega' i) > 0$ $a_{minbrake} < acc(\omega' i) \leq 0$ $speed(\omega i) + ?a_0 * \rho > 0$ $d_{safe} \omega i acc(\omega' i) + \varepsilon' \leq d_{real} \omega i$

Table 7.3: Table with Abstract Test Cases for minimal distance of  $drive_{safe}$  (RSS++)

No	Test Case description	Conds
1	r-car stops f-car breaking faster f-car stops	$speed(\omega' i) = 0$ $\langle b_f < b_r \rangle$ $speed(\omega' (i+1)) = 0$
2	r-car doesn't stop (breaking or accelerating) f-car breaking faster f-car stops	$speed(\omega' i) > 0$ $\langle b_f < b_r \rangle$ $speed(\omega' (i+1)) = 0$
3	f-car breaking faster safe distance f-car doesn't stop (breaking or accelerating) f-car very slow (could stop)	$\langle b_f < b_r \rangle$ $d_{safe} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) + a_{maxbrake} * \delta t \leq 0$
4	f-car breaking faster non-safe distance minimal distance f-car doesn't stop (breaking or accelerating) f-car very slow (could stop)	$\langle b_f < b_r \rangle$ $d_{safe} \omega i > d_{real} \omega i$ $d_{min} \omega i \leq d_{real} \omega i$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t \leq 0$
5	r-car stops f-car breaking faster f-car doesn't stop f-car not very slow (could not stop)	$speed(\omega' i) = 0$ $\langle b_f < b_r \rangle$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t > 0$
6	r-car doesn't stop r-car was breaking f-car breaking faster f-car doesn't stop f-car not very slow (could not stop)	$speed(\omega' i) > 0$ $acc(\omega' i) \leq a_{minbrake}$ $\langle b_f < b_r \rangle$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t > 0$
7	r-car doesn't stop r-car was not breaking (slowing or accelerating) f-car breaking faster f-car doesn't stop f-car not very slow (could not stop)	$speed(\omega' i) > 0$ $acc(\omega' i) > a_{minbrake}$ $\langle b_f < b_r \rangle$ $speed(\omega' (i+1)) > 0$ $speed(\omega (i+1)) - a_{maxbrake} * \delta t > 0$

Table 7.4: Table with Abstract Test Cases for no-collision of  $drive_{safe}$  (RSS++)





## Bibliography

- [1] T. C. I. 22 and S. S. 32. Road vehicles — safety of the intended functionality. techreport iso/pas 21448:2019, International Organization for Standardization, 2019.
- [2] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating sequential processes. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 281–305, Berlin, Heidelberg, 1985. Springer. ISBN 978-3-540-39593-5.
- [3] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.
- [4] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985. ISBN 0-13-153271-5.
- [5] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Prism-games 3.0: Stochastic game verification with concurrency, equilibria and time. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 475–487. Springer, 2020. doi: 10.1007/978-3-030-53291-8\_25. URL [https://doi.org/10.1007/978-3-030-53291-8\\_25](https://doi.org/10.1007/978-3-030-53291-8_25).
- [6] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. A calculus for hybrid CSP. In K. Ueda, editor, *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings*, volume 6461 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2010. doi: 10.1007/978-3-642-17164-2\_1. URL [https://doi.org/10.1007/978-3-642-17164-2\\_1](https://doi.org/10.1007/978-3-642-17164-2_1).
- [7] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. doi: 10.1007/3-540-45949-9.
- [8] A. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1997. ISBN 0-13-674409-5.
- [9] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a Formal Model of Safe and Scalable Self-driving Cars. *arXiv e-prints*, art. arXiv:1708.06374, Aug. 2017.
- [10] S. Taha, B. Wolff, and L. Ye. Philosophers may dine - definitively! In B. Dongol and E. Troubitsyna, editors, *Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings*, volume 12546 of *Lecture Notes in Computer Science*, pages 419–439. Springer, 2020. doi: 10.1007/978-3-030-63461-2\_23. URL [https://doi.org/10.1007/978-3-030-63461-2\\_23](https://doi.org/10.1007/978-3-030-63461-2_23).