A Formally Verified Checker of the Safe Distance Traffic Rules for Autonomous Vehicles

Albert Rizaldi, Fabian Immler

March 17, 2025

Abstract

The Vienna Convention on Road Traffic defines the safe distance traffic rules informally. This could make autonomous vehicle liable for safe-distance-related accidents because there is no clear definition of how large a safe distance is. We provide a formally proven prescriptive definition of a safe distance, and checkers which can decide whether an autonomous vehicle is obeying the safe distance rule. Not only does our work apply to the domain of law, but it also serves as a specification for autonomous vehicle manufacturers and for online verification of path planners. This formalization accompanies our paper "A Formally Verified Checker of the Safe Distance Traffic Rules for Autonomous Vehicles". [1]

Contents

1 Safe Distance

theory *Safe-Distance*
imports
HOL−Analysis.Multivariate-Analysis
HOL−Decision-Procs.Approximation
Sturm-Sequences.Sturm
begin

This theory is about formalising the safe distance rule. The safe distance rule is obtained from Vienna Convention which basically states the following thing.

"The car at all times must maintain a safe distance towards the vehicle in front of it, such that whenever the vehicle in front and the ego vehicle apply maximum deceleration, there will not be a collision."

To formalise this safe distance rule we have to define first what is a safe distance. To define this safe distance, we have to model the physics of the movement of the vehicle. The following model is sufficient.

$s = s_0 + v_0 * t + 1 / 2 * a_0 * t^2$

Assumptions in this model are :

- Both vehicles are assumed to be point mass. The exact location of the ego vehicle is the front-most occupancy of the ego vehicle. Similarly for the other vehicle, its exact location is the rearmost occupancy of the other vehicle.
- Both cars can never drive backward.

lemmas [*simp del*] = *div-mult-self1 div-mult-self2 div-mult-self3 div-mult-self4*

1.1 Quadratic Equations

lemma *discriminant*: $a * x^2 + b * x + c = (0::real) \Longrightarrow 0 \leq b^2 - 4 * a * c$
by (*sos* (((*A*<*0* ∗ *R*<*1*) + (*R*<*1* ∗ (*R*<*1* ∗ [*2*∗*a*∗*x* + *b*]^*2*)))))

lemma *quadratic-eq-factoring*:

assumes $D : D = b^2 - 4 * a * c$
assumes *nn*: $0 \leq D$
assumes *x1*: $x_1 = (-b + sqrt\ D)\ /\ (2 * a)$
assumes *x2*: $x_2 = (-b - sqrt\ D)\ /\ (2 * a)$
assumes $a : a \neq 0$
shows $a * x^2 + b * x + c = a * (x - x_1) * (x - x_2)$
using *nn*
by (*simp add*: *D x1 x2*)
(*simp add*: *assms power2-eq-square power3-eq-cube field-split-simps*)

lemma *quadratic-eq-zeroes-iff*:
assumes $D : D = b^2 - 4 * a * c$
assumes *x1*: $x_1 = (-b + sqrt\ D)\ /\ (2 * a)$
assumes *x2*: $x_2 = (-b - sqrt\ D)\ /\ (2 * a)$
assumes $a : a \neq 0$
shows $a * x^2 + b * x + c = 0 \longleftrightarrow (D \geq 0 \wedge (x = x_1 \vee x = x_2))$ (**is** *?z* $\longleftrightarrow$ -)
using *quadratic-eq-factoring*[*OF D - x1 x2 a*, *of x*] *discriminant*[*of a x b c*] *a*
by (*auto simp*: *D*)

1.2 Convexity Condition

lemma *p-convex*:
fixes *a b c x y z* :: *real*
assumes *p-def*: $p = (\lambda x.\ a * x^2 + b * x + c)$
assumes *less* : $x < y\ y < z$
and *ge* : $p\ x > p\ y\ p\ y \leq p\ z$
shows $a > 0$
using *less ge* **unfolding** *p-def*
by (*sos* ((((*A*<*0* * (*A*<*1* * *A*<*2*)) * *R*<*1*) + (((*A*<*2* * *R*<*1*) * (*R*<*1/4* * [*y* + ~*1***z*]^*2*)) +
(((*A*<=*1* * *R*<*1*) * (*R*<*1* * [*x* + ~*1***y*]^*2*)) + (((*A*<=*1* * (*A*<*0* * (*A*<*1* * *R*<*1*))) * (*R*<*1/4* * [*1*]^*2*)) +
(((*A*<=*0* * *R*<*1*) * (*R*<*1/4* * [~*1***y*^*2* + *x***y* + ~*1***x***z* + *y***z*]^*2*)) +
((*A*<=*0* * (*A*<*0* * (*A*<*1* * *R*<*1*))) * (*R*<*1* * [*x* + ~*1/2***y* + ~*1/2***z*]^*2*)))))))))

definition *root-in* :: *real* ⇒ *real* ⇒ (*real* ⇒ *real*) ⇒ *bool* **where**
root-in m M f = (∃ *x*∈{*m* .. *M*}. *f x* = *0*)

definition *quadroot-in* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
quadroot-in m M a b c = *root-in m M* (λ*x*. *a* * *x*^*2* + *b* * *x* + *c*)

lemma *card-iff-exists*: *0* < *card X* ⟷ *finite X* ∧ (∃ *x*. *x* ∈ *X*)
by (*auto simp*: *card-gt-0-iff*)

lemma *quadroot-in-sturm*[*code*]:
quadroot-in m M a b c ⟷ (*a* = *0* ∧ *b* = *0* ∧ *c* = *0* ∧ *m* ≤ *M*) ∨
(*m* ≤ *M* ∧ *poly* [:*c*, *b*, *a*:] *m* = *0*) ∨
count-roots-between [:*c*, *b*, *a*:] *m M* > *0*
apply (*cases a* = *0* ∧ *b* = *0* ∧ *c* = *0* ∧ *m* ≤ *M*)

```
   apply (force simp: quadroot-in-def root-in-def)
  apply (cases m ≤ M ∧ poly [:c, b, a:] m = 0)
   apply (force simp: quadroot-in-def root-in-def algebra-simps power2-eq-square
count-roots-between-correct card-iff-exists)
proof −
  assume H: ¬ (a = 0 ∧ b = 0 ∧ c = 0 ∧ m ≤ M) ¬ (m ≤ M ∧ poly [:c, b, a:]
m = 0)
  hence poly [:c, b, a:] m ≠ 0 ∨ m > M
    by auto
  then have quadroot-in m M a b c ⟷ 0 < count-roots-between [:c, b, a:] m M
  proof (rule disjE)
    assume pnz: poly [:c, b, a:] m ≠ 0
    then have nz: [:c, b, a:] ≠ 0 by auto
    show ?thesis
      unfolding count-roots-between-correct card-iff-exists
      apply safe
        apply (rule finite-subset[where B={x. poly [:c, b, a:] x = 0}])
         apply force
        apply (rule poly-roots-finite)
        apply (rule nz)
      using pnz
      apply (auto simp add: count-roots-between-correct quadroot-in-def root-in-def
card-iff-exists
          algebra-simps power2-eq-square)
      apply (case-tac x = m)
       apply (force simp: algebra-simps)
      apply force
      done
 qed (auto simp: quadroot-in-def count-roots-between-correct root-in-def card-eq-0-iff)
  then show quadroot-in m M a b c = (a = 0 ∧ b = 0 ∧ c = 0 ∧ m ≤ M ∨
     m ≤ M ∧ poly [:c, b, a:] m = 0 ∨
     0 < count-roots-between [:c, b, a:] m M)
    using H by metis
qed

lemma check-quadroot-linear:
  fixes a b c :: real
  assumes a = 0
  shows ¬ quadroot-in m M a b c ⟷
    ((b = 0 ∧ c = 0 ∧ M < m) ∨ (b = 0 ∧ c ≠ 0) ∨
     (b ≠ 0 ∧ (let x = − c / b in m > x ∨ x > M)))
proof −
  have quadroot-in m M a b c ⟷ (b = 0 ⟶ quadroot-in m M a b c) ∧ (b ≠ 0
⟶ quadroot-in m M a b c)
    by auto
  also have (b = 0 ⟶ quadroot-in m M a b c) ⟷
    ((b = 0 ⟶ c = 0 ⟶ m ≤ M) ∧ (b ≠ 0 ∨ c = 0))
    by (auto simp: quadroot-in-def Let-def root-in-def assms field-split-simps
      intro!: bexI[where x=−c / b])
```

also have $(b \neq 0 \longrightarrow$ *quadroot-in m M a b c*$) \longleftrightarrow (b = 0 \vee ($*let* $x = -c$ / b *in* $m \leq x \wedge x \leq M))$
apply (*auto simp*: *quadroot-in-def Let-def root-in-def assms field-split-simps intro*!: *bexI*[**where** $x{=}-c$ / b])
by (*metis mult.commute mult-le-cancel-left-neg add-eq-0-iff mult-le-cancel-left-pos*)+
finally show *?thesis*
by (*simp add*: *Let-def not-less not-le*)
qed

lemma *check-quadroot-nonlinear*:
assumes $a \neq 0$
shows *quadroot-in m M a b c* =
(*let* $D = b\hat{}2 - 4 * a * c$ *in* $D \geq 0 \wedge$
$((\textit{let } x = (-b + \textit{sqrt } D)/(2{*}a) \textit{ in } m \leq x \wedge x \leq M) \vee$
(*let* $x = (-b - \textit{sqrt } D)/(2{*}a)$ *in* $m \leq x \wedge x \leq M)))$
by (*auto simp*: *quadroot-in-def Let-def root-in-def quadratic-eq-zeroes-iff*[*OF refl refl refl assms*])

lemma *ncheck-quadroot*:
shows $\neg$*quadroot-in m M a b c* $\longleftrightarrow$
$(a = 0 \longrightarrow \neg$*quadroot-in m M a b c*$) \wedge$
$(a = 0 \vee \neg$*quadroot-in m M a b c*$)$
by *auto*

1.3 Movement

locale *movement* =
fixes *a v s0* :: *real*
begin

Function to compute the distance using the equation
$s(t) = s_0 + v_0 * t + 1 / 2 * a_0 * t^2$
Input parameters:

- s_0: initial distance
- v_0: initial velocity (positive means forward direction and the converse is true)
- a: acceleration (positive for increasing and negative for decreasing)
- t: time

For the time $t < 0$, we assume the output of the function is s_0. Otherwise, the output is calculated according to the equation above.

1.3.1 Continuous Dynamics

definition $p :: real \Rightarrow real$ **where**
$p\ t = s0 + v * t + 1/2 * a * t^2$

lemma *p-all-zeroes*:
assumes *D*: $D = v^2 - 2 * a * s0$
shows $p\ t = 0 \longleftrightarrow ((a \neq 0 \wedge 0 \leq D \wedge ((t = (-\ v + sqrt\ D)\ /\ a) \vee t = (-\ v - sqrt\ D)\ /\ a)) \vee$
$(a = 0 \wedge v = 0 \wedge s0 = 0) \vee (a = 0 \wedge v \neq 0 \wedge t = (-\ s0\ /\ v)))$
using *quadratic-eq-zeroes-iff*[*OF refl refl refl*, *of a / 2 t v s0*]
by (*auto simp*: *movement.p-def D power2-eq-square field-split-simps*)

lemma *p-zero*[*simp*]: $p\ 0 = s0$
by (*simp add*: *p-def*)

lemma *p-continuous*[*continuous-intros*]: *continuous-on T p*
unfolding *p-def* **by** (*auto intro*!: *continuous-intros*)

lemma *isCont-p*[*continuous-intros*]: *isCont p x*
using *p-continuous*[*of UNIV*]
by (*auto simp*: *continuous-on-eq-continuous-at*)

definition $p' :: real \Rightarrow real$ **where**
$p'\ t = v + a * t$

lemma *p'-zero*: $p'\ 0 = v$
by (*simp add*: *p'-def*)

lemma *p-has-vector-derivative*[*derivative-intros*]: (*p has-vector-derivative p' t*) (*at t within s*)
by (*auto simp*: *p-def*[*abs-def*] *p'-def has-vector-derivative-def algebra-simps intro*!: *derivative-eq-intros*)

lemma *p-has-real-derivative*[*derivative-intros*]: (*p has-real-derivative p' t*) (*at t within s*)
using *p-has-vector-derivative*
by (*simp add*: *has-real-derivative-iff-has-vector-derivative*)

definition $p'' :: real \Rightarrow real$ **where**
$p''\ t = a$

lemma *p'-has-vector-derivative*[*derivative-intros*]: (*p' has-vector-derivative p'' t*) (*at t within s*)
by (*auto simp*: *p'-def*[*abs-def*] *p''-def has-vector-derivative-def algebra-simps intro*!: *derivative-eq-intros*)

lemma *p'-has-real-derivative*[*derivative-intros*]: (*p' has-real-derivative p'' t*) (*at t within s*)
using *p'-has-vector-derivative*

by (*simp add*: *has-real-derivative-iff-has-vector-derivative*)

definition *t-stop* :: *real* **where**
t-stop = − *v* / *a*

lemma *p′-stop-zero*: *p′ t-stop* = (*if a* = *0 then v else 0*) **by** (*auto simp*: *p′-def t-stop-def*)

lemma *p′-pos-iff*: *p′ x* > *0* ⟷ (*if a* > *0 then x* > −*v* / *a else if a* < *0 then x* < −*v* / *a else v* > *0*)
by (*auto simp*: *p′-def field-split-simps*)

lemma *le-t-stop-iff*: *a* ≠ *0* ⟹ *x* ≤ *t-stop* ⟷ (*if a* < *0 then p′ x* ≥ *0 else p′ x* ≤ *0*)
by (*auto simp*: *p′-def field-split-simps t-stop-def*)

lemma *p′-continuous*[*continuous-intros*]: *continuous-on T p′*
unfolding *p′-def* **by** (*auto intro*: *continuous-intros*)

lemma *isCont-p′*[*continuous-intros*]: *isCont p′ x*
using *p′-continuous*[*of UNIV*] **by** (*auto simp*: *continuous-on-eq-continuous-at*)

definition *p-max* :: *real* **where**
p-max = *p t-stop*

lemmas *p-t-stop* = *p-max-def*[*symmetric*]

lemma *p-max-eq*: *p-max* = *s0* − *v*² / *a* / *2*
by (*auto simp*: *p-max-def p-def t-stop-def field-split-simps power2-eq-square*)

1.3.2 Hybrid Dynamics

definition *s* :: *real* ⇒ *real* **where**
s t = (*if t* ≤ *0 then s0*
else if t ≤ *t-stop then p t*
else p-max)

definition *q* :: *real* ⇒ *real* **where**
q t = *s0* + *v* ∗ *t*

definition *q′* :: *real* ⇒ *real* **where**
q′ t = *v*

lemma *init-q*: *q 0* = *s0* **unfolding** *q-def* **by** *auto*

lemma *q-continuous*[*continuous-intros*]: *continuous-on T q*
unfolding *q-def* **by** (*auto intro*: *continuous-intros*)

lemma *isCont-q*[*continuous-intros*]: *isCont q x*

using *q-continuous*[*of UNIV*]
by (*auto simp*:*continuous-on-eq-continuous-at*)

lemma *q-has-vector-derivative*[*derivative-intros*]: (*q has-vector-derivative q' t*) (*at t within u*)
by (*auto simp*: *q-def*[*abs-def*] *q'-def has-vector-derivative-def algebra-simps intro*!: *derivative-eq-intros*)

lemma *q-has-real-derivative*[*derivative-intros*]: (*q has-real-derivative q' t*) (*at t within u*)
using *q-has-vector-derivative*
by (*simp add*:*has-real-derivative-iff-has-vector-derivative*)

lemma *s-cond-def*:
$t \leq 0 \Longrightarrow s\ t = s0$
$0 \leq t \Longrightarrow t \leq t\text{-}stop \Longrightarrow s\ t = p\ t$
by (*simp-all add*: *s-def*)

end

locale *braking-movement* = *movement* +
assumes *decel*: $a < 0$
assumes *nonneg-vel*: $v \geq 0$
begin

lemma *t-stop-nonneg*: $0 \leq t\text{-}stop$
using *decel nonneg-vel*
by (*auto simp*: *t-stop-def divide-simps*)

lemma *t-stop-pos*:
assumes $v \neq 0$
shows $0 < t\text{-}stop$
using *decel nonneg-vel assms*
by (*auto simp*: *t-stop-def divide-simps*)

lemma *t-stop-zero*:
assumes $t\text{-}stop = 0$
shows $v = 0$
using *assms decel*
by (*auto simp*: *t-stop-def*)

lemma *t-stop-zero-not-moving*: $t\text{-}stop = 0 \Longrightarrow q\ t = s0$
unfolding *q-def* **using** *t-stop-zero* **by** *auto*

abbreviation $s\text{-}stop \equiv s\ t\text{-}stop$

lemma *s-t-stop*: $s\text{-}stop = p\text{-}max$
using *t-stop-nonneg*
by (*auto simp*: *s-def t-stop-def p-max-def p-def*)

lemma *s0-le-s-stop*: $s0 \leq s\text{-}stop$
proof (*rule subst*[**where** t=*s-stop* **and** s=*p-max*])
  show *p-max* = *s-stop* **by** (*rule sym*[*OF s-t-stop*])
next
  show $s0 \leq p\text{-}max$
  proof (*rule subst*[**where** t=*p-max* **and** s=$s0 - v^2 / a / 2$])
    show $s0 - v^2 / a / 2 = p\text{-}max$ **using** *p-max-eq* **by** *auto*
  next
    have $0 \leq - v^2 / a / 2$ **using** *decel zero-le-square*[*of v*]
    proof −
      have *f1*: $a \leq 0$
        using ‹$a < 0$› **by** *linarith*
      have $(- 1 * v^2 \leq 0) = (0 \leq v^2)$
        by *auto*
      then have $0 \leq - 1 * v^2 / a$
        using *f1* **by** (*meson zero-le-divide-iff zero-le-power2*)
      then show *?thesis*
        by *force*
    qed
    thus $s0 \leq s0 - v^2 / a / 2$ **by** *auto*
  qed
qed

lemma *p-mono*: $x \leq y \Longrightarrow y \leq t\text{-}stop \Longrightarrow p\ x \leq p\ y$
  using *decel*
proof −
  assume $x \leq y$ **and** $y \leq t\text{-}stop$ **and** $a < 0$
  hence $x + y \leq - 2 * v / a$
    unfolding *t-stop-def* **by** *auto*
  hence $-1 / 2 * a * (x + y) \leq v$ (**is** *?lhs0* ≤ *?rhs0*)
    using ‹$a < 0$› **by** (*auto simp add*: *field-simps*)
  hence *?lhs0* $* (x - y) \geq$ *?rhs0* $* (x - y)$
    using ‹$x \leq y$› **by** *sos*
  hence $v * x + 1 / 2 * a * x^2 \leq v * y + 1 / 2 * a * y^2$
    by (*auto simp add*: *field-simps power-def*)
  thus $p\ x \leq p\ y$
    unfolding *p-max-def p-def t-stop-def* **by** *auto*
qed

lemma *p-antimono*: $x \leq y \Longrightarrow t\text{-}stop \leq x \Longrightarrow p\ y \leq p\ x$
  using *decel*
proof −
  assume $x \leq y$ **and** *t-stop* ≤ x **and** $a < 0$
  hence $- 2 * v / a \leq x + y$
    unfolding *t-stop-def* **by** *auto*
  hence $v \leq - 1/ 2 * a * (x + y)$
    using ‹$a < 0$› **by** (*auto simp add*: *field-simps*)
  hence $v * (x - y) \geq -1/2 * a * (x^2 - y^2)$

using ‹$x \leq y$› **by** *sos*
hence $v * y + 1/2 * a * y^2 \leq v * x + 1/2 * a * x^2$
by (*auto simp add*: *field-simps*)
thus $p\ y \leq p\ x$
unfolding *p-max-def p-def t-stop-def* **by** *auto*
qed

lemma *p-strict-mono*: $x < y \Longrightarrow y \leq$ *t-stop* $\Longrightarrow p\ x < p\ y$
using *decel*
proof −
assume $x < y$ **and** $y \leq$ *t-stop* **and** $a < 0$
hence $x + y < -\ 2 * v\ /\ a$
unfolding *t-stop-def* **by** *auto*
hence $-1\ /\ 2 * a * (x + y) < v$ (**is** *?lhs0* < *?rhs0*)
using ‹$a < 0$› **by** (*auto simp add*: *field-simps*)
hence *?lhs0* $* (x- y) >$ *?rhs0* $* (x - y)$
using ‹$x < y$› **by** *sos*
hence $v * x + 1\ /\ 2 * a * x^2 < v * y + 1\ /\ 2 * a * y^2$
by (*auto simp add*: *field-simps power-def*)
thus $p\ x < p\ y$
unfolding *p-max-def p-def t-stop-def* **by** *auto*
qed

lemma *p-strict-antimono*: $x < y \Longrightarrow$ *t-stop* $\leq x \Longrightarrow p\ y < p\ x$
using *decel*
proof −
assume $x < y$ **and** *t-stop* $\leq x$ **and** $a < 0$
hence $-\ 2 * v\ /\ a < x + y$
unfolding *t-stop-def* **by** *auto*
hence $v < -\ 1/\ 2 * a * (x + y)$
using ‹$a < 0$› **by** (*auto simp add*: *field-simps*)
hence $v * (x - y) > -1/2 * a * (x^2 - y^2)$
using ‹$x < y$› **by** *sos*
hence $v * y + 1/2 * a * y^2 < v * x + 1/2 * a * x^2$
by (*auto simp add*: *field-simps*)
thus $p\ y < p\ x$
unfolding *p-max-def p-def t-stop-def* **by** *auto*
qed

lemma *p-max*: $p\ x \leq$ *p-max*
unfolding *p-max-def*
by (*cases* $x \leq$ *t-stop*) (*auto intro*: *p-mono p-antimono*)

lemma *continuous-on-s*[*continuous-intros*]: *continuous-on T s*
unfolding *s-def*[*abs-def*]
using *t-stop-nonneg*
by (*intro continuous-on-subset*[**where** *t*=*T* **and** $s = \{..\ 0\} \cup (\{0\ ..\ t\text{-}stop\} \cup \{t\text{-}stop\ ..\})$] *continuous-on-If*)
(*auto simp*: *p-continuous p-max-def antisym-conv*[**where** *x*=*0*])

lemma *isCont-s*[*continuous-intros*]: *isCont s x*
using *continuous-on-s*[*of UNIV*]
by (*auto simp*: *continuous-on-eq-continuous-at*)

definition s' :: *real* ⇒ *real* **where**
$s'\ t$ = (*if* $t \leq$ *t-stop then* $p'\ t$ *else 0*)

lemma *s-has-real-derivative*:
assumes $t \geq 0\ v\ /\ a \leq 0\ a \neq 0$
shows (*s has-real-derivative* $s'\ t$) (*at t within* {*0*..})
proof −
from *assms* **have** ∗: $t \leq$ *t-stop* ⟷ $t \in \{0\ ..$ *t-stop*$\}$ **by** *simp*
from *assms* **have** $0 \leq$ *t-stop* **by** (*auto simp*: *t-stop-def*)
have ((λ*t*. *if* $t \in \{0\ ..$ *t-stop*$\}$ *then p t else p-max*) *has-real-derivative*
(*if* $t \in \{0..$*t-stop*$\}$ *then* $p'\ t$ *else 0*)) (*at t within* {*0*..})
unfolding *s-def*[*abs-def*] s'*-def*
has-real-derivative-iff-has-vector-derivative
apply (*rule has-vector-derivative-If-within-closures*[**where** $T = \{$*t-stop* ..}])
using ‹*0* ≤ *t-stop*› ‹$a \neq 0$›
by (*auto simp*: *assms p*′*-stop-zero p-t-stop max-def insert-absorb*
intro!: *p-has-vector-derivative*)
from - - *this* **show** *?thesis*
unfolding *has-vector-derivative-def has-real-derivative-iff-has-vector-derivative*
s'*-def s-def*[*abs-def*] ∗
by (*rule has-derivative-transform*)
(*auto simp*: *assms s-def p-max-def t-stop-def*)
qed

lemma *s-has-vector-derivative*[*derivative-intros*]:
assumes $t \geq 0\ v\ /\ a \leq 0\ a \neq 0$
shows (*s has-vector-derivative* $s'\ t$) (*at t within* {*0*..})
using *s-has-real-derivative*[*OF assms*]
by (*simp add*: *has-real-derivative-iff-has-vector-derivative*)

lemma *s-has-field-derivative*[*derivative-intros*]:
assumes $t \geq 0\ v\ /\ a \leq 0\ a \neq 0$
shows (*s has-field-derivative* $s'\ t$) (*at t within* {*0*..})
using *s-has-vector-derivative*[*OF assms*]
by(*simp add*: *has-real-derivative-iff-has-vector-derivative*)

lemma *s-has-real-derivative-at*:
assumes $0 < x\ 0 \leq v\ a < 0$
shows (*s has-real-derivative* $s'\ x$) (*at x*)
proof −
from *assms* **have** (*s has-real-derivative* $s'\ x$) (*at x within* {*0* ..})
by (*intro s-has-real-derivative*) (*auto intro*!: *divide-nonneg-nonpos*)
then have (*s has-real-derivative* $s'\ x$) (*at x within* {*0*<..})
by (*rule DERIV-subset*) *auto*

then show (*s has-real-derivative s′ x*) (*at x*) **using** *assms*
by (*subst* (*asm*) *at-within-open*) *auto*
qed

lemma *s-delayed-has-field-derivative*[*derivative-intros*]:
assumes $\delta < t\ 0 \leq v\ a < 0$
shows ((λ*x*. *s* (*x* − δ)) *has-field-derivative s′* (*t* − δ)) (*at t within* {δ<..})
proof −
from *assms* **have** ((λ*x*. *s* (*x* + − δ)) *has-real-derivative s′* (*t* − δ)) (*at t*)
using *DERIV-shift*[*of s* (*s′* (*t* − δ)) *t* −δ] *s-has-real-derivative-at*
by *auto*

thus ((λ*x*. *s* (*x* − δ)) *has-field-derivative s′* (*t* − δ)) (*at t within* {δ<..})
using *has-field-derivative-at-within* **by** *auto*
qed

lemma *s-delayed-has-vector-derivative*[*derivative-intros*]:
assumes $\delta < t\ 0 \leq v\ a < 0$
shows ((λ*x*. *s* (*x* − δ)) *has-vector-derivative s′* (*t* − δ)) (*at t within* {δ<..})
using *s-delayed-has-field-derivative*[*OF assms*]
by(*simp add*:*has-real-derivative-iff-has-vector-derivative*)

lemma *s′-nonneg*: $0 \leq v \Longrightarrow a \leq 0 \Longrightarrow 0 \leq s'\ x$
by (*auto simp*: *s′-def p′-def t-stop-def field-split-simps*)

lemma *s′-pos*: $0 \leq x \Longrightarrow x < t\text{-}stop \Longrightarrow 0 \leq v \Longrightarrow a \leq 0 \Longrightarrow 0 < s'\ x$
by (*intro le-neq-trans s′-nonneg*)
(*auto simp*: *s′-def p′-def t-stop-def field-split-simps*)

1.3.3 Monotonicity of Movement

lemma *s-mono*:
assumes $t \geq u\ u \geq 0$
shows $s\ t \geq s\ u$
using *p-mono*[*of u t*] *assms p-max*[*of u*] **by** (*auto simp*: *s-def*)

lemma *s-strict-mono*:
assumes $u < t\ t \leq t\text{-}stop\ u \geq 0$
shows $s\ u < s\ t$
using *p-strict-mono*[*of u t*] *assms p-max*[*of u*] **by** (*auto simp*: *s-def*)

lemma *s-antimono*:
assumes $x \leq y$
assumes $t\text{-}stop \leq x$
shows $s\ y \leq s\ x$
proof −
from *assms* **have** $t\text{-}stop \leq y$ **by** *auto*
hence $s\ y \leq p\text{-}max$ **unfolding** *s-def p-max-eq*
using *p-max-def p-max-eq s0-le-s-stop s-t-stop* **by** *auto*

also have ... $\leq s\ x$
using ‹*t-stop* $\leq x$› *s-mono s-t-stop t-stop-nonneg* **by** *fastforce*
ultimately show $s\ y \leq s\ x$ **by** *auto*
qed

lemma *init-s* : $t \leq 0 \Longrightarrow s\ t = s0$
using *decel nonneg-vel* **by** (*simp add*: *divide-simps s-def*)

lemma *q-min*: $0 \leq t \Longrightarrow s0 \leq q\ t$
unfolding *q-def* **using** *nonneg-vel* **by** *auto*

lemma *q-mono*: $x \leq y \Longrightarrow q\ x \leq q\ y$
unfolding *q-def* **using** *nonneg-vel* **by** (*auto simp*: *mult-left-mono*)

1.3.4 Maximum at Stopping Time

lemma *s-max*: $s\ x \leq$ *s-stop*
using *p-max*[*of x*] *p-max*[*of 0*] **unfolding** *s-t-stop* **by** (*auto simp*: *s-def*)

lemma *s-eq-s-stop*: *NO-MATCH t-stop x* $\Longrightarrow x \geq$ *t-stop* $\Longrightarrow s\ x =$ *s-stop*
using *t-stop-nonneg* **by** (*auto simp*: *s-def p-max-def*)

end

1.4 Safe Distance

locale *safe-distance* =
fixes $a_e\ v_e\ s_e$ :: *real*
fixes $a_o\ v_o\ s_o$ :: *real*
assumes *nonneg-vel-ego* : $0 \leq v_e$
assumes *nonneg-vel-other* : $0 \leq v_o$
assumes *decelerate-ego* : $a_e < 0$
assumes *decelerate-other* : $a_o < 0$
assumes *in-front* : $s_e < s_o$
begin

lemmas *hyps* =
nonneg-vel-ego
nonneg-vel-other
decelerate-ego
decelerate-other
in-front

sublocale *ego*: *braking-movement* $a_e\ v_e\ s_e$ **by** (*unfold-locales*; *rule hyps*)
sublocale *other*: *braking-movement* $a_o\ v_o\ s_o$ **by** (*unfold-locales*; *rule hyps*)
sublocale *ego-other*: *movement* $a_o - a_e\ v_o - v_e\ s_o - s_e$ **by** *unfold-locales*

1.4.1 Collision

definition *collision* :: *real set* ⇒ *bool* **where**

collision time-set ≡ (∃ *t* ∈ *time-set*. *ego.s t* = *other.s t*)

abbreviation *no-collision* :: *real set* ⇒ *bool* **where**
no-collision time-set ≡ ¬ *collision time-set*

lemma *no-collision-initially* : *no-collision* {.. *0*}
using *decelerate-ego nonneg-vel-ego*
using *decelerate-other nonneg-vel-other in-front*
by (*auto simp*: *divide-simps collision-def ego.s-def other.s-def*)

lemma *no-collisionI*:
(⋀*t*. *t* ∈ *S* ⟹ *ego.s t* ≠ *other.s t*) ⟹ *no-collision S*
unfolding *collision-def* **by** *blast*

theorem *cond-1*: *ego.s-stop* < s_o ⟹ *no-collision* {*0*..}
proof (*rule no-collisionI*, *simp*)
fix *t*::*real*
assume *t* ≥ *0*
have *ego.s t* ≤ *ego.s-stop*
by (*rule ego.s-max*)
also assume ... < s_o
also have ... = *other.s 0*
by (*simp add*: *other.init-s*)
also have ... ≤ *other.s t*
using ‹*0* ≤ *t*› *hyps*
by (*intro other.s-mono*) *auto*
finally show *ego.s t* ≠ *other.s t*
by *simp*
qed

lemma *ego-other-strict-ivt*:
assumes *ego.s t* > *other.s t*
shows *collision* {*0* ..< *t*}
proof *cases*
have [*simp*]: s_e < s_o ⟹ *ego.s 0* ≤ *other.s 0*
by (*simp add*: *movement.s-def*)
assume *0* ≤ *t*
with *assms in-front*
have ∃ *x*≥*0*. *x* ≤ *t* ∧ *other.s x* − *ego.s x* = *0*
by (*intro IVT2*, *auto simp*: *continuous-diff other.isCont-s ego.isCont-s*)
then show *?thesis*
using *assms*
by (*auto simp add*: *algebra-simps collision-def Bex-def order.order-iff-strict*)
qed (*insert assms hyps*, *auto simp*: *collision-def ego.init-s other.init-s intro*!: *bexI*[**where** *x=0*])

lemma *collision-subset*: *collision s* ⟹ *s* ⊆ *t* ⟹ *collision t*
by (*auto simp*: *collision-def*)

lemma *ego-other-ivt*:
assumes *ego.s t* $\geq$ *other.s t*
shows *collision* {*0* .. *t*}
proof *cases*
assume *ego.s t* > *other.s t*
from *ego-other-strict-ivt*[*OF this*]
show *?thesis*
by (*rule collision-subset*) *auto*
qed (*insert hyps assms*; *cases t* $\geq$ *0*; *force simp*: *collision-def ego.init-s other.init-s*)

theorem *cond-2*:
assumes *ego.s-stop* $\geq$ *other.s-stop*
shows *collision* {*0* ..}
using *assms*
apply (*intro collision-subset*[**where** *t*={*0* ..} **and** *s* = {*0* .. *max ego.t-stop other.t-stop*}])
apply (*intro ego-other-ivt*[**where** *t* = *max ego.t-stop other.t-stop*])
apply (*auto simp*: *ego.s-eq-s-stop other.s-eq-s-stop*)
done

abbreviation *D2* :: *real* **where**
D2 $\equiv (v_o - v_e)\hat{}2 - 2 * (a_o - a_e) * (s_o - s_e)$

abbreviation $t_D{}'$ :: *real* **where**
$t_D{}' \equiv$ *sqrt* (*2* * (*ego.s-stop* − *other.s-stop*) / a_o)

lemma *pos-via-half-dist*:
dist a b < *b* / *2* $\Longrightarrow$ *b* > *0* $\Longrightarrow$ *a* > *0*
by (*auto simp*: *dist-real-def abs-real-def split*: *if-splits*)

lemma *collision-within-p*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* < *other.s-stop*
shows *collision* {*0*..} $\longleftrightarrow$ ($\exists$ *t*$\geq$*0*. *ego.p t* = *other.p t* $\wedge$ *t* < *ego.t-stop* $\wedge$ *t* < *other.t-stop*)
proof (*auto simp*: *collision-def*, *goal-cases*)
case (*2 t*)
then show *?case*
by (*intro bexI*[**where** *x* = *t*]) (*auto simp*: *ego.s-def other.s-def*)
next
case (*1 t*)
then show *?case* **using** *assms hyps ego.t-stop-nonneg other.t-stop-nonneg*
apply (*auto simp*: *ego.s-def other.s-def ego.s-t-stop other.s-t-stop ego.p-t-stop other.p-t-stop not-le*
split: *if-splits*)
defer
proof *goal-cases*
case *1*
from *1* **have** *le*: *ego.t-stop* $\leq$ *other.t-stop* **by** *auto*
from *1* **have** *ego.t-stop* < *t* **by** *simp*

```
    from other.s-strict-mono[OF this] 1
    have other.s ego.t-stop < other.s t
      by auto
    also have ... = ego.s ego.t-stop
      using ego.s-t-stop ego.t-stop-nonneg 1 other.s-def by auto
    finally have other.s ego.t-stop < ego.s ego.t-stop .
    from ego-other-strict-ivt[OF this] le in-front
    show ?case
      by (auto simp add: collision-def) (auto simp: movement.s-def split: if-splits)
  next
    case 2
    from 2 have other.p-max = ego.p t by simp
    also have ... ≤ ego.p ego.t-stop
      using 2
      by (intro ego.p-mono) auto
    also have ... = ego.p-max
      by (simp add: ego.p-t-stop)
    also note ‹... < other.p-max›
    finally show ?case by arith
  next
    case 3
    thus ∃ t≥0. ego.p t = other.p t ∧ t < ego.t-stop ∧ t < other.t-stop
      apply (cases t = other.t-stop)
       apply (simp add: other.p-t-stop )
       apply (metis (no-types) ego.p-max not-le)
      apply (cases t = ego.t-stop)
       apply (simp add: ego.p-t-stop)
       defer
       apply force
    proof goal-cases
      case (1)
      let ?d = λt. other.p' t − ego.p' t
      define d' where d' = ?d ego.t-stop / 2
      have d-cont: isCont ?d ego.t-stop
        unfolding ego.t-stop-def other.p'-def ego.p'-def by simp
      have ?d ego.t-stop > 0
        using 1
          by (simp add: ego.p'-stop-zero other.p'-pos-iff) (simp add: ego.t-stop-def
other.t-stop-def)
      then have d' > 0 by (auto simp: d'-def)
       from d-cont[unfolded continuous-at-eps-delta, THEN spec, rule-format, OF
‹d' > 0›]
      obtain e where e: e > 0 ⋀x. dist x ego.t-stop < e ⟹ ?d x > 0
        unfolding d'-def
        using ‹?d ego.t-stop > 0› pos-via-half-dist
        by force
      define t' where t' = ego.t-stop − min (ego.t-stop / 2) (e / 2)
      have 0 < ego.t-stop using 1 by auto
      have other.p t' − ego.p t' < other.p ego.t-stop − ego.p ego.t-stop
```

```
      apply (rule DERIV-pos-imp-increasing[of t'])
       apply (force simp: t'-def e min-def ‹0 < ego.t-stop›)
      apply (auto intro!: exI[where x = ?d x for x] intro!: derivative-intros e)
      using ‹e > 0›
      apply (auto simp: t'-def dist-real-def algebra-simps)
      done
    also have ... = 0 using 1 by (simp add: ego.p-t-stop)
    finally have less: other.p t' < ego.p t' by simp
    have t' > 0
      using 1 by (auto simp: t'-def algebra-simps min-def)
    have t' < ego.t-stop by (auto simp: t'-def ‹e > 0› ‹ego.t-stop > 0›)
    from less-le-trans[OF ‹t' < ego.t-stop› ‹ego.t-stop ≤ other.t-stop›]
    have t' < other.t-stop .
    from ego-other-strict-ivt[of t'] less
    have collision {0..<t'}
      using ‹t' > 0› ‹t' < ego.t-stop› ‹t' < other.t-stop›
      by (auto simp: other.s-def ego.s-def split: if-splits)
    thus ?case
      using ‹t' > 0› ‹t' < ego.t-stop› ‹t' < other.t-stop›
      apply (auto simp: collision-def ego.s-def other.s-def movement.p-def
          split: if-splits)
      apply (rule-tac x = t in exI)
      apply (auto simp: movement.p-def)[]
      done
  qed
 qed
qed

lemma collision-within-eq:
 assumes s_o ≤ ego.s-stop ego.s-stop < other.s-stop
 shows collision {0..} ⟷ collision {0 ..< min ego.t-stop other.t-stop}
 unfolding collision-within-p[OF assms]
 unfolding collision-def
 by (safe; force
   simp: ego.s-def other.s-def movement.p-def ego.t-stop-def other.t-stop-def
   split: if-splits)

lemma collision-excluded: (⋀t. t ∈ T ⟹ ego.s t ≠ other.s t) ⟹ collision S ⟷
collision (S − T)
 by (auto simp: collision-def)

lemma collision-within-less:
 assumes s_o ≤ ego.s-stop ego.s-stop < other.s-stop
 shows collision {0..} ⟷ collision {0 <..< min ego.t-stop other.t-stop}
proof −
 note collision-within-eq[OF assms]
 also have collision {0 ..< min ego.t-stop other.t-stop} ⟷
   collision ({0 ..< min ego.t-stop other.t-stop} − {0})
   using hyps assms
```

```
    by (intro collision-excluded) (auto simp: ego.s-def other.s-def)
  also have {0 ..< min ego.t-stop other.t-stop} − {0} = {0 <..< min ego.t-stop
other.t-stop}
    by auto
  finally show ?thesis
    unfolding collision-def
    by (safe;
      force
        simp: ego.s-def other.s-def movement.p-def ego.t-stop-def other.t-stop-def
        split: if-splits)
qed

theorem cond-3:
  assumes s_o ≤ ego.s-stop ego.s-stop < other.s-stop
  shows collision {0..} ⟷ (a_o > a_e ∧ v_o < v_e ∧ 0 ≤ D2 ∧ sqrt D2 > v_e − a_e
/ a_o * v_o)
proof −
  have v_o ≠ 0
    using assms(1) assms(2) movement.s-def movement.t-stop-def by auto
  with hyps have v_o > 0 by auto
  note hyps = hyps this
  define t1 where t1 = (− (v_o − v_e) + sqrt D2) / (a_o − a_e)
  define t2 where t2 = (− (v_o − v_e) − sqrt D2) / (a_o − a_e)
 define bounded where bounded ≡ λt. (0 ≤ t ∧ t ≤ ego.t-stop ∧ t ≤ other.t-stop)
  have ego-other-conv:
    ⋀t. bounded t ⟹ ego.p t = other.p t ⟷ ego-other.p t = 0
    by (auto simp: movement.p-def field-split-simps)
  let ?r = {0 <..< min ego.t-stop other.t-stop}
  have D2: D2 = (v_o − v_e)^2 − 4 * ((a_o − a_e) / 2) * (s_o − s_e) by simp
  define D where D = D2
  note D = D-def[symmetric]
  define x1 where x1 ≡ (− (v_o − v_e) + sqrt D2) / (2 * ((a_o − a_e) / 2))
  define x2 where x2 ≡ (− (v_o − v_e) − sqrt D2) / (2 * ((a_o − a_e) / 2))
  have x2: x2 =(− (v_o − v_e) − sqrt D2) / (a_o − a_e)
    by (simp add: x2-def field-split-simps)
  have x1: x1 =(− (v_o − v_e) + sqrt D2) / (a_o − a_e)
    by (simp add: x1-def field-split-simps)
  from collision-within-less[OF assms]
  have coll-eq: collision {0..} = collision ?r
    by (auto simp add: bounded-def)
  also have ... ⟷ (a_o > a_e ∧ v_o < v_e ∧ 0 ≤ D2 ∧ sqrt D2 > v_e − a_e / a_o *
v_o)
  proof safe
    assume H: a_e < a_o v_o < v_e 0 ≤ D2
    assume sqrt: sqrt D2 > v_e − a_e / a_o * v_o
    have nz: (a_o − a_e) / 2 ≠ 0 using ‹a_e < a_o› by simp
   note sol = quadratic-eq-zeroes-iff[OF D2 x1-def[THEN meta-eq-to-obj-eq] x2-def[THEN
meta-eq-to-obj-eq] nz]
    from sol[of x2] ‹0 ≤ D2›
```

```
    have other.p x2 = ego.p x2
      by (auto simp: ego.p-def other.p-def field-split-simps)
    moreover
    have x2 > 0
    proof (rule ccontr)
      assume ¬ 0 < x2
      then have ego-other.p x2 ≥ ego-other.p 0
        using H hyps
        by (intro DERIV-nonpos-imp-nonincreasing[of x2])
          (auto intro!: exI[where x=ego-other.p' x for x] derivative-eq-intros
            simp: ego-other.p'-def add-nonpos-nonpos mult-nonneg-nonpos)
      also have ego-other.p 0 > 0 using hyps by (simp add: ego-other.p-def)
      finally (xtrans) show False using ‹other.p x2 = ego.p x2›
        by (simp add: movement.p-def field-split-simps power2-eq-square)
    qed
    moreover
    have x2 < other.t-stop
      using sqrt H hyps
      by (auto simp: x2 other.t-stop-def field-split-simps power2-eq-square)

    ultimately
    show collision {0<..<min ego.t-stop other.t-stop}
    proof (cases x2 < ego.t-stop, goal-cases)
      case 2
      then have other.s x2 = other.p x2
        by (auto simp: other.s-def)
      also from 2 have ... ≤ ego.p ego.t-stop
        by (auto intro!: ego.p-antimono)
      also have ... = ego.s x2
        using 2 by (auto simp: ego.s-def ego.p-t-stop)
      finally have other.s x2 ≤ ego.s x2 .
      from ego-other-ivt[OF this]
      show ?thesis
        unfolding coll-eq[symmetric]
        by (rule collision-subset) auto
    qed (auto simp: collision-def ego.s-def other.s-def not-le intro!: bexI[where
x=x2])
  next
    let ?max = max ego.t-stop other.t-stop
    let ?min = min ego.t-stop other.t-stop
    assume collision ?r
    then obtain t where t: ego.p t = other.p t 0 < t t < ?min
      by (auto simp: collision-def ego.s-def other.s-def)
    then have t < - (v_e / a_e) t < - (v_o / a_o) t < other.t-stop
      by (simp-all add: ego.t-stop-def other.t-stop-def)
    from t have ego-other.p t = 0
      by (auto simp: movement.p-def field-split-simps)
    from t have t < ?max by auto
    from hyps assms have 0 < ego-other.p 0
```

```
    by simp
  from ego-other.p-def[abs-def, THEN meta-eq-to-obj-eq]
  have eop-eq: ego-other.p = (λt. 1 / 2 * (a_o − a_e) * t^2 + (v_o − v_e) * t + (s_o
− s_e))
    by (simp add: algebra-simps)
  show a_o > a_e
  proof −
    have ego.p other.t-stop ≤ ego.p-max
      by (rule ego.p-max)
    also have ... ≤ other.p other.t-stop using hyps assms
      by (auto simp:other.s-def ego.s-def ego.p-t-stop split:if-splits)
    finally have 0 ≤ ego-other.p other.t-stop
      by (auto simp add:movement.p-def field-simps)
    from p-convex[OF eop-eq, of 0 t other.t-stop, simplified ‹ego-other.p t = 0›,
        OF ‹0 < t› ‹t < other.t-stop› ‹0 < ego-other.p 0› ‹0 ≤ ego-other.p
other.t-stop›]
    show a_o > a_e by (simp add: algebra-simps)
  qed
  have rewr: 4 * ((a_o − a_e) / 2) = 2 * (a_o − a_e) by simp
  from ‹a_o > a_e› ‹ego-other.p t = 0› ego-other.p-all-zeroes[OF D2[symmetric],
of t]
  have 0 ≤ D2 and disj: (t = (− (v_o − v_e) + sqrt D2) / (a_o − a_e) ∨ t = (−
(v_o − v_e) − sqrt D2) / (a_o − a_e))
    using hyps assms
    unfolding rewr by simp-all
  show 0 ≤ D2 by fact
  from add-strict-mono[OF ‹t < − (v_e / a_e)› ‹t < − (v_o / a_o)›] ‹0 < t› ‹a_o >
a_e›
  have 0 < − (v_e / a_e) + − (v_o / a_o) by (simp add: divide-simps)
  then have 0 > v_e * a_o + a_e * v_o using hyps
    by (simp add: field-split-simps split: if-splits)
  show v_o < v_e
    using ‹a_e < a_o› ‹movement.p (a_o − a_e) (v_o − v_e) (s_o − s_e) t = 0› in-front
t(2)
  apply (auto simp: movement.p-def divide-less-0-iff algebra-simps power2-eq-square)
    by (smt divide-less-0-iff mult-le-cancel-right mult-mono mult-nonneg-nonneg
nonneg-vel-ego)
  from disj have x2 < ?min
  proof rule
    assume t = (− (v_o − v_e) − sqrt D2) / (a_o − a_e)
    then show ?thesis
      using ‹t < ?min›
      by (simp add: x2)
  next
    assume t = (− (v_o − v_e) + sqrt D2) / (a_o − a_e)
    also have ... ≥ x2
      unfolding x2
      apply (rule divide-right-mono)
       apply (subst (2) diff-conv-add-uminus)
```

```
      apply (rule add-left-mono)
     using ‹a_o > a_e› ‹D2 ≥ 0›
     by auto
   also (xtrans) note ‹t < ?min›
   finally show ?thesis .
  qed
  then show sqrt D2 > v_e − a_e / a_o * v_o
   using hyps ‹a_o > a_e›
   by (auto simp: x2 field-split-simps other.t-stop-def)
 qed
 finally show ?thesis .
qed
```

1.4.2 Formalising Safe Distance

First definition for Safe Distance based on *cond-1*.

```
definition absolute-safe-distance :: real where
 absolute-safe-distance = − v_e² / (2 * a_e)

lemma absolute-safe-distance:
 assumes s_o − s_e > absolute-safe-distance
 shows no-collision {0..}
 proof −
 from assms hyps absolute-safe-distance-def have ego.s-stop < s_o
  by (auto simp add:ego.s-def ego.p-def ego.t-stop-def power-def)
 thus ?thesis by (rule cond-1)
 qed
```

First Fallback for Safe Distance.

```
definition fst-safe-distance :: real where
 fst-safe-distance = v_o² / (2 * a_o) − v_e² / (2 * a_e)

definition distance-leq-d2 :: real where
 distance-leq-d2 = (a_e + a_o) / (2 * a_o²) * v_o² − v_o * v_e / a_o

lemma snd-leq-fst-exp: distance-leq-d2 ≤ fst-safe-distance
proof −
 have 0 ≤ (other.t-stop − ego.t-stop)² by auto
  hence − ego.t-stop² ≤ other.t-stop² − 2 * other.t-stop * ego.t-stop by (simp
add:power-def algebra-simps)
 with hyps(3) have − ego.t-stop² * (a_e / 2) ≥ (other.t-stop² − 2 * other.t-stop
* ego.t-stop) * (a_e / 2)
   by (smt half-gt-zero-iff mult-le-cancel-right)
 with ego.t-stop-def other.t-stop-def hyps
  have − v_e² / (2 * a_e) ≥ a_e * v_o² / (2 * a_o²) − v_o * v_e / a_o by (simp
add:power-def algebra-simps)
 with fst-safe-distance-def distance-leq-d2-def
 have 1: fst-safe-distance ≥  a_e * v_o² / (2 * a_o²) − v_o * v_e / a_o + v_o² / (2 *
a_o) by (auto simp add:algebra-simps)
```

have $a_e * {v_o}^2 \ / \ (2 * {a_o}^2) - v_o * v_e \ / \ a_o + {v_o}^2 \ / \ (2 * a_o) = \textit{distance-leq-d2}$ (**is** *?LHS = -*)
proof −
have $\textit{?LHS} = a_e * {v_o}^2 \ / \ (2 * {a_o}^2) - v_o * v_e \ / \ a_o + a_o * {v_o}^2 \ / \ (2 * {a_o}^2)$
by (*auto simp add*: *algebra-simps power-def*)
also have ... $=$ *distance-leq-d2*
by (*auto simp add*: *power-def field-split-simps distance-leq-d2-def*)
finally show *?thesis* **by** *auto*
qed
with *1* **show** *?thesis* **by** *auto*
qed

lemma *sqrt-D2-leq-stop-time-diff*:
assumes $a_e < a_o$
assumes $0 \leq v_e - a_e \ / \ a_o * v_o$
assumes $s_o - s_e \geq \textit{distance-leq-d2}$
shows $\textit{sqrt D2} \leq v_e - a_e \ / \ a_o * v_o$
proof −
from *assms* **have** $- 2 * (a_o - a_e) * (s_o - s_e) \leq - 2 * (a_o - a_e) * \textit{distance-leq-d2}$ (**is** $\textit{?L} \leq \textit{?R}$)
by *simp*
hence $\textit{D2} \leq (v_o - v_e)^2 - 2 * (a_o - a_e) * \textit{distance-leq-d2}$ **by** (*simp add*: *algebra-simps*)
also have ... $= (v_e - a_e \ / \ a_o * v_o)^2$
proof −
from *distance-leq-d2-def*
have *1*: $(v_o - v_e)^2 - 2 * (a_o - a_e) * \textit{distance-leq-d2} =$
$(v_o - v_e)^2 - (a_o - a_e) * (a_e + a_o) \ / \ {a_o}^2 * {v_o}^2 + 2 * (a_o - a_e) * v_o * v_e \ / \ a_o$
by (*auto simp add*: *field-split-simps*)
with *hyps*(*4*) **have** ... $= (v_e - a_e \ / \ a_o * v_o)^2$
by (*auto simp add*: *power-def field-split-simps*)
with *1* **show** *?thesis* **by** *auto*
qed
finally show *?thesis* **by** (*smt assms*(*2*) *real-le-lsqrt real-sqrt-le-0-iff*)
qed

lemma *cond2-imp-pos-vo*:
assumes $s_o \leq \textit{ego.s-stop}$ $\textit{ego.s-stop} < \textit{other.s-stop}$
shows $v_o \neq 0$
proof (*rule ccontr*)
assume $\neg \ v_o \neq 0$
with *other.s-def other.t-stop-def* **have** $\textit{other.s-stop} = s_o$ **by** *auto*
with *assms*(*2*) **have** $\textit{ego.s-stop} < s_o$ **by** *auto*
with *assms*(*1*) **show** *False* **by** *auto*
qed

lemma *cond2-imp-gt-fst-sd*:
assumes $s_o \leq \textit{ego.s-stop}$ $\textit{ego.s-stop} < \textit{other.s-stop}$

shows *fst-safe-distance* $< s_o - s_e$
proof (*cases* $v_e \neq 0$)
case *True*
from *fst-safe-distance-def assms ego.s-def ego.t-stop-pos*[*OF* ‹$v_e \neq 0$›] *ego.p-def ego.t-stop-def*
other.s-def other.t-stop-pos[*OF cond2-imp-pos-vo*[*OF assms*]] *other.p-def other.t-stop-def hyps*
show *?thesis* **by** (*simp add*: *power-def algebra-simps*)
next
case *False*
with *fst-safe-distance-def* **have** *fst-safe-distance* $= {v_o}^2 \,/\, (2 * a_o)$ **by** *auto*
also have ... ≤ 0 **by** (*simp add*: *divide-nonneg-neg hyps*)
also have ... $< s_o - s_e$ **by** (*simp add*: *algebra-simps hyps*)
finally show *?thesis* **by** *auto*
qed

Second Fallback for Safe Distance.

definition *snd-safe-distance* :: *real* **where**
snd-safe-distance $= (v_o - v_e)^2 \,/\, (2 * (a_o - a_e))$

lemma *fst-leq-snd-safe-distance*:
assumes $a_e < a_o$
shows*fst-safe-distance* $\leq$ *snd-safe-distance*
proof −
have $0 \leq (v_o \,/\, a_o - v_e \,/\, a_e)^2$ **by** *auto*
hence *1*: $0 \leq (v_o \,/\, a_o)^2 - 2 * v_o * v_e \,/\, (a_o * a_e) + (v_e \,/\, a_e)^2$ **by** (*auto simp add*: *power-def algebra-simps*)
from *hyps* **have** $0 \leq a_o * a_e$ **by** (*simp add*: *mult-nonpos-nonpos*)
from *mult-right-mono*[*OF 1 this*] *hyps*
have $0 \leq {v_o}^2 * a_e \,/\, a_o - 2 * v_o * v_e + {v_e}^2 * a_o \,/\, a_e$ **by** (*auto simp add*: *power-def algebra-simps*)
with *hyps* **have** *2*: $({v_o}^2 \,/\, (2 * a_o) - {v_e}^2 \,/\, (2 * a_e)) * (2 * (a_o - a_e)) \leq (v_o - v_e)^2$
by (*auto simp add*: *power-def field-split-simps*)
from *assms* **have** $0 \leq 2 * (a_o - a_e)$ **by** *auto*
from *divide-right-mono*[*OF 2 this*] *assms fst-safe-distance-def snd-safe-distance-def*
show *?thesis* **by** *auto*
qed

lemma *snd-safe-distance-iff-nonneg-D2*:
assumes $a_e < a_o$
shows $s_o - s_e \leq$ *snd-safe-distance* $\longleftrightarrow$ $0 \leq D2$
proof −
from *snd-safe-distance-def assms pos-le-divide-eq*[*of* $2 * (a_o - a_e)$]
have $s_o - s_e \leq$ *snd-safe-distance* $\longleftrightarrow$ $(s_o - s_e) * (2 * (a_o - a_e)) \leq (v_o - v_e)^2$
by *auto*
also have ... $\longleftrightarrow$ $0 \leq D2$ **by** (*auto simp add*: *algebra-simps*)
finally show *?thesis* **by** *auto*
qed

lemma *t-stop-diff-neg-means-leq-D2*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop* $a_e < a_o$ $0 \leq$ *D2*
shows $v_e - a_e$ / $a_o * v_o < 0 \longleftrightarrow$ *sqrt D2* $> v_e - a_e$ / $a_o * v_o$
proof
assume *only-if*: $v_e - a_e$ / $a_o * v_o < 0$
from *assms* **have** ... $\leq$ *sqrt D2* **by** *auto*
with *only-if* **show** $v_e - a_e$ / $a_o * v_o <$ *sqrt D2* **by** *linarith*
next
assume *if-part*: $v_e - a_e$ / $a_o * v_o <$ *sqrt D2*
from *cond2-imp-gt-fst-sd*[*OF assms*(*1*) *assms*(*2*)] *snd-leq-fst-exp* **have** *distance-leq-d2* $\leq s_o - s_e$ **by** *auto*
from *if-part* **and** *sqrt-D2-leq-stop-time-diff* [*OF* ‹$a_e < a_o$› - ‹*distance-leq-d2* $\leq s_o - s_e$›]
show $v_e - a_e$ / $a_o * v_o < 0$ **by** *linarith*
qed

theorem *cond-3′*:
assumes $s_o \leq$ *ego.s-stop ego.s-stop* $<$ *other.s-stop*
shows *collision* $\{0..\} \longleftrightarrow (a_o > a_e \wedge v_o < v_e \wedge s_o - s_e \leq$ *snd-safe-distance* $\wedge$ $v_e - a_e$ / $a_o * v_o < 0)$
proof (*cases* $a_o \leq a_e \vee v_o \geq v_e$)
case *True*
with *cond-3*[*OF assms*] **show** *?thesis* **by** *auto*
next
case *False*
from ‹$\neg$ $(a_o \leq a_e \vee v_e \leq v_o)$› **have** $a_o > a_e$ **by** *auto*
from ‹$\neg$ $(a_o \leq a_e \vee v_e \leq v_o)$› **have** $v_o < v_e$ **by** *auto*
show *?thesis*
proof −
from *snd-safe-distance-iff-nonneg-D2* [*OF* ‹$a_o > a_e$›]
have *1*: $(a_e < a_o \wedge v_o < v_e \wedge s_o - s_e \leq$ *snd-safe-distance* $\wedge$ $v_e - a_e$ / $a_o * v_o < 0) \longleftrightarrow$
$(a_e < a_o \wedge v_o < v_e \wedge 0 \leq D2 \wedge v_e - a_e$ / $a_o * v_o < 0)$ **by** *auto*

from *t-stop-diff-neg-means-leq-D2*[*OF assms* ‹$a_e < a_o$›]
have ... $= (a_e < a_o \wedge v_o < v_e \wedge 0 \leq D2 \wedge$ *sqrt D2* $> v_e - a_e$ / $a_o * v_o)$ **by** *auto*

with *1 cond-3*[*OF assms*] **show** *?thesis* **by** *blast*
qed
qed

definition $d ::$ *real* $\Rightarrow$ *real* **where**
$d\ t = ($
if $t \leq 0$ *then* $s_o - s_e$
else if $t \leq$ *ego.t-stop* $\wedge$ $t \leq$ *other.t-stop then ego-other.p t*
else if ego.t-stop $\leq t \wedge t \leq$ *other.t-stop then other.p t* $-$ *ego.s-stop*
else if other.t-stop $\leq t \wedge t \leq$ *ego.t-stop then other.s-stop* $-$ *ego.p t*

```
    else other.s-stop − ego.s-stop
  )

lemma d-diff: d t = other.s t − ego.s t
  by (auto simp: d-def ego.s-eq-s-stop other.s-eq-s-stop ego.s-cond-def other.s-cond-def
    movement.p-def field-split-simps)

lemma collision-d: collision S ⟷ (∃ t∈S. d t = 0)
  by (force simp: d-diff collision-def )

lemma collision-restrict: collision {0..} ⟷ collision {0..max ego.t-stop other.t-stop}

  by (auto simp: max.coboundedI1 ego.t-stop-nonneg min-def
    ego.s-eq-s-stop other.s-eq-s-stop collision-def
    intro!: bexI[where x = min t (max (movement.t-stop a_e v_e) (movement.t-stop
a_o v_o)) for t])

lemma collision-union: collision (A ∪ B) ⟷ collision A ∨ collision B
  by (auto simp: collision-def)

lemma symbolic-checker:
  collision {0..} ⟷
    (quadroot-in 0 (min ego.t-stop other.t-stop) (1/2 * (a_o − a_e)) (v_o − v_e) (s_o −
s_e)) ∨
    (quadroot-in ego.t-stop other.t-stop (1/2 * a_o) v_o (s_o − ego.s-stop)) ∨
    (quadroot-in other.t-stop ego.t-stop (1/2 * a_e) v_e (s_e − other.s-stop))
 (is - ⟷ ?q1 ∨ ?q2 ∨ ?q3)
proof −
  have ∗: {0..max ego.t-stop other.t-stop} =
    {0 .. min ego.t-stop other.t-stop} ∪ {ego.t-stop .. other.t-stop} ∪ {other.t-stop
.. ego.t-stop}
    using ego.t-stop-nonneg other.t-stop-nonneg
    by auto
  have collision {0..min (movement.t-stop a_e v_e) (movement.t-stop a_o v_o)} = ?q1
    by (force simp: collision-def quadroot-in-def root-in-def d-def
      power2-eq-square field-split-simps movement.p-def movement.s-cond-def)
  moreover
  have collision {ego.t-stop .. other.t-stop} = ?q2
    using ego.t-stop-nonneg
    by (force simp: collision-def quadroot-in-def root-in-def d-def
      ego.s-eq-s-stop movement.s-cond-def movement.p-def)
  moreover
  have collision {other.t-stop .. ego.t-stop} = ?q3
    using other.t-stop-nonneg
    by (force simp: collision-def quadroot-in-def root-in-def d-def
      other.s-eq-s-stop movement.s-cond-def movement.p-def)
  ultimately
  show ?thesis
    unfolding collision-restrict ∗ collision-union
```

by *auto*
qed

end

1.5 Checker Design

definition *rel-dist-to-stop* :: *real* ⇒ *real* ⇒ *real* **where**
rel-dist-to-stop v a ≡ − v^2 / (*2* ∗ *a*)

context includes *floatarith-syntax* **begin**
definition *rel-dist-to-stop-expr* :: *nat* ⇒ *nat* ⇒ *floatarith* **where**
rel-dist-to-stop-expr v a = *Mult* (*Minus* (*Power* (*Var v*) *2*)) (*Inverse* (*Mult* (*Num 2*) (*Var a*)))

definition *rel-dist-to-stop′* :: *nat* ⇒ *float interval option* ⇒ *float interval option* ⇒ *float interval option* **where**
rel-dist-to-stop′ p v a = *approx p* (*rel-dist-to-stop-expr 0 1*) [*v*, *a*]

lemma *rel-dist-to-stop′*: *interpret-floatarith* (*rel-dist-to-stop-expr 0 1*) [*v*, *a*] = *rel-dist-to-stop v a*
by (*simp add*: *rel-dist-to-stop-def rel-dist-to-stop-expr-def inverse-eq-divide*)

definition *first-safe-dist* :: *real* ⇒ *real* ⇒ *real* **where**
first-safe-dist v_e a_e ≡ *rel-dist-to-stop* v_e a_e

definition *second-safe-dist* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
second-safe-dist v_e a_e v_o a_o ≡ *rel-dist-to-stop* v_e a_e − *rel-dist-to-stop* v_o a_o

definition *second-safe-dist-expr* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *floatarith* **where**
second-safe-dist-expr ve ae vo ao = *Add* (*rel-dist-to-stop-expr ve ae*) (*Minus* (*rel-dist-to-stop-expr vo ao*))

definition *second-safe-dist′* :: *nat* ⇒ *float interval option* ⇒ *float interval option* ⇒ *float interval option* ⇒ *float interval option* ⇒ *float interval option* **where**
second-safe-dist′ p v_e a_e v_o a_o = *approx p* (*second-safe-dist-expr 0 1 2 3*) [v_e, a_e, v_o, a_o]

lemma *second-safe-dist′*:
interpret-floatarith (*second-safe-dist-expr 0 1 2 3*) [*v*, *a*, *v′*, *a′*] = *second-safe-dist v a v′ a′*
by (*simp add*: *second-safe-dist-def second-safe-dist-expr-def rel-dist-to-stop-def rel-dist-to-stop-expr-def inverse-eq-divide*)

definition *t-stop* :: *real* ⇒ *real* ⇒ *real* **where**
t-stop v a ≡ − *v* / *a*

definition *t-stop-expr* :: *nat* ⇒ *nat* ⇒ *floatarith* **where**
t-stop-expr v a = *Minus* (*Mult* (*Var v*) (*Inverse* (*Var a*)))

end

definition *s-stop* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
s-stop s v a ≡ *s* + *rel-dist-to-stop v a*

definition *discriminant* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
discriminant s_e v_e a_e s_o v_o a_o ≡ $(v_o - v_e)^2 - 2 * (a_o - a_e) * (s_o - s_e)$

definition *suff-cond-safe-dist2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool*
where
suff-cond-safe-dist2 s_e v_e a_e s_o v_o a_o ≡
let D2 = *discriminant* s_e v_e a_e s_o v_o a_o
in ¬ ($a_e < a_o \wedge v_o < v_e \wedge 0 \leq D2 \wedge v_e - a_e$ / $a_o * v_o < sqrt\ D2$
)

lemma *less-sqrt-iff*: $y \geq 0 \Longrightarrow x < sqrt\ y \longleftrightarrow (x \geq 0 \longrightarrow x^2 < y)$
by (*smt real-le-lsqrt real-less-rsqrt real-sqrt-ge-zero*)

lemma *suff-cond-safe-dist2-code*[*code*]:
suff-cond-safe-dist2 s_e v_e a_e s_o v_o a_o =
(*let D2* = *discriminant* s_e v_e a_e s_o v_o a_o *in*
$(a_e < a_o \longrightarrow v_o < v_e \longrightarrow 0 \leq D2 \longrightarrow (v_e - a_e$ / $a_o * v_o \geq 0 \wedge (v_e - a_e$ / $a_o * v_o)^2 \geq D2)))$
using *real-sqrt-ge-zero real-less-rsqrt less-sqrt-iff*
by (*auto simp*: *suff-cond-safe-dist2-def Let-def*)

There are two expressions for safe distance. The first safe distance *first-safe-dist* is always valid. Whenever the distance is bigger than *first-safe-dist*, it is guarantee to be collision free. The second one is *second-safe-dist*. If the sufficient condition *suff-cond-safe-dist2* is satisfied and the distance is bigger than *second-safe-dist*, it is guaranteed to be collision free.

definition *check-precond* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool*
where
check-precond s_e v_e a_e s_o v_o $a_o \longleftrightarrow s_o > s_e \wedge 0 \leq v_e \wedge 0 \leq v_o \wedge a_e < 0 \wedge a_o < 0$

lemma *check-precond-safe-distance*:
check-precond s_e v_e a_e s_o v_o a_o = *safe-distance* a_e v_e s_e a_o v_o s_o
proof
assume *safe-distance* a_e v_e s_e a_o v_o s_o
then interpret *safe-distance* a_e v_e s_e a_o v_o s_o .
show *check-precond* s_e v_e a_e s_o v_o a_o
by (*auto simp*: *check-precond-def in-front nonneg-vel-ego other.nonneg-vel ego.decel other.decel*)
qed (*unfold-locales*; *auto simp*: *check-precond-def*)

1.5.1 Prescriptive Checker

definition *checker* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**

```
  checker s_e v_e a_e s_o v_o a_o ≡
    let distance   = s_o − s_e;
        precond    = check-precond s_e v_e a_e s_o v_o a_o;
        safe-dist1 = first-safe-dist v_e a_e;
        safe-dist2 = second-safe-dist v_e a_e v_o a_o;
        cond2      = suff-cond-safe-dist2 s_e v_e a_e s_o v_o a_o
     in precond ∧ (safe-dist1 < distance ∨ (safe-dist2 < distance ∧ distance ≤
safe-dist1 ∧ cond2))

lemma aux-logic:
  assumes a ⟹ b
  assumes b ⟹ a ⟷ c
  shows a ⟷ b ∧ c
  using assms by blast

theorem soundness-correctness:
  checker s_e v_e a_e s_o v_o a_o ⟷ check-precond s_e v_e a_e s_o v_o a_o ∧ safe-distance.no-collision
a_e v_e s_e a_o v_o s_o {0..}
proof (rule aux-logic, simp add: checker-def Let-def)
  assume cp: check-precond s_e v_e a_e s_o v_o a_o
  then have in-front': s_o > s_e
    and nonneg-vel-ego: 0 ≤ v_e
    and nonneg-vel-other: 0 ≤ v_o
    and decelerate-ego: a_e < 0
    and decelerate-other: a_o < 0
    by (auto simp: check-precond-def)

  from in-front' have in-front: 0 < s_o − s_e by arith

  interpret safe-distance a_e v_e s_e a_o v_o s_o by (unfold-locales; fact)
  interpret ego: braking-movement a_e v_e s_e by (unfold-locales; fact)
  interpret other: braking-movement a_o v_o s_o by (unfold-locales; fact)

  have ego.p-max < s_o ∨ other.p-max ≤ ego.p-max ∨ s_o ≤ ego.p-max ∧ ego.p-max
< other.p-max
    by arith
  then show checker s_e v_e a_e s_o v_o a_o = safe-distance.no-collision a_e v_e s_e a_o v_o
s_o {0..}
  proof (elim disjE)
    assume ego.p-max < s_o
    then have checker s_e v_e a_e s_o v_o a_o
      using ‹a_e < 0› cp
    by (simp add: checker-def Let-def first-safe-dist-def rel-dist-to-stop-def ego.p-max-def
        ego.p-def ego.t-stop-def algebra-simps power2-eq-square)
    moreover
    have no-collision {0..}
      using ‹ego.p-max < s_o›
      by (intro cond-1) (auto simp: ego.s-t-stop)
    ultimately show ?thesis by auto
```

```
  next
    assume other.p-max ≤ ego.p-max
    then have ¬ checker s_e v_e a_e s_o v_o a_o
      using ‹a_e < 0› ‹a_o < 0› other.nonneg-vel
      by (auto simp add: checker-def Let-def first-safe-dist-def second-safe-dist-def
        rel-dist-to-stop-def movement.p-max-def
        movement.p-def movement.t-stop-def algebra-simps power2-eq-square)
         (smt divide-nonneg-neg mult-nonneg-nonneg)
    moreover have collision {0..}
      using ‹other.p-max ≤ ego.p-max›
      by (intro cond-2) (auto simp: other.s-t-stop ego.s-t-stop)
    ultimately show ?thesis by auto
  next
    assume H: s_o ≤ ego.p-max ∧ ego.p-max < other.p-max
    then have checker s_e v_e a_e s_o v_o a_o = (¬ (a_e < a_o ∧ v_o < v_e ∧ 0 ≤ D2 ∧ v_e
− a_e / a_o * v_o < sqrt D2))
      using ‹a_e < 0› ‹a_o < 0› cp
    by (simp add: checker-def Let-def first-safe-dist-def rel-dist-to-stop-def ego.p-max-def
        ego.p-def ego.t-stop-def algebra-simps power2-eq-square second-safe-dist-def
       suff-cond-safe-dist2-def discriminant-def not-less other.p-max-def other.p-def
other.t-stop-def)
    also have ... = no-collision {0..}
      using H
      unfolding Not-eq-iff
      by (intro cond-3[symmetric]) (auto simp: ego.s-t-stop other.s-t-stop)
    finally show ?thesis by auto
  qed
qed

definition checker2 :: real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ bool where
  checker2 s_e v_e a_e s_o v_o a_o ≡
    let distance  = s_o − s_e;
        precond   = check-precond s_e v_e a_e s_o v_o a_o;
        safe-dist1 = first-safe-dist v_e a_e;
        safe-dist2 = second-safe-dist v_e a_e v_o a_o;
        safe-dist3 = − rel-dist-to-stop (v_o − v_e) (a_o − a_e)
    in
      if ¬ precond then False
      else if distance > safe-dist1 then True
      else if a_o > a_e ∧ v_o < v_e ∧ v_e − a_e / a_o * v_o < 0 then distance > safe-dist3
      else distance > safe-dist2

theorem checker-eq-checker2: checker s_e v_e a_e s_o v_o a_o ⟷ checker2 s_e v_e a_e s_o
v_o a_o
proof (cases check-precond s_e v_e a_e s_o v_o a_o)
  case False
  with checker-def checker2-def
  show ?thesis by auto
next
```

case *True*
with *check-precond-def safe-distance-def*
have *safe-distance* a_e v_e s_e a_o v_o s_o **by** (*simp add*: *check-precond-safe-distance*)

from *this* **interpret** *safe-distance* a_e v_e s_e a_o v_o s_o **by** *auto*
interpret *ego*: *braking-movement* a_e v_e s_e **by** (*unfold-locales*; *fact*)
interpret *other*: *braking-movement* a_o v_o s_o **by** (*unfold-locales*; *fact*)

from ‹*check-precond* s_e v_e a_e s_o v_o a_o› *cond-3 cond-3'*[*symmetric*] *fst-leq-snd-safe-distance ego.s-t-stop ego.p-max-def ego.p-def ego.t-stop-def hyps other.s-t-stop other.p-max-def other.p-def other.t-stop-def checker2-def checker-def suff-cond-safe-dist2-def fst-safe-distance-def first-safe-dist-def snd-safe-distance-def second-safe-dist-def rel-dist-to-stop-def discriminant-def*
show *?thesis*
by (*auto simp add*:*power-def Let-def split*: *if-splits*)
qed

definition *checker3* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
checker3 s_e v_e a_e s_o v_o a_o ≡
let distance = $s_o - s_e$;
precond = *check-precond* s_e v_e a_e s_o v_o a_o;
s-stop-e = s_e + *rel-dist-to-stop* v_e a_e;
s-stop-o = s_o + *rel-dist-to-stop* v_o a_o
in precond ∧ (*s-stop-e* < s_o
∨ (s_o ≤ *s-stop-e* ∧ *s-stop-e* < *s-stop-o* ∧
(¬(a_o > a_e ∧ v_o < v_e ∧ $v_e - a_e$ / a_o * v_o < *0* ∧ *distance* * ($a_o - a_e$) ≤ $(v_o - v_e)^2$ / *2*))))

theorem *checker2-eq-checker3*:
checker2 s_e v_e a_e s_o v_o a_o ⟷ *checker3* s_e v_e a_e s_o v_o a_o
apply (*auto simp*: *checker2-def checker3-def Let-def first-safe-dist-def not-less suff-cond-safe-dist2-def second-safe-dist-def rel-dist-to-stop-def check-precond-def*)
proof *goal-cases*
case *1*
then interpret *safe-distance*
by *unfold-locales auto*
from *fst-leq-snd-safe-distance 1*
show *?case*
by (*auto simp*: *fst-safe-distance-def snd-safe-distance-def*)
next
case *2*
then interpret *safe-distance*
by *unfold-locales auto*
from *fst-leq-snd-safe-distance 2*
show *?case*
by (*auto simp*: *fst-safe-distance-def snd-safe-distance-def field-split-simps*)
next

case *3*
then interpret *safe-distance*
by *unfold-locales auto*
from *fst-leq-snd-safe-distance 3*
show *?case*
by (*auto simp*: *fst-safe-distance-def snd-safe-distance-def field-split-simps*)
qed

1.5.2 Approximate Checker

lemma *checker2-def'*: *checker2 a b c d e f* = (
let distance = *d* − *a*;
precond = *check-precond a b c d e f*;
safe-dist1 = *first-safe-dist b c*;
safe-dist2 = *second-safe-dist b c e f*;
C = *c* < *f* ∧ *e* < *b* ∧ *b* ∗ *f* > *c* ∗ *e*;
P1 = (*e* − *b*)^2 < *2* ∗ *distance* ∗ (*f* − *c*);
P2 = − *b*^{2} / *c* + *e*^{2} / *f* < *2* ∗ *distance*
in precond ∧ (*safe-dist1* < *distance* ∨
safe-dist1 ≥ *distance* ∧ (*C* ∧ *P1* ∨ ¬*C* ∧ *P2*)))
unfolding *checker2-def*
by (*auto simp*: *Let-def field-split-simps check-precond-def second-safe-dist-def*
rel-dist-to-stop-def)

lemma *power2-less-sqrt-iff*: (*x*::*real*)^2 < *y* ⟷ (*y* ≥ *0* ∧ *abs x* < *sqrt y*)
apply (*auto simp*: *real-less-rsqrt abs-real-def less-sqrt-iff*)
apply (*meson le-less le-less-trans not-less power2-less-0*)+
done

schematic-goal *checker-form*: *interpret-form ?x ?y* ⟹ *checker* s_e v_e a_e s_o v_o a_o
unfolding *checker-eq-checker2 checker2-eq-checker3 checker3-def check-precond-def first-safe-dist-def second-safe-dist-def*
suff-cond-safe-dist2-def Let-def t-stop-def s-stop-def
rel-dist-to-stop-def
discriminant-def
not-le not-less
de-Morgan-conj
de-Morgan-disj
power2-less-sqrt-iff
apply (*tactic* ‹(*Reification.tac* @{*context*} @{*thms interpret-form.simps interpret-floatarith.simps interpret-floatarith-divide interpret-floatarith-diff*}) *NONE 1*›)
apply *assumption*
done

context includes *floatarith-syntax* **begin**
definition *checker'* *p* s_e v_e a_e s_o v_o a_o = *approx-form p*
(*Conj* (*Conj* (*Less* (*Var* (*Suc* (*Suc 0*))) (*Var* (*Suc* (*Suc* (*Suc 0*)))))
(*Conj* (*LessEqual* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))))))
(*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))))

(*Conj* (*LessEqual* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))) (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))))
(*Conj* (*Less* (*Var* *0*) (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))))))
(*Less* (*Var* (*Suc* *0*)) (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))))))))
(*Disj* (*Less* (*Add* (*Var* (*Suc* (*Suc* *0*)))
(*Mult* (*Minus* (*Power* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))) *2*)) (*Inverse* (*Mult* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))) (*Var* *0*)))))
(*Var* (*Suc* (*Suc* (*Suc* *0*)))))
(*Conj* (*LessEqual* (*Var* (*Suc* (*Suc* (*Suc* *0*))))
(*Add* (*Var* (*Suc* (*Suc* *0*)))
(*Mult* (*Minus* (*Power* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))) *2*)) (*Inverse* (*Mult* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))) (*Var* *0*))))))
(*Conj* (*Less* (*Add* (*Var* (*Suc* (*Suc* *0*)))
(*Mult* (*Minus* (*Power* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))) *2*)) (*Inverse* (*Mult* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))) (*Var* *0*)))))
(*Add* (*Var* (*Suc* (*Suc* (*Suc* *0*))))
(*Mult* (*Minus* (*Power* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))) *2*))
(*Inverse* (*Mult* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))) (*Var* (*Suc* *0*)))))
(*Disj* (*LessEqual* (*Var* (*Suc* *0*)) (*Var* *0*))
(*Disj* (*LessEqual* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*)))))) (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))))
(*Disj* (*LessEqual* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))))
(*Add* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))
(*Minus* (*Mult* (*Mult* (*Var* *0*) (*Inverse* (*Var* (*Suc* *0*)))) (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))))))
(*Less* (*Mult* (*Power* (*Add* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))) (*Minus* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))))) *2*)
(*Inverse* (*Var* (*Suc* (*Suc* (*Suc* (*Suc* *0*)))))))
(*Mult* (*Add* (*Var* (*Suc* (*Suc* (*Suc* *0*)))) (*Minus* (*Var* (*Suc* (*Suc* *0*))))) (*Add* (*Var* (*Suc* *0*)) (*Minus* (*Var* *0*))))))))))))
([a_e, a_o, s_e, s_o, *Interval′* (*Float* *2* *0*) (*Float* *2* *0*), v_e, v_o, *Interval′* (*Float* *0* *1*) (*Float* *0* *1*)]) (*replicate* *8* *0*)

lemma *less-Suc-iff-disj*: $i < Suc\ x \longleftrightarrow i = x \vee i < x$
by *auto*

lemma *checker′-soundness-correctness*:
assumes $a \in \{real\text{-}of\text{-}float\ al\ ..\ real\text{-}of\text{-}float\ au\}$
assumes $b \in \{real\text{-}of\text{-}float\ bl\ ..\ real\text{-}of\text{-}float\ bu\}$
assumes $c \in \{real\text{-}of\text{-}float\ cl\ ..\ real\text{-}of\text{-}float\ cu\}$
assumes $d \in \{real\text{-}of\text{-}float\ dl\ ..\ real\text{-}of\text{-}float\ du\}$
assumes $e \in \{real\text{-}of\text{-}float\ el\ ..\ real\text{-}of\text{-}float\ eu\}$
assumes $f \in \{real\text{-}of\text{-}float\ fl\ ..\ real\text{-}of\text{-}float\ fu\}$
assumes *chk*: *checker′ p* (*Interval′ al au*) (*Interval′ bl bu*) (*Interval′ cl cu*)

(*Interval′ dl du*) (*Interval′ el eu*) (*Interval′ fl fu*)
shows *checker a b c d e f*
apply (*rule checker-form*)
apply (*rule approx-form-aux*)
apply (*rule chk*[*unfolded checker′-def*])
using *assms*(*1−6*)
unfolding *bounded-by-def*
proof (*auto split*: *option.splits*)
fix *i x2*
assume ∗: [*Interval′ cl cu*, *Interval′ fl fu*, *Interval′ al au*, *Interval′ dl du*, *Interval′* (*Float 2 0*) (*Float 2 0*), *Interval′ bl bu*, *Interval′ el eu*, *Interval′ 0 0*] ! *i* = *Some x2*
assume *i* < *Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))))
then consider *i* = *0* | *i* = *1* | *i* = *2* | *i* = *3* | *i* = *4* | *i* = *5* | *i* = *6* | *i* = *7*
by *linarith*
thus [*c*, *f*, *a*, *d*, *2*, *b*, *e*, *0*] ! *i* ∈$_r$ *x2*
apply *cases* **using** *assms*(*1−6*) ∗
by (*auto intro*!: *in-real-intervalI dest*!: *Interval′-eq-Some*)
qed

lemma *approximate-soundness-correctness*:
assumes *a* ∈ {*real-of-float al* .. *real-of-float au*}
assumes *b* ∈ {*real-of-float bl* .. *real-of-float bu*}
assumes *c* ∈ {*real-of-float cl* .. *real-of-float cu*}
assumes *d* ∈ {*real-of-float dl* .. *real-of-float du*}
assumes *e* ∈ {*real-of-float el* .. *real-of-float eu*}
assumes *f* ∈ {*real-of-float fl* .. *real-of-float fu*}
assumes *chk*: *checker′ p* (*Interval′ al au*) (*Interval′ bl bu*) (*Interval′ cl cu*) (*Interval′ dl du*) (*Interval′ el eu*) (*Interval′ fl fu*)
shows *checker′-precond*: *check-precond a b c d e f*
and *checker′-no-collision*: *safe-distance.no-collision c b a f e d* {*0*..}
unfolding *atomize-conj*
apply (*subst soundness-correctness*[*symmetric*])
using *checker′-soundness-correctness*[*OF assms*]
by (*auto simp*: *checker-def Let-def*)

1.5.3 Symbolic Checker

definition *symbolic-checker* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
symbolic-checker s_e v_e a_e s_o v_o a_o ≡
let e-stop = − v_e / a_e;
o-stop = − v_o / a_o
in check-precond s_e v_e a_e s_o v_o a_o ∧
(¬*quadroot-in 0* (*min e-stop o-stop*) (*1/2* ∗ (a_o − a_e)) (v_o − v_e) (s_o − s_e) ∧
¬*quadroot-in e-stop o-stop* (*1/2* ∗ a_o) v_o (s_o − *movement.p* a_e v_e s_e *e-stop*) ∧
¬*quadroot-in o-stop e-stop* (*1/2* ∗ a_e) v_e (s_e − *movement.p* a_o v_o s_o *o-stop*))

theorem *symbolic-soundness-correctness*:
symbolic-checker s_e v_e a_e s_o v_o a_o $\longleftrightarrow$ *check-precond* s_e v_e a_e s_o v_o a_o $\wedge$ *safe-distance.no-collision* a_e v_e s_e a_o v_o s_o $\{0..\}$
proof −
{
assume *c*: *check-precond* s_e v_e a_e s_o v_o a_o
then interpret *safe-distance* a_e v_e s_e a_o v_o s_o
by (*simp add*: *check-precond-safe-distance*)
have *symbolic-checker* s_e v_e a_e s_o v_o a_o = *no-collision* $\{0..\}$
using *c*
unfolding *symbolic-checker symbolic-checker-def ego.s-t-stop other.s-t-stop ego.p-max-def other.p-max-def*
by (*auto simp*: *Let-def movement.t-stop-def*)
}
then show *?thesis*
by (*auto simp*: *symbolic-checker-def Let-def*)
qed
end

end

2 Safe Distance with Reaction Time

theory *Safe-Distance-Reaction*
imports
Safe-Distance
begin

2.1 Normal Safe Distance

locale *safe-distance-normal* = *safe-distance* +
fixes δ :: *real*
assumes *pos-react*: $0 < \delta$
begin

sublocale *ego2*: *braking-movement* a_e v_e (*ego.q* δ) **..**

lemma *ego2-s-init*: *ego2.s* 0 = *ego.q* δ **unfolding** *ego2.s-def* **by** *auto*

definition τ :: *real* ⇒ *real* **where**
$\tau\ t = t - \delta$

definition τ' :: *real* ⇒ *real* **where**
$\tau'\ t = 1$

lemma τ-*continuous*[*continuous-intros*]: *continuous-on T* τ
unfolding τ-*def* **by** (*auto intro*: *continuous-intros*)

lemma *isCont-*τ[*continuous-intros*]: *isCont* τ *x*

using τ-*continuous*[*of UNIV*] **by** (*auto simp*: *continuous-on-eq-continuous-at*)

lemma *del-has-vector-derivative*[*derivative-intros*]: (τ *has-vector-derivative* τ' *t*) (*at t within u*)
by (*auto simp*: τ-*def*[*abs-def*] τ'-*def has-vector-derivative-def algebra-simps intro*!: *derivative-eq-intros*)

lemma *del-has-real-derivative*[*derivative-intros*]: (τ *has-real-derivative* τ' *t*) (*at t within u*)
using *del-has-vector-derivative*
by (*simp add*:*has-real-derivative-iff-has-vector-derivative*)

lemma *delay-image*: τ ‘ $\{\delta..\} = \{0..\}$
proof (*rule subset-antisym*, *unfold image-def*, *unfold* τ-*def*)
show $\{y.\ \exists x{\in}\{\delta..\}.\ y = x - \delta\} \subseteq \{0..\}$ **by** *auto*
next
show $\{0..\} \subseteq \{y.\ \exists x{\in}\{\delta..\}.\ y = x - \delta\}$
proof (*rule subsetI*)
fix *a*
assume (*a*::*real*) $\in \{0..\}$
hence $0 \leq a$ **by** *simp*
hence $\exists x{\in}\{\delta..\}.\ a = x - \delta$ **using** *bexI*[**where** $x = a + \delta$] **by** *auto*
thus $a \in \{y.\ \exists x{\in}\{\delta..\}.\ y = x - \delta\}$ **by** *auto*
qed
qed

lemma *s-delayed-has-real-derivative*[*derivative-intros*]:
assumes $\delta \leq t$
shows ((*ego2.s* $\circ$ τ) *has-field-derivative ego2.s'* $(t - \delta) * \tau'$ *t*) (*at t within* $\{\delta..\}$)
proof (*rule DERIV-image-chain*)
from *assms* **have** *0*: $0 \leq t - \delta$ **by** *simp*
from *ego2.t-stop-nonneg* **have** *1*: v_e / $a_e \leq 0$ **unfolding** *ego2.t-stop-def* **by** *simp*
from *ego2.decel* **have** *2*: $a_e \neq 0$ **by** *simp*
show (*ego2.s has-real-derivative ego2.s'* $(t - \delta)$) (*at* (τ *t*) *within* τ ‘ $\{\delta..\}$)
using *ego2.s-has-real-derivative*[*OF 0 1 2*] *sym*[*OF delay-image*]
unfolding τ-*def* **by** *simp*
next
from *del-has-real-derivative* **show** (τ *has-real-derivative* τ' *t*) (*at t within* $\{\delta..\}$)
by *auto*
qed

lemma *s-delayed-has-real-derivative'* [*derivative-intros*]:
assumes $\delta \leq t$
shows ((*ego2.s* $\circ$ τ) *has-field-derivative* (*ego2.s'* $\circ$ τ) *t*) (*at t within* $\{\delta..\}$)
proof −
from *s-delayed-has-real-derivative*[*OF assms*] **have**
((*ego2.s* $\circ$ τ) *has-field-derivative ego2.s'* $(t - \delta) * \tau'$ *t*) (*at t within* $\{\delta..\}$)
by *auto*

hence $((ego2.s \circ \tau)$ *has-field-derivative* $ego2.s'\ (t - \delta) * 1)\ (at\ t\ within\ \{\delta..\})$
using τ'*-def*[*of t*] **by** *metis*
hence $((ego2.s \circ \tau)$ *has-field-derivative* $ego2.s'\ (t - \delta))\ (at\ t\ within\ \{\delta..\})$
by (*simp add*:*algebra-simps*)
thus *?thesis* **unfolding** *comp-def* τ*-def* **by** *auto*
qed

lemma *s-delayed-has-vector-derivative'* [*derivative-intros*]:
assumes $\delta \leq t$
shows $((ego2.s \circ \tau)$ *has-vector-derivative* $(ego2.s' \circ \tau)\ t)\ (at\ t\ within\ \{\delta..\})$
using *s-delayed-has-real-derivative'*[*OF assms*]
by (*simp add*:*has-real-derivative-iff-has-vector-derivative*)

definition $u :: real \Rightarrow real$ **where**
$u\ t = (\quad if\ t \leq 0\ then\ s_e$
$\quad else\ if\ t \leq \delta\ then\ ego.q\ t$
$\quad else\quad (ego2.s \circ \tau)\ t)$

lemma *init-u*: $t \leq 0 \Longrightarrow u\ t = s_e$ **unfolding** *u-def* **by** *auto*

lemma *u-delta*: $u\ \delta = ego2.s\ 0$
proof −
have $u\ \delta = ego.q\ \delta$ **using** *pos-react* **unfolding** *u-def* **by** *auto*
also have $... = ego2.s\ 0$ **unfolding** *ego2.s-def* **by** *auto*
finally show $u\ \delta = ego2.s\ 0$ **.**
qed

lemma *q-delta*: $ego.q\ \delta = ego2.s\ 0$ **using** *u-delta pos-react* **unfolding** *u-def* **by** *auto*

definition $u' :: real \Rightarrow real$ **where**
$u'\ t = (if\ t \leq \delta\ then\ ego.q'\ t\ else\ ego2.s'\ (t - \delta))$

lemma u'*-delta*: $u'\ \delta = ego2.s'\ 0$
proof −
have $u'\ \delta = ego.q'\ \delta$ **unfolding** u'*-def* **by** *auto*
also have $... = v_e$ **unfolding** $ego2.q'$*-def* **by** *simp*
also have $... = ego2.p'\ 0$ **unfolding** $ego2.p'$*-def* **by** *simp*
also have $... = ego2.s'\ 0$ **using** *ego2.t-stop-nonneg* **unfolding** $ego2.s'$*-def* **by** *auto*
finally show $u'\ \delta = ego.s'\ 0$ **.**
qed

lemma q'*-delta*: $ego.q'\ \delta = ego2.s'\ 0$ **using** u'*-delta* **unfolding** u'*-def* **by** *auto*

lemma *u-has-real-derivative*[*derivative-intros*]:
assumes *nonneg-t*: $t \geq 0$
shows $(u$ *has-real-derivative* $u'\ t)\ (at\ t\ within\ \{0..\})$
proof −

from *pos-react* **have** $0 \leq \delta$ **by** *simp*

have *temp*: $((\lambda t.\ if\ t \in \{0\ ..\ \delta\}\ then\ ego.q\ t\ else\ (ego2.s \circ \tau)\ t)\ has\text{-}real\text{-}derivative$
$(if\ t \in \{0..\delta\}\ then\ ego.q'\ t\ else\ (ego2.s' \circ \tau)\ t))\ (at\ t\ within\ \{0..\})$ (**is** $(?f1\ has\text{-}real\text{-}derivative\ ?f2)\ (?net)$)
unfolding *u-def*[*abs-def*] *u'-def*
has-real-derivative-iff-has-vector-derivative
apply (*rule has-vector-derivative-If-within-closures*[**where** $T = \{\delta..\}$])
using $\langle 0 \leq \delta \rangle$ *q-delta q'-delta ego.s-has-vector-derivative*[*OF assms*] *ego.decel ego.t-stop-nonneg*
s-delayed-has-vector-derivative'[*of t*] *τ-def*
unfolding *comp-def*
by (*auto simp*: *assms max-def insert-absorb*
intro!: *ego.q-has-vector-derivative*)
show *?thesis*
unfolding *has-vector-derivative-def has-real-derivative-iff-has-vector-derivative*
u'-def u-def[*abs-def*]
proof (*rule has-derivative-transform*[**where** $f=(\lambda t.\ if\ t \in \{0..\delta\}\ then\ ego.q\ t\ else\ (ego2.s \circ \tau)\ t)$])
from *nonneg-t* **show** $t \in \{0..\}$ **by** *auto*
next
fix *x*
assume $(x::real) \in \{0..\}$
hence $x \leq \delta \longleftrightarrow x \in \{0\ ..\ \delta\}$ **by** *simp*
thus $(if\ x \leq 0\ then\ s_e\ else\ if\ x \leq \delta\ then\ ego.q\ x\ else\ (ego2.s \circ \tau)\ x) =$
$(if\ x \in \{0..\delta\}\ then\ ego.q\ x\ else\ (ego2.s \circ \tau)\ x)$ **using** *pos-react* **unfolding** *ego.q-def* **by** *auto*
next
from *temp* **have** $(?f1\ has\text{-}vector\text{-}derivative\ ?f2)\ ?net$
using *has-real-derivative-iff-has-vector-derivative* **by** *auto*
moreover with *assms* **have** $t \in \{0\ ..\ \delta\} \longleftrightarrow t \leq \delta$ **by** *auto*
ultimately show $((\lambda t.\ if\ t \in \{0..\delta\}\ then\ ego.q\ t\ else\ (ego2.s \circ \tau)\ t)\ has\text{-}derivative$
$(\lambda x.\ x *_R (if\ t \leq \delta\ then\ ego2.q'\ t\ else\ ego2.s'\ (t - \delta))))\ (at\ t\ within\ \{0..\})$
unfolding *comp-def τ-def has-vector-derivative-def* **by** *auto*
qed
qed

definition *t-stop* :: *real* **where**
t-stop = *ego2.t-stop* + δ

lemma *t-stop-nonneg*: $0 \leq$ *t-stop*
unfolding *t-stop-def*
using *ego2.t-stop-nonneg pos-react*
by *auto*

lemma *t-stop-pos*: $0 <$ *t-stop*
unfolding *t-stop-def*

using *ego2.t-stop-nonneg pos-react*
by *auto*

lemma *t-stop-zero*:
assumes $t\text{-}stop \leq x$
assumes $x \leq \delta$
shows $v_e = 0$
using *assms* **unfolding** *t-stop-def* **using** *ego2.t-stop-nonneg pos-react ego2.t-stop-zero*
by *auto*

lemma *u'-stop-zero*: $u'\ t\text{-}stop = 0$
unfolding *u'-def t-stop-def ego2.q'-def ego2.s'-def*
using *ego2.t-stop-nonneg ego2.p'-stop-zero decelerate-ego ego2.t-stop-zero* **by** *auto*

definition *u-max* :: *real* **where**
$u\text{-}max = u\ (ego2.t\text{-}stop + \delta)$

lemma *u-max-eq*: $u\text{-}max = ego.q\ \delta - {v_e}^2\ /\ a_e\ /\ 2$
proof (*cases ego2.t-stop = 0*)
assume *ego2.t-stop = 0*
hence $v_e = 0$ **using** *ego2.t-stop-zero* **by** *simp*
with ‹*ego2.t-stop = 0*› **show** $u\text{-}max = ego.q\ \delta - {v_e}^2\ /\ a_e\ /\ 2$ **unfolding** *u-max-def u-def* **using** *pos-react* **by** *auto*
next
assume *ego2.t-stop ≠ 0*
hence $u\text{-}max = (ego2.s \circ \tau)\ (ego2.t\text{-}stop + \delta)$
unfolding *u-max-def u-def* **using** *ego2.t-stop-nonneg pos-react* **by** *auto*
moreover have ... = *ego2.s ego2.t-stop* **unfolding** *comp-def τ-def* **by** *auto*
moreover have ... = *ego2.p-max*
unfolding *ego2.s-def ego2.p-max-def* **using** ‹*ego2.t-stop ≠ 0*› *ego2.t-stop-nonneg*
by *auto*
moreover have ... = $ego.q\ \delta - {v_e}^2\ /\ a_e\ /\ 2$ **using** *ego2.p-max-eq* **.**
ultimately show *?thesis* **by** *auto*
qed

lemma *u-mono*:
assumes $x \leq y\ y \leq t\text{-}stop$
shows $u\ x \leq u\ y$
proof −
have $y \leq 0 \vee (0 < y \wedge y \leq \delta) \vee \delta < y$ **by** *auto*

moreover
{ assume $y \leq 0$
with *assms* **have** $x \leq 0$ **by** *auto*
with ‹$y \leq 0$› **have** $u\ x \leq u\ y$ **unfolding** *u-def* **by** *auto* **}**

moreover
{ assume $0 < y \wedge y \leq \delta$

```
    with assms have x ≤ δ by auto
    hence u x ≤ u y
    proof (cases x ≤ 0)
      assume x ≤ 0
      with ‹x ≤ δ› and ‹0 < y ∧ y ≤ δ› show u x ≤ u y  unfolding u-def using
ego.q-min by auto
    next
      assume ¬ x ≤ 0
      with ‹0 < y ∧ y ≤ δ› and assms show u x ≤ u y
        unfolding u-def  using ego.q-mono by auto
    qed }

  moreover
  { assume δ < y
    have u x ≤ u y
    proof (cases δ < x)
      assume δ < x
      with pos-react have ¬ x ≤ 0 by auto
      moreover from ‹δ < y› and pos-react have ¬ y ≤ 0 by auto
      ultimately show u x ≤ u y  unfolding u-def comp-def
          using assms ego2.s-mono[of x − δ y − δ] ‹δ < y› ‹δ < x› by (auto
simp:τ-def)
    next
      assume ¬ δ < x
      hence x ≤ δ by simp
      hence u x ≤ ego.q δ unfolding u-def using pos-react nonneg-vel-ego
        by (auto simp add:ego.q-def mult-left-mono)
      also have ... = ego2.s (τ δ) unfolding ego2.s-def unfolding τ-def by auto
      also have ... ≤ ego2.s (τ y) unfolding τ-def using ‹δ < y› by (auto simp
add:ego2.s-mono)
      also have ... = u y unfolding u-def using ‹δ < y› pos-react by auto
      ultimately show u x ≤ u y by auto
    qed }

  ultimately show u x ≤ u y by auto
qed

lemma u-antimono: x ≤ y ⟹ t-stop ≤ x ⟹ u y ≤ u x
proof −
  assume 1: x ≤ y
  assume 2: t-stop ≤ x
  hence δ ≤ x unfolding τ-def t-stop-def using pos-react ego2.t-stop-nonneg by
auto
  with 1 have δ ≤ y by auto
  from 1 and 2 have 3: t-stop ≤ y by auto
  show u y ≤ u x
  proof (cases x ≠ δ ∧ y ≠ δ)
    assume x ≠ δ ∧ y ≠ δ
    hence x ≠ δ and y ≠ δ by auto
```

```
    have u y ≤ (ego2.s ∘ τ) y unfolding u-def using ‹δ ≤ y› ‹y ≠ δ› pos-react
by auto
    also have ... ≤ (ego2.s ∘ τ) x unfolding comp-def
    proof (intro ego2.s-antimono)
      show τ x ≤ τ y unfolding τ-def using ‹x ≤ y› by auto
    next
      show ego2.t-stop ≤ τ x unfolding τ-def using ‹t-stop ≤ x› by (auto simp:
t-stop-def)
    qed
    also have ... ≤ u x unfolding u-def using ‹δ ≤ x›‹x ≠ δ› pos-react by auto
    ultimately show u y ≤ u x by auto
  next
    assume ¬ (x ≠ δ ∧ y ≠ δ)
    have x ≠ δ ⟶ y ≠ δ
    proof (rule impI; erule contrapos-pp[where Q=¬ x = δ])
      assume ¬ y ≠ δ
      hence y = δ by simp
      with ‹t-stop ≤ y› have ego2.t-stop = 0 unfolding t-stop-def
        using ego2.t-stop-nonneg by auto
      with ‹t-stop ≤ x› have x = δ unfolding t-stop-def using ‹x ≤ y› ‹y = δ›
by auto
      thus ¬ x ≠ δ by auto
    qed
    with ‹¬ (x ≠ δ ∧ y ≠ δ)› have (x = δ ∧ y = δ) ∨ (x = δ) by auto

    moreover
    { assume x = δ ∧ y = δ
      hence x = δ and y = δ by auto
      hence u y ≤ ego.q δ unfolding u-def using pos-react by auto
      also have ... ≤ u x unfolding u-def using ‹x = δ› pos-react by auto
      ultimately have u y ≤ u x by auto }

    moreover
    { assume x = δ
        hence ego2.t-stop = 0 using ‹t-stop ≤ x› ego2.t-stop-nonneg by (auto
simp:t-stop-def)
      hence v_e = 0 by (rule ego2.t-stop-zero)
      hence u y ≤ ego.q δ
        using pos-react ‹x = δ› ‹x ≤ y› ‹v_e = 0›
          unfolding u-def comp-def τ-def ego2.s-def ego2.p-def ego2.p-max-def
ego2.t-stop-def
        by auto
      also have ... ≤ u x using ‹x = δ› pos-react unfolding u-def by auto
      ultimately have u y ≤ u x by auto }

    ultimately show ?thesis by auto
  qed
qed
```

lemma *u-max*: $u\ x \leq$ *u-max*
unfolding *u-max-def* **using** *t-stop-def*
by (*cases* $x \leq$ *t-stop*) (*auto intro*: *u-mono u-antimono*)

lemma *u-eq-u-stop*: *NO-MATCH t-stop* $x \Longrightarrow x \geq$ *t-stop* $\Longrightarrow u\ x =$ *u-max*
proof −
assume *t-stop* $\leq x$
with *t-stop-pos* **have** $0 < x$ **by** *auto*
from ‹*t-stop* $\leq x$› **have** $\delta \leq x$ **unfolding** *t-stop-def* **using** *ego2.t-stop-nonneg* **by** *auto*
show $u\ x =$ *u-max*
proof (*cases* $x \leq \delta$)
assume $x \leq \delta$
with ‹*t-stop* $\leq x$› **have** $v_e = 0$ **by** (*rule t-stop-zero*)
also have $x = \delta$ **using** ‹$x \leq \delta$› **and** ‹$\delta \leq x$› **by** *auto*
ultimately have $u\ x =$ *ego.q* δ **unfolding** *u-def* **using** *pos-react* **by** *auto*
also have ... = *u-max* **unfolding** *u-max-eq* **using** ‹$v_e = 0$› **by** *auto*
ultimately show $u\ x =$ *u-max* **by** *simp*
next
assume $\neg\ x \leq \delta$
hence $\delta < x$ **by** *auto*
hence $u\ x = (ego2.s \circ \tau)\ x$ **unfolding** *u-def* **using** *pos-react* **by** *auto*
also have ... = *ego2.s ego2.t-stop*
proof (*unfold comp-def*; *unfold* τ*-def*; *intro order.antisym*)
have $x - \delta \geq$ *ego2.t-stop* **using** ‹*t-stop* $\leq x$› **unfolding** *t-stop-def* **by** *auto*
thus *ego2.s* $(x - \delta) \leq$ *ego2.s ego2.t-stop* **by** (*rule ego2.s-antimono*) *simp*
next
have $x - \delta \geq$ *ego2.t-stop* **using** ‹*t-stop* $\leq x$› **unfolding** *t-stop-def* **by** *auto*
thus *ego2.s ego2.t-stop* $\leq$ *ego2.s* $(x - \delta)$ **using** *ego2.t-stop-nonneg* **by** (*rule ego2.s-mono*)
qed
also have ... = *u-max* **unfolding** *u-max-eq ego2.s-t-stop ego2.p-max-eq* **by** *auto*
ultimately show $u\ x =$ *u-max* **by** *auto*
qed
qed

lemma *at-least-delta*:
assumes $x \leq \delta$
assumes *t-stop* $\leq x$
shows *ego.q* $x =$ *ego2.s* $(x - \delta)$
using *assms ego2.t-stop-nonneg*
unfolding *t-stop-def ego2.s-def less-eq-real-def* **by** *auto*

lemma *continuous-on-u*[*continuous-intros*]: *continuous-on T u*
unfolding *u-def*[*abs-def*]
using *t-stop-nonneg pos-react at-least-delta*
proof (*intro continuous-on-subset*[**where** *t*=*T* **and** $s = \{..0\} \cup (\{0..\delta\} \cup (\{\delta\ ..\ t\text{-}stop\} \cup \{t\text{-}stop\ ..\}))$] *continuous-on-If continuous-intros*)

fix x
assume $\neg\ x \leq \delta$
assume $x \in \{0..\delta\}$
hence $0 \leq x$ **and** $x \leq \delta$ **by** *auto*
thus *ego.q* $x = (ego2.s \circ \tau)\ x$
unfolding *comp-def* τ*-def ego2.s-def*
using $\langle \neg\ x \leq \delta \rangle$ **by** *auto*
next
fix x
assume $x \in \{\delta..t\text{-}stop\} \cup \{t\text{-}stop..\}$
hence $\delta \leq x$ **unfolding** *t-stop-def* **using** *pos-react ego.t-stop-nonneg* **by** *auto*
also assume $x \leq \delta$
ultimately have $x = \delta$ **by** *auto*
thus *ego.q* $x = (ego2.s \circ \tau)\ x$ **unfolding** *comp-def* τ*-def ego2.s-def* **by** *auto*
next
fix t::*real*
assume $t \in \{..\ 0\}$
hence $t \leq 0$ **by** *auto*
also assume $\neg\ t \leq 0$
ultimately have $t = 0$ **by** *auto*
hence $s_e = ego.q\ t$ **unfolding** *ego.q-def* **by** *auto*
with *pos-react* $\langle t = 0 \rangle$ **show** $s_e = (if\ t \leq \delta\ then\ ego.q\ t\ else\ (ego2.s \circ \tau)\ t)$ **by** *auto*
next
fix t::*real*
assume $t \in \{0..\delta\} \cup (\{\delta..t\text{-}stop\} \cup \{t\text{-}stop..\})$
hence $0 \leq t$ **using** *pos-react ego2.t-stop-nonneg* **by** (*auto simp*: *t-stop-def*)
also assume $t \leq 0$
ultimately have $t = 0$ **by** *auto*
hence $s_e = (if\ t \leq \delta\ then\ ego.q\ t\ else\ (ego2.s \circ \tau)\ t)$ **using** *pos-react ego.init-q* **by** *auto*
thus $s_e = (if\ t \leq \delta\ then\ ego.q\ t\ else\ (ego2.s \circ \tau)\ t)$ **by** *auto*
next
show $T \subseteq \{..0\} \cup (\{0..\delta\} \cup (\{\delta..t\text{-}stop\} \cup \{t\text{-}stop..\}))$ **by** *auto*
qed

lemma *isCont-u*[*continuous-intros*]: *isCont u x*
using *continuous-on-u*[*of UNIV*]
by (*auto simp*:*continuous-on-eq-continuous-at*)

definition *collision-react* :: *real set* ⇒ *bool* **where**
collision-react time-set ≡ $(\exists\, t{\in}$*time-set*. *u t* = *other.s t*)

abbreviation *no-collision-react* :: *real set* ⇒ *bool* **where**
no-collision-react time-set ≡ ¬ *collision-react time-set*

lemma *no-collision-reactI*:
assumes $\bigwedge t.\ t \in S \Longrightarrow u\ t \neq other.s\ t$
shows *no-collision-react S*

using *assms*
unfolding *collision-react-def*
by *blast*

lemma *no-collision-union*:
assumes *no-collision-react S*
assumes *no-collision-react T*
shows *no-collision-react* $(S \cup T)$
using *assms*
unfolding *collision-react-def*
by *auto*

lemma *collision-trim-subset*:
assumes *collision-react S*
assumes *no-collision-react T*
assumes $T \subseteq S$
shows *collision-react* $(S - T)$
using *assms*
unfolding *collision-react-def* **by** *auto*

theorem *cond-1r* : *u-max* $< s_o \Longrightarrow$ *no-collision-react* $\{0..\}$
proof (*rule no-collision-reactI*, *simp*)
fix *t* :: *real*
assume $0 \leq t$
have *u t* $\leq$ *u-max* **by** (*rule u-max*)
also assume ... $< s_o$
also have ... = *other.s 0*
by (*simp add*: *other.init-s*)
also have ... $\leq$ *other.s t*
using ‹$0 \leq t$› *hyps*
by (*intro other.s-mono*) *auto*
finally show *u t* $\neq$ *other.s t*
by *simp*
qed

definition *safe-distance-1r* :: *real* **where**
safe-distance-1r = $v_e * \delta - {v_e}^2$ / a_e / *2*

lemma *sd-1r-eq*: $(s_o - s_e >$ *safe-distance-1r*$) = ($*u-max* $< s_o)$
proof −
have $(s_o - s_e >$ *safe-distance-1r*$) = (s_o - s_e > v_e * \delta - {v_e}^2$ / a_e / *2*$)$ **unfolding** *safe-distance-1r-def* **by** *auto*
moreover have ... $= (s_e + v_e * \delta - {v_e}^2$ / a_e / *2* $< s_o)$ **by** *auto*
ultimately show *?thesis* **using** *u-max-eq ego.q-def* **by** *auto*
qed

lemma *sd-1r-correct*:
assumes $s_o - s_e >$ *safe-distance-1r*
shows *no-collision-react* $\{0..\}$

proof −
from *assms* **have** *u-max* < s_o **using** *sd-1r-eq* **by** *auto*
thus *?thesis* **by** (*rule cond-1r*)
qed

lemma *u-other-strict-ivt*:
assumes *u t* > *other.s t*
shows *collision-react* {*0*..<*t*}
proof *cases*
assume *0* ≤ *t*
with *assms in-front*
have ∃ *x*≥*0*. *x* ≤ *t* ∧ *other.s x* − *u x* = *0*
by (*intro IVT2*) (*auto simp*: *init-u other.init-s continuous-diff isCont-u other.isCont-s*)
then show *?thesis*
using *assms*
by (*auto simp add*: *algebra-simps collision-react-def Bex-def order.order-iff-strict*)
qed(*insert assms hyps*, *auto simp*: *collision-react-def init-u other.init-s*)

lemma *collision-react-subset*: *collision-react s* ⟹ *s* ⊆ *t* ⟹ *collision-react t*
by (*auto simp*:*collision-react-def*)

lemma *u-other-ivt*:
assumes *u t* ≥ *other.s t*
shows *collision-react* {*0* .. *t*}
proof *cases*
assume *u t* > *other.s t*
from *u-other-strict-ivt*[*OF this*]
show *?thesis*
by (*rule collision-react-subset*) *auto*
qed (*insert hyps assms*; *cases t* ≥ *0*; *force simp*: *collision-react-def init-u other.init-s*)

theorem *cond-2r*:
assumes *u-max* ≥ *other.s-stop*
shows *collision-react* {*0* ..}
using *assms*
apply(*intro collision-react-subset*[**where** *t*={*0*..} **and** *s* ={*0* .. *max t-stop other.t-stop*}])
apply(*intro u-other-ivt*[**where** *t* =*max t-stop other.t-stop*])
apply(*auto simp*: *u-eq-u-stop other.s-eq-s-stop*)
done

definition *ego-other2* :: *real* ⇒ *real* **where**
ego-other2 t = *other.s t* − *u t*

lemma *continuous-on-ego-other2*[*continuous-intros*]: *continuous-on T ego-other2*
unfolding *ego-other2-def*[*abs-def*]
by (*intro continuous-intros*)

lemma *isCont-ego-other2*[*continuous-intros*]: *isCont ego-other2 x*
using *continuous-on-ego-other2*[*of UNIV*]

by (*auto simp*: *continuous-on-eq-continuous-at*)

definition *ego-other2'* :: *real* ⇒ *real* **where**
ego-other2' t = *other.s' t* − *u' t*

lemma *ego-other2-has-real-derivative*[*derivative-intros*]:
assumes *0* ≤ *t*
shows (*ego-other2 has-real-derivative ego-other2' t*) (*at t within* {*0*..})
using *assms other.t-stop-nonneg decelerate-other*
unfolding *other.t-stop-def*
by (*auto simp*: *ego-other2-def*[*abs-def*] *ego-other2'-def algebra-simps*
intro!: *derivative-eq-intros*)

theorem *cond-3r-1*:
assumes *u δ* ≥ *other.s δ*
shows *collision-react* {*0* .. *δ*}
proof (*unfold collision-react-def*)
have *1*: ∃ *t*≥*0*. *t* ≤ *δ* ∧ *ego-other2 t* = *0*
proof (*intro IVT2*)
show *ego-other2 δ* ≤ *0* **unfolding** *ego-other2-def* **using** *assms* **by** *auto*
next
show *0* ≤ *ego-other2 0* **unfolding** *ego-other2-def*
using *other.init-s*[*of 0*] *init-u*[*of 0*] *in-front* **by** *auto*
next
show *0* ≤ *δ* **using** *pos-react* **by** *auto*
next
show ∀ *t*. *0* ≤ *t* ∧ *t* ≤ *δ* ⟶ *isCont ego-other2 t*
using *isCont-ego-other2* **by** *auto*
qed
then obtain *t* **where** *0* ≤ *t* ∧ *t* ≤ *δ* ∧ *ego-other2 t* = *0* **by** *auto*
hence *t* ∈ {*0* .. *δ*} **and** *u t* = *other.s t* **unfolding** *ego-other2-def* **by** *auto*
thus ∃ *t*∈{*0*..*δ*}. *u t* = *other.s t* **by** (*intro bexI*)
qed

definition *distance0* :: *real* **where**
distance0 = $v_e * \delta - v_o * \delta - a_o * \delta^2 / 2$

definition *distance0-2* :: *real* **where**
distance0-2 = $v_e * \delta + 1 / 2 * {v_o}^2 / a_o$

theorem *cond-3r-1'*:
assumes $s_o - s_e \leq$ *distance0*
assumes *δ* ≤ *other.t-stop*
shows *collision-react* {*0* .. *δ*}
proof −
from *assms* **have** *u δ* ≥ *other.s δ* **unfolding** *distance0-def other.s-def*
other.p-def u-def ego.q-def **using** *pos-react* **by** *auto*
thus *?thesis* **using** *cond-3r-1* **by** *auto*
qed

theorem *distance0-2-eq*:
assumes $\delta > other.t\text{-}stop$
shows $(u\ \delta < other.s\ \delta) = (s_o - s_e > distance0\text{-}2)$
proof −
from *assms* **have** $(u\ \delta < other.s\ \delta) = (ego.q\ \delta < other.p\text{-}max)$
using *u-def other.s-def pos-react* **by** *auto*
also have ... $= (s_e + v_e * \delta < s_o + v_o * (-\ v_o\ /\ a_o) + 1\ /\ 2 * a_o * (-\ v_o\ /\ a_o)^2)$
using *ego.q-def other.p-max-def other.p-def other.t-stop-def* **by** *auto*
also have ... $= (v_e * \delta - v_o * (-\ v_o\ /\ a_o) - 1\ /\ 2 * a_o * (-\ v_o\ /\ a_o)^2 < s_o - s_e)$ **by** *linarith*
also have ... $= (v_e * \delta + {v_o}^2\ /\ a_o - 1\ /\ 2 * {v_o}^2\ /\ a_o < s_o - s_e)$
using *other.p-def other.p-max-def other.p-max-eq other.t-stop-def* **by** *auto*
also have ... $= (v_e * \delta + 1\ /\ 2 * {v_o}^2\ /\ a_o < s_o - s_e)$ **by** *linarith*
thus *?thesis* **using** *distance0-2-def* **by** (*simp add*: *calculation*)
qed

theorem *cond-3r-1'-2*:
assumes $s_o - s_e \leq distance0\text{-}2$
assumes $\delta > other.t\text{-}stop$
shows *collision-react* $\{0\ ..\ \delta\}$
proof −
from *assms distance0-2-eq* **have** $u\ \delta \geq other.s\ \delta$ **unfolding** *distance0-def other.s-def other.p-def u-def ego.q-def* **using** *pos-react* **by** *auto*
thus *?thesis* **using** *cond-3r-1* **by** *auto*
qed

definition *safe-distance-3r* :: *real* **where**
$safe\text{-}distance\text{-}3r = v_e * \delta - {v_e}^2\ /\ 2\ /\ a_e - v_o * \delta - 1/2 * a_o * \delta^2$

lemma *distance0-at-most-sd3r*:
$distance0 \leq safe\text{-}distance\text{-}3r$
unfolding *distance0-def safe-distance-3r-def* **using** *nonneg-vel-ego decelerate-ego*
by (*auto simp add*:*field-simps*)

definition *safe-distance-4r* :: *real* **where**
$safe\text{-}distance\text{-}4r = (v_o + a_o * \delta - v_e)^2\ /\ 2\ /\ (a_o - a_e) - v_o * \delta - 1/2 * a_o * \delta^2 + v_e * \delta$

lemma *distance0-at-most-sd4r*:
assumes $a_o > a_e$
shows $distance0 \leq safe\text{-}distance\text{-}4r$
proof −
from *assms* **have** $a_o \geq a_e$ **by** *auto*
have $0 \leq (v_o + a_o * \delta - v_e)^2\ /\ (2 * a_o - 2 * a_e)$
by (*rule divide-nonneg-nonneg*) (*auto simp add*:*assms* ‹$a_e \leq a_o$›)
thus *?thesis* **unfolding** *distance0-def safe-distance-4r-def*
by *auto*

qed

definition *safe-distance-2r* :: *real* **where**
safe-distance-2r = $v_e * \delta - {v_e}^2 \;/\; 2 \;/\; a_e + {v_o}^2 \;/\; 2 \;/\; a_o$

lemma *vo-start-geq-ve*:
assumes $\delta \leq$ *other.t-stop*
assumes *other.s'* $\delta \geq v_e$
shows *u* $\delta <$ *other.s* δ
proof −
from *assms* **have** $v_e \leq v_o + a_o * \delta$ **unfolding** *other.s'-def other.p'-def* **by** *auto*
with *mult-right-mono*[*OF this*, *of* δ] **have** $v_e * \delta \leq v_o * \delta + a_o * \delta^2$ (**is** *?l0* ≤ *?r0*)
using *pos-react* **by** (*auto simp add*:*field-simps power-def*)
hence s_e + *?l0* ≤ s_e + *?r0* **by** *auto*
also have ... < s_o + *?r0* **using** *in-front* **by** *auto*
also have ... < $s_o + v_o * \delta + a_o * \delta^2 \;/\; 2$ **using** *decelerate-other pos-react* **by** *auto*
finally show *?thesis* **using** *pos-react assms*(*1*)
unfolding *u-def ego.q-def other.s-def other.t-stop-def other.p-def* **by** *auto*
qed

theorem *so-star-stop-leq-se-stop*:
assumes $\delta \leq$ *other.t-stop*
assumes *other.s'* $\delta < v_e$
assumes ¬ ($a_o > a_e \wedge$ *other.s'* $\delta < v_e \wedge v_e - a_e \;/\; a_o *$ *other.s'* $\delta < 0$)
shows $0 \leq - {v_e}^2 \;/\; a_e \;/\; 2 + (v_o + a_o * \delta)^2 \;/\; a_o \;/\; 2$
proof −
consider $v_e - a_e \;/\; a_o *$ *other.s'* $\delta \geq 0 \mid \neg\, (v_e - a_e \;/\; a_o *$ *other.s'* $\delta \geq 0)$ **by** *auto*
thus *?thesis*
proof (*cases*)
case *1*
hence $v_e - a_e \;/\; a_o * (v_o + a_o * \delta) \geq 0$ **unfolding** *other.s'-def other.p'-def*
by (*auto simp add*:*assms*(*1*))
hence $v_e - a_e \;/\; a_o * v_o - a_e * \delta \geq 0$ (**is** *?l0* ≥ *0*) **using** *decelerate-other*
by (*auto simp add*:*field-simps*)
hence *?l0* / $a_e \leq 0$ **using** *divide-right-mono-neg*[*OF* ‹*?l0* ≥ *0*›] *decelerate-ego* **by** *auto*
hence $0 \geq v_e \;/\; a_e - v_o \;/\; a_o - \delta$ **using** *decelerate-ego* **by** (*auto simp add*:*field-simps*)
hence ∗: $- v_e \;/\; a_e \geq - (v_o + a_o * \delta) \;/\; a_o$ **using** *decelerate-other* **by** (*auto simp add*:*field-simps*)
from *assms* **have** ∗∗: $v_o + a_o * \delta \leq v_e$ **unfolding** *other.s'-def other.p'-def* **by** *auto*
have *vo-star-nneg*: $v_o + a_o * \delta \geq 0$
proof −
from *assms*(*1*) **have** − $v_o \leq a_o * \delta$ **unfolding** *other.t-stop-def* **using**

decelerate-other
by (*auto simp add*:*field-simps*)
thus *?thesis* **by** *auto*
qed
from *mult-mono*[*OF* * ** - ‹$0 \leq v_o + a_o * \delta$›]
have $-(v_o + a_o * \delta) / a_o * (v_o + a_o * \delta) \leq - v_e / a_e * v_e$ **using** *nonneg-vel-ego decelerate-ego*
by (*auto simp add*:*field-simps*)
hence $-(v_o + a_o * \delta)^2 / a_o \leq - {v_e}^2 / a_e$ **by** (*auto simp add*: *field-simps power-def*)
thus *?thesis* **by** (*auto simp add*:*field-simps*)
next
case *2*
with *assms* **have** $a_o \leq a_e$ **by** *auto*
from *assms*(*2*) **have** $(v_o + a_o * \delta) \leq v_e$ **unfolding** *other.s'-def* **using** *assms* **unfolding** *other.p'-def*
by *auto*
have *vo-star-nneg*: $v_o + a_o * \delta \geq 0$
proof −
from *assms*(*1*) **have** $- v_o \leq a_o * \delta$ **unfolding** *other.t-stop-def* **using** *decelerate-other*
by (*auto simp add*:*field-simps*)
thus *?thesis* **by** *auto*
qed
with *mult-mono*[*OF* ‹$v_o + a_o * \delta \leq v_e$› ‹$v_o + a_o * \delta \leq v_e$›] **have** *: $(v_o + a_o * \delta)^2 \leq {v_e}^2$
using *nonneg-vel-ego* **by** (*auto simp add*:*power-def*)
from ‹$a_o \leq a_e$› **have** $-1 / a_o \leq -1 / a_e$ **using** *decelerate-ego decelerate-other*
by (*auto simp add*:*field-simps*)
from *mult-mono*[*OF* * *this*] **have** $(v_o + a_o * \delta)^2 * (-1 / a_o) \leq {v_e}^2 * (-1 / a_e)$
using *nonneg-vel-ego decelerate-other* **by** (*auto simp add*:*field-simps*)
then show *?thesis* **by** *auto*
qed
qed

theorem *distance0-at-most-distance2r*:
assumes $\delta \leq$ *other.t-stop*
assumes *other.s'* $\delta < v_e$
assumes $\neg (a_o > a_e \wedge$ *other.s'* $\delta < v_e \wedge v_e - a_e / a_o *$ *other.s'* $\delta < 0)$
shows *distance0* $\leq$ *safe-distance-2r*
proof −
from *so-star-stop-leq-se-stop*[*OF assms*] **have** $0 \leq - {v_e}^2 / a_e / 2 + (v_o + a_o * \delta)^2 / a_o / 2$ (**is** $0 \leq$ *?term*)
by *auto*
have *safe-distance-2r* $= v_e * \delta - {v_e}^2 / 2 / a_e + {v_o}^2 / 2 / a_o$ **unfolding** *safe-distance-2r-def* **by** *auto*
also have ... $= v_e * \delta - {v_e}^2 / 2 / a_e + (v_o + a_o * \delta)^2 / 2 / a_o - v_o * \delta - a_o * \delta^2 / 2$

using *decelerate-other* **by** (*auto simp add:field-simps power-def*)
also have ... = $v_e * \delta - v_o * \delta - a_o * \delta^2$ / 2 + *?term* (**is** - = *?left* + *?term*)
by (*auto simp add:field-simps*)
finally have *safe-distance-2r* = *distance0* + *?term* **unfolding** *distance0-def* **by** *auto*
with ‹*0* ≤ *?term*› **show** *distance0* ≤ *safe-distance-2r* **by** *auto*
qed

theorem *dist0-sd2r-1*:
assumes $\delta \leq$ *other.t-stop*
assumes $\neg$ ($a_o > a_e \wedge$ *other.s'* $\delta < v_e \wedge v_e - a_e$ / $a_o *$ *other.s'* $\delta < 0$)
assumes $s_o - s_e >$ *safe-distance-2r*
shows $s_o - s_e >$ *distance0*
proof (*cases other.s'* $\delta \geq v_e$)
assume $v_e \leq$ *other.s'* δ
from *vo-start-geq-ve*[*OF assms*(*1*) *this*] **have** *u* $\delta <$ *other.s* δ **by** *auto*
thus *?thesis* **unfolding** *distance0-def u-def* **using** *pos-react assms*(*1*) **unfolding** *ego.q-def*
other.s-def other.p-def **by** *auto*
next
assume $\neg$ $v_e \leq$ *other.s'* δ
hence $v_e >$ *other.s'* δ **by** *auto*
from *distance0-at-most-distance2r*[*OF assms*(*1*) *this assms*(*2*)] **have** *distance0* ≤ *safe-distance-2r*
by *auto*
with *assms*(*3*) **show** *?thesis* **by** *auto*
qed

theorem *sd2r-eq*:
assumes $\delta >$ *other.t-stop*
shows (*u-max* < *other.s* δ) = ($s_o - s_e >$ *safe-distance-2r*)
proof −
from *assms* **have** (*u-max* < *other.s* δ) = (*ego2.s* (− v_e / a_e) < *other.p-max*)
using *u-max-def ego2.t-stop-def u-def other.s-def* τ*-def pos-react ego2.p-max-eq ego2.s-t-stop u-max-eq* **by** *auto*
also have ... = ($s_e + v_e * \delta + v_e *$ (− v_e / a_e) + *1* / *2* $* a_e *$ (− v_e / a_e)^2 < $s_o + v_o *$ (− v_o / a_o) + *1* / *2* $* a_o *$ (− v_o / a_o)^2)
using *ego2.s-def ego2.p-def ego.q-def other.p-max-def other.p-def other.t-stop-def ego2.p-max-def ego2.s-t-stop ego2.t-stop-def* **by** *auto*
also have ... = ($v_e * \delta + v_e *$ (− v_e / a_e) + *1* / *2* $* a_e *$ (− v_e / a_e)^2 − $v_o *$ (− v_o / a_o) − *1* / *2* $* a_o *$ (− v_o / a_o)^2 < $s_o - s_e$) **by** *linarith*
also have ... = ($v_e * \delta - {v_e}^2$ / a_e + *1* / *2* $* {v_e}^2$ / $a_e + {v_o}^2$ / a_o − *1* / *2* $* {v_o}^2$ / $a_o < s_o - s_e$)
using *ego2.p-def ego2.p-max-def ego2.p-max-eq ego2.t-stop-def other.p-def other.p-max-def other.p-max-eq other.t-stop-def* **by** *auto*
also have ... = ($v_e * \delta - {v_e}^2$ / *2* / $a_e + {v_o}^2$ / *2* / $a_o < s_o - s_e$) **by** *linarith*
thus *?thesis* **using** *distance0-2-def* **by** (*simp add*: *calculation safe-distance-2r-def*)
qed

theorem *dist0-sd2r-2*:
assumes $\delta > - v_o \ / \ a_o$
assumes $s_o - s_e > safe\text{-}distance\text{-}2r$
shows $s_o - s_e > distance0\text{-}2$
proof −
have $- {v_e}^2 \ / \ 2 \ / \ a_e \geq 0$ **using** *zero-le-power2 hyps*(3) *divide-nonneg-neg* **by** (*auto simp add*:*field-simps*)
hence $v_e * \delta - {v_e}^2 \ / \ 2 \ / \ a_e + {v_o}^2 \ / \ 2 \ / \ a_o \geq v_e * \delta + {v_o}^2 \ / \ 2 \ / \ a_o$ **by** *simp*
hence *safe-distance-2r* $\geq$ *distance0-2* **using** *safe-distance-2r-def distance0-2-def* **by** *auto*
thus *?thesis* **using** *assms*(2) **by** *linarith*
qed
end

2.2 Safe Distance Delta

locale *safe-distance-no-collsion-delta* = *safe-distance-normal* +
assumes *no-collision-delta*: $u\ \delta < other.s\ \delta$
begin

sublocale *delayed-safe-distance*: *safe-distance* $a_e\ v_e$ *ego.q* $\delta\ a_o$ *other.s'* δ *other.s* δ
proof (*unfold-locales*)
from *nonneg-vel-ego* **show** $0 \leq v_e$ **by** *auto*
next
from *nonneg-vel-other* **show** $0 \leq other.s'\ \delta$ **unfolding** *other.s'-def other.p'-def other.t-stop-def*
using *decelerate-other* **by** (*auto simp add*: *field-split-simps*)
next
from *decelerate-ego* **show** $a_e < 0$ **by** *auto*
next
from *decelerate-other* **show** $a_o < 0$ **by** *auto*
next
from *no-collision-delta* **show** *ego.q* $\delta <$ *other.s* δ **unfolding** *u-def* **using** *pos-react* **by** *auto*
qed

lemma *no-collision-react-initially-strict*:
assumes $s_o \leq u\text{-}max$
assumes $u\text{-}max < other.s\text{-}stop$
shows *no-collision-react* $\{0 <..< \delta\}$
proof (*rule no-collision-reactI*)
fix *t*::*real*
assume $t \in \{0 <..< \delta\}$
show $u\ t \neq other.s\ t$
proof (*rule ccontr*)
assume $\neg\ u\ t \neq other.s\ t$
hence *ego-other2 t* = *0* **unfolding** *ego-other2-def* **by** *auto*
from ‹$t \in \{0 <..< \delta\}$› **have** *ego-other2 t* = *other.s t* − *ego.q t*
unfolding *ego-other2-def u-def* **using** *ego.init-q* **by** *auto*

have $\delta \leq$ *other.t-stop* $\vee$ *other.t-stop* $< \delta$ **by** *auto*

moreover
{ assume *le-t-stop*: $\delta \leq$ *other.t-stop*
with ‹*ego-other2 t* = *other.s t* − *ego.q t*› **have** *ego-other2 t* = *other.p t* − *ego.q t*
unfolding *other.s-def* **using** ‹$t \in \{0 <..< \delta\}$› **by** *auto*
with ‹*ego-other2 t* = *0*› **have** *other.p t* − *ego.q t* = *0* **by** *auto*
hence *eq*: $(s_o - s_e) + (v_o - v_e) * t + (1/2 * a_o) * t^2 = 0$
unfolding *other.p-def ego.q-def* **by** (*auto simp*: *algebra-simps*)
define *p* **where** $p \equiv \lambda x.\ (1/2 * a_o) * x^2 + (v_o - v_e) * x + (s_o - s_e)$
have $0 < 1/2 * a_o$
proof (*intro p-convex*[**where** *p*=*p* **and** *b*=$v_o - v_e$ **and** *c*=$s_o - s_e$])
show *0* < *t* **using** ‹$t \in \{0 <..< \delta\}$› **by** *auto*
next
show *t* < δ **using** ‹$t \in \{0 <..< \delta\}$› **by** *auto*
next
show *p t* < *p 0* **unfolding** *p-def* **using** *eq in-front* **by** (*auto simp*: *algebra-simps*)
next
from *eq* **have** *p t* = *0* **unfolding** *p-def* **by** *auto*
also have ... < *p* δ **using** *no-collision-delta pos-react le-t-stop*
unfolding *p-def u-def other.s-def ego.q-def other.p-def* **by** (*auto simp*:*algebra-simps*)
finally have *p t* < *p* δ **by** *simp*
thus *p t* ≤ *p* δ **by** *auto*
next
show $p = (\lambda x.\ 1\ /\ 2 * a_o * x^2 + (v_o - v_e) * x + (s_o - s_e))$ **unfolding** *p-def*
by (*rule refl*)
qed
hence $0 < a_o$ **by** *auto*
with *decelerate-other* **have** *False* **by** *simp* **}**

moreover
{ assume *gt-t-stop*: $\delta >$ *other.t-stop*
have *t-lt-t-stop*: *t* < *other.t-stop*
proof (*rule ccontr*)
assume ¬ *t* < *other.t-stop*
hence *other.t-stop* ≤ *t* **by** *simp*
from ‹*ego-other2 t* = *0*› **have** *ego.q t* = *other.p-max*
unfolding *ego-other2-def u-def other.s-def comp-def* τ*-def other.p-max-def*
using ‹$t \in \{0 <..< \delta\}$› ‹*other.t-stop* ≤ *t*› *gt-t-stop* **by** (*auto split*:*if-splits*)
have *ego.q t* = *u t* **unfolding** *u-def* **using** ‹$t \in \{0 <..< \delta\}$› **by** *auto*
also have ... ≤ *u-max* **using** *u-max* **by** *auto*
also have ... < *other.p-max* **using** *assms*(*2*) *other.s-t-stop* **by** *auto*
finally have *ego.q t* < *other.p-max* **by** *auto*
with ‹*ego.q t* = *other.p-max*› **show** *False* **by** *auto*
qed

```
      with ‹ego-other2 t = other.s t − ego.q t› have ego-other2 t = other.p t −
ego.q t
        unfolding other.s-def using ‹t ∈ {0 <..< δ}› by auto
      with ‹ego-other2 t = 0› have other.p t − ego.q t = 0 by auto
      hence eq: (s_o− s_e) + (v_o − v_e) * t + (1/2 * a_o) * t^2 = 0
        unfolding other.p-def ego.q-def by (auto simp: algebra-simps)
      define p where p ≡ λx. (1/2 * a_o) * x^2 + (v_o − v_e) * x + (s_o − s_e)
      have 0 < 1/2 * a_o
      proof (intro p-convex[where p=p and b=v_o − v_e and c=s_o − s_e])
         show 0 < t using ‹t ∈ {0 <..< δ}› by auto
        next
         show t < other.t-stop using t-lt-t-stop by auto
        next
            show p t < p 0 unfolding p-def using eq in-front by (auto simp:
algebra-simps)
        next
         from eq have zero: p t = 0 unfolding p-def by auto
         have eq: p other.t-stop = ego-other2 other.t-stop
           unfolding ego-other2-def other.s-t-stop u-def ego.q-def
                   other.s-def other.p-def p-def
           using ‹δ > other.t-stop› other.t-stop-nonneg other.t-stop-def
           by (auto simp: diff-divide-distrib left-diff-distrib')
         have u other.t-stop ≤ u-max using u-max by auto
         also have ... < other.s-stop using assms by auto
         finally have 0 ≤ other.s-stop − u other.t-stop by auto
         hence 0 ≤ ego-other2 other.t-stop unfolding ego-other2-def by auto
         hence 0 ≤ p other.t-stop using eq by auto
         with zero show p t ≤ p other.t-stop by auto
        next
         show p = (λx. 1 / 2 * a_o * x^2 + (v_o − v_e) * x + (s_o − s_e))
         unfolding p-def by (rule refl)
      qed
      hence False using decelerate-other by auto }

    ultimately show False by auto
  qed
qed

lemma no-collision-react-initially:
  assumes s_o ≤ u-max
  assumes u-max < other.s-stop
  shows no-collision-react {0 .. δ}
proof −
  have no-collision-react {0 <..< δ} by (rule no-collision-react-initially-strict[OF
assms])
  have u 0 ≠ other.s 0 using init-u other.init-s in-front by auto
  hence no-collision-react {0} unfolding collision-react-def by auto
  with ‹no-collision-react {0 <..< δ}› have no-collision-react ({0} ∪ {0 <..< δ})
```

```
    using no-collision-union[of {0} {0 <..< δ}] by auto
  moreover have {0} ∪ {0 <..< δ} = {0 ..< δ} using pos-react by auto
  ultimately have no-collision-react {0 ..< δ} by auto

  have u δ ≠ other.s δ using no-collision-delta by auto
  hence no-collision-react {δ} unfolding collision-react-def by auto
  with ‹no-collision-react {0 ..< δ}› have no-collision-react ({δ} ∪ {0 ..< δ})
    using no-collision-union[of {δ} {0 ..< δ}] by auto
  moreover have {δ} ∪ {0 ..< δ} = {0 .. δ} using pos-react by auto
  ultimately show no-collision-react {0 .. δ} by auto
qed

lemma collision-after-delta:
  assumes s_o ≤ u-max
  assumes u-max < other.s-stop
  shows collision-react {0 ..} ⟷ collision-react {δ..}
proof
  assume collision-react {0 ..}
  have no-collision-react {0 .. δ} by (rule no-collision-react-initially[OF assms])
  with ‹collision-react {0..}› have collision-react ({0..} − {0 .. δ})
  using pos-react by (auto intro: collision-trim-subset)

  moreover have {0..} − {0 .. δ} = {δ <..} using pos-react by auto
  ultimately have collision-react {δ <..} by auto
  thus collision-react {δ ..} by (auto intro:collision-react-subset)
next
  assume collision-react {δ..}
  moreover have {δ..} ⊆ {0 ..} using pos-react by auto
  ultimately show collision-react {0 ..} by (rule collision-react-subset)
qed

lemma collision-react-strict:
  assumes s_o ≤ u-max
  assumes u-max < other.s-stop
  shows collision-react {δ ..} ⟷ collision-react {δ <..}
proof
  assume asm: collision-react {δ ..}
 have no-collision-react {δ} using no-collision-delta unfolding collision-react-def
by auto
  moreover have {δ <..} ⊆ {δ ..} by auto
  ultimately have collision-react ({δ ..} − {δ}) using asm collision-trim-subset
by simp
  moreover have {δ <..} = {δ ..} − {δ} by auto
  ultimately show collision-react {δ <..} by auto
next
  assume collision-react {δ <..}
  thus collision-react {δ ..}
    using collision-react-subset[where t={δ ..} and s={δ <..}] by fastforce
qed
```

lemma *delayed-other-s-stop-eq*: *delayed-safe-distance.other.s-stop* = *other.s-stop*
proof (*unfold other.s-t-stop*; *unfold delayed-safe-distance.other.s-t-stop*; *unfold movement.p-max-eq*)
have $\delta \leq$ *other.t-stop* $\vee$ *other.t-stop* $< \delta$ **by** *auto*

moreover
{ assume $\delta \leq$ *other.t-stop*
hence *other.s* $\delta - (other.s' \delta)^2 / a_o / 2 = s_o - {v_o}^2 / a_o / 2$
unfolding *other.s-def other.s'-def*
using *pos-react decelerate-other*
by (*auto simp add*: *other.p-def other.p'-def power2-eq-square field-split-simps*)
}

moreover
{ assume *other.t-stop* $< \delta$
hence *other.s* $\delta - (other.s' \delta)^2 / a_o / 2 = s_o - {v_o}^2 / a_o / 2$
unfolding *other.s-def other.s'-def other.p-max-eq*
using *pos-react decelerate-other*
by (*auto*) **}**

ultimately show *other.s* $\delta - (other.s' \delta)^2 / a_o / 2 = s_o - {v_o}^2 / a_o / 2$ **by** *auto*
qed

lemma *delayed-cond3'*:
assumes *other.s* $\delta \leq$ *u-max*
assumes *u-max* $<$ *other.s-stop*
shows *delayed-safe-distance.collision* $\{0 ..\} \longleftrightarrow$
$(a_o > a_e \wedge other.s' \delta < v_e \wedge other.s \delta - ego.q \delta \leq$ *delayed-safe-distance.snd-safe-distance* $\wedge\ v_e - a_e / a_o * other.s' \delta < 0)$
proof (*rule delayed-safe-distance.cond-3'*)
have *other.s* $\delta \leq$ *u-max* **using** ‹*other.s* $\delta \leq$ *u-max*› **.**
also have ... = *ego2.s-stop* **unfolding** *u-max-eq ego2.s-t-stop ego2.p-max-eq* **by** (*rule refl*)
finally show *other.s* $\delta \leq$ *ego2.s-stop* **by** *auto*
next
have *ego2.s-stop* = *u-max* **unfolding** *ego2.s-t-stop ego2.p-max-eq u-max-eq* **by** (*rule refl*)
also have ... $<$ *other.s-stop* **using** *assms* **by** *auto*
also have ... $\leq$ *delayed-safe-distance.other.s-stop* **using** *delayed-other-s-stop-eq* **by** *auto*
finally show *ego2.s-stop* $<$ *delayed-safe-distance.other.s-stop* **by** *auto*
qed

lemma *delayed-other-t-stop-eq*:
assumes $\delta \leq$ *other.t-stop*
shows *delayed-safe-distance.other.t-stop* + δ = *other.t-stop*
using *assms decelerate-other*

```
  unfolding delayed-safe-distance.other.t-stop-def other.t-stop-def other.s'-def
        movement.t-stop-def other.p'-def
  by (auto simp add: field-split-simps)

lemma delayed-other-s-eq:
  assumes 0 ≤ t
  shows delayed-safe-distance.other.s t = other.s (t + δ)
proof (cases δ ≤ other.t-stop)
  assume 1: δ ≤ other.t-stop
  have t + δ ≤ other.t-stop ∨ other.t-stop < t + δ by auto
  moreover
  { assume t + δ ≤ other.t-stop
    hence delayed-safe-distance.other.s t = delayed-safe-distance.other.p t
      using delayed-other-t-stop-eq [OF 1] assms
      unfolding delayed-safe-distance.other.s-def by auto

    also have ... = other.p (t + δ)
      unfolding movement.p-def other.s-def other.s'-def other.p'-def
      using pos-react 1
      by (auto simp add: power2-eq-square field-split-simps)

    also have ... = other.s (t + δ)
      unfolding other.s-def
      using assms pos-react ‹t + δ ≤ other.t-stop› by auto

    finally have delayed-safe-distance.other.s t = other.s (t + δ) by auto }

  moreover
  { assume other.t-stop < t + δ
    hence delayed-safe-distance.other.s t = delayed-safe-distance.other.p-max
    using delayed-other-t-stop-eq [OF 1] assms delayed-safe-distance.other.t-stop-nonneg
      unfolding delayed-safe-distance.other.s-def by auto

    also have ... = other.p-max
     unfolding movement.p-max-eq other.s-def other.s'-def other.p-def other.p'-def
      using pos-react 1 decelerate-other
      by (auto simp add: power2-eq-square field-split-simps)

    also have ... = other.s (t + δ)
      unfolding other.s-def
      using assms pos-react ‹other.t-stop < t + δ› by auto

    finally have delayed-safe-distance.other.s t = other.s (t + δ) by auto }

  ultimately show ?thesis by auto
next
  assume ¬ δ ≤ other.t-stop
  hence other.t-stop < δ by auto
  hence other.s' δ = 0 and other.s δ = other.p-max
```

unfolding *other.s'-def other.s-def* **using** *pos-react* **by** *auto*
hence *delayed-safe-distance.other.s t = delayed-safe-distance.other.p-max*
unfolding *delayed-safe-distance.other.s-def* **using** *assms decelerate-other*
by (*auto simp add:movement.p-max-eq movement.p-def movement.t-stop-def*)
also have ... = *other.p-max*
unfolding *movement.p-max-eq* **using** ‹*other.s' δ = 0*› ‹*other.s δ = other.p-max*›
using *other.p-max-eq* **by** *auto*
also have ... = *other.s* (*t + δ*)
unfolding *other.s-def* **using** *pos-react assms* ‹*other.t-stop < δ*› **by** *auto*
finally show *delayed-safe-distance.other.s t = other.s* (*t + δ*) **by** *auto*
qed

lemma *translate-collision-range*:
assumes $s_o \leq$ *u-max*
assumes *u-max < other.s-stop*
shows *delayed-safe-distance.collision* {*0 ..*} ⟷ *collision-react* {*δ ..*}
proof
assume *delayed-safe-distance.collision* {*0 ..*}
then obtain *t* **where** *eq*: *ego2.s t = delayed-safe-distance.other.s t* **and** *0 ≤ t*
unfolding *delayed-safe-distance.collision-def* **by** *auto*

have *ego2.s t* = (*ego2.s* ∘ *τ*) (*t + δ*) **unfolding** *comp-def τ-def* **by** *auto*
also have ... = *u* (*t + δ*) **unfolding** *u-def* **using** ‹*0 ≤ t*› *pos-react*
by (*auto simp*: *τ-def ego2.init-s*)
finally have *left*:*ego2.s t = u* (*t + δ*) **by** *auto*

have *right*: *delayed-safe-distance.other.s t = other.s* (*t + δ*)
using *delayed-other-s-eq pos-react* ‹*0 ≤ t*› **by** *auto*

with *eq* **and** *left* **have** *u* (*t + δ*) = *other.s* (*t + δ*) **by** *auto*
moreover have *δ ≤ t + δ* **using** ‹*0 ≤ t*› **by** *auto*
ultimately show *collision-react* {*δ ..*} **unfolding** *collision-react-def* **by** *auto*
next
assume *collision-react* {*δ ..*}
hence *collision-react* {*δ <..*} **using** *collision-react-strict*[*OF assms*] **by** *simp*
then obtain *t* **where** *eq*: *u t = other.s t* **and** *δ < t*
unfolding *collision-react-def* **by** *auto*
moreover hence *u t* = (*ego2.s* ∘ *τ*) *t* **unfolding** *u-def* **using** *pos-react* **by** *auto*
moreover have *other.s t = delayed-safe-distance.other.s* (*t − δ*)
using *delayed-other-s-eq* ‹*δ < t*› **by** *auto*
ultimately have *ego2.s* (*t − δ*) = *delayed-safe-distance.other.s* (*t − δ*)
unfolding *comp-def τ-def* **by** *auto*
with ‹*δ < t*› **show** *delayed-safe-distance.collision* {*0 ..*}
unfolding *delayed-safe-distance.collision-def* **by** *auto*
qed

theorem *cond-3r-2*:
assumes $s_o \leq$ *u-max*
assumes *u-max < other.s-stop*

assumes *other.s* $\delta \leq$ *u-max*
shows *collision-react* $\{0\ ..\} \longleftrightarrow$
$(a_o > a_e \wedge$ *other.s'* $\delta <$ *v_e* $\wedge$ *other.s* $\delta -$ *ego.q* $\delta \leq$ *delayed-safe-distance.snd-safe-distance* $\wedge$ *v_e* $-$ *a_e* / *a_o* $*$ *other.s'* $\delta < 0)$
proof −
have *collision-react* $\{0\ ..\} \longleftrightarrow$ *collision-react* $\{\delta\ ..\}$ **by** (*rule collision-after-delta*[*OF assms*(*1*) *assms*(*2*)])
also have ... $\longleftrightarrow$ *delayed-safe-distance.collision* $\{0\ ..\}$ **by** (*simp add*: *translate-collision-range*[*OF assms*(*1*) *assms*(*2*)])
also have ... $\longleftrightarrow$ $(a_o > a_e \wedge$ *other.s'* $\delta <$ *v_e* $\wedge$ *other.s* $\delta -$ *ego.q* $\delta \leq$ *delayed-safe-distance.snd-safe-distance* $\wedge$ *v_e* $-$ *a_e* / *a_o* $*$ *other.s'* $\delta < 0)$
by (*rule delayed-cond3'*[*OF assms*(*3*) *assms*(*2*)])
finally show *collision-react* $\{0\ ..\} \longleftrightarrow$ $(a_o > a_e \wedge$ *other.s'* $\delta <$ *v_e* $\wedge$ *other.s* $\delta -$ *ego.q* $\delta \leq$ *delayed-safe-distance.snd-safe-distance* $\wedge$ *v_e* $-$ *a_e* / *a_o* $*$ *other.s'* $\delta < 0)$
by *auto*
qed

lemma *sd-2r-correct-for-3r-2*:
assumes *s_o* $-$ *s_e* $>$ *safe-distance-2r*
assumes *other.s* $\delta \leq$ *u-max*
assumes $\neg$ $(a_o > a_e \wedge$ *other.s'* $\delta <$ *v_e* $\wedge$ *v_e* $-$ *a_e* / *a_o* $*$ *other.s'* $\delta < 0)$
shows *no-collision-react* $\{0..\}$
proof −
from *assms* **have** *s_o* $-$ *s_e* $>$ *v_e* $*$ δ $-$ *v_e*^{2} / *2* / *a_e* $+$ *v_o*^{2} / *2* / *a_o* **unfolding** *safe-distance-2r-def* **by** *auto*
hence *s_o* $-$ *v_o*^{2} / *2* / *a_o* $>$ *s_e* $+$ *v_e* $*$ δ $-$ *v_e*^{2} / *2* / *a_e* **by** *auto*
hence *s_o* $-$ *v_o*^{2} / *a_o* $+$ *v_o*^{2} / *2* / *a_o* $>$ *s_e* $+$ *v_e* $*$ δ $-$ *v_e*^{2} / *2* / *a_e* **by** *auto*
hence *s_o* $+$ *v_o* $*$ $(-$ *v_o* / *a_o*$)$ $+$ *1/2* $*$ *a_o* $*$ $(-$*v_o* / *a_o*$)^2 >$ *s_e* $+$ *v_e* $*$ δ $-$ *v_e*^{2} / *2* / *a_e*
using *other.p-def other.p-max-def other.p-max-eq other.t-stop-def* **by** *auto*
hence *other.s-stop* $>$ *u-max* **unfolding** *other.s-def* **using** *u-max-eq other.t-stop-def*
using *ego.q-def other.p-def other.p-max-def other.s-def other.s-t-stop* **by** *auto*
thus *?thesis*
using *assms*(*2*) *assms*(*3*) *collision-after-delta cond-1r delayed-cond3' translate-collision-range* **by** *linarith*
qed

lemma *sd2-at-most-sd4*:
assumes *a_o* $>$ *a_e*
shows *safe-distance-2r* $\leq$ *safe-distance-4r*
proof −
have *a_o* $\neq$ *0* **and** *a_e* $\neq$ *0* **and** *a_o* $-$ *a_e* $\neq$ *0* **and** *0* $<$ *2* $*$ $(a_o - a_e)$ **using** *hyps assms*(*1*) **by** *auto*
have *0* $\leq$ $(-$ *v_e* $*$ *a_o* $+$ *v_o* $*$ *a_e* $+$ *a_o* $*$ *a_e* $*$ $\delta)$ $*$ $(-$ *v_e* $*$ *a_o* $+$ *v_o* $*$ *a_e* $+$ *a_o* $*$ *a_e* $*$ $\delta)$
(**is** *0* $\leq$ *(?l1 + ?l2 + ?l3)* $*$ *?r*) **by** *auto*
also have ... $=$ *v_e*^{2} $*$ *a_o*^{2} $+$ *v_o*^{2} $*$ *a_e*^{2} $+$ *a_o*^{2} $*$ *a_e*^{2} $*$ δ^2 $-$ *2* $*$ *v_e* $*$ *a_o* $*$ *v_o* $*$ *a_e* $-$ *2* $*$ *a_o*^{2} $*$ *a_e* $*$ δ $*$ *v_e* $+$ *2* $*$ *a_o* $*$ *a_e*^{2} $*$ δ $*$ *v_o*

(**is** *?lhs = ?rhs*)
by (*auto simp add:algebra-simps power-def*)
finally have *0 ≤ ?rhs* **by** *auto*
hence $(- {v_e}^2 * a_o / a_e - {v_o}^2 * a_e / a_o) * (a_o * a_e) \leq (a_o * a_e * \delta^2 - 2 * v_e * v_o - 2 * a_o * \delta * v_e + 2 * a_e * \delta * v_o) * (a_o * a_e)$
by (*auto simp add: algebra-simps power-def*)
hence $2 * v_e * \delta * (a_o - a_e) - {v_e}^2 * a_o / a_e + {v_e}^2 + {v_o}^2 - {v_o}^2 * a_e / a_o \leq {v_o}^2 + {a_o}^2 * \delta^2 + {v_e}^2 + 2 * v_o * \delta * a_o - 2 * v_e * v_o - 2 * a_o * \delta * v_e - 2 * v_o * \delta * a_o + 2 * a_e * \delta * v_o - {a_o}^2 * \delta^2 + a_o * a_e * \delta^2 + 2 * v_e * \delta * (a_o - a_e)$
by (*auto simp add: ego2.decel other.decel*)
hence $2 * v_e * \delta * (a_o - a_e) - {v_e}^2 * a_o / a_e + {v_e}^2 + {v_o}^2 - {v_o}^2 * a_e / a_o \leq (v_o + \delta * a_o - v_e)^2 - 2 * v_o * \delta * a_o + 2 * a_e * \delta * v_o - {a_o}^2 * \delta^2 + a_o * a_e * \delta^2 + 2 * v_e * \delta * (a_o - a_e)$
by (*auto simp add: algebra-simps power-def*)
hence $v_e * \delta * 2 * (a_o - a_e) - {v_e}^2 / 2 / a_e * 2 * a_o + {v_e}^2 / 2 / a_e * 2 * a_e + {v_o}^2 / 2 / a_o * 2 * a_o - {v_o}^2 / 2 / a_o * 2 * a_e \leq (v_o + \delta * a_o - v_e)^2 - v_o * \delta * 2 * a_o - v_o * \delta * 2 * -a_e - a_o * \delta^2 / 2 * 2 * a_o - a_o * \delta^2 / 2 * 2 * -a_e + v_e * \delta * 2 * (a_o - a_e)$
(**is** *?lhs1 ≤ ?rhs1*)
by (*simp add*: ‹$a_o \neq 0$› ‹$a_e \neq 0$› *power2-eq-square algebra-simps*)
hence $v_e * \delta * 2 * (a_o - a_e) - {v_e}^2 / 2 / a_e * 2 * (a_o - a_e) + {v_o}^2 / 2 / a_o * 2 * (a_o - a_e) \leq (v_o + a_o * \delta - v_e)^2 / 2 / (a_o - a_e) * 2 * (a_o - a_e) - v_o * \delta * 2 * (a_o - a_e) - a_o * \delta^2 / 2 * 2 * (a_o - a_e) + v_e * \delta * 2 *(a_o - a_e)$
(**is** *?lhs2 ≤ ?rhs2*)
proof −
assume *?lhs1 ≤ ?rhs1*
have *?lhs1 = ?lhs2* **by** (*auto simp add:field-simps*)
moreover
have *?rhs1 = ?rhs2* **using** ‹$a_o - a_e \neq 0$› **by** (*auto simp add:field-simps*)
ultimately show *?thesis* **using** ‹*?lhs1 ≤ ?rhs1*› **by** *auto*
qed
hence $(v_e * \delta - {v_e}^2 / 2 / a_e + {v_o}^2 / 2 / a_o) * 2 * (a_o - a_e) \leq ((v_o + a_o * \delta - v_e)^2 / 2 / (a_o - a_e) - v_o * \delta - 1/2 * a_o * \delta^2 + v_e * \delta) * 2 *(a_o - a_e)$
by (*simp add*: *algebra-simps*)
hence $v_e * \delta - {v_e}^2 / 2 / a_e + {v_o}^2 / 2 / a_o \leq (v_o + a_o * \delta - v_e)^2 / 2 / (a_o - a_e) - v_o * \delta - 1/2 * a_o * \delta^2 + v_e * \delta$
using ‹$a_o > a_e$› *mult-le-cancel-right-pos*[*OF* ‹$0 < 2 * (a_o - a_e)$›, *of* $(v_e * \delta - {v_e}^2 / 2 / a_e + {v_o}^2 / 2 / a_o)$
$(v_o + a_o * \delta - v_e)^2 / 2 / (a_o - a_e) - v_o * \delta - 1/2 * a_o * \delta^2 + v_e * \delta$] *semiring-normalization-rules*(*18*)
by (*metis* (*no-types*, *lifting*))
thus *?thesis* **using** *safe-distance-2r-def safe-distance-4r-def* **by** *auto*
qed

lemma *sd-4r-correct*:
assumes $s_o - s_e >$ *safe-distance-4r*
assumes *other.s* $\delta \leq$ *u-max*
assumes $\delta \leq$ *other.t-stop*
assumes $a_o > a_e$

shows *no-collision-react* $\{0..\}$
proof −
from *assms* **have** $s_o - s_e > (v_o + a_o * \delta - v_e)^2 \;/\; 2 \;/\; (a_o - a_e) - v_o * \delta - 1/2 * a_o * \delta^2 + v_e * \delta$
unfolding *safe-distance-4r-def* **by** *auto*
hence $s_o + v_o * \delta + 1/2 * a_o * \delta^2 - s_e - v_e * \delta > (v_o + a_o * \delta - v_e)^2 \;/\; 2 \;/\; (a_o - a_e)$ **by** *linarith*
hence *other.s* δ − *ego.q* $\delta > (other.s' \; \delta - v_e)^2 \;/\; 2 \;/\; (a_o - a_e)$
using *assms*(3) *ego.q-def other.p-def other.s-def other.p'-def other.s'-def pos-react* **by** *auto*
hence *other.s* δ − *ego.q* $\delta >$ *delayed-safe-distance.snd-safe-distance*
by (*simp add*: *delayed-safe-distance.snd-safe-distance-def*)
hence *c*: ¬ (*other.s* δ − *ego.q* $\delta \leq$ *delayed-safe-distance.snd-safe-distance*) **by** *linarith*
have *u-max* < *other.s-stop*
unfolding *u-max-eq other.s-t-stop other.p-max-eq ego.q-def* **using** *assms*(1) *sd2-at-most-sd4*[*OF assms*(4)]
unfolding *safe-distance-4r-def safe-distance-2r-def* **by** *auto*
consider $s_o \leq$ *u-max* | $s_o >$ *u-max* **by** *linarith*
thus *?thesis*
proof (*cases*)
case *1*
from *cond-3r-2*[*OF this* ‹*u-max* < *other.s-stop*› *assms*(2)] **show** *?thesis*
using *c* **by** *auto*
next
case *2*
then show *?thesis* **using** *cond-1r* **by** *auto*
qed
qed

Irrelevant, this Safe Distance is unreachable in the checker.

definition *safe-distance-5r* :: *real* **where**
safe-distance-5r = ${v_e}^2 \;/\; 2 \;/\; (a_o - a_e) + {v_o}^2 \;/\; 2 \;/\; a_o + v_e * \delta$

lemma *sd-5r-correct*:
assumes $s_o - s_e >$ *safe-distance-5r*
assumes *u-max* < *other.s-stop*
assumes *other.s* $\delta \leq$ *u-max*
assumes $\delta >$ *other.t-stop*
shows *no-collision-react* $\{0..\}$
proof −
from *assms* **have** $s_o - s_e > {v_e}^2 \;/\; 2 \;/\; (a_o - a_e) + {v_o}^2 \;/\; 2 \;/\; a_o + v_e * \delta$
unfolding *safe-distance-5r-def* **by** *auto*
hence $s_o + {v_o}^2 \;/\; 2 \;/\; a_o - s_e - v_e * \delta > (0 - v_e)^2 \;/\; 2 \;/\; (a_o - a_e)$
using *assms*(2−4) **unfolding** *other.s-def other.s-t-stop*
apply (*auto simp*: *movement.p-t-stop split*: *if-splits*)
using *cond-1r cond-2r other.s-t-stop* **by** *linarith*+
hence *other.s* δ − *ego.q* $\delta > (other.s' \; \delta - v_e)^2 \;/\; 2 \;/\; (a_o - a_e)$
using *assms*(2) *assms*(3) *assms*(4) *other.s-def other.s-t-stop* **by** *auto*

hence *other.s δ − ego.q δ > delayed-safe-distance.snd-safe-distance*
by (*simp add*: *delayed-safe-distance.snd-safe-distance-def*)
hence ¬ (*other.s δ − ego.q δ ≤ delayed-safe-distance.snd-safe-distance*) **by** *linarith*
thus *?thesis* **using** *assms(2) assms(3) cond-1r cond-3r-2* **by** *linarith*
qed

lemma *translate-no-collision-range*:
delayed-safe-distance.no-collision {0 ..} ⟷ no-collision-react {δ ..}
proof
assume *left*: *delayed-safe-distance.no-collision {0 ..}*
show *no-collision-react {δ ..}*
proof (*unfold collision-react-def*; *simp*; *rule ballI*)
fix *t*::*real*
assume *t ∈ {δ ..}*
hence *δ ≤ t* **by** *simp*
with *pos-react* **have** *0 ≤ t − δ* **by** *simp*
with *left* **have** *ineq*: *ego2.s (t − δ) ≠ delayed-safe-distance.other.s (t − δ)*
unfolding *delayed-safe-distance.collision-def* **by** *auto*
have *ego2.s (t − δ) = (ego2.s ∘ τ) t* **unfolding** *comp-def τ-def* **by** *auto*
also have *... = u t* **unfolding** *u-def* **using** ‹*δ ≤ t*› *pos-react*
by (*auto simp*: *τ-def ego2.init-s*)
finally have *ego2.s (t − δ) = u t* **by** *auto*

moreover have *delayed-safe-distance.other.s (t − δ) = other.s t*
using *delayed-other-s-eq pos-react* ‹*δ ≤ t*› **by** *auto*

ultimately show *u t ≠ other.s t* **using** *ineq* **by** *auto*
qed
next
assume *right*: *no-collision-react {δ ..}*
show *delayed-safe-distance.no-collision {0 ..}*
proof (*unfold delayed-safe-distance.collision-def*; *simp*; *rule ballI*)
fix *t* ::*real*
assume *t ∈ {0 ..}*
hence *0 ≤ t* **by** *auto*
hence *δ ≤ t + δ* **by** *auto*
with *right* **have** *ineq*: *u (t + δ) ≠ other.s (t + δ)* **unfolding** *collision-react-def* **by** *auto*

have *u (t + δ) = ego2.s t* **unfolding** *u-def comp-def τ-def*
using ‹*0 ≤ t*› *pos-react* ‹*δ ≤ t+ δ*› **by** (*auto simp add*:*ego2.init-s*)
moreover have *other.s (t + δ) = delayed-safe-distance.other.s t*
using *delayed-other-s-eq*[*of t*] **using** ‹*0 ≤ t*› **by** *auto*
ultimately show *ego2.s t ≠ delayed-safe-distance.other.s t* **using** *ineq* **by** *auto*
qed
qed

lemma *delayed-cond1*:

assumes *other.s* δ > *u-max*
shows *delayed-safe-distance.no-collision* {*0* ..}
proof −
have *ego2.s-stop* = *u-max* **unfolding** *ego2.s-t-stop ego2.p-max-eq u-max-eq* **by** *auto*
also have ... < *other.s* δ **using** *assms* **by** *simp*
finally have *ego2.s-stop* < *other.s* δ **by** *auto*
thus *delayed-safe-distance.no-collision* {*0* ..} **by** (*simp add*: *delayed-safe-distance.cond-1*)
qed

theorem *cond-3r-3*:
assumes $s_o \leq$ *u-max*
assumes *u-max* < *other.s-stop*
assumes *other.s* δ > *u-max*
shows *no-collision-react* {*0* ..}
proof −
have *eq*: {*0* ..} = {*0* .. δ} ∪ {δ ..} **using** *pos-react* **by** *auto*
show *?thesis* **unfolding** *eq*
proof (*intro no-collision-union*)
show *no-collision-react* {*0* .. δ} **by** (*rule no-collision-react-initially*[*OF assms*(*1*) *assms*(*2*)])
next
have *delayed-safe-distance.no-collision* {*0* ..} **by** (*rule delayed-cond1*[*OF assms*(*3*)])
with *translate-no-collision-range* **show** *no-collision-react* {δ ..} **by** *auto*
qed
qed

lemma *sd-2r-correct-for-3r-3*:
assumes $s_o - s_e >$ *safe-distance-2r*
assumes *other.s* δ > *u-max*
shows *no-collision-react* {*0*..}
proof −
from *assms* **have** $s_o - s_e > v_e * \delta - {v_e}^2$ / *2* / $a_e + {v_o}^2$ / *2* / a_o **unfolding** *safe-distance-2r-def* **by** *auto*
hence $s_o - {v_o}^2$ / *2* / $a_o > s_e + v_e * \delta - {v_e}^2$ / *2* / a_e **by** *auto*
hence $s_o - {v_o}^2$ / $a_o + {v_o}^2$ / *2* / $a_o > s_e + v_e * \delta - {v_e}^2$ / *2* / a_e **by** *auto*
hence $s_o + v_o * (-\ v_o$ / $a_o)$ + *1/2* $* a_o * (-v_o$ / $a_o)^2 > s_e + v_e * \delta - {v_e}^2$ / *2* / a_e
using *other.p-def other.p-max-def other.p-max-eq other.t-stop-def* **by** *auto*
hence *other.s-stop* > *u-max* **unfolding** *other.s-def* **using** *u-max-eq other.t-stop-def*
using *ego.q-def other.p-def other.p-max-def other.s-def other.s-t-stop* **by** *auto*
thus *?thesis*
using *assms*(*2*) *cond-1r cond-3r-3* **by** *linarith*
qed

lemma *sd-3r-correct*:
assumes $s_o - s_e >$ *safe-distance-3r*
assumes $\delta \leq$ *other.t-stop*
shows *no-collision-react* {*0* ..}

proof −
from *assms* **have** $s_o - s_e > v_e * \delta - {v_e}^2 \ / \ 2 \ / \ a_e - v_o * \delta - 1/2 * a_o * \delta^2$
unfolding *safe-distance-3r-def* **by** *auto*
hence $s_o + v_o * \delta + 1/2 * a_o * \delta^2 > s_e + v_e * \delta - {v_e}^2 \ / \ 2 \ / \ a_e$ **by** *auto*
hence *other.s* $\delta >$ *u-max* **using** *other.s-def u-max-eq assms*(2) *ego.q-def other.p-def pos-react* **by** *auto*
thus *?thesis* **using** *cond-1r cond-3r-3 delayed-other-s-stop-eq delayed-safe-distance.other.s0-le-s-stop* **by** *linarith*
qed

lemma *sd-2-at-least-sd-3*:
assumes $\delta \leq$ *other.t-stop*
shows *safe-distance-3r* $\geq$ *safe-distance-2r*
proof −
from *assms* **have** $\delta =$ *other.t-stop* $\vee \ \delta <$ *other.t-stop* **by** *auto*
then have *safe-distance-3r* = *safe-distance-2r* $\vee$ *safe-distance-3r* > *safe-distance-2r*
proof (*rule Meson.disj-forward*)
assume $\delta =$ *other.t-stop*
hence $\delta = - \ v_o \ / \ a_o$ **unfolding** *other.t-stop-def* **by** *auto*
hence $- \ v_o * \delta - 1/2 * a_o * \delta^2 = - \ v_o \ *$ *other.t-stop* $- \ 1/2 * a_o *$ *other.t-stop*^{2} **by** (*simp add*: *movement.t-stop-def*)
thus *safe-distance-3r* = *safe-distance-2r*
using *other.p-def other.p-max-def other.p-max-eq safe-distance-2r-def safe-distance-3r-def* **by** *auto*
next
assume $\delta <$ *other.t-stop*
hence $\delta < - \ v_o \ / \ a_o$ **unfolding** *other.t-stop-def* **by** *auto*
hence $0 < v_o + a_o * \delta$
using *other.decel other.p'-def other.p'-pos-iff* **by** *auto*
hence $0 < v_o + 1/2 * a_o * (\delta +$ *other.t-stop*$)$ **by** (*auto simp add*:*field-simps other.t-stop-def*)
hence $0 > v_o * (\delta -$ *other.t-stop*$) + 1/2 * a_o * (\delta +$ *other.t-stop*$) * (\delta -$ *other.t-stop*$)$
using ‹$\delta <$ *other.t-stop*› *mult-less-cancel-left-neg*[**where** $c{=}\delta -$ *other.t-stop* **and** $a{=}v_o + 1 \ / \ 2 * a_o * (\delta +$ *other.t-stop*$)$ **and** $b{=}0$]
by (*auto simp add*: *field-simps*)
hence $(\delta +$ *other.t-stop*$) * (\delta -$ *other.t-stop*$) = (\delta^2 -$ *other.t-stop*$^2)$
by (*simp add*: *power2-eq-square square-diff-square-factored*)
hence $0 > v_o * (\delta -$ *other.t-stop*$) + 1/2 * a_o * (\delta^2 -$ *other.t-stop*$^2)$
by (*metis* (*no-types*, *opaque-lifting*) ‹$v_o * (\delta -$ *other.t-stop*$) + 1 \ / \ 2 * a_o * (\delta +$ *other.t-stop*$) * (\delta -$ *other.t-stop*$) < 0$› *divide-divide-eq-left divide-divide-eq-right times-divide-eq-left*)
hence $0 > v_o * \delta - v_o *$ *other.t-stop* $+ 1/2 * a_o * \delta^2 - \ 1/2 * a_o *$ *other.t-stop*^{2}
by (*simp add*: *right-diff-distrib*)
hence $- \ v_o * \delta - 1/2 * a_o * \delta^2 > - \ v_o \ * (- \ v_o \ / \ a_o) - 1/2 * a_o * (- \ v_o \ / \ a_o)^2$
unfolding *movement.t-stop-def* **by** *argo*
thus *safe-distance-3r* > *safe-distance-2r*

using *other.p-def other.p-max-def other.p-max-eq other.t-stop-def safe-distance-2r-def safe-distance-3r-def* **by** *auto*
qed
thus *?thesis* **by** *auto*
qed
end

2.3 Checker Design

We define two checkers for different cases:

- one checker for the case that $\delta \leq$ *other.t-stop* (*other.t-stop* $= -\ v_o\ /\ a_o$)
- a second checker for the case that $\delta >$ *other.t-stop*

definition *check-precond-r1* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
check-precond-r1 $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \longleftrightarrow s_o > s_e \wedge 0 \leq v_e \wedge 0 \leq v_o \wedge a_e < 0 \wedge a_o < 0 \wedge 0 < \delta \wedge \delta \leq -\ v_o\ /\ a_o$

definition *safe-distance0* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance0 $v_e\ a_o\ v_o\ \delta = v_e * \delta - v_o * \delta - a_o * \delta^2\ /\ 2$

definition *safe-distance-1r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-1r $a_e\ v_e\ \delta = v_e * \delta - {v_e}^2\ /\ a_e\ /\ 2$

definition *safe-distance-2r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-2r $a_e\ v_e\ a_o\ v_o\ \delta = v_e * \delta - {v_e}^2\ /\ 2\ /\ a_e + {v_o}^2\ /\ 2\ /\ a_o$

definition *safe-distance-4r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-4r $a_e\ v_e\ a_o\ v_o\ \delta = (v_o + a_o * \delta - v_e)^2\ /\ 2\ /\ (a_o - a_e) - v_o * \delta - 1\ /\ 2 * a_o * \delta^2 + v_e * \delta$

definition *safe-distance-3r* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance-3r $a_e\ v_e\ a_o\ v_o\ \delta = v_e * \delta - {v_e}^2\ /\ 2\ /\ a_e - v_o * \delta - 1\ /\ 2 * a_o * \delta^2$

definition *checker-r1* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
checker-r1 $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \equiv$
let distance $= s_o - s_e$;
precond $=$ *check-precond-r1* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$;
vo-star $= v_o + a_o * \delta$;
t-stop-o-star $= -\ vo\text{-}star\ /\ a_o$;
t-stop-e $= -\ v_e\ /\ a_e$;
safe-dist0 $=$ *safe-distance-1r* $a_e\ v_e\ \delta$;
safe-dist1 $=$ *safe-distance-2r* $a_e\ v_e\ a_o\ v_o\ \delta$;
safe-dist2 $=$ *safe-distance-4r* $a_e\ v_e\ a_o\ v_o\ \delta$;

```
      safe-dist3    = safe-distance-3r a_e v_e a_o v_o δ
    in
      if ¬ precond then False
      else if distance > safe-dist0 ∨ distance > safe-dist3 then True
       else if (a_o > a_e ∧ vo-star < v_e ∧ t-stop-e < t-stop-o-star) then distance >
safe-dist2
      else distance > safe-dist1

theorem checker-r1-correctness:
  (checker-r1 s_e v_e a_e s_o v_o a_o δ ⟷ check-precond-r1 s_e v_e a_e s_o v_o a_o δ ∧
   safe-distance-normal.no-collision-react a_e v_e s_e a_o v_o s_o δ {0..})
proof
  assume asm: checker-r1 s_e v_e a_e s_o v_o a_o δ
  have pre: check-precond-r1 s_e v_e a_e s_o v_o a_o δ
  proof (rule ccontr)
    assume ¬ check-precond-r1 s_e v_e a_e s_o v_o a_o δ
    with asm show False unfolding checker-r1-def Let-def by auto
  qed
  from pre have sdn': safe-distance-normal a_e v_e s_e a_o v_o s_o δ
    by (unfold-locales) (auto simp add: check-precond-r1-def)
  interpret sdn: safe-distance-normal a_e v_e s_e a_o v_o s_o δ
    rewrites sdn.distance0 = safe-distance0 v_e a_o v_o δ and
             sdn.safe-distance-1r = safe-distance-1r a_e v_e δ and
             sdn.safe-distance-2r = safe-distance-2r a_e v_e a_o v_o δ and
             sdn.safe-distance-4r = safe-distance-4r a_e v_e a_o v_o δ and
             sdn.safe-distance-3r = safe-distance-3r a_e v_e a_o v_o δ
  proof −
    from sdn' show safe-distance-normal a_e v_e s_e a_o v_o s_o δ by auto
  next
    show safe-distance-normal.distance0 v_e a_o v_o δ = safe-distance0 v_e a_o v_o δ
     unfolding safe-distance-normal.distance0-def[OF sdn'] safe-distance0-def by
auto
  next
    show safe-distance-normal.safe-distance-1r a_e v_e δ = safe-distance-1r a_e v_e δ
    unfolding safe-distance-normal.safe-distance-1r-def[OF sdn'] safe-distance-1r-def
by auto
  next
   show safe-distance-normal.safe-distance-2r a_e v_e a_o v_o δ = safe-distance-2r a_e
v_e a_o v_o δ
    unfolding safe-distance-normal.safe-distance-2r-def[OF sdn'] safe-distance-2r-def
by auto
  next
   show safe-distance-normal.safe-distance-4r a_e v_e a_o v_o δ = safe-distance-4r a_e
v_e a_o v_o δ
    unfolding safe-distance-normal.safe-distance-4r-def[OF sdn'] safe-distance-4r-def
by auto
  next
   show safe-distance-normal.safe-distance-3r a_e v_e a_o v_o δ = safe-distance-3r a_e
v_e a_o v_o δ
```

unfolding *safe-distance-normal.safe-distance-3r-def*[*OF sdn'*] *safe-distance-3r-def*
by *auto*
qed
have $0 < \delta$ **and** $\delta \leq - v_o \ / \ a_o$ **using** *pre* **unfolding** *check-precond-r1-def* **by** *auto*
define *so-delta* **where** *so-delta* $= s_o + v_o * \delta + a_o * \delta^2 \ / \ 2$
define *q-e-delta* **where** *q-e-delta* $\equiv s_e + v_e * \delta$
define *u-stop-e* **where** *u-stop-e* $\equiv$ *q-e-delta* $- {v_e}^2 \ / \ (2 * a_e)$
define *vo-star* **where** *vo-star* $= v_o + a_o * \delta$
define *t-stop-o-star* **where** *t-stop-o-star* $\equiv -$ *vo-star* $/ \ a_o$
define *t-stop-e* **where** *t-stop-e* $= - v_e \ / \ a_e$
define *distance* **where** *distance* $\equiv s_o - s_e$
define *distance0* **where** *distance0* $=$ *safe-distance0* $v_e \ a_o \ v_o \ \delta$
define *safe-dist0* **where** *safe-dist0* $=$ *safe-distance-1r* $a_e \ v_e \ \delta$
define *safe-dist2* **where** *safe-dist2* $\equiv$ *safe-distance-4r* $a_e \ v_e \ a_o \ v_o \ \delta$
define *safe-dist1* **where** *safe-dist1* $\equiv$ *safe-distance-2r* $a_e \ v_e \ a_o \ v_o \ \delta$
define *safe-dist3* **where** *safe-dist3* $=$ *safe-distance-3r* $a_e \ v_e \ a_o \ v_o \ \delta$
note *abb* $=$ *so-delta-def q-e-delta-def u-stop-e-def vo-star-def t-stop-o-star-def t-stop-e-def*
distance-def safe-dist2-def safe-dist1-def safe-dist0-def safe-dist3-def distance0-def
consider *distance* $>$ *safe-dist0* $\mid$ *distance* $>$ *safe-dist3* $\mid$ *distance* $\leq$ *safe-dist0* $\wedge$ *distance* $\leq$ *safe-dist3*
by *linarith*
hence *sdn.no-collision-react* $\{0..\}$
proof (*cases*)
case *1*
then show *?thesis* **using** *sdn.sd-1r-correct* **unfolding** *abb* **by** *auto*
next
case *2*
hence *pre2*: *distance* $>$ *distance0* **using** *sdn.distance0-at-most-sd3r* **unfolding** *abb* **by** *auto*
hence *sdn.u* $\delta <$ *sdn.other.s* δ **using** *pre* **unfolding** *sdn.u-def sdn.ego.q-def*
sdn.other.s-def sdn.other.t-stop-def sdn.other.p-def abb check-precond-r1-def sdn.distance0-def
by *auto*
from *pre* **interpret** *sdr*: *safe-distance-no-collsion-delta* $a_e \ v_e \ s_e \ a_o \ v_o \ s_o \ \delta$
by (*unfold-locales*) (*auto simp add*:*check-precond-r1-def* ‹*sdn.u* $\delta <$ *sdn.other.s* δ›)
show *?thesis* **using** *sdr.sd-3r-correct 2 pre* **unfolding** *check-precond-r1-def abb sdn.other.t-stop-def*
by *auto*
next
case *3*
hence *distance* $\leq$ *safe-dist3* **by** *auto*
hence *sdn.other.s* $\delta \leq$ *sdn.u-max* **using** *pre* **unfolding** *check-precond-r1-def sdn.other.s-def sdn.other.t-stop-def*
sdn.other.p-def sdn.u-max-eq sdn.ego.q-def abb sdn.safe-distance-3r-def **by** *auto*

have $(a_o > a_e \wedge \textit{vo-star} < v_e \wedge \textit{t-stop-e} < \textit{t-stop-o-star}) \vee \neg\ (a_o > a_e \wedge \textit{vo-star} < v_e \wedge \textit{t-stop-e} < \textit{t-stop-o-star})$
by *auto*
moreover
{ assume *cond*: $(a_o > a_e \wedge \textit{vo-star} < v_e \wedge \textit{t-stop-e} < \textit{t-stop-o-star})$
with *3 pre* **have** *distance* > *safe-dist2* **using** *asm* **unfolding** *checker-r1-def Let-def abb* **by** *auto*
with *sdn.distance0-at-most-sd4r* **have** *distance* > *distance0* **unfolding** *abb* **using** *cond* **by** *auto*
hence *sdn.u* δ < *sdn.other.s* δ **using** *pre* **unfolding** *sdn.u-def sdn.ego.q-def sdn.other.s-def sdn.other.t-stop-def sdn.other.p-def abb check-precond-r1-def sdn.distance0-def*
by *auto*
from *pre* **interpret** *sdr*: *safe-distance-no-collsion-delta* $a_e\ v_e\ s_e\ a_o\ v_o\ s_o\ \delta$
by (*unfold-locales*) (*auto simp add*:*check-precond-r1-def* ‹*sdn.u* δ < *sdn.other.s* δ›)
from *sdr.sd-4r-correct*[*OF* - ‹*sdn.other.s* $\delta \leq$ *sdn.u-max*›] ‹*distance* > *safe-dist2*›
have *?thesis* **using** *pre cond* **unfolding** *check-precond-r1-def sdn.other.t-stop-def abb* **by** *auto* **}**
moreover
{ assume *not-cond*: $\neg\ (a_o > a_e \wedge \textit{vo-star} < v_e \wedge \textit{t-stop-e} < \textit{t-stop-o-star})$
with *3 pre* **have** *distance* > *safe-dist1* **using** *asm* **unfolding** *checker-r1-def Let-def abb* **by** *auto*
with *sdn.dist0-sd2r-1* **have** *distance* > *distance0* **using** *pre not-cond* **unfolding** *check-precond-r1-def sdn.other.t-stop-def sdn.other.s'-def sdn.other.p'-def abb* **by** (*auto simp add*:*field-simps*)
hence *sdn.u* δ < *sdn.other.s* δ **using** *pre* **unfolding** *sdn.u-def sdn.ego.q-def sdn.other.s-def sdn.other.t-stop-def sdn.other.p-def abb check-precond-r1-def sdn.distance0-def*
by *auto*
from *pre* **interpret** *sdr*: *safe-distance-no-collsion-delta* $a_e\ v_e\ s_e\ a_o\ v_o\ s_o\ \delta$
by (*unfold-locales*) (*auto simp add*:*check-precond-r1-def* ‹*sdn.u* δ < *sdn.other.s* δ›)
from *sdr.sd-2r-correct-for-3r-2*[*OF* - ‹*sdn.other.s* $\delta \leq$ *sdn.u-max*›] *not-cond* ‹*distance* > *safe-dist1*›
have *?thesis* **using** *pre* **unfolding** *abb sdn.other.s'-def check-precond-r1-def sdn.other.t-stop-def sdn.other.p'-def*
by (*auto simp add*:*field-simps*) **}**
ultimately show *?thesis* **by** *auto*
qed
with *pre* **show** *check-precond-r1* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge$ *sdn.no-collision-react* $\{0..\}$ **by** *auto*
next
assume *check-precond-r1* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge$ *safe-distance-normal.no-collision-react* $a_e\ v_e\ s_e\ a_o\ v_o\ s_o\ \delta\ \{0..\}$
hence *pre*: *check-precond-r1* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$ **and** *as2*: *safe-distance-normal.no-collision-react* $a_e\ v_e\ s_e\ a_o\ v_o\ s_o\ \delta\ \{0..\}$
by *auto*

show *checker-r1* s_e v_e a_e s_o v_o a_o δ
proof (*rule ccontr*)
assume *as1*: ¬ *checker-r1* s_e v_e a_e s_o v_o a_o δ
from *pre* **have** $0 < \delta$ **and** $\delta \leq - v_o / a_o$ **unfolding** *check-precond-r1-def* **by** *auto*
define *so-delta* **where** *so-delta* = $s_o + v_o * \delta + a_o * \delta^2 / 2$
define *q-e-delta* **where** *q-e-delta* ≡ $s_e + v_e * \delta$
define *u-stop-e* **where** *u-stop-e* ≡ *q-e-delta* − ${v_e}^2 / (2 * a_e)$
define *vo-star* **where** *vo-star* ≡ $v_o + a_o * \delta$
define *t-stop-o-star* **where** *t-stop-o-star* ≡ − *vo-star* / a_o
define *t-stop-e* **where** *t-stop-e* ≡ − v_e / a_e
define *distance* **where** *distance* ≡ $s_o - s_e$
define *distance0* **where** *distance0* ≡ *safe-distance0* v_e a_o v_o δ
define *safe-dist0* **where** *safe-dist0* ≡ *safe-distance-1r* a_e v_e δ
define *safe-dist2* **where** *safe-dist2* ≡ *safe-distance-4r* a_e v_e a_o v_o δ
define *safe-dist1* **where** *safe-dist1* ≡ *safe-distance-2r* a_e v_e a_o v_o δ
define *safe-dist3* **where** *safe-dist3* ≡ *safe-distance-3r* a_e v_e a_o v_o δ
note *abb* = *so-delta-def q-e-delta-def u-stop-e-def vo-star-def t-stop-o-star-def t-stop-e-def*
distance-def safe-dist2-def safe-dist1-def safe-dist0-def safe-dist3-def distance0-def
from *pre* **have** *sdn'*: *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ
by (*unfold-locales*) (*auto simp add*: *check-precond-r1-def*)
interpret *sdn*: *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ
rewrites *sdn.distance0* = *safe-distance0* v_e a_o v_o δ **and**
sdn.safe-distance-1r = *safe-distance-1r* a_e v_e δ **and**
sdn.safe-distance-2r = *safe-distance-2r* a_e v_e a_o v_o δ **and**
sdn.safe-distance-4r = *safe-distance-4r* a_e v_e a_o v_o δ **and**
sdn.safe-distance-3r = *safe-distance-3r* a_e v_e a_o v_o δ
proof −
from *sdn'* **show** *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ **by** *auto*
next
show *safe-distance-normal.distance0* v_e a_o v_o δ = *safe-distance0* v_e a_o v_o δ
unfolding *safe-distance-normal.distance0-def*[*OF sdn'*] *safe-distance0-def* **by** *auto*
next
show *safe-distance-normal.safe-distance-1r* a_e v_e δ = *safe-distance-1r* a_e v_e δ
unfolding *safe-distance-normal.safe-distance-1r-def*[*OF sdn'*] *safe-distance-1r-def* **by** *auto*
next
show *safe-distance-normal.safe-distance-2r* a_e v_e a_o v_o δ = *safe-distance-2r* a_e v_e a_o v_o δ
unfolding *safe-distance-normal.safe-distance-2r-def*[*OF sdn'*] *safe-distance-2r-def* **by** *auto*
next
show *safe-distance-normal.safe-distance-4r* a_e v_e a_o v_o δ = *safe-distance-4r* a_e v_e a_o v_o δ
unfolding *safe-distance-normal.safe-distance-4r-def*[*OF sdn'*] *safe-distance-4r-def* **by** *auto*

```
  next
    show safe-distance-normal.safe-distance-3r a_e v_e a_o v_o δ = safe-distance-3r
a_e v_e a_o v_o δ
    unfolding safe-distance-normal.safe-distance-3r-def[OF sdn'] safe-distance-3r-def
by auto
  qed
  have ¬ distance > distance0 ∨  distance > distance0 by auto
  moreover
  { assume ¬ distance > distance0
    hence distance ≤ distance0 by auto
     with sdn.cond-3r-1' have sdn.collision-react {0..δ} using pre unfolding
check-precond-r1-def abb
      sdn.other.t-stop-def by auto
    with sdn.collision-react-subset have sdn.collision-react {0..} by auto
    with as2 have False by auto }
  moreover
  { assume if2: distance > distance0
    have ¬ (distance > safe-dist0 ∨ distance > safe-dist3)
    proof (rule ccontr)
      assume ¬ ¬ (safe-dist0 < distance ∨ safe-dist3 < distance)
      hence (safe-dist0 < distance ∨ safe-dist3 < distance) by auto
      with as1 show False using pre if2 unfolding checker-r1-def Let-def abb
        by auto
    qed
    hence if31: distance ≤ safe-dist0 and if32: distance ≤ safe-dist3 by auto
   have sdn.u δ < sdn.other.s δ using if2 pre unfolding sdn.u-def sdn.ego.q-def
      sdn.other.s-def sdn.other.t-stop-def sdn.other.p-def abb check-precond-r1-def
sdn.distance0-def
       by auto
    from pre interpret sdr: safe-distance-no-collsion-delta a_e v_e s_e a_o v_o s_o δ
    by (unfold-locales) (auto simp add:check-precond-r1-def ‹sdn.u δ < sdn.other.s
δ›)
    have  s_o ≤ sdn.u-max using if31 unfolding sdn.u-max-eq sdn.ego.q-def abb
      sdn.safe-distance-1r-def by auto
     have sdn.other.s δ ≤ sdn.u-max using if32 pre unfolding sdn.other.s-def
check-precond-r1-def
    sdn.other.t-stop-def sdn.other.p-def sdn.u-max-eq sdn.ego.q-def abb sdn.safe-distance-3r-def
      by auto
    consider (a_o > a_e ∧ vo-star < v_e ∧ t-stop-e < t-stop-o-star) |
          ¬ (a_o > a_e ∧ vo-star < v_e ∧ t-stop-e < t-stop-o-star) by auto
    hence False
    proof (cases)
      case 1
       hence rest-conjunct:(a_e < a_o ∧ sdn.other.s' δ < v_e ∧ v_e - a_e / a_o *
sdn.other.s' δ < 0)
         using pre unfolding check-precond-r1-def unfolding sdn.other.s'-def
sdn.other.t-stop-def
        sdn.other.p'-def abb by (auto simp add:field-simps)
       from 1 have distance ≤ safe-dist2 using as1 pre if2 if31 if32 unfolding
```

checker-r1-def
Let-def abb **by** *auto*
hence *cond-f*: *sdn.other.s* δ − *sdn.ego.q* $\delta \leq$ *sdr.delayed-safe-distance.snd-safe-distance*
using *pre* **unfolding** *check-precond-r1-def sdn.other.s-def sdn.other.t-stop-def sdn.other.p-def*
sdn.ego.q-def sdr.delayed-safe-distance.snd-safe-distance-def **using** *sdn.other.s'-def*[*of* δ]
unfolding *sdn.other.t-stop-def sdn.other.p'-def abb sdn.safe-distance-4r-def*
by *auto*
have *distance* > *safe-dist1* ∨ *distance* ≤ *safe-dist1* **by** *auto*
moreover
{ assume *distance* > *safe-dist1*
hence *sdn.u-max* < *sdn.other.s-stop* **unfolding** *sdn.u-max-eq sdn.ego.q-def sdn.other.s-t-stop*
sdn.other.p-max-eq abb sdn.safe-distance-2r-def **by** (*auto simp add*:*field-simps*)
from *sdr.cond-3r-2*[*OF* ‹$s_o \leq$ *sdn.u-max*› *this* ‹*sdn.other.s* $\delta \leq$ *sdn.u-max*›]
have *sdn.collision-react* {*0*..} **using** *cond-f rest-conjunct* **by** *auto*
with *as2* **have** *False* **by** *auto* **}**
moreover
{ assume *distance* ≤ *safe-dist1*
hence *sdn.u-max* ≥ *sdn.other.s-stop* **unfolding** *sdn.u-max-eq sdn.ego.q-def sdn.other.s-t-stop*
sdn.other.p-max-eq abb sdn.safe-distance-2r-def **by** (*auto simp add*:*field-simps*)
with *sdn.cond-2r*[*OF this*] **have** *sdn.collision-react* {*0*..} **by** *auto*
with *as2* **have** *False* **by** *auto* **}**
ultimately show *?thesis* **by** *auto*
next
case *2*
hence *distance* ≤ *safe-dist1* **using** *as1 pre if2 if31 if32* **unfolding** *checker-r1-def*
Let-def abb **by** *auto*
hence *sdn.u-max* ≥ *sdn.other.s-stop* **unfolding** *sdn.u-max-eq sdn.ego.q-def sdn.other.s-t-stop*
sdn.other.p-max-eq abb sdn.safe-distance-2r-def **by** (*auto simp add*:*field-simps*)
with *sdn.cond-2r*[*OF this*] **have** *sdn.collision-react* {*0*..} **by** *auto*
with *as2* **show** *False* **by** *auto*
qed }
ultimately show *False* **by** *auto*
qed
qed

definition *check-precond-r2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool* **where**
check-precond-r2 s_e v_e a_e s_o v_o a_o $\delta \longleftrightarrow s_o > s_e \wedge 0 \leq v_e \wedge 0 \leq v_o \wedge a_e < 0 \wedge a_o < 0 \wedge 0 < \delta \wedge \delta > - v_o / a_o$

definition *safe-distance0-2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
safe-distance0-2 v_e a_o v_o $\delta = v_e * \delta + 1 / 2 * {v_o}^2 / a_o$

definition *checker-r2* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *bool*
where
checker-r2 s_e v_e a_e s_o v_o a_o δ ≡
let distance = $s_o - s_e$;
precond = *check-precond-r2* s_e v_e a_e s_o v_o a_o δ;
safe-dist0 = *safe-distance-1r* a_e v_e δ;
safe-dist1 = *safe-distance-2r* a_e v_e a_o v_o δ
in
if ¬ *precond then False*
else if distance > *safe-dist0 then True*
else distance > *safe-dist1*

theorem *checker-r2-correctness*:
(*checker-r2* s_e v_e a_e s_o v_o a_o δ ⟷ *check-precond-r2* s_e v_e a_e s_o v_o a_o δ ∧
safe-distance-normal.no-collision-react a_e v_e s_e a_o v_o s_o δ {0..})
proof
assume *asm*: *checker-r2* s_e v_e a_e s_o v_o a_o δ
have *pre*: *check-precond-r2* s_e v_e a_e s_o v_o a_o δ
proof (*rule ccontr*)
assume ¬ *check-precond-r2* s_e v_e a_e s_o v_o a_o δ
with *asm* **show** *False* **unfolding** *checker-r2-def Let-def* **by** *auto*
qed
from *pre* **have** *sdn'*: *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ
by (*unfold-locales*) (*auto simp add*: *check-precond-r2-def*)
interpret *sdn*: *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ
rewrites *sdn.distance0-2* = *safe-distance0-2* v_e a_o v_o δ **and**
sdn.safe-distance-1r = *safe-distance-1r* a_e v_e δ **and**
sdn.safe-distance-2r = *safe-distance-2r* a_e v_e a_o v_o δ
proof −
from *sdn'* **show** *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ **by** *auto*
next
show *safe-distance-normal.distance0-2* v_e a_o v_o δ = *safe-distance0-2* v_e a_o v_o δ
unfolding *safe-distance-normal.distance0-2-def*[*OF sdn'*] *safe-distance0-2-def*
by *auto*
next
show *safe-distance-normal.safe-distance-1r* a_e v_e δ = *safe-distance-1r* a_e v_e δ
unfolding *safe-distance-normal.safe-distance-1r-def*[*OF sdn'*] *safe-distance-1r-def*
by *auto*
next
show *safe-distance-normal.safe-distance-2r* a_e v_e a_o v_o δ = *safe-distance-2r* a_e v_e a_o v_o δ
unfolding *safe-distance-normal.safe-distance-2r-def*[*OF sdn'*] *safe-distance-2r-def*
by *auto*
qed
have 0 < δ **and** δ > − v_o / a_o **using** *pre* **unfolding** *check-precond-r2-def* **by** *auto*
define *distance* **where** *distance* ≡ $s_o - s_e$
define *distance0-2* **where** *distance0-2* = *safe-distance0-2* v_e a_o v_o δ

define *safe-dist0* **where** *safe-dist0 = safe-distance-1r* a_e v_e δ
define *safe-dist1* **where** *safe-dist1* $\equiv$ *safe-distance-2r* a_e v_e a_o v_o δ
note *abb = distance-def safe-dist1-def safe-dist0-def distance0-2-def*
consider *distance > safe-dist0* | *distance* $\leq$ *safe-dist0*
by *linarith*
hence *sdn.no-collision-react* $\{0..\}$
proof (*cases*)
case *1*
then show *?thesis* **using** *sdn.sd-1r-correct* **unfolding** *abb* **by** *auto*
next
case *2*
hence (s_o $\leq$ *sdn.u-max*) **using** *distance-def safe-dist0-def sdn.sd-1r-eq* **by** *linarith*
with *2 pre* **have** *distance > safe-dist1* **using** *asm* **unfolding** *checker-r2-def Let-def abb* **by** *auto*
with *sdn.dist0-sd2r-2* **have** *distance > distance0-2* **using** *abb* ‹$-$ v_o / a_o < δ› **by** *auto*
hence *sdn.u* δ < *sdn.other.s* δ **using** *abb sdn.distance0-2-eq* ‹δ > $-$ v_o / a_o› *sdn.other.t-stop-def* **by** *auto*
have *sdn.u-max < sdn.other.s* δ **using** *abb sdn.sd2r-eq* ‹δ > $-$ v_o / a_o› *sdn.other.t-stop-def* ‹*distance > safe-dist1*› **by** *auto*
from *pre* **interpret** *sdr*: *safe-distance-no-collsion-delta* a_e v_e s_e a_o v_o s_o δ
by (*unfold-locales*) (*auto simp add*:*check-precond-r2-def* ‹*sdn.u* δ < *sdn.other.s* δ›)
from *sdr.sd-2r-correct-for-3r-3*[*OF*] ‹*distance > safe-dist1*› ‹*sdn.u* δ < *sdn.other.s* δ› ‹*sdn.u-max < sdn.other.s* δ›
show *?thesis* **using** *pre* **unfolding** *abb sdn.other.s'-def check-precond-r2-def sdn.other.t-stop-def sdn.other.p'-def*
by (*auto simp add*:*field-simps*)
qed
with *pre* **show** *check-precond-r2* s_e v_e a_e s_o v_o a_o δ $\wedge$ *sdn.no-collision-react* $\{0..\}$ **by** *auto*
next
assume *check-precond-r2* s_e v_e a_e s_o v_o a_o δ $\wedge$ *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ $\{0..\}$
hence *pre*: *check-precond-r2* s_e v_e a_e s_o v_o a_o δ **and** *as2*: *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ $\{0..\}$
by *auto*
show *checker-r2* s_e v_e a_e s_o v_o a_o δ
proof (*rule ccontr*)
assume *as1*: $\neg$ *checker-r2* s_e v_e a_e s_o v_o a_o δ
from *pre* **have** $0 < \delta$ **and** $\delta > -$ v_o / a_o **unfolding** *check-precond-r2-def* **by** *auto*
define *distance* **where** *distance* $\equiv$ $s_o - s_e$
define *distance0-2* **where** *distance0-2 = safe-distance0-2* v_e a_o v_o δ
define *safe-dist0* **where** *safe-dist0 = safe-distance-1r* a_e v_e δ
define *safe-dist1* **where** *safe-dist1* $\equiv$ *safe-distance-2r* a_e v_e a_o v_o δ
note *abb = distance-def safe-dist1-def safe-dist0-def distance0-2-def*
from *pre* **have** *sdn'*: *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ

by (*unfold-locales*) (*auto simp add*: *check-precond-r2-def*)
interpret *sdn*: *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ
rewrites *sdn.distance0-2* = *safe-distance0-2* v_e a_o v_o δ **and**
sdn.safe-distance-1r = *safe-distance-1r* a_e v_e δ **and**
sdn.safe-distance-2r = *safe-distance-2r* a_e v_e a_o v_o δ
proof −
from *sdn'* **show** *safe-distance-normal* a_e v_e s_e a_o v_o s_o δ **by** *auto*
next
show *safe-distance-normal.distance0-2* v_e a_o v_o δ = *safe-distance0-2* v_e a_o v_o δ
unfolding *safe-distance-normal.distance0-2-def*[*OF sdn'*] *safe-distance0-2-def* **by** *auto*
next
show *safe-distance-normal.safe-distance-1r* a_e v_e δ = *safe-distance-1r* a_e v_e δ
unfolding *safe-distance-normal.safe-distance-1r-def*[*OF sdn'*] *safe-distance-1r-def* **by** *auto*
next
show *safe-distance-normal.safe-distance-2r* a_e v_e a_o v_o δ = *safe-distance-2r* a_e v_e a_o v_o δ
unfolding *safe-distance-normal.safe-distance-2r-def*[*OF sdn'*] *safe-distance-2r-def* **by** *auto*
qed
have ¬ *distance* > *distance0-2* ∨ *distance* > *distance0-2* **by** *auto*
moreover
{ assume ¬ *distance* > *distance0-2*
hence *distance* ≤ *distance0-2* **by** *auto*
with *sdn.cond-3r-1'-2* **have** *sdn.collision-react* {*0*..*δ*} **using** *pre* **unfolding** *check-precond-r2-def abb sdn.other.t-stop-def* **by** *auto*
with *sdn.collision-react-subset* **have** *sdn.collision-react* {*0*..} **by** *auto*
with *as2* **have** *False* **by** *auto* **}**
moreover
{ assume *if2*: *distance* > *distance0-2*
have ¬ (*distance* > *safe-dist0*)
proof (*rule ccontr*)
assume ¬ ¬ (*safe-dist0* < *distance*)
hence (*safe-dist0* < *distance*) **by** *auto*
with *as1* **show** *False* **using** *pre if2* **unfolding** *checker-r2-def Let-def abb* **by** *auto*
qed
hence *if3*: *distance* ≤ *safe-dist0* **by** *auto*
with *pre* **have** *distance* ≤ *safe-dist1* **using** *as1* **unfolding** *checker-r2-def Let-def abb* **by** *auto*

have *sdn.u δ* < *sdn.other.s δ* **using** *abb if2 sdn.distance0-2-eq* ‹*δ* > − v_o / a_o› *sdn.other.t-stop-def* **by** *auto*
from *pre* **interpret** *sdr*: *safe-distance-no-collsion-delta* a_e v_e s_e a_o v_o s_o δ
by (*unfold-locales*) (*auto simp add*:*check-precond-r2-def* ‹*sdn.u δ* < *sdn.other.s δ*›)
have *sdn.u-max* ≥ *sdn.other.s δ* **using** *abb sdn.sd2r-eq* ‹*δ* > − v_o / a_o›

```
sdn.other.t-stop-def ‹distance ≤ safe-dist1› by auto
      with ‹δ > − v_o / a_o› have sdn.u-max ≥ sdn.other.s-stop
      using sdn.other.s-mono sdn.other.t-stop-nonneg sdn.other.p-t-stop sdn.other.p-zero
sdn.other.t-stop-def
        apply (auto simp: sdn.other.s-def movement.t-stop-def split: if-splits)
        using sdn.other.p-zero by auto
      hence sdn.collision-react {0..} using sdn.cond-2r by auto
      with as2 have False by auto }
    ultimately show False by auto
  qed
qed
```

Combine the two checkers into one.

```
definition check-precond-r :: real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ real ⇒
bool where
  check-precond-r s_e v_e a_e s_o v_o a_o δ ⟷ s_o > s_e ∧ 0 ≤ v_e ∧ 0 ≤ v_o ∧ a_e < 0 ∧
a_o < 0 ∧ 0 < δ

definition checker-r :: real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ real ⇒ bool
where
  checker-r s_e v_e a_e s_o v_o a_o δ ≡
    let distance      = s_o − s_e;
    precond       = check-precond-r s_e v_e a_e s_o v_o a_o δ;
        vo-star       = v_o + a_o ∗ δ;
        t-stop-o-star = −vo-star / a_o;
        t-stop-e      = −v_e / a_e;
        t-stop-o      = −v_o / a_o;
        safe-dist0    = safe-distance-1r a_e v_e δ;
        safe-dist1    = safe-distance-2r a_e v_e a_o v_o δ;
        safe-dist2    = safe-distance-4r a_e v_e a_o v_o δ;
        safe-dist3    = safe-distance-3r a_e v_e a_o v_o δ
    in
      if ¬ precond then False
      else if distance > safe-dist0 then True
      else if δ ≤ t-stop-o ∧ distance > safe-dist3 then True
       else if δ ≤ t-stop-o ∧ (a_o > a_e ∧ vo-star < v_e ∧ t-stop-e < t-stop-o-star)
then distance > safe-dist2
      else distance > safe-dist1

theorem checker-eq-1:
  checker-r s_e v_e a_e s_o v_o a_o δ ∧ δ ≤ − v_o / a_o  ⟷ checker-r1 s_e v_e a_e s_o v_o a_o
δ
proof −
  have checker-r s_e v_e a_e s_o v_o a_o δ ∧ δ ≤ − v_o / a_o ⟷ check-precond-r s_e v_e
a_e s_o v_o a_o δ
    ∧ (s_o − s_e > safe-distance-1r a_e v_e δ
        ∨ s_o − s_e > safe-distance-3r a_e v_e a_o v_o δ
        ∨ (((a_o > a_e ∧ v_o + a_o ∗ δ < v_e ∧ − v_e / a_e < − (v_o + a_o ∗ δ) / a_o) ⟶
s_o − s_e > safe-distance-4r a_e v_e a_o v_o δ)
```

$\wedge$ ($\neg$ ($a_o > a_e \wedge v_o + a_o * \delta < v_e \wedge - v_e \,/\, a_e < - (v_o + a_o * \delta) \,/\, a_o$) $\longrightarrow$ $s_o - s_e >$ *safe-distance-2r* $a_e\ v_e\ a_o\ v_o\ \delta$)))
$\wedge\ \delta \leq - v_o \,/\, a_o$ **using** *checker-r-def* **by** *metis*
also have ... $\longleftrightarrow$ *check-precond-r1* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$
$\wedge$ ($s_o - s_e >$ *safe-distance-1r* $a_e\ v_e\ \delta$
$\vee\ s_o - s_e >$ *safe-distance-3r* $a_e\ v_e\ a_o\ v_o\ \delta$
$\vee$ (($a_o > a_e \wedge v_o + a_o * \delta < v_e \wedge - v_e \,/\, a_e < - (v_o + a_o * \delta) \,/\, a_o$) $\longrightarrow$ $s_o - s_e >$ *safe-distance-4r* $a_e\ v_e\ a_o\ v_o\ \delta$)
$\wedge$ ($\neg$ ($a_o > a_e \wedge v_o + a_o * \delta < v_e \wedge - v_e \,/\, a_e < - (v_o + a_o * \delta) \,/\, a_o$) $\longrightarrow$ $s_o - s_e >$ *safe-distance-2r* $a_e\ v_e\ a_o\ v_o\ \delta$)))
by (*auto simp add*:*check-precond-r-def check-precond-r1-def*)
also have ... $\longleftrightarrow$ *checker-r1* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$ **by** (*metis checker-r1-def*)
finally show *?thesis* **by** *auto*
qed

theorem *checker-eq-2*:
checker-r $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge \delta > - v_o \,/\, a_o \longleftrightarrow$ *checker-r2* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$
proof −
have *checker-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge \delta > - v_o \,/\, a_o \longleftrightarrow$ *check-precond-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge$ ($\neg$ *check-precond-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \vee$
$s_o - s_e >$ *safe-distance-1r* $a_e\ v_e\ \delta \vee$
($\delta \leq - v_o \,/\, a_o \wedge s_o - s_e >$ *safe-distance-3r* $a_e\ v_e\ a_o\ v_o\ \delta$) $\vee$
($\delta \leq - v_o \,/\, a_o \wedge a_o > a_e \wedge v_o + a_o * \delta < v_e \wedge - v_e \,/\, a_e < - (v_o + a_o * \delta) \,/\, a_o \wedge s_o - s_e >$ *safe-distance-4r* $a_e\ v_e\ a_o\ v_o\ \delta$) $\vee$
$s_o - s_e >$ *safe-distance-2r* $a_e\ v_e\ a_o\ v_o\ \delta$) $\wedge\ \delta > - v_o \,/\, a_o$ **unfolding** *checker-r-def Let-def if-splits* **by** *auto*
also have
... $\longleftrightarrow$ *check-precond-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$
$\wedge$ ($s_o - s_e >$ *safe-distance-1r* $a_e\ v_e\ \delta \vee s_o - s_e >$ *safe-distance-2r* $a_e\ v_e\ a_o\ v_o\ \delta$)
$\wedge\ \delta > - v_o \,/\, a_o$ **by** (*auto simp add*:*HOL.disjE*)
also have
... $\longleftrightarrow$ *check-precond-r2* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$
$\wedge$ ($s_o - s_e >$ *safe-distance-1r* $a_e\ v_e\ \delta \vee s_o - s_e >$ *safe-distance-2r* $a_e\ v_e\ a_o\ v_o\ \delta$)
by (*auto simp add*:*check-precond-r-def check-precond-r2-def*)
also have ... $\longleftrightarrow$ *checker-r2* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$ **by** (*auto simp add*:*checker-r2-def Let-def if-splits*)
thus *?thesis* **using** *calculation* **by** *auto*
qed

theorem *checker-r-correctness*:
(*checker-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \longleftrightarrow$ *check-precond-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge$ *safe-distance-normal.no-collision-react* $a_e\ v_e\ s_e\ a_o\ v_o\ s_o\ \delta\ \{0..\}$)
proof −
have *checker-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \longleftrightarrow$ (*checker-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge \delta \leq - v_o \,/\, a_o$) $\vee$ (*checker-r* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \wedge \delta > - v_o \,/\, a_o$) **by** *auto*
also have ... $\longleftrightarrow$ *checker-r1* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta \vee$ *checker-r2* $s_e\ v_e\ a_e\ s_o\ v_o\ a_o\ \delta$ **using** *checker-eq-1 checker-eq-2* **by** *auto*

also have ... $\longleftrightarrow$ (*check-precond-r1* s_e v_e a_e s_o v_o a_o δ $\wedge$ *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ $\{0..\}$)
$\vee$ (*check-precond-r2* s_e v_e a_e s_o v_o a_o δ $\wedge$ *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ $\{0..\}$)
using *checker-r1-correctness checker-r2-correctness* **by** *auto*
also have ... $\longleftrightarrow$ ($\delta \leq -\ v_o\ /\ a_o\ \wedge$ *check-precond-r* s_e v_e a_e s_o v_o a_o δ $\wedge$ *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ $\{0..\}$)
$\vee$ ($\delta > -\ v_o\ /\ a_o \wedge$ *check-precond-r* s_e v_e a_e s_o v_o a_o δ $\wedge$ *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ $\{0..\}$)
by (*auto simp add*:*check-precond-r-def check-precond-r1-def check-precond-r2-def*)
also have ... $\longleftrightarrow$ *check-precond-r* s_e v_e a_e s_o v_o a_o δ $\wedge$ *safe-distance-normal.no-collision-react* a_e v_e s_e a_o v_o s_o δ $\{0..\}$
by *auto*
finally show *?thesis* **by** *auto*
qed

end

3 Evaluation

theory *Evaluation*
imports
Safe-Distance
HOL−Library.Float
begin

3.1 Code Generation Setup for Numeric Values

definition *real-div-down* :: *nat* ⇒ *int* ⇒ *int* ⇒ *real* **where**
real-div-down p i j = *truncate-down* (*Suc p*) (*i* / *j*)

definition *real-div-up* :: *nat* ⇒ *int* ⇒ *int* ⇒ *real* **where**
real-div-up p i j = *truncate-up* (*Suc p*) (*i* / *j*)

context includes *float.lifting* **begin**
lift-definition *float-div-down* :: *nat* ⇒ *int* ⇒ *int* ⇒ *float* **is** *real-div-down*
by (*simp add*: *real-div-down-def*)

lift-definition *float-div-up* :: *nat* ⇒ *int* ⇒ *int* ⇒ *float* **is** *real-div-up*
by (*simp add*: *real-div-up-def*)
end

lemma *compute-float-div-up*[*code*]: *float-div-up p i j* = − *float-div-down p* (−*i*) *j*
including *float.lifting*
by *transfer* (*simp add*: *real-div-up-def real-div-down-def truncate-up-eq-truncate-down*)

lemma *compute-float-div-down*[*code*]:
float-div-down prec m1 m2 = *lapprox-rat* (*Suc prec*) *m1 m2*
including *float.lifting* **by** *transfer* (*simp add*: *real-div-down-def*)

definition *real2-of-real* :: *nat* ⇒ *real* ⇒ (*real* ∗ *real*) **where**
real2-of-real p x = (*truncate-down* (*Suc p*) *x*, *truncate-up* (*Suc p*) *x*)

context includes *float.lifting* **begin**
lift-definition *float2-of-real* :: *nat* ⇒ *real* ⇒ *float* × *float* **is** *real2-of-real*
by (*auto simp*: *real2-of-real-def*)
end

definition *float2-opt-of-real* :: *nat* ⇒ *real* ⇒ *float interval option* **where**
float2-opt-of-real prec x = *Interval'* (*fst* (*float2-of-real prec x*)) (*snd* (*float2-of-real prec x*))

hide-const (**open**) *Fraction-Field.Fract*
lemma *real-of-rat-Fract*[*simp*]: *real-of-rat* (*Fract a b*) = *a* / *b*
by (*simp add*: *Fract-of-int-quotient of-rat-divide*)

lemma [*code*]: *float2-of-real p* (*Ratreal r*) =
(*let* (*a*, *b*) = *quotient-of r in*
(*float-div-down p a b*, *float-div-up p a b*))
including *float.lifting*
apply *transfer*
apply (*auto split*: *prod.split simp*: *real2-of-real-def real-div-down-def real-div-up-def*)
apply (*metis of-rat-divide of-rat-of-int-eq quotient-of-div*)+
done

fun *real-of-dec* :: *integer* × *integer* ⇒ *real* **where**
real-of-dec (*m*, *e*) =
real-of-int (*int-of-integer m*) ∗
(*if e* ≥ *0 then 10* ^ (*nat-of-integer e*) *else inverse* (*10* ^ (*nat* (−(*int-of-integer e*)))))

lemma *real-of-dec* (*m*, *e*) = *int-of-integer m* ∗ *10 powr* (*int-of-integer e*)
proof −
have *1*: *e* ≥ *0* ⟹ *real* (*nat-of-integer e*) = *real-of-int* (*int-of-integer e*)
using *less-eq-integer.rep-eq nat-of-integer.rep-eq* **by** *auto*
have *2*: *e* ≤ *0* ⟹ *real-of-int* (*int-of-integer e*) = − *real* (*nat* (− *int-of-integer e*))
using *less-eq-integer.rep-eq* **by** *auto*
show *?thesis*
using *1*
apply (*auto simp*: *powr-realpow*[*symmetric*] *divide-simps*)
apply (*subst* (*asm*) *2*)
apply (*auto simp*: *powr-add*[*symmetric*])
done
qed

3.2 Data Evaluation

definition *trans6* **where**
  trans6 c chk se ve ae so vo ao =
    chk (c se) (c ve) (c ae) (c so) (c vo) (c ao)

definition *checker-dec* **where**
  checker-dec chk p u =
    trans6 (float2-opt-of-real (nat-of-integer u) o real-of-dec) (chk (nat-of-integer p))

definition *checker-interval = checker-dec checker'*
definition *checker-symbolic = trans6 real-of-dec symbolic-checker*
definition *checker-rational = trans6 real-of-dec checker*
lemmas[*code*] = *movement.p-def*

ML ‹
  exception InvalidArgument of string;

  fun split-string s = String.fields (fn c => c = the (Char.fromString s))

  fun dec-of-string s =
    case split-string . s
    of [r] => (the (IntInf.fromString r), 0)
    | [d1, d2] => (the (IntInf.fromString (d1 ^ d2)), ~ (String.size d2))
    | - => raise (InvalidArgument s)

  fun check-string chk data =
    case split-string , data of
      [-, so, -, ve, ae, -, vo, ao] => chk data (0, 0)
        (dec-of-string ve) (dec-of-string ae) (dec-of-string so) (dec-of-string vo) (dec-of-string ao)
    | - => raise (InvalidArgument data)
›

The precision of the input data is roughly 12 and yields similar performance as Sturm

ML ‹
val prec = 12

local

*exception Result of int * int*;

fun check-line chk n l (y, i) =
  let
    val l = String.substring (l, 0, String.size l - 1)
    val c = check-string chk l
  in if i < n then (if c then y + 1 else y, i + 1) else raise Result (y, i) end

```
in

fun check-file chk path n =
  let
    val data =
      path
      |> Bytes.read
      |> XZ.uncompress
      |> Bytes.trim-split-lines
  in
    fold (check-line chk n) data (0, 0)
  end
    handle Result res => res;

end

val check-file-symbolic             = check-file (fn - => code ‹checker-symbolic›)
fun check-file-interval prec uncer = check-file (fn - => code ‹checker-interval› prec
uncer)
val check-file-rational             = check-file (fn - => code ‹checker-rational›)
›
```

Number of data points:

- data01: 1121215
- data02: 1341135
- data03: 1452656

```
ML ‹
  val data01 = master-dir + path ‹data/data01.csv.xz›
  val data02 = master-dir + path ‹data/data02.csv.xz›
  val data03 = master-dir + path ‹data/data03.csv.xz›
›

ML ‹
    val t-start1 = Timing.start ();
    val result1 = check-file-rational data01 100;
    val t-end1 = Timing.result t-start1;
    assert (result1 = (96, 100));
›

ML ‹
    val t-start2 = Timing.start ();
    val result2 = check-file-rational data02 100;
    val t-end2 = Timing.result t-start2;
    assert (result2 = (100, 100));
›
```

ML ‹
val t-start3 = Timing.start ();
val result3 = check-file-rational data03 100;
val t-end3 = Timing.result t-start3;
assert *(result3 = (76, 100));*
›

Precision: 12, Uncertainty: 7 digits

ML ‹***assert*** *(check-file-interval 12 7 data01 100 = (95, 100))*›
ML ‹***assert*** *(check-file-rational data01 100 = (96, 100))*›
ML ‹***assert*** *(check-file-symbolic data01 100 = (96, 100))*›

end

References

[1] A. Rizaldi, F. Immler, and M. Althoff. A formally verified checker of the safe distance traffic rules for autonomous vehicles. In S. Rayadurgam and O. Tkachuk, editors, *NASA Formal Methods*, pages 175–190, Cham, 2016. Springer International Publishing.