CPSVerification

CPSVerification

July 4, 2019

Contents

theory *hs-prelims*
imports *Ordinary-Differential-Equations.Initial-Value-Problem*

begin

Chapter 1

Hybrid Systems Preliminaries

This chapter contains preliminary lemmas for verification of Hybrid Systems.

1.1 Miscellaneous

1.1.1 Functions

lemma *case-of-fst*[*simp*]:($\lambda x.\ case\ x\ of\ (t, x) \Rightarrow f\ t) = (\lambda\ x.\ (f \circ fst)\ x)$
by *auto*

lemma *case-of-snd*[*simp*]:($\lambda x.\ case\ x\ of\ (t, x) \Rightarrow f\ x) = (\lambda\ x.\ (f \circ snd)\ x)$
by *auto*

1.1.2 Orders

lemma *cSup-eq-linorder*:
fixes c::$'a$::*conditionally-complete-linorder*
assumes $X \neq \{\}$ **and** $\forall\, x \in X.\ x \leq c$
and *bdd-above* X **and** $\forall\, y{<}c.\ \exists\, x{\in}X.\ y < x$
shows *Sup* $X = c$
apply(*rule order-antisym*)
using *assms* **apply**(*simp add*: *cSup-least*)
using *assms* **by**(*subst le-cSup-iff*)

lemma *cSup-eq*:
fixes c::$'a$::*conditionally-complete-lattice*
assumes $\forall\, x \in X.\ x \leq c$ **and** $\exists\, x \in X.\ c \leq x$
shows *Sup* $X = c$
apply(*rule order-antisym*)
apply(*rule cSup-least*)
using *assms* **apply**(*blast*, *blast*)
using *assms*(*2*) **apply** *safe*

apply(*subgoal-tac* $x \leq Sup\ X$, *simp*)
by (*metis assms*(*1*) *cSup-eq-maximum eq-iff*)

lemma *bdd-above-ltimes*:
fixes $c::'a::linordered\text{-}ring\text{-}strict$
assumes $c \geq 0$ **and** *bdd-above X*
shows *bdd-above* $\{c * x \mid x.\ x \in X\}$
using *assms* **unfolding** *bdd-above-def* **apply** *clarsimp*
apply(*rule-tac* $x{=}c * M$ **in** *exI*, *clarsimp*)
using *mult-left-mono* **by** *blast*

lemma *finite-nat-minimal-witness*:
fixes $P :: ('a::finite) \Rightarrow nat \Rightarrow bool$
assumes $\forall\, i.\ \exists\, N::nat.\ \forall\, n \geq N.\ P\ i\ n$
shows $\exists\, N.\ \forall\, i.\ \forall\, n \geq N.\ P\ i\ n$
proof−
let $?bound\ i = (LEAST\ N.\ \forall\, n \geq N.\ P\ i\ n)$
let $?N = Max\ \{?bound\ i \mid i.\ i \in UNIV\}$
{fix $n::nat$ **and** $i::'a$
obtain M **where** $\forall\, n \geq M.\ P\ i\ n$
using *assms* **by** *blast*
hence *obs*: $\forall\ m \geq ?bound\ i.\ P\ i\ m$
using *LeastI*[*of* $\lambda N.\ \forall\, n \geq N.\ P\ i\ n$] **by** *blast*
assume $n \geq ?N$
have *finite* $\{?bound\ i \mid i.\ i \in UNIV\}$
using *finite-Atleast-Atmost-nat* **by** *fastforce*
hence $?N \geq ?bound\ i$
using *Max-ge* **by** *blast*
hence $n \geq ?bound\ i$
using $\langle n \geq ?N \rangle$ **by** *linarith*
hence $P\ i\ n$
using *obs* **by** *blast***}**
thus $\exists\, N.\ \forall\, i\ n.\ N \leq n \longrightarrow P\ i\ n$
by *blast*
qed

1.1.3 Real Numbers

lemma *sqrt-le-itself*: $1 \leq x \Longrightarrow sqrt\ x \leq x$
by (*metis basic-trans-rules*(*23*) *monoid-mult-class.power2-eq-square more-arith-simps*(*6*)

mult-left-mono real-sqrt-le-iff' zero-le-one)

lemma *sqrt-real-nat-le*:*sqrt* (*real n*) $\leq$ *real n*
by (*metis* (*full-types*) *abs-of-nat le-square of-nat-mono of-nat-mult real-sqrt-abs2 real-sqrt-le-iff*)

lemma *sq-le-cancel*:
shows $(a::real) \geq 0 \Longrightarrow b \geq 0 \Longrightarrow a\hat{\ }2 \leq b * a \Longrightarrow a \leq b$

and $(a{::}real) \geq 0 \Longrightarrow b \geq 0 \Longrightarrow a\hat{\ }2 \leq a * b \Longrightarrow a \leq b$
apply(*metis less-eq-real-def mult.commute mult-le-cancel-left semiring-normalization-rules*(*29*))
by(*metis less-eq-real-def mult-le-cancel-left semiring-normalization-rules*(*29*))

named-theorems *trig-simps simplification rules for trigonometric identities*

lemmas *trig-identities* = *sin-squared-eq*[*THEN sym*] *cos-squared-eq*[*symmetric*] *cos-diff*[*symmetric*] *cos-double*

declare *sin-minus* [*trig-simps*]
and *cos-minus* [*trig-simps*]
and *trig-identities*(*1*,*2*) [*trig-simps*]
and *sin-cos-squared-add* [*trig-simps*]
and *sin-cos-squared-add2* [*trig-simps*]
and *sin-cos-squared-add3* [*trig-simps*]
and *trig-identities*(*3*) [*trig-simps*]

lemma *sin-cos-squared-add4* [*trig-simps*]:
fixes $x :: 'a:: \{banach,real\text{-}normed\text{-}field\}$
shows $x * (sin\ t)^2 + x * (cos\ t)^2 = x$
by (*metis mult.right-neutral semiring-normalization-rules*(*34*) *sin-cos-squared-add*)

lemma [*trig-simps*, *simp*]:
fixes $x :: 'a:: \{banach,real\text{-}normed\text{-}field\}$
shows $(x * cos\ t - y * sin\ t)^2 + (x * sin\ t + y * cos\ t)^2 = x^2 + y^2$
proof−
have $(x * cos\ t - y * sin\ t)^2 = x^2 * (cos\ t)^2 + y^2 * (sin\ t)^2 - 2 * (x * cos\ t) * (y * sin\ t)$
by(*simp add*: *power2-diff power-mult-distrib*)
also have $(x * sin\ t + y * cos\ t)^2 = y^2 * (cos\ t)^2 + x^2 * (sin\ t)^2 + 2 * (x * cos\ t) * (y * sin\ t)$
by(*simp add*: *power2-sum power-mult-distrib*)
ultimately show $(x * cos\ t - y * sin\ t)^2 + (x * sin\ t + y * cos\ t)^2 = x^2 + y^2$

by (*simp add*: *Groups.mult-ac*(*2*) *Groups.mult-ac*(*3*) *right-diff-distrib sin-squared-eq*)

qed

thm *trig-simps*

1.2 Calculus

1.2.1 Single variable Derivatives

notation *has-derivative* ((*1*(*D* - ↦ (-))/ -) [*65*,*65*] *61*)
notation *has-vderiv-on* ((*1 D* - = (-)/ *on* -) [*65*,*65*] *61*)
notation *norm* ((*1*‖-‖) [*65*] *61*)

lemma *closed-segment-mvt*:

fixes f :: *real* ⇒ *real*
assumes $(\bigwedge r.\ r\in\{a{-}{-}b\} \Longrightarrow (D\ f \mapsto (f'\ r)\ (at\ r\ within\ \{a{-}{-}b\})))$ **and** $a \le b$
shows $\exists\, r\in\{a{-}{-}b\}.\ f\ b - f\ a = f'\ r\ (b - a)$
using *assms closed-segment-eq-real-ivl mvt-very-simple* **by** *auto*

lemma *exp-scaleR-has-derivative-right*[*derivative-intros*]:
fixes f::*real* ⇒ *real*
assumes $D\ f \mapsto f'\ at\ x\ within\ s$ **and** $(\lambda h.\ f'\ h *_R (exp\ (f\ x *_R A) * A)) = g'$
shows $D\ (\lambda x.\ exp\ (f\ x *_R A)) \mapsto g'\ at\ x\ within\ s$
proof −
from *assms* **have** *bounded-linear* f' **by** *auto*
with *real-bounded-linear* **obtain** m **where** f': $f' = (\lambda h.\ h * m)$ **by** *blast*
show *?thesis*
using *vector-diff-chain-within*[*OF - exp-scaleR-has-vector-derivative-right*, *of f m x s A*] *assms* f'
by (*auto simp*: *has-vector-derivative-def o-def*)
qed

named-theorems *poly-derivatives compilation of derivatives for kinematics and polynomials.*

declare *has-vderiv-on-const* [*poly-derivatives*]

lemma *has-vector-derivative-mult-const*: $((*)\ a\ has\text{-}vector\text{-}derivative\ a)\ (at\ x\ within\ T)$
by (*auto intro*: *derivative-eq-intros*)

lemma *has-derivative-mult-const*: $D\ (*)\ a \mapsto (\lambda x.\ x *_R a)\ at\ x\ within\ T$
using *has-vector-derivative-mult-const* **unfolding** *has-vector-derivative-def* **by** *simp*

lemma *has-derivative-quadratic-monomial*:
fixes a :: *real*
shows $D\ (\lambda t.\ a * t^2) \mapsto (\lambda t.\ a * (2 * x * t))\ at\ x\ within\ T$
apply(*rule-tac* $g'1{=}\lambda\ t.\ 2 * x * t$ **in** *derivative-eq-intros*(*6*))
apply(*rule-tac* $f'1{=}\lambda\ t.\ t$ **in** *derivative-eq-intros*(*15*))
by (*auto intro*: *derivative-eq-intros*)

lemma *has-derivative-quadratic-monomial-halfed*:
fixes a :: *real*
shows $D\ (\lambda t.\ a * t^2\ /\ 2) \mapsto (*)\ (a * x)\ at\ x\ within\ T$
apply(*rule-tac* $f'1{=}\lambda t.\ a * (2 * x * t)$ **and** $g'1{=}\lambda\ x.\ 0$ **in** *derivative-eq-intros*(*18*))
using *has-derivative-quadratic-monomial* **by** *auto*

lemma [*poly-derivatives*]: $D\ (\lambda t.\ a * t^2\ /\ 2) = (*)\ a\ on\ T$
apply(*simp add*: *has-vderiv-on-def has-vector-derivative-def*, *clarify*)
using *has-derivative-quadratic-monomial-halfed* **by** (*simp add*: *mult-commute-abs*)

lemma [*poly-derivatives*]: $D\ (\lambda t.\ a * t^2\ /\ 2 + v * t + x) = (\lambda t.\ a * t + v)\ on\ T$
  apply(*rule-tac* $f'=\lambda\ x.\ a * x + v$ **and** $g'1=\lambda\ x.\ 0$ **in** *derivative-intros*(*191*))
    apply(*rule-tac* $f'1=\lambda\ x.\ a * x$ **and** $g'1=\lambda\ x.\ v$ **in** *derivative-intros*(*191*))
  using *poly-derivatives*(*2*) **by**(*auto intro*: *derivative-intros*)

lemma [*poly-derivatives*]: $D\ (\lambda r.\ a * r + v) = (\lambda t.\ a)\ on\ T$
  apply(*rule-tac* $f'1=\lambda\ x.\ a$ **and** $g'1=\lambda\ x.\ 0$ **in** *derivative-intros*(*191*))
  unfolding *has-vderiv-on-def* **by**(*auto intro*: *derivative-eq-intros*)

lemma [*poly-derivatives*]: $D\ (\lambda t.\ v * t - a * t^2\ /\ 2 + x) = (\lambda x.\ v - a * x)\ on\ T$
  apply(*subgoal-tac* $D\ (\lambda t.\ - a * t^2\ /\ 2 + v * t\ +x) = (\lambda x.\ - a * x + v)\ on\ T$,
simp)
  by(*rule poly-derivatives*)

lemma [*poly-derivatives*]: $D\ (\lambda t.\ v - a * t) = (\lambda x.\ - a)\ on\ T$
  apply(*subgoal-tac* $D\ (\lambda t.\ - a * t + v) = (\lambda x.\ - a)\ on\ T$, *simp*)
  by(*rule poly-derivatives*)

declare *has-derivative-mult-const* [*poly-derivatives*]
    and *has-derivative-quadratic-monomial* [*poly-derivatives*]
    and *has-derivative-quadratic-monomial-halfed* [*poly-derivatives*]

lemma [*poly-derivatives*]:
  assumes $t \in T$
  shows $D\ (\lambda\tau.\ a * \tau^2\ /\ 2 + v * \tau + x) \mapsto (\lambda x.\ x *_R (a * t + v))\ at\ t\ within\ T$
 using *assms poly-derivatives* **unfolding** *has-vderiv-on-def has-vector-derivative-def*
by *simp*

thm *poly-derivatives*

1.2.2 Multivariable Derivatives

lemma *eventually-all-finite2*:
  fixes $P :: ('a{::}finite) \Rightarrow 'b \Rightarrow bool$
  assumes $h{:}\forall\, i.\ eventually\ (P\ i)\ F$
  shows $eventually\ (\lambda x.\ \forall\, i.\ P\ i\ x)\ F$
proof(*unfold eventually-def*)
  let $?F = Rep\text{-}filter\ F$
  have *obs*: $\forall\, i.\ ?F\ (P\ i)$
    using *h* **by** *auto*
  have $?F\ (\lambda x.\ \forall\, i \in UNIV.\ P\ i\ x)$
    apply(*rule finite-induct*)
    by(*auto intro*: *eventually-conj simp*: *obs h*)
  thus $?F\ (\lambda x.\ \forall\, i.\ P\ i\ x)$
    by *simp*
qed

lemma *eventually-all-finite-mono*:
  fixes $P :: ('a{::}finite) \Rightarrow 'b \Rightarrow bool$

assumes $h1$: $\forall i.$ *eventually* $(P\ i)\ F$
and $h2$: $\forall x.\ (\forall i.\ (P\ i\ x)) \longrightarrow Q\ x$
shows *eventually* $Q\ F$
proof−
have *eventually* $(\lambda x.\ \forall i.\ P\ i\ x)\ F$
using *h1 eventually-all-finite2* **by** *blast*
thus *eventually* $Q\ F$
unfolding *eventually-def*
using *h2 eventually-mono* **by** *auto*
qed

lemma *frechet-vec-lambda*:
fixes f::*real* $\Rightarrow$ $('a$::*banach*$)$^$('m$::*finite*$)$ **and** x::*real* **and** T::*real set*
defines $x_0 \equiv$ *netlimit* (*at* x *within* T) **and** $m \equiv$ *real CARD*$('m)$
assumes $\forall i.\ ((\lambda y.\ (f\ y\ \$\ i - f\ x_0\ \$\ i - (y - x_0) *_R f'\ x\ \$\ i)\ /_R\ (\|y - x_0\|)) \longrightarrow 0)$ (*at* x *within* T)
shows $((\lambda y.\ (f\ y - f\ x_0 - (y - x_0) *_R f'\ x)\ /_R\ (\|y - x_0\|)) \longrightarrow 0)$ (*at* x *within* T)
proof(*simp add*: *tendsto-iff*, *clarify*)
fix ε::*real* **assume** $0 < \varepsilon$
let $?\Delta = \lambda y.\ y - x_0$ **and** $?\Delta f = \lambda y.\ f\ y - f\ x_0$
let $?P = \lambda i\ e\ y.$ *inverse* $|?\Delta\ y| * (\|f\ y\ \$\ i - f\ x_0\ \$\ i - ?\Delta\ y *_R f'\ x\ \$\ i\|) < e$
and $?Q = \lambda y.$ *inverse* $|?\Delta\ y| * (\|?\Delta f\ y - ?\Delta\ y *_R f'\ x\|) < \varepsilon$
have $0 < \varepsilon$ / *sqrt* m
using $\langle 0 < \varepsilon\rangle$ **by** (*auto simp*: *assms*)
hence $\forall i.$ *eventually* $(\lambda y.\ ?P\ i\ (\varepsilon$ / *sqrt* $m)\ y)$ (*at* x *within* T)
using *assms* **unfolding** *tendsto-iff* **by** *simp*
thus *eventually* $?Q$ (*at* x *within* T)
proof(*rule eventually-all-finite-mono*, *simp add*: *norm-vec-def L2-set-def*, *clarify*)
fix t::*real*
let $?c =$ *inverse* $|t - x_0|$ **and** $?u\ t = \lambda i.\ f\ t\ \$\ i - f\ x_0\ \$\ i - ?\Delta\ t *_R f'\ x\ \$\ i$
assume *hyp*:$\forall i.\ ?c * (\|?u\ t\ i\|) < \varepsilon$ / *sqrt* m
hence $\forall i.\ (?c *_R (\|?u\ t\ i\|))^2 < (\varepsilon$ / *sqrt* $m)^2$
by (*simp add*: *power-strict-mono*)
hence $\forall i.\ ?c^2 * ((\|?u\ t\ i\|))^2 < \varepsilon^2 / m$
by (*simp add*: *power-mult-distrib power-divide assms*)
hence $\forall i.\ ?c^2 * ((\|?u\ t\ i\|))^2 < \varepsilon^2 / m$
by (*auto simp*: *assms*)
also have $(\{\}::'m\ set) \neq UNIV \wedge$ *finite* $(UNIV :: 'm\ set)$
by *simp*
ultimately have $(\sum i{\in}UNIV.\ ?c^2 * ((\|?u\ t\ i\|))^2) < (\sum (i::'m){\in}UNIV.\ \varepsilon^2 / m)$
by (*metis* (*lifting*) *sum-strict-mono*)
moreover have $?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) = (\sum i{\in}UNIV.\ ?c^2 * (\|?u\ t\ i\|)^2)$
using *sum-distrib-left* **by** *blast*
ultimately have $?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) < \varepsilon^2$
by (*simp add*: *assms*)
hence *sqrt* $(?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2)) <$ *sqrt* (ε^2)

using *real-sqrt-less-iff* **by** *blast*
also have ... $= \varepsilon$
using $\langle 0 < \varepsilon\rangle$ **by** *auto*
moreover have $?c * sqrt\ (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) = sqrt\ (?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2))$
by (*simp add*: *real-sqrt-mult*)
ultimately show $?c * sqrt\ (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2) < \varepsilon$
by *simp*
qed
qed

lemma *has-derivative-vec-lambda*:
fixes f::*real* $\Rightarrow$ (*'a*::*banach*) ^(*'m*::*finite*)
assumes $\forall\, i.\ D\ (\lambda t.\ f\ t\ \$\ i) \mapsto (\lambda\ h.\ h *_R f'\ x\ \$\ i)\ (at\ x\ within\ T)$
shows $D\, f \mapsto (\lambda h.\ h *_R f'\ x)\ at\ x\ within\ T$
apply(*unfold has-derivative-def*, *safe*)
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
using *assms frechet-vec-lambda*[*of x T*] **unfolding** *has-derivative-def* **by** *auto*

lemma *has-vderiv-on-vec-lambda*:
fixes f::((*'a*::*banach*) ^(*'n*::*finite*)) $\Rightarrow$ (*'a*^*'n*)
assumes $\forall\, i.\ D\ (\lambda t.\ x\ t\ \$\ i) = (\lambda t.\ f\ (x\ t)\ \$\ i)\ on\ T$
shows $D\ x = (\lambda t.\ f\ (x\ t))\ on\ T$
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*
by(*rule has-derivative-vec-lambda*, *simp*)

lemma *frechet-vec-nth*:
fixes f::*real* $\Rightarrow$ (*'a*::*real-normed-vector*) ^*'m* **and** x::*real* **and** T::*real set*
defines $x_0 \equiv netlimit\ (at\ x\ within\ T)$
assumes $((\lambda y.\ (f\ y - f\ x_0 - (y - x_0) *_R f'\ x)\ /_R\ (\|y - x_0\|)) \longrightarrow 0)\ (at\ x\ within\ T)$
shows $((\lambda y.\ (f\ y\ \$\ i - f\ x_0\ \$\ i - (y - x_0) *_R f'\ x\ \$\ i)\ /_R\ (\|y - x_0\|)) \longrightarrow 0)\ (at\ x\ within\ T)$
proof(*unfold tendsto-iff dist-norm*, *clarify*)
let $?\Delta = \lambda y.\ y - x_0$ **and** $?\Delta f = \lambda y.\ f\ y - f\ x_0$
fix ε::*real* **assume** $0 < \varepsilon$
let $?P = \lambda y.\ \|(?\Delta f\ y - ?\Delta\ y *_R f'\ x)\ /_R\ (\|?\Delta\ y\|) - 0\| < \varepsilon$
and $?Q = \lambda y.\ \|(f\ y\ \$\ i - f\ x_0\ \$\ i - ?\Delta\ y *_R f'\ x\ \$\ i)\ /_R\ (\|?\Delta\ y\|) - 0\| < \varepsilon$
have *eventually* $?P$ $(at\ x\ within\ T)$
using $\langle 0 < \varepsilon\rangle$ *assms* **unfolding** *tendsto-iff* **by** *auto*
thus *eventually* $?Q$ $(at\ x\ within\ T)$
proof(*rule-tac* $P{=}?P$ **in** *eventually-mono*, *simp-all*)
let $?u\ y\ i = f\ y\ \$\ i - f\ x_0\ \$\ i - ?\Delta\ y *_R f'\ x\ \$\ i$
fix y **assume** *hyp*:*inverse* $|?\Delta\ y| * (\|?\Delta f\ y - ?\Delta\ y *_R f'\ x\|) < \varepsilon$
have $\|(?\Delta f\ y - ?\Delta\ y *_R f'\ x)\ \$\ i\| \le \|?\Delta f\ y - ?\Delta\ y *_R f'\ x\|$
using *Finite-Cartesian-Product.norm-nth-le* **by** *blast*
also have $\|?u\ y\ i\| = \|(?\Delta f\ y - ?\Delta\ y *_R f'\ x)\ \$\ i\|$
by *simp*
ultimately have $\|?u\ y\ i\| \le \|?\Delta f\ y - ?\Delta\ y *_R f'\ x\|$

by *linarith*
hence *inverse* $|?\Delta\ y| * (\|?u\ y\ i\|) \leq$ *inverse* $|?\Delta\ y| * (\|?\Delta f\ y\ -\ ?\Delta\ y *_R f'\ x\|)$
by (*simp add*: *mult-left-mono*)
thus *inverse* $|?\Delta\ y| * (\|f\ y\ \$\ i - f\ x_0\ \$\ i\ -\ ?\Delta\ y *_R f'\ x\ \$\ i\|) < \varepsilon$
using *hyp* **by** *linarith*
qed
qed

lemma *has-derivative-vec-nth*:
assumes $D\ f \mapsto (\lambda h.\ h *_R f'\ x)$ *at x within T*
shows $D\ (\lambda t.\ f\ t\ \$\ i) \mapsto (\lambda h.\ h *_R f'\ x\ \$\ i)$ *at x within T*
apply(*unfold has-derivative-def*, *safe*)
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
using *frechet-vec-nth*[*of x T f*] *assms* **unfolding** *has-derivative-def* **by** *auto*

lemma *has-vderiv-on-vec-nth*:
fixes f::(('*a*::*banach*) ^('*n*::*finite*)) $\Rightarrow$ ('*a* ^'*n*)
assumes $D\ x = (\lambda t.\ f\ (x\ t))$ *on T*
shows $D\ (\lambda t.\ x\ t\ \$\ i) = (\lambda t.\ f\ (x\ t)\ \$\ i)$ *on T*
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*
by(*rule has-derivative-vec-nth*, *simp*)

1.3 Ordinary Differential Equations

1.3.1 Picard-Lindeloef

named-theorems *ubc-definitions definitions used in the locale unique-on-bounded-closed*

declare *unique-on-bounded-closed-def* [*ubc-definitions*]
and *unique-on-bounded-closed-axioms-def* [*ubc-definitions*]
and *unique-on-closed-def* [*ubc-definitions*]
and *compact-interval-def* [*ubc-definitions*]
and *compact-interval-axioms-def* [*ubc-definitions*]
and *self-mapping-def* [*ubc-definitions*]
and *self-mapping-axioms-def* [*ubc-definitions*]
and *continuous-rhs-def* [*ubc-definitions*]
and *closed-domain-def* [*ubc-definitions*]
and *global-lipschitz-def* [*ubc-definitions*]
and *interval-def* [*ubc-definitions*]
and *nonempty-set-def* [*ubc-definitions*]

lemma(**in** *unique-on-bounded-closed*) *unique-on-bounded-closed-on-compact-subset*:
assumes $t0 \in T'$ **and** $x0 \in X$ **and** $T' \subseteq T$ **and** *compact-interval* T'
shows *unique-on-bounded-closed* $t0\ T'\ x0\ f\ X\ L$
apply(*unfold-locales*)
using ‹*compact-interval* T'› **unfolding** *ubc-definitions* **apply** *simp+*
using ‹$t0 \in T'$› **apply** *simp*
using ‹$x0 \in X$› **apply** *simp*

using ‹$T' \subseteq T$› *self-mapping* **apply** *blast*
using ‹$T' \subseteq T$› *continuous* **apply**(*meson Sigma-mono continuous-on-subset subsetI*)
using ‹$T' \subseteq T$› *lipschitz* **apply** *blast*
using ‹$T' \subseteq T$› *lipschitz-bound* **by** *blast*

The next locale makes explicit the conditions for applying the Picard-Lindeloef theorem. This guarantees a unique solution for every initial value problem represented with a vector field f and an initial time t_0. It is mostly a simplified reformulation of the approach taken by the people who created the Ordinary Differential Equations entry in the AFP.

locale *picard-lindeloef* =
fixes f::*real* ⇒ ($'a$::*banach*) ⇒ $'a$ **and** T::*real set* **and** L t_0::*real*
assumes *init-time*: $t_0 \in T$
and *cont-vec-field*: *continuous-on* ($T \times$ *UNIV*) ($\lambda(t, x).\ f\ t\ x$)
and *lipschitz-vec-field*: $\bigwedge t.\ t \in T \Longrightarrow L$*−lipschitz-on UNIV* ($\lambda x.\ f\ t\ x$)
and *nonempty-time*: $T \neq \{\}$
and *interval-time*: *is-interval* T
and *compact-time*: *compact* T
and *lipschitz-bound*: $\bigwedge s\ t.\ s \in T \Longrightarrow t \in T \Longrightarrow$ *abs* $(s - t) * L < 1$
begin

sublocale *continuous-rhs T UNIV*
using *cont-vec-field* **unfolding** *continuous-rhs-def* **by** *simp*

sublocale *global-lipschitz T UNIV*
using *lipschitz-vec-field* **unfolding** *global-lipschitz-def* **by** *simp*

sublocale *closed-domain UNIV*
unfolding *closed-domain-def* **by** *simp*

sublocale *compact-interval*
using *interval-time nonempty-time compact-time* **by**(*unfold-locales*, *auto*)

lemma *is-ubc*:
shows *unique-on-bounded-closed* t_0 *T s f UNIV L*
using *nonempty-time* **unfolding** *ubc-definitions* **apply** *safe*
by(*auto simp*: *compact-time interval-time init-time*
lipschitz-vec-field lipschitz-bound cont-vec-field)

lemma *min-max-interval*:
obtains *m M* **where** $T = \{m\ ..\ M\}$
using *T-def* **by** *blast*

lemma *subinterval*:
assumes $t \in T$
obtains *t1* **where** $\{t\ ..\ t1\} \subseteq T$
using *assms interval-subset-is-interval interval-time* **by** *fastforce*

```
lemma subsegment:
  assumes t1 ∈ T and t2 ∈ T
  shows {t1 −− t2} ⊆ T
  using assms closed-segment-subset-domain by blast

lemma unique-solution:
  assumes D x = (λt. f t (x t)) on T and x t₀ = s
    and D y = (λt. f t (y t)) on T and y t₀ = s and t ∈ T
  shows x t = y t
  apply(rule unique-on-bounded-closed.unique-solution)
  using is-ubc[of s] apply blast
  using assms unfolding solves-ode-def by auto

abbreviation phi t s ≡ (apply-bcontfun (unique-on-bounded-closed.fixed-point t₀
T s f UNIV)) t

lemma fixpoint-solves-ivp:
  shows D (λt. phi t s) = (λt. f t (phi t s)) on T and phi t₀ s = s
  using is-ubc[of s] unique-on-bounded-closed.fixed-point-solution[of t₀ T s f UNIV
L]
    unique-on-bounded-closed.fixed-point-iv[of t₀ T s f UNIV L]
  unfolding solves-ode-def by auto

lemma fixpoint-usolves-ivp:
  assumes D x = (λt. f t (x t)) on T and x t₀ = s and t ∈ T
  shows x t = phi t s
  using unique-solution[OF assms(1,2)] fixpoint-solves-ivp assms by blast

end
```

1.3.2 Flows for ODEs

This locale is a particular case of the previous one. It makes the unique solution for initial value problems explicit, it restricts the vector field to reflect autonomous systems (those that do not depend explicitly on time), and it sets the initial time equal to 0. This is the first step towards formalizing the flow of a differential equation, i.e. the function that maps every point to the unique trajectory tangent to the vector field.

```
locale local-flow = picard-lindeloef (λ t. f) T L 0 for f::('a::banach) ⇒ 'a and
T L +
  fixes φ :: real ⇒ 'a ⇒ 'a
  assumes ivp: D (λt. φ t s) = (λt. f (φ t s)) on T φ 0 s = s
begin

lemma is-fixpoint:
  assumes t ∈ T
  shows φ t s = phi t s
  apply(rule fixpoint-usolves-ivp)
```

using *ivp assms init-time* **by** *simp-all*

lemma *solves-ode*:
shows $((\lambda\ t.\ \varphi\ t\ s)$ *solves-ode* $(\lambda\ t.\ f))\,T$ *UNIV*
unfolding *solves-ode-def* **using** *ivp*(*1*) **by** *auto*

lemma *usolves-ivp*:
assumes $D\ x = (\lambda t.\ f\ (x\ t))$ *on* T **and** $x\ 0 = s$ **and** $t \in T$
shows $x\ t = \varphi\ t\ s$
proof−
have $x\ t =$ *phi t s*
using *assms fixpoint-usolves-ivp* **by** *blast*
also have ... $= \varphi\ t\ s$
using *assms is-fixpoint* **by** *force*
finally show *?thesis* **.**
qed

lemma *usolves-on-compact-subset*:
assumes $T' \subseteq T$ **and** *compact-interval* T' **and** $0 \in T'$ **and** $t \in T'$
and *x-solves*: $(x$ *solves-ode* $(\lambda t.\ f))\ T'$ *UNIV*
shows $\varphi\ t\ (x\ 0) = x\ t$
proof−
have *obs*:$((\lambda\ \tau.\ \varphi\ \tau\ (x\ 0))$ *solves-ode* $(\lambda\ \tau.\ f))\,T'$ *UNIV*
using $\langle T' \subseteq T\rangle$ *solves-ode-on-subset solves-ode* **by** (*metis subset-eq*)
have *unique-on-bounded-closed* $0\ T\ (x\ 0)\ (\lambda\ \tau.\ f)$ *UNIV L*
using *is-ubc* **by** *blast*
hence *unique-on-bounded-closed* $0\ T'\ (x\ 0)\ (\lambda\ \tau.\ f)$ *UNIV L*
using *unique-on-bounded-closed.unique-on-bounded-closed-on-compact-subset*
$\langle 0 \in T'\rangle$ $\langle T' \subseteq T\rangle$ **and** $\langle$*compact-interval* $T'\rangle$ **by** *blast*
moreover have $\varphi\ 0\ (x\ 0) = x\ 0$
using *ivp* **by** *blast*
ultimately show $\varphi\ t\ (x\ 0) = x\ t$
using *unique-on-bounded-closed.unique-solution obs x-solves* $\langle t \in T'\rangle$ **by** *blast*
qed

end

lemma *flow-on-compact-subset*:
assumes *flow-on-big*: *local-flow* $f\ T'\ L\ \varphi$ **and** $T \subseteq T'$
and *compact-interval* T **and** $0 \in T$
shows *local-flow* $f\ T\ L\ \varphi$
proof(*unfold local-flow-def local-flow-axioms-def*, *safe*)
fix s **show** $\varphi\ 0\ s = s$
using *local-flow.ivp*(*2*) *flow-on-big* **by** *blast*
show $D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))$ *on* T
using *assms solves-ode-on-subset*[**where** $T{=}T$ **and** $S{=}T'$ **and** $x{=}\lambda t.\ \varphi\ t\ s$
and $X{=}$*UNIV*]
unfolding *local-flow-def local-flow-axioms-def solves-ode-def* **by** *force*
next

```
  show picard-lindeloef (λt. f) T L 0
    using assms apply(unfold local-flow-def local-flow-axioms-def)
    apply(unfold picard-lindeloef-def ubc-definitions)
    apply(meson Sigma-mono continuous-on-subset subsetI)
    by(simp-all add: subset-eq)
qed
```

Finally, the flow exists when the unique solution from the last locale is defined in all of $\mathbb{R}$. Here we prove that it is a dyanmical system, i.e. a group action on the additive group of the real numbers.

```
locale global-flow = local-flow f UNIV L φ for f L φ
begin

lemma add-flow-solves: D (λτ. φ (τ + t) s) = (λτ. f (φ (τ + t) s)) on UNIV
  apply(subgoal-tac D ((λτ. φ τ s) ∘ (λτ. τ + t)) =
   (λx. (λτ. 1) x *R (λt. f (φ t s)) ((λτ. τ + t) x)) on UNIV, simp add: comp-def)
  apply(rule has-vderiv-on-compose)
  using solves-ode min-max-interval unfolding solves-ode-def apply force
  apply(rule-tac f'1=λ x. 1  and g'1=λ x. 0 in derivative-intros(191))
  by(rule derivative-intros, simp)+ simp-all

lemma is-group-action:
  shows φ 0 s = s
    and φ (t1 + t2) s = φ t1 (φ t2 s)
proof−
  show φ 0 s = s
    using ivp by simp
  have φ (0 + t2) s = φ t2 s
    by simp
  moreover have D (λτ. φ (τ + t2) s) = (λτ. f (φ (τ + t2) s)) on UNIV
    using add-flow-solves by simp
  moreover have φ 0 (φ t2 s) = φ t2 s
    using ivp by simp
  ultimately have ⋀ t. φ (t + t2) s = φ t (φ t2 s)
    using usolves-ivp by blast
  thus φ (t1 + t2) s = φ t1 (φ t2 s)
    by simp
qed

end

lemma localize-global-flow:
  assumes global-flow f L φ and compact-interval T
  shows local-flow f T L φ
  using assms unfolding global-flow-def local-flow-def picard-lindeloef-def by simp
```

Example

Below there is an example showing the general methodology to introduce pairs of vector fields and their respective flows using the previous locales.

lemma *picard-lindeloef-constant*: $0 \leq t \Longrightarrow$ *picard-lindeloef* $(\lambda t\ s.\ c)\ \{0..t\}\ (1\ /\ (t + 1))\ 0$
  unfolding *picard-lindeloef-def* **by**(*simp add*: *nonempty-set-def lipschitz-on-def*, *clarsimp*, *simp*)

lemma *line-vderiv-constant*: $D\ (\lambda\tau.\ x + \tau *_R c) = (\lambda t.\ c)$ *on* $\{0..t\}$
  apply(*rule-tac* $f'1{=}\lambda\ x.\ 0$ **and** $g'1{=}\lambda\ x.\ c$ **in** *derivative-intros*(*191*))
  apply(*rule derivative-intros*, *simp*)+
  by *simp-all*

lemma *line-is-local-flow*:
  fixes $c{::}'a{::}banach$
  assumes $0 \leq t$
  shows *local-flow* $(\lambda\ s.\ c)\ \{0..t\}\ (1/(t + 1))\ (\lambda\ t\ x.\ x + t *_R c)$
  unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
  using *assms picard-lindeloef-constant* **apply** *blast*
  using *line-vderiv-constant* **by** *auto*

end
theory *hs-prelims-matrices*
  imports *hs-prelims*

begin

Chapter 2

Linear Algebra for Hybrid Systems

Linear systems of ordinary differential equations (ODEs) are those whose vector fields are a linear operator. That is, there is a matrix A such that the system $x'\ t = f\ (x\ t)$ can be rewritten as $x'\ t = A *v\ x\ t$. The end goal of this section is to prove that every linear system of ODEs has a unique solution, and to obtain a characterization of said solution. For that we start by formalising various properties of vector spaces.

2.1 Vector operations

abbreviation e $k \equiv$ *axis k 1*

abbreviation $entries\ (A{::}'a\hat{\ }'n\hat{\ }'m) \equiv \{A\ \$\ i\ \$\ j \mid i\ j.\ i \in UNIV \wedge j \in UNIV\}$

abbreviation $kronecker\text{-}delta :: 'a \Rightarrow 'a \Rightarrow 'b \Rightarrow ('b{::}zero)\ (\delta_K\ \text{- - -}\ [55,\ 55,\ 55]\ 55)$
  where $\delta_K\ i\ j\ q \equiv (if\ i = j\ then\ q\ else\ 0)$

lemma *finite-sum-univ-singleton*: $(sum\ g\ UNIV) = sum\ g\ \{i\} + sum\ g\ (UNIV - \{i\})$ **for** $i{::}'a{::}finite$
  by (*metis add.commute finite-class.finite-UNIV sum.subset-diff top-greatest*)

lemma *kronecker-delta-simps*[*simp*]:
  fixes $q{::}('a{::}semiring\text{-}0)$ **and** $i{::}'n{::}finite$
  shows $(\sum j{\in}UNIV.\ f\ j * (\delta_K\ j\ i\ q)) = f\ i * q$
    and $(\sum j{\in}UNIV.\ f\ j * (\delta_K\ i\ j\ q)) = f\ i * q$
    and $(\sum j{\in}UNIV.\ (\delta_K\ i\ j\ q) * f\ j) = q * f\ i$
    and $(\sum j{\in}UNIV.\ (\delta_K\ j\ i\ q) * f\ j) = q * f\ i$
  by (*auto simp*: *finite-sum-univ-singleton*[*of - i*])

lemma *sum-axis*[*simp*]:

fixes q::('a::semiring-0)
shows $(\sum j{\in}UNIV.\ f\ j\ *\ axis\ i\ q\ \$\ j) = f\ i\ *\ q$
and $(\sum j{\in}UNIV.\ axis\ i\ q\ \$\ j\ *\ f\ j) = q\ *\ f\ i$
unfolding *axis-def* **by**(*auto simp*: *vec-eq-iff*)

lemma *sum-scalar-nth-axis*: $sum\ (\lambda i.\ (x\ \$\ i)\ {*}s\ \mathrm{e}\ i)\ UNIV = x$ **for** x :: ('a::semiring-1)^'n
unfolding *vec-eq-iff axis-def* **by** *simp*

lemma *scalar-eq-scaleR*[*simp*]: $c\ {*}s\ x = c\ {*}_R\ x$ **for** c :: *real*
unfolding *vec-eq-iff* **by** *simp*

lemma *matrix-add-rdistrib*: $((B + C)\ {**}\ A) = (B\ {**}\ A) + (C\ {**}\ A)$
by (*vector matrix-matrix-mult-def sum.distrib*[*symmetric*] *field-simps*)

lemma *vec-mult-inner*: $(A\ {*}v\ v) \cdot w = v \cdot (transpose\ A\ {*}v\ w)$ **for** A::*real* ^'n^'n
unfolding *matrix-vector-mult-def transpose-def inner-vec-def*
apply(*simp add*: *sum-distrib-right sum-distrib-left*)
apply(*subst sum.swap*)
apply(*subgoal-tac* $\forall\ i\ j.\ A\ \$\ i\ \$\ j\ *\ v\ \$\ j\ *\ w\ \$\ i = v\ \$\ j\ *\ (A\ \$\ i\ \$\ j\ *\ w\ \$\ i)$)
by *presburger* (*simp*)

lemma *norm-axis-eq*[*simp*]: $\|axis\ i\ k\| = \|k\|$
proof(*simp add*: *axis-def norm-vec-def L2-set-def*)
have $(\sum j{\in}UNIV.\ (\|(\delta_K\ j\ i\ k)\|)^2) = (\sum j{\in}\{i\}.\ (\|(\delta_K\ j\ i\ k)\|)^2) + (\sum j{\in}(UNIV{-}\{i\}).\ (\|(\delta_K\ j\ i\ k)\|)^2)$
using *finite-sum-univ-singleton* **by** *blast*
also have ... $= (\|k\|)^2$ **by** *simp*
finally show $sqrt\ (\sum j{\in}UNIV.\ (norm\ (if\ j = i\ then\ k\ else\ 0))^2) = norm\ k$ **by** *simp*
qed

lemma *matrix-axis-0*:
fixes A :: ('a::idom)^'n^'m
assumes $k \neq 0$ **and** h:$\forall\ i.\ (A\ {*}v\ (axis\ i\ k)) = 0$
shows $A = 0$
proof−
{**fix** i::'n
have $0 = (\sum j{\in}UNIV.\ (axis\ i\ k)\ \$\ j\ {*}s\ column\ j\ A)$
using *h matrix-mult-sum*[*of A axis i k*] **by** *simp*
also have ... $= k\ {*}s\ column\ i\ A$
by(*simp add*: *axis-def vector-scalar-mult-def column-def vec-eq-iff mult.commute*)
finally have $k\ {*}s\ column\ i\ A = 0$
unfolding *axis-def* **by** *simp*
hence $column\ i\ A = 0$
using *vector-mul-eq-0* ‹$k \neq 0$› **by** *blast*}
thus $A = 0$
unfolding *column-def vec-eq-iff* **by** *simp*
qed

lemma *scaleR-norm-sgn-eq*: $(\|x\|) *_R \; sgn\ x = x$
  by (*metis divideR-right norm-eq-zero scale-eq-0-iff sgn-div-norm*)

lemma *vector-scaleR-commute*: $A *v\ c *_R x = c *_R (A *v\ x)$ **for** x :: $('a::real\text{-}normed\text{-}algebra\text{-}1)\hat{\ }'n$
  unfolding *scaleR-vec-def matrix-vector-mult-def* **by**(*auto simp*: *vec-eq-iff scaleR-right.sum*)

lemma *scaleR-vector-assoc*: $c *_R (A *v\ x) = (c *_R A) *v\ x$ **for** x :: $('a::real\text{-}normed\text{-}algebra\text{-}1)\hat{\ }'n$
  unfolding *matrix-vector-mult-def* **by**(*auto simp*: *vec-eq-iff scaleR-right.sum*)

lemma *mult-norm-matrix-sgn-eq*:
  fixes x :: $('a::real\text{-}normed\text{-}algebra\text{-}1)\hat{\ }'n$
  shows $(\|A *v\ sgn\ x\|) * (\|x\|) = \|A *v\ x\|$
proof−
  have $\|A *v\ x\| = \|A *v\ ((\|x\|) *_R sgn\ x)\|$
    by(*simp add*: *scaleR-norm-sgn-eq*)
  also have ... $= (\|A *v\ sgn\ x\|) * (\|x\|)$
    by(*simp add*: *vector-scaleR-commute*)
  finally show *?thesis* **..**
qed

2.2 Matrix norms

Here we develop the foundations for obtaining the Lipschitz constant for every linear system of ODEs $x'\ t = A *v\ x\ t$. For that we derive some properties of two matrix norms.

2.2.1 Matrix operator norm

abbreviation *op-norm* $(A::('a::real\text{-}normed\text{-}algebra\text{-}1)\hat{\ }'n\hat{\ }'m) \equiv Sup\ \{\|A *v\ x\|$
$|\ x.\ \|x\| = 1\}$

notation *op-norm* $((1\|\text{-}\|_{op})\ [65]\ 61)$

lemma *norm-matrix-bound*:
  fixes $A::('a::real\text{-}normed\text{-}algebra\text{-}1)\hat{\ }'n\hat{\ }'m$
  shows $\|x\| = 1 \Longrightarrow \|A *v\ x\| \le \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$
proof−
  fix $x::('a,\ 'n)$ *vec* **assume** $\|x\| = 1$
  hence *xi-le1*:$\bigwedge i.\ \|x\ \$\ i\| \le 1$
    by (*metis Finite-Cartesian-Product.norm-nth-le*)
  {**fix** $j::'m$
    have $\|(\sum i\in UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \le (\sum i\in UNIV.\ \|A\ \$\ j\ \$\ i * x\ \$\ i\|)$
      using *norm-sum* **by** *blast*
    also have ... $\le (\sum i\in UNIV.\ (\|A\ \$\ j\ \$\ i\|) * (\|x\ \$\ i\|))$
      by (*simp add*: *norm-mult-ineq sum-mono*)
    also have ... $\le (\sum i\in UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$
      using *xi-le1* **by** (*simp add*: *sum-mono mult-left-le*)

finally have $\|(\sum i{\in}UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \leq (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$ **by** *simp*}
from *this* **have** $\bigwedge j.\ \|(A *v\ x)\ \$\ j\| \leq ((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j$
unfolding *matrix-vector-mult-def* **by** *simp*
hence $(\sum j{\in}UNIV.\ (\|(A *v\ x)\ \$\ j\|)^2) \leq (\sum j{\in}UNIV.\ (\|((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j\|)^2)$
by (*metis* (*mono-tags*, *lifting*) *norm-ge-zero power2-abs power-mono real-norm-def sum-mono*)
thus $\|A *v\ x\| \leq \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$
unfolding *norm-vec-def L2-set-def* **by** *simp*
qed

lemma *op-norm-set-proptys*:
fixes A::('*a*::*real-normed-algebra-1*)^'*n*^'*m*
shows *bounded* $\{\|A *v\ x\| \mid x.\ \|x\| = 1\}$
and *bdd-above* $\{\|A *v\ x\| \mid x.\ \|x\| = 1\}$
and $\{\|A *v\ x\| \mid x.\ \|x\| = 1\} \neq \{\}$
unfolding *bounded-def bdd-above-def* **apply** *safe*
apply(*rule-tac* $x=0$ **in** *exI*, *rule-tac* $x=\|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$ **in** *exI*)
apply(*force simp*: *norm-matrix-bound dist-real-def*)
apply(*rule-tac* $x=\|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$ **in** *exI*, *force simp*: *norm-matrix-bound*)
using *ex-norm-eq-1* **by** *blast*

lemma *norm-matrix-le-op-norm*: $\|x\| = 1 \Longrightarrow \|A *v\ x\| \leq \|A\|_{op}$
by(*rule cSup-upper*, *auto simp*: *op-norm-set-proptys*)

lemma *norm-matrix-le-op-norm-ge-0*: $0 \leq \|A\|_{op}$
using *ex-norm-eq-1 norm-ge-zero norm-matrix-le-op-norm basic-trans-rules*(*23*)
by *blast*

lemma *norm-sgn-le-op-norm*: $\|A *v\ sgn\ x\| \leq \|A\|_{op}$
by(*cases* $x=0$, *simp-all add*: *norm-sgn norm-matrix-le-op-norm norm-matrix-le-op-norm-ge-0*)

lemma *norm-matrix-le-mult-op-norm*: $\|A *v\ x\| \leq (\|A\|_{op}) * (\|x\|)$ **for** A :: *real*^'*n*^'*m*
proof−
have $\|A *v\ x\| = (\|A *v\ sgn\ x\|) * (\|x\|)$
by(*simp add*: *mult-norm-matrix-sgn-eq*)
also have ... $\leq (\|A\|_{op}) * (\|x\|)$
using *norm-sgn-le-op-norm*[*of A*] **by** (*simp add*: *mult-mono'*)
finally show *?thesis* **by** *simp*
qed

lemma *ltimes-op-norm*:
Sup $\{|c| * (\|A *v\ x\|)\ |x.\ \|x\| = 1\} = |c| * (\|A\|_{op})$ (**is** *Sup ?cA* $= |c| * (\|A\|_{op})$)
proof(*cases* $c = 0$, *simp add*: *ex-norm-eq-1*)
let *?S* $= \{(\|A *v\ x\|)\ |x.\ \|x\| = 1\}$
note *op-norm-set-proptys*(*2*)[*of A*]
also have *?cA* $= \{|c| * x|x.\ x \in\ ?S\}$
by *force*

ultimately have *bdd-cA*:*bdd-above ?cA*
  using *bdd-above-ltimes*[*of* $|c|$ *?S*] **by** *simp*
assume $c \neq 0$
show *Sup ?cA* $= |c| * (\|A\|_{op})$
proof(*rule cSup-eq-linorder*)
  show *nempty-cA*:*?cA* $\neq \{\}$
    using *op-norm-set-proptys*(*3*)[*of A*] **by** *blast*
  show *bdd-above ?cA*
    using *bdd-cA* **by** *blast*
  {**fix** *m* **assume** $m \in$ *?cA*
    then obtain *x* **where** *x-def*:$\|x\| = 1 \wedge m = |c| * (\|A *v\ x\|)$
      by *blast*
    hence $(\|A *v\ x\|) \leq (\|A\|_{op})$
      using *norm-matrix-le-op-norm* **by** *force*
    hence $m \leq |c| * (\|A\|_{op})$
      using *x-def* **by** (*simp add*: *mult-left-mono*)}
  thus $\forall x \in$*?cA*. $x \leq |c| * (\|A\|_{op})$
    by *blast*
next
  show $\forall y<|c|*(\|A\|_{op}).\ \exists x \in$*?cA*. $y < x$
  proof(*clarify*)
    fix *m* **assume** $m < |c| * (\|A\|_{op})$
    hence $(m\ /\ |c|) < (\|A\|_{op})$
      using *pos-divide-less-eq*[*of* $|c|$ *m* $(\|A\|_{op})$] ‹$c \neq 0$›
        semiring-normalization-rules(*7*)[*of* $|c|$] **by** *auto*
    then obtain *x* **where** $\|x\| = 1 \wedge (m\ /\ |c|) < (\|A *v\ x\|)$
      using *less-cSup-iff*[*of ?S m* / $|c|$] *op-norm-set-proptys* **by** *force*
    hence $\|x\| = 1 \wedge m < |c| * (\|A *v\ x\|)$
      using ‹$c \neq 0$› *pos-divide-less-eq*[*of - m -*] **by** (*simp add*: *mult.commute*)
    thus $\exists n \in$*?cA*. $m < n$ **by** *blast*
  qed
qed
qed

lemma *op-norm-le-sum-column*:
  $\|A\|_{op} \leq (\sum i \in UNIV.\ \|column\ i\ A\|)$ **for** *A*::*real^'n^'m*
  using *op-norm-set-proptys*(*3*) **proof**(*rule cSup-least*)
  fix *m* **assume** $m \in \{\|A *v\ x\|\ |\ x.\ \|x\| = 1\}$
    then obtain *x* **where** *x-def*:$\|x\| = 1 \wedge m = (\|A *v\ x\|)$ **by** *blast*
    hence *x-hyp*:$\bigwedge i.$ *norm* $(x\ \$\ i) \leq 1$
      by (*simp add*: *norm-bound-component-le-cart*)
    have $(\|A *v\ x\|) =$ *norm* $(\sum i \in UNIV.\ (x\ \$\ i *s\ column\ i\ A))$
      by(*subst matrix-mult-sum*[*of A*], *simp*)
    also have ... $\leq (\sum i \in UNIV.$ *norm* $(x\ \$\ i *s\ column\ i\ A))$
      by (*simp add*: *sum-norm-le*)
    also have ... $= (\sum i \in UNIV.$ *norm* $(x\ \$\ i) *$ *norm* $(column\ i\ A))$
      by (*simp add*: *mult-norm-matrix-sgn-eq*)
    also have ... $\leq (\sum i \in UNIV.$ *norm* $(column\ i\ A))$
      using *x-hyp* **by** (*simp add*: *mult-left-le-one-le sum-mono*)

finally show $m \leq (\sum i{\in}UNIV.\ norm\ (column\ i\ A))$
using *x-def* **by** *linarith*
qed

lemma *op-norm-zero-iff*: $(\|A\|_{op} = 0) = (A = 0)$ **for** A::('a::*real-normed-field*)^'n^'m
proof
assume $A = 0$ **thus** $\|A\|_{op} = 0$
by(*simp add*: *ex-norm-eq-1*)
next
assume $\|A\|_{op} = 0$
note *cSup-upper*[*of* - $\{\|A *v\ x\| \mid x.\ \|x\| = 1\}$]
hence $\bigwedge r.\ r \in \{\|A *v\ x\| \mid x.\ \|x\| = 1\} \Longrightarrow r \leq (\|A\|_{op})$
using *op-norm-set-proptys*(*2*) **by** *force*
also have $\bigwedge r.\ r \in (\{\|A *v\ x\| \mid x.\ \|x\| = 1\}) \Longrightarrow 0 \leq r$
using *norm-ge-zero* **by** *blast*
ultimately have $\bigwedge r.\ r \in (\{\|A *v\ x\| \mid x.\ \|x\| = 1\}) \Longrightarrow r = 0$
using ‹$\|A\|_{op} = 0$› **by** *fastforce*
hence $\bigwedge x.\ \|x\| = 1 \Longrightarrow x \neq 0 \wedge (\|A *v\ x\|) = 0$
by *force*
hence $\bigwedge i.\ norm\ (A *v\ \mathrm{e}\ i) = 0$
by *simp*
from *this* **show** $A = 0$
using *matrix-axis-0*[*of 1 A*] *norm-eq-zero* **by** *simp*
qed

lemma *op-norm-triangle*:
fixes A::('a::*real-normed-algebra-1*)^'n^'m
shows $\|A + B\|_{op} \leq (\|A\|_{op}) + (\|B\|_{op})$
using *op-norm-set-proptys*(*3*)[*of A + B*] **proof**(*rule cSup-least*)
fix m **assume** $m \in \{\|(A + B) *v\ x\| \mid x.\ \|x\| = 1\}$
then obtain x::'a^'n **where** $\|x\| = 1$ **and** $m = \|(A + B) *v\ x\|$
by *blast*
have $\|(A + B) *v\ x\| \leq (\|A *v\ x\|) + (\|B *v\ x\|)$
by (*simp add*: *matrix-vector-mult-add-rdistrib norm-triangle-ineq*)
also have ... $\leq (\|A\|_{op}) + (\|B\|_{op})$
by (*simp add*: ‹$\|x\| = 1$› *add-mono norm-matrix-le-op-norm*)
finally show $m \leq (\|A\|_{op}) + (\|B\|_{op})$
using ‹$m = \|(A + B) *v\ x\|$› **by** *blast*
qed

lemma *op-norm-scaleR*: $\|c *_R A\|_{op} = |c| * (\|A\|_{op})$
proof−
let $?N = \{|c| * (\|A *v\ x\|)\ |x.\ \|x\| = 1\}$
have $\{\|(c *_R A) *v\ x\| \mid x.\ \|x\| = 1\} = ?N$
by (*metis* (*no-types*, *hide-lams*) *norm-scaleR scaleR-vector-assoc*)
also have *Sup* $?N = |c| * (\|A\|_{op})$
using *ltimes-op-norm*[*of c A*] **by** *blast*
ultimately show *op-norm* $(c *_R A) = |c| * (\|A\|_{op})$
by *auto*

qed

lemma *op-norm-matrix-matrix-mult-le*: $\|A ** B\|_{op} \leq (\|A\|_{op}) * (\|B\|_{op})$ **for** A::*real*^′*n*^′*m*
using *op-norm-set-proptys*(*3*)[*of A* ** *B*]
proof(*rule cSup-least*)
  have $0 \leq (\|A\|_{op})$ **using** *norm-matrix-le-op-norm-ge-0* **by** *force*
  fix n **assume** $n \in \{\|(A ** B) *v\ x\| \mid x.\ \|x\| = 1\}$
    then obtain x **where** *x-def*:$n = \|A ** B *v\ x\| \wedge \|x\| = 1$ **by** *blast*
    have $\|A ** B *v\ x\| = \|A *v\ (B *v\ x)\|$
      by (*simp add*: *matrix-vector-mul-assoc*)
    also have ... $\leq (\|A\|_{op}) * (\|B *v\ x\|)$
      by (*simp add*: *norm-matrix-le-mult-op-norm*[*of* - $B *v\ x$])
    also have ... $\leq (\|A\|_{op}) * ((\|B\|_{op}) * (\|x\|))$
      using *norm-matrix-le-mult-op-norm*[*of B x*] ‹$0 \leq (\|A\|_{op})$› *mult-left-mono* **by** *blast*
    also have ... $= (\|A\|_{op}) * (\|B\|_{op})$ **using** *x-def* **by** *simp*
    finally show $n \leq (\|A\|_{op}) * (\|B\|_{op})$ **using** *x-def* **by** *blast*
qed

lemma *norm-matrix-vec-mult-le-transpose*:
$\|x\| = 1 \Longrightarrow (\|A *v\ x\|) \leq sqrt\ (\|transpose\ A ** A\|_{op}) * (\|x\|)$ **for** A::*real*^′*a*^′*a*
proof−
  assume $\|x\| = 1$
  have $(\|A *v\ x\|)^2 = (A *v\ x) \cdot (A *v\ x)$
    using *dot-square-norm*[*of* $(A *v\ x)$] **by** *simp*
  also have ... $= x \cdot (transpose\ A *v\ (A *v\ x))$
    using *vec-mult-inner* **by** *blast*
  also have ... $\leq (\|x\|) * (\|transpose\ A *v\ (A *v\ x)\|)$
    using *norm-cauchy-schwarz* **by** *blast*
  also have ... $\leq (\|transpose\ A ** A\|_{op}) * (\|x\|)$^*2*
    apply(*subst matrix-vector-mul-assoc*) **using** *norm-matrix-le-mult-op-norm*[*of transpose A* ** *A x*]
    by (*simp add*: ‹$\|x\| = 1$›)
  finally have $((\|A *v\ x\|))$^*2* $\leq (\|transpose\ A ** A\|_{op}) * (\|x\|)$^*2*
    by *linarith*
  thus $(\|A *v\ x\|) \leq sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
    by (*simp add*: ‹$\|x\| = 1$› *real-le-rsqrt*)
qed

2.2.2 Matrix maximum norm

abbreviation *max-norm* (A::*real*^′*n*^′*m*) ≡ *Max* (*abs* ‘ (*entries A*))

notation *max-norm* $((1\|\text{-}\|_{max})\ [65]\ 61)$

lemma *max-norm-def*: $\|A\|_{max} = Max\ \{|A\ \$\ i\ \$\ j| \mid i\ j.\ i{\in}UNIV \wedge j{\in}UNIV\}$
  by(*simp add*: *image-def*, *rule arg-cong*[*of* - - *Max*], *blast*)

lemma *max-norm-set-proptys*:
  fixes $A::real\hat{}('n::finite)\hat{}('m::finite)$
  shows $finite\ \{|A\ \$\ i\ \$\ j|\ |i\ j.\ i \in UNIV \wedge j \in UNIV\}$ (**is** $finite\ ?X$)
proof$-$
  have $\bigwedge i.\ finite\ \{|A\ \$\ i\ \$\ j|\ |\ j.\ j \in UNIV\}$
    using *finite-Atleast-Atmost-nat* **by** *fastforce*
  hence $finite\ (\bigcup i{\in}UNIV.\ \{|A\ \$\ i\ \$\ j|\ |\ j.\ j \in UNIV\})$ (**is** $finite\ ?Y$)
    using *finite-class.finite-UNIV* **by** *blast*
  also have $?X \subseteq ?Y$ **by** *auto*
  ultimately show *?thesis*
    using *finite-subset* **by** *blast*
qed

lemma *max-norm-ge-0*: $0 \leq \|A\|_{max}$
proof$-$
  have $\bigwedge\ i\ j.\ |A\ \$\ i\ \$\ j| \geq 0$ **by** *simp*
  also have $\bigwedge\ i\ j.\ |A\ \$\ i\ \$\ j| \leq \|A\|_{max}$
    unfolding *max-norm-def* **using** *max-norm-set-proptys Max-ge max-norm-def*
by *blast*
  finally show $0 \leq \|A\|_{max}$ **.**
qed

lemma *op-norm-le-max-norm*:
  fixes $A::real\hat{}('n::finite)\hat{}('m::finite)$
  shows $\|A\|_{op} \leq real\ CARD('n) * real\ CARD('m) * (\|A\|_{max})$ (**is** $\|A\|_{op} \leq ?n *$
$?m * (\|A\|_{max})$)
proof(*rule cSup-least*)
  show $\{\|A *v\ x\|\ |x.\ \|x\| = 1\} \neq \{\}$
    using *op-norm-set-proptys*(*3*) **by** *blast*
  {**fix** n **assume** $n \in \{\|A *v\ x\|\ |x.\ \|x\| = 1\}$
    then obtain $x::(real,\ 'n)\ vec$ **where** $n\text{-}def{:}\|x\| = 1 \wedge \|A *v\ x\| = n$
      by *blast*
    hence $comp\text{-}le\text{-}1{:}\forall\ \ i::'n.\ |x\ \$\ i| \leq 1$
      by (*simp add*: *norm-bound-component-le-cart*)
    have $A *v\ x = (\sum i{\in}UNIV.\ x\ \$\ i *s\ column\ i\ A)$
      using *matrix-mult-sum* **by** *blast*
    hence $\|A *v\ x\| \leq (\sum i{\in}UNIV.\ \|x\ \$\ i *s\ column\ i\ A\|)$
      by (*simp add*: *sum-norm-le*)
    also have $... = (\sum i{\in}UNIV.\ |x\ \$\ i| * (\|column\ i\ A\|))$
      by *simp*
    also have $... \leq (\sum i{\in}UNIV.\ \|column\ i\ A\|)$
    by (*metis* (*no-types*, *lifting*) *Groups.mult-ac*(*2*) *comp-le-1 mult-left-le norm-ge-zero*
sum-mono)
    also have $... \leq (\sum (i::'n){\in}UNIV.\ ?m * (\|A\|_{max}))$
    proof(*unfold norm-vec-def L2-set-def real-norm-def*)
      have $\bigwedge i\ j.\ |column\ i\ A\ \$\ j| \leq \|A\|_{max}$
        using *max-norm-set-proptys Max-ge* **unfolding** *column-def max-norm-def*
by(*simp*, *blast*)
      hence $\bigwedge i\ j.\ |column\ i\ A\ \$\ j|^2 \leq (\|A\|_{max})^2$

by (*metis* (*no-types*, *lifting*) *One-nat-def abs-ge-zero numerals*(*2*) *order-trans-rules*(*23*)

power2-abs power2-le-iff-abs-le)
then have $\bigwedge i.\ (\sum j{\in}UNIV.\ |column\ i\ A\ \$\ j|^2) \leq (\sum (j{::}'m){\in}UNIV.\ (\|A\|_{max})^2)$
by (*meson sum-mono*)
also have $(\sum (j{::}'m){\in}UNIV.\ (\|A\|_{max})^2) = ?m * (\|A\|_{max})^2$ **by** *simp*
ultimately have $\bigwedge i.\ (\sum j{\in}UNIV.\ |column\ i\ A\ \$\ j|^2) \leq ?m * (\|A\|_{max})^2$ **by** *force*
hence $\bigwedge i.\ sqrt\ (\sum j{\in}UNIV.\ |column\ i\ A\ \$\ j|^2) \leq sqrt\ (?m * (\|A\|_{max})^2)$
by(*simp add*: *real-sqrt-le-mono*)
also have $sqrt\ (?m * (\|A\|_{max})^2) \leq sqrt\ ?m * (\|A\|_{max})$
using *max-norm-ge-0 real-sqrt-mult* **by** *auto*
also have ... $\leq ?m * (\|A\|_{max})$
using *sqrt-real-nat-le max-norm-ge-0 mult-right-mono* **by** *blast*
finally show $(\sum i{\in}UNIV.\ sqrt\ (\sum j{\in}UNIV.\ |column\ i\ A\ \$\ j|^2)) \leq (\sum (i{::}'n){\in}UNIV.\ ?m * (\|A\|_{max}))$
by (*meson sum-mono*)
qed
also have $(\sum (i{::}'n){\in}UNIV.\ (\|A\|_{max})) = ?n * (\|A\|_{max})$
using *sum-constant-scale* **by** *auto*
ultimately have $n \leq ?n * ?m * (\|A\|_{max})$
by (*simp add*: *n-def*)**}**
thus $\bigwedge n.\ n \in \{\|A *v\ x\|\ |x.\ \|x\| = 1\} \Longrightarrow n \leq ?n * ?m * (\|A\|_{max})$
by *blast*
qed

2.3 Picard Lindeloef for linear systems

Now we prove our first objective. First we obtain the Lipschitz constant for linear systems of ODEs, and then we prove that IVPs arising from these satisfy the conditions for Picard-Lindeloef theorem (hence, they have a unique solution).

lemma *matrix-lipschitz-constant*:
fixes $A{::}real\hat{}('n{::}finite)\hat{}'n$
shows $dist\ (A *v\ x)\ (A *v\ y) \leq (real\ CARD('n))^2 * (\|A\|_{max}) * dist\ x\ y$
unfolding *dist-norm matrix-vector-mult-diff-distrib*[*symmetric*]
proof(*subst mult-norm-matrix-sgn-eq*[*symmetric*])
have $\|A\|_{op} \leq (\|A\|_{max}) * (real\ CARD('n) * real\ CARD('n))$
by (*metis* (*no-types*) *Groups.mult-ac*(*2*) *op-norm-le-max-norm*)
then have $(\|A\|_{op}) * (\|x - y\|) \leq (real\ CARD('n))^2 * (\|A\|_{max}) * (\|x - y\|)$
by (*simp add*: *cross3-simps*(*11*) *mult-left-mono semiring-normalization-rules*(*29*))
also have $(\|A *v\ sgn\ (x - y)\|) * (\|x - y\|) \leq (\|A\|_{op}) * (\|x - y\|)$
by (*simp add*: *norm-sgn-le-op-norm cross3-simps*(*11*) *mult-left-mono*)
ultimately show $(\|A *v\ sgn\ (x - y)\|) * (\|x - y\|) \leq (real\ CARD('n))^2 * (\|A\|_{max}) * (\|x - y\|)$
using *order-trans-rules*(*23*) **by** *blast*
qed

lemma *picard-lindeloef-linear-system*:
  fixes *A*::*real*^('*n*::*finite*)^'*n*
  assumes $0 < ((real\ CARD('n))^2 * (\|A\|_{max}))$ (**is** $0 < ?L$)
  assumes $0 \leq t$ **and** $t < 1/?L$
  shows *picard-lindeloef* (λ *t s*. *A* $*v$ *s*) {*0*..*t*} *?L 0*
  apply *unfold-locales* **apply**(*simp add*: ‹$0 \leq t$›)
 subgoal by(*simp*, *metis continuous-on-compose2 continuous-on-cong continuous-on-id*

    continuous-on-snd matrix-vector-mult-linear-continuous-on top-greatest)
 subgoal using *matrix-lipschitz-constant max-norm-ge-0 zero-compare-simps*(*4*,*12*)

   unfolding *lipschitz-on-def* **by** *blast*
  apply(*simp-all add*: *assms*)
  subgoal for *r s* **apply**(*subgoal-tac* $|r - s| < 1/?L$)
   apply(*subst* (*asm*) *pos-less-divide-eq*[*of ?L* $|r - s|$ *1*])
   using *assms* **by** *auto*
  done

2.4 Matrix Exponential

The general solution for linear systems of ODEs is an exponential function. Unfortunately, this operation is only available in Isabelle for Banach spaces which are formalised as a class. Hence we need to prove that a specific type is an instance of this class. We define the type and build towards this instantiation in this section.

2.4.1 Squared matrices operations

typedef '*m sqrd-matrix* = *UNIV*::(*real*^'*m*^'*m*) *set*
  morphisms *to-vec sq-mtx-chi* **by** *simp*

declare *sq-mtx-chi-inverse* [*simp*]
   and *to-vec-inverse* [*simp*]

lemma *galois-to-vec-mtx-chi*[*simp*]:(*to-vec A* = *B*) = (*A* = *sq-mtx-chi B*)
  by *auto*

setup-lifting *type-definition-sqrd-matrix*

lift-definition *sq-mtx-ith*::'*m sqrd-matrix* ⇒ '*m* ⇒ (*real*^'*m*) (**infixl** $$ *90*) **is** *vec-nth* .

lift-definition *sq-mtx-vec-prod*::'*m sqrd-matrix* ⇒ (*real*^'*m*) ⇒ (*real*^'*m*) (**infixl** $*_V$ *90*)
  is *matrix-vector-mult* .

lift-definition *sq-mtx-column*::'*m* ⇒ '*m sqrd-matrix* ⇒ (*real*^'*m*)

is $\lambda i\ X.\ column\ i\ (to\text{-}vec\ X)$ **.**

lift-definition *vec-sq-mtx-prod*::$(real\hat{\ }'m) \Rightarrow 'm\ sqrd\text{-}matrix \Rightarrow (real\hat{\ }'m)$ **is** *vector-matrix-mult* **.**

lift-definition *sq-mtx-diag*::$real \Rightarrow ('m::finite)\ sqrd\text{-}matrix$ (diag) **is** *mat* **.**

lift-definition *sq-mtx-transpose*::$('m::finite)\ sqrd\text{-}matrix \Rightarrow 'm\ sqrd\text{-}matrix$ ($\text{-}^{\dagger}$) **is** *transpose* **.**

lift-definition *sq-mtx-row*::$'m \Rightarrow ('m::finite)\ sqrd\text{-}matrix \Rightarrow real\hat{\ }'m$ (row) **is** *row* **.**

lift-definition *sq-mtx-col*::$'m \Rightarrow ('m::finite)\ sqrd\text{-}matrix \Rightarrow real\hat{\ }'m$ (col) **is** *column* **.**

lift-definition *sq-mtx-rows*::$('m::finite)\ sqrd\text{-}matrix \Rightarrow (real\hat{\ }'m)\ set$ **is** *rows* **.**

lift-definition *sq-mtx-cols*::$('m::finite)\ sqrd\text{-}matrix \Rightarrow (real\hat{\ }'m)\ set$ **is** *columns* **.**

lemma *sq-mtx-eq-iff*:
shows $(\bigwedge i.\ A\ \$\$\ i = B\ \$\$\ i) \Longrightarrow A = B$
and $(\bigwedge i\ j.\ A\ \$\$\ i\ \$\ j = B\ \$\$\ i\ \$\ j) \Longrightarrow A = B$
by(*transfer*, *simp add*: *vec-eq-iff*)+

lemma *sq-mtx-vec-prod-eq*: $m *_V x = (\chi\ i.\ sum\ (\lambda j.\ ((m\$\$i)\$j) * (x\$j))\ UNIV)$
by(*transfer*, *simp add*: *matrix-vector-mult-def*)

lemma *sq-mtx-transpose-transpose*[*simp*]:$(A^{\dagger})^{\dagger} = A$
by(*transfer*, *simp*)

lemma *transpose-mult-vec-canon-row*[*simp*]:$(A^{\dagger}) *_V (\mathrm{e}\ i) = \mathrm{row}\ i\ A$
by *transfer* (*simp add*: *row-def transpose-def axis-def matrix-vector-mult-def*)

lemma *row-ith*[*simp*]:$\mathrm{row}\ i\ A = A\ \$\$\ i$
by *transfer* (*simp add*: *row-def*)

lemma *mtx-vec-prod-canon*:$A *_V (\mathrm{e}\ i) = \mathrm{col}\ i\ A$
by (*transfer*, *simp add*: *matrix-vector-mult-basis*)

2.4.2 Squared matrices form Banach space

instantiation *sqrd-matrix* :: (*finite*) *ring*
begin

lift-definition *plus-sqrd-matrix* :: $'a\ sqrd\text{-}matrix \Rightarrow 'a\ sqrd\text{-}matrix \Rightarrow 'a\ sqrd\text{-}matrix$ **is** $(+)$ **.**

lift-definition *zero-sqrd-matrix* :: $'a\ sqrd\text{-}matrix$ **is** 0 **.**

lift-definition *uminus-sqrd-matrix* ::$'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix* **is** *uminus* **.**

lift-definition *minus-sqrd-matrix* :: $'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix*
is $(-)$ **.**

lift-definition *times-sqrd-matrix* :: $'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix*
is $(**)$ **.**

declare *plus-sqrd-matrix.rep-eq* [*simp*]
and *minus-sqrd-matrix.rep-eq* [*simp*]

instance apply *intro-classes*
by(*transfer*, *simp add*: *algebra-simps matrix-mul-assoc matrix-add-rdistrib matrix-add-ldistrib*)+

end

lemma *sq-mtx-plus-ith*[*simp*]:$(A + B)$ \$\$ $i = A$ \$\$ $i + B$ \$\$ i
by(*unfold plus-sqrd-matrix-def*, *transfer*, *simp*)

lemma *sq-mtx-minus-ith*[*simp*]:$(A - B)$ \$\$ $i = A$ \$\$ $i - B$ \$\$ i
by(*unfold minus-sqrd-matrix-def*, *transfer*, *simp*)

lemma *mtx-vec-prod-add-rdistr*:$(A + B) *_V x = A *_V x + B *_V x$
unfolding *plus-sqrd-matrix-def* **apply**(*transfer*)
by (*simp add*: *matrix-vector-mult-add-rdistrib*)

lemma *mtx-vec-prod-minus-rdistrib*:$(A - B) *_V x = A *_V x - B *_V x$
unfolding *minus-sqrd-matrix-def* **by**(*transfer*, *simp add*: *matrix-vector-mult-diff-rdistrib*)

lemma *sq-mtx-times-vec-assoc*: $(A * B) *_V x0 = A *_V (B *_V x0)$
by (*transfer*, *simp add*: *matrix-vector-mul-assoc*)

lemma *sq-mtx-vec-mult-sum-cols*:$A *_V x = sum\ (\lambda i.\ x \$ i *_R \mathrm{col}\ i\ A)\ UNIV$
by(*transfer*) (*simp add*: *matrix-mult-sum scalar-mult-eq-scaleR*)

instantiation *sqrd-matrix* :: (*finite*) *real-normed-vector*
begin

definition *norm-sqrd-matrix* :: $'a$ *sqrd-matrix* $\Rightarrow$ *real* **where** $\|A\| = \|to\text{-}vec\ A\|_{op}$

lift-definition *scaleR-sqrd-matrix*::*real* $\Rightarrow$ $'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix* **is** *scaleR*
.

definition *sgn-sqrd-matrix* :: $'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix*
where *sgn-sqrd-matrix* $A = (inverse\ (\|A\|)) *_R A$

definition *dist-sqrd-matrix* :: $'a$ *sqrd-matrix* $\Rightarrow$ $'a$ *sqrd-matrix* $\Rightarrow$ *real*
where *dist-sqrd-matrix* $A\ B = \|A - B\|$

definition *uniformity-sqrd-matrix* :: $('a\ \mathit{sqrd\text{-}matrix} \times 'a\ \mathit{sqrd\text{-}matrix})$ *filter*
where *uniformity-sqrd-matrix* $= (\mathit{INF}\ e{:}\{0{<}..\}.\ \mathit{principal}\ \{(x, y).\ \mathit{dist}\ x\ y < e\})$

definition *open-sqrd-matrix* :: $'a$ *sqrd-matrix set* $\Rightarrow$ *bool*
where *open-sqrd-matrix* $U = (\forall x{\in}U.\ \forall_F\ (x', y)\ \mathit{in\ uniformity}.\ x' = x \longrightarrow y \in U)$

instance apply *intro-classes*
unfolding *sgn-sqrd-matrix-def open-sqrd-matrix-def dist-sqrd-matrix-def uniformity-sqrd-matrix-def*
prefer *10* **apply**(*transfer*, *simp add*: *norm-sqrd-matrix-def op-norm-triangle*)
prefer *9* **apply**(*simp-all add*: *norm-sqrd-matrix-def zero-sqrd-matrix-def op-norm-zero-iff*)
by(*transfer*, *simp add*: *norm-sqrd-matrix-def op-norm-scaleR algebra-simps*)+

end

lemma *sq-mtx-scaleR-ith*[*simp*]: $(c *_R A)\ \$\$\ i = (c\ *_R (A\ \$\$\ i))$
by(*unfold scaleR-sqrd-matrix-def*, *transfer*, *simp*)

lemma *le-mtx-norm*: $m \in \{\|A *_V x\|\ |x.\ \|x\| = 1\} \Longrightarrow m \le \|A\|$
using *cSup-upper*[*of* - $\{\|(\mathit{to\text{-}vec}\ A) *v\ x\|\ |\ x.\ \|x\| = 1\}$]
by (*simp add*: *op-norm-set-proptys*(*2*) *norm-sqrd-matrix-def sq-mtx-vec-prod.rep-eq*)

lemma *norm-vec-mult-le*: $\|A *_V x\| \le (\|A\|) * (\|x\|)$
by (*simp add*: *norm-matrix-le-mult-op-norm norm-sqrd-matrix-def sq-mtx-vec-prod.rep-eq*)

lemma *sq-mtx-norm-le-sum-col*: $\|A\| \le (\sum i{\in}\mathit{UNIV}.\ \|\mathrm{col}\ i\ A\|)$
using *op-norm-le-sum-column*[*of to-vec A*] **apply**(*simp add*: *norm-sqrd-matrix-def*)
by(*transfer*, *simp add*: *op-norm-le-sum-column*)

lemma *norm-le-transpose*: $\|A\| \le \|A^\dagger\|$
apply(*simp add*: *norm-sqrd-matrix-def*, *transfer*, *simp add*: *transpose-def*)
using *op-norm-set-proptys*(*3*) **apply**(*rule cSup-least*)
proof(*clarsimp*)
fix $A{::}\mathit{real}\hat{\ }'a\hat{\ }'a$ **and** $x{::}\mathit{real}\ \hat{\ }\ 'a$ **assume** $\|x\| = 1$
have $\mathit{obs}{:}\forall x.\ \|x\| = 1 \longrightarrow (\|A *v\ x\|) \le \mathit{sqrt}\ ((\|\mathit{transpose}\ A ** A\|_{op})) * (\|x\|)$
using *norm-matrix-vec-mult-le-transpose* **by** *blast*
have $(\|A\|_{op}) \le \mathit{sqrt}\ ((\|\mathit{transpose}\ A ** A\|_{op}))$
using *op-norm-set-proptys*(*3*) **apply**(*rule cSup-least*) **using** *obs* **by** *clarsimp*
then have $((\|A\|_{op}))^2 \le (\|\mathit{transpose}\ A ** A\|_{op})$
using *power-mono*[*of* $(\|A\|_{op})$ - *2*] *norm-matrix-le-op-norm-ge-0* **by** *force*
also have ... $\le (\|\mathit{transpose}\ A\|_{op}) * (\|A\|_{op})$
using *op-norm-matrix-matrix-mult-le* **by** *blast*
finally have $((\|A\|_{op}))^2 \le (\|\mathit{transpose}\ A\|_{op}) * (\|A\|_{op})$ **by** *linarith*
hence $(\|A\|_{op}) \le (\|\mathit{transpose}\ A\|_{op})$
using *sq-le-cancel*[*of* $(\|A\|_{op})$] *norm-matrix-le-op-norm-ge-0* **by** *blast*
thus $(\|A *v\ x\|) \le \mathit{op\text{-}norm}\ (\chi\ i\ j.\ A\ \$\ j\ \$\ i)$
unfolding *transpose-def* **using** $\langle\|x\| = 1\rangle$ *order-trans norm-matrix-le-op-norm*
by *blast*

qed

lemma *norm-eq-norm-transpose*[*simp*]: $\|A^\dagger\| = \|A\|$
using *norm-le-transpose*[*of A*] **and** *norm-le-transpose*[*of* $A^\dagger$] **by** *simp*

lemma *norm-column-le-norm*: $\|A$ $$ $i\| \leq \|A\|$
using *norm-vec-mult-le*[*of* $A^\dagger$ e *i*] **by** *simp*

instantiation *sqrd-matrix* :: (*finite*) *real-normed-algebra-1*
begin

lift-definition *one-sqrd-matrix* :: *'a sqrd-matrix* **is** *sq-mtx-chi* (*mat 1*) **.**

lemma *sq-mtx-one-idty*: $1 * A = A$ $A * 1 = A$ **for** *A*::*'a sqrd-matrix*
by(*transfer*, *transfer*, *unfold mat-def matrix-matrix-mult-def*, *simp add*: *vec-eq-iff*)+

lemma *sq-mtx-norm-1*: $\|(1::'a\ sqrd\text{-}matrix)\| = 1$
unfolding *one-sqrd-matrix-def norm-sqrd-matrix-def* **apply** *simp*
apply(*subst cSup-eq*[*of - 1*])
using *ex-norm-eq-1* **by** *auto*

lemma *sq-mtx-norm-times*: $\|A * B\| \leq (\|A\|) * (\|B\|)$ **for** *A*::*'a sqrd-matrix*
unfolding *norm-sqrd-matrix-def times-sqrd-matrix-def* **by**(*simp add*: *op-norm-matrix-matrix-mult-le*)

instance apply *intro-classes*
apply(*simp-all add*: *sq-mtx-one-idty sq-mtx-norm-1 sq-mtx-norm-times*)
apply(*simp-all add*: *sq-mtx-chi-inject vec-eq-iff one-sqrd-matrix-def zero-sqrd-matrix-def mat-def*)
by(*transfer*, *simp add*: *scalar-matrix-assoc matrix-scalar-ac*)+

end

lemma *sq-mtx-one-vec*: $1 *_V s = s$
by (*auto simp*: *sq-mtx-vec-prod-def one-sqrd-matrix-def*
mat-def vec-eq-iff matrix-vector-mult-def)

lemma *Cauchy-cols*:
fixes *X* :: *nat* ⇒ (*'a*::*finite*) *sqrd-matrix*
assumes *Cauchy X*
shows *Cauchy* ($\lambda n.$ col i (X n))
proof(*unfold Cauchy-def dist-norm*, *clarsimp*)
fix ε::*real* **assume** $\varepsilon > 0$
from *this* **obtain** *M* **where** *M-def*:$\forall m \geq M.\ \forall n \geq M.\ \|X\ m - X\ n\| < \varepsilon$
using ‹*Cauchy X*› **unfolding** *Cauchy-def* **by**(*simp add*: *dist-sqrd-matrix-def*)
blast
{**fix** *m n* **assume** $m \geq M$ **and** $n \geq M$
hence $\varepsilon > \|X\ m - X\ n\|$
using *M-def* **by** *blast*

moreover have $\|X\ m - X\ n\| \geq \|(X\ m - X\ n) *_V \mathrm{e}\ i\|$
by(*rule le-mtx-norm*[*of - X m − X n*], *force*)
moreover have $\|(X\ m - X\ n) *_V \mathrm{e}\ i\| = \|X\ m *_V \mathrm{e}\ i - X\ n *_V \mathrm{e}\ i\|$
by (*simp add*: *mtx-vec-prod-minus-rdistrib*)
moreover have ... $= \|\mathrm{col}\ i\ (X\ m) - \mathrm{col}\ i\ (X\ n)\|$
by (*simp add*: *mtx-vec-prod-minus-rdistrib mtx-vec-prod-canon*)
ultimately have $\|\mathrm{col}\ i\ (X\ m) - \mathrm{col}\ i\ (X\ n)\| < \varepsilon$
by *linarith*}
thus $\exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ \|\mathrm{col}\ i\ (X\ m) - \mathrm{col}\ i\ (X\ n)\| < \varepsilon$
by *blast*
qed

lemma *col-convergent*:
assumes $\forall i.\ (\lambda n.\ \mathrm{col}\ i\ (X\ n)) \longrightarrow L\ \$\ i$
shows *convergent X*
unfolding *convergent-def* **proof**(*rule-tac x=sq-mtx-chi* (*transpose L*) **in** *exI*)
let *?L* = *sq-mtx-chi* (*transpose L*)
show $X \longrightarrow\ ?L$
proof(*unfold LIMSEQ-def dist-norm*, *clarsimp*)
fix ε::*real* **assume** $\varepsilon > 0$
let *?a* = *CARD*(*'a*) **fix** ε::*real* **assume** $\varepsilon > 0$
hence $\varepsilon\ /\ ?a > 0$
by *simp*
from *this* **and** *assms* **have** $\forall i.\ \exists\ N.\ \forall n{\geq}N.\ \|\mathrm{col}\ i\ (X\ n) - L\ \$\ i\| < \varepsilon/?a$
unfolding *LIMSEQ-def dist-norm convergent-def* **by** *blast*
then obtain N **where** $\forall i.\ \forall n{\geq}N.\ \|\mathrm{col}\ i\ (X\ n) - L\ \$\ i\| < \varepsilon/?a$
using *finite-nat-minimal-witness*[*of* $\lambda\ i\ n.\ \|\mathrm{col}\ i\ (X\ n) - L\ \$\ i\| < \varepsilon/?a$] **by** *blast*
also have $\bigwedge i\ n.\ (\mathrm{col}\ i\ (X\ n) - L\ \$\ i) = (\mathrm{col}\ i\ (X\ n - ?L))$
unfolding *minus-sqrd-matrix-def* **by**(*transfer*, *simp add*: *transpose-def vec-eq-iff column-def*)
ultimately have *N-def*: $\forall i.\ \forall n{\geq}N.\ \|\mathrm{col}\ i\ (X\ n - ?L)\| < \varepsilon/?a$
by *auto*
have $\forall n{\geq}N.\ \|X\ n - ?L\| < \varepsilon$
proof(*rule allI*, *rule impI*)
fix n::*nat* **assume** $N \leq n$
hence $\forall\ i.\ \|\mathrm{col}\ i\ (X\ n - ?L)\| < \varepsilon/?a$
using *N-def* **by** *blast*
hence $(\sum i{\in}UNIV.\ \|\mathrm{col}\ i\ (X\ n - ?L)\|) < (\sum (i{::}'a){\in}UNIV.\ \varepsilon/?a)$
using *sum-strict-mono*[*of -* $\lambda i.\ \|\mathrm{col}\ i\ (X\ n - ?L)\|$] **by** *force*
moreover have $\|X\ n - ?L\| \leq (\sum i{\in}UNIV.\ \|\mathrm{col}\ i\ (X\ n - ?L)\|)$
using *sq-mtx-norm-le-sum-col* **by** *blast*
moreover have $(\sum (i{::}'a){\in}UNIV.\ \varepsilon/?a) = \varepsilon$
by *force*
ultimately show $\|X\ n - ?L\| < \varepsilon$
by *linarith*
qed
thus $\exists no.\ \forall n{\geq}no.\ \|X\ n - ?L\| < \varepsilon$
by *blast*

qed
qed

instance *sqrd-matrix* :: (*finite*) *banach*
proof(*standard*)
fix X::*nat* $\Rightarrow$ *'a sqrd-matrix*
assume *Cauchy X*
have $\bigwedge i$. *Cauchy* (λn. col i (X n))
using ‹*Cauchy X*› *Cauchy-cols* **by** *blast*
hence *obs*:$\forall i$. $\exists!$ L. (λn. col i (X n)) $\longrightarrow$ L
using *Cauchy-convergent convergent-def LIMSEQ-unique* **by** *fastforce*
define L **where** $L = (\chi\ i.\ lim\ (\lambda n.\ \text{col}\ i\ (X\ n)))$
from *this* **and** *obs* **have** $\forall i$. (λn. col i (X n)) $\longrightarrow$ L \$ i
using *theI-unique*[*of* λL. (λn. col - (X n)) $\longrightarrow$ L L \$ -] **by** (*simp add*: *lim-def*)
thus *convergent X*
using *col-convergent* **by** *blast*
qed

2.5 Flow for squared matrix systems

Finally, we can use the *exp* operation to characterize the general solutions for linear systems of ODEs. After this, we show that IVPs with these systems have a unique solution (using the Picard Lindeloef locale) and explicitly write it via the local flow locale.

lemma *mtx-vec-prod-has-derivative-mtx-vec-prod*:
assumes $\bigwedge$ i j. D (λt. (A t) \$\$ i \$ j) $\mapsto$ ($\lambda\tau$. τ $*_R$ (A' t) \$\$ i \$ j) (*at t within s*)
and ($\lambda\tau$. τ $*_R$ (A' t) $*_V$ x) = g'
shows D (λt. A t $*_V$ x) $\mapsto$ g' *at t within s*
using *assms*(*2*) **apply** *safe* **apply**(*rule ssubst*[*of* g' ($\lambda\tau$. τ $*_R$ (A' t) $*_V$ x)], *simp*)
unfolding *sq-mtx-vec-mult-sum-cols*
apply(*rule-tac* $f'1$=λi τ. τ $*_R$ (x \$ i $*_R$ col i (A' t)) **in** *derivative-eq-intros*(*9*))
apply(*simp-all add*: *scaleR-right.sum*)
apply(*rule-tac* $g'1$=$\lambda\tau$. τ $*_R$ col i (A' t) **in** *derivative-eq-intros*(*4*), *simp-all add*: *mult.commute*)
using *assms* **unfolding** *sq-mtx-col-def column-def* **apply**(*transfer*, *simp*)
apply(*rule has-derivative-vec-lambda*)
by(*simp add*: *scaleR-vec-def*)

lemma *has-derivative-mtx-ith*:
assumes D A $\mapsto$ (λh. h $*_R$ A' x) *at x within s*
shows D (λt. A t \$\$ i) $\mapsto$ (λh. h $*_R$ A' x \$\$ i) *at x within s*
unfolding *has-derivative-def tendsto-iff dist-norm* **apply** *safe*
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
proof(*clarsimp*)
fix ε::*real* **assume** $0 < \varepsilon$

```
  let ?x = netlimit (at x within s) let ?Δ y = y − ?x and ?ΔA y = A y − A ?x
  let ?P e = λy. inverse |?Δ y| ∗ (‖?ΔA y − ?Δ y ∗R A′ x‖) < e
  let ?Q = λy. inverse |?Δ y| ∗ (‖A y $$ i − A ?x $$ i − ?Δ y ∗R A′ x $$ i‖)
< ε
  from assms have ∀ e>0. eventually (?P e) (at x within s)
    unfolding has-derivative-def tendsto-iff by auto
  hence eventually (?P ε) (at x within s)
    using ‹0 < ε› by blast
  thus eventually ?Q (at x within s)
  proof(rule-tac P=?P ε in eventually-mono, simp-all)
    let ?u y i = A y $$ i − A ?x $$ i − ?Δ y ∗R A′ x $$ i
    fix y assume hyp: inverse |?Δ y| ∗ (‖?ΔA y − ?Δ y ∗R A′ x‖) < ε
    have ‖?u y i‖ = ‖(?ΔA y − ?Δ y ∗R A′ x) $$ i‖
      by simp
    also have ... ≤ (‖?ΔA y − ?Δ y ∗R A′ x‖)
      using norm-column-le-norm by blast
    ultimately have ‖?u y i‖ ≤ ‖?ΔA y − ?Δ y ∗R A′ x‖
      by linarith
    hence inverse |?Δ y| ∗ (‖?u y i‖) ≤ inverse |?Δ y| ∗ (‖?ΔA y − ?Δ y ∗R
A′ x‖)
      by (simp add: mult-left-mono)
    thus inverse |?Δ y| ∗ (‖?u y i‖) < ε
      using hyp by linarith
  qed
qed

lemma exp-has-vderiv-on-linear:
  fixes A::(('a::finite) sqrd-matrix)
  shows D (λt. exp ((t − t0) ∗R A) ∗V x0) = (λt. A ∗V (exp ((t − t0) ∗R A) ∗V
x0)) on T
  unfolding has-vderiv-on-def has-vector-derivative-def apply clarsimp
 apply(rule-tac A′=λt. A ∗ exp ((t − t0) ∗R A) in mtx-vec-prod-has-derivative-mtx-vec-prod)
  apply(rule has-derivative-vec-nth)
  apply(rule has-derivative-mtx-ith)
  apply(rule-tac f′=id in exp-scaleR-has-derivative-right)
   apply(rule-tac f′1=id and g′1=λx. 0 in derivative-eq-intros(11))
    apply(rule derivative-eq-intros)
  by(simp-all add: fun-eq-iff exp-times-scaleR-commute sq-mtx-times-vec-assoc)

lemma picard-lindeloef-sq-mtx:
  fixes A::('n::finite) sqrd-matrix
  assumes 0 < ((real CARD('n))^2 ∗ (‖to-vec A‖_max)) (is 0 < ?L)
  assumes 0 ≤ t and t < 1/?L
  shows picard-lindeloef (λ t s. A ∗V s) {0..t} ?L 0
  apply unfold-locales apply(simp add: ‹0 ≤ t›)
   subgoal by(transfer, simp, metis continuous-on-compose2 continuous-on-cong
continuous-on-id
        continuous-on-snd matrix-vector-mult-linear-continuous-on top-greatest)
 subgoal apply transfer using matrix-lipschitz-constant max-norm-ge-0 zero-compare-simps(4,12)
```

```
    unfolding lipschitz-on-def by blast
  apply(simp-all add: assms)
  subgoal for r s apply(subgoal-tac |r − s| < 1/?L)
     apply(subst (asm) pos-less-divide-eq[of ?L |r − s| 1])
    using assms by auto
  done

lemma local-flow-exp:
  fixes A::('n::finite) sqrd-matrix
  assumes 0 < ((real CARD('n))^2 ∗ (‖to-vec A‖_max)) (is 0 < ?L)
  assumes 0 ≤ t and t < 1/?L
  shows local-flow (λs. A ∗_V s) {0..t} ((real CARD('n))^2 ∗ (‖to-vec A‖_max)) ((λt
s. exp (t ∗_R A) ∗_V s))
  unfolding local-flow-def local-flow-axioms-def apply safe
  using picard-lindeloef-sq-mtx assms apply blast
  using exp-has-vderiv-on-linear[of 0] apply force
  by(auto simp: sq-mtx-one-vec)

end
theory cat2funcset
  imports ../hs-prelims Transformer-Semantics.Kleisli-Quantale

begin
```

Chapter 3

Hybrid System Verification

— We start by deleting some conflicting notation and introducing some new.
type-synonym *'a pred* = *'a* ⇒ *bool*

3.1 Verification of regular programs

First we add lemmas for computation of weakest liberal preconditions (wlps).

lemma *ffb-eta*[*simp*]:$fb_{\mathcal{F}}$ η $X = X$
  unfolding *ffb-def* **by**(*simp add*: *kop-def klift-def map-dual-def*)

lemma *ffb-eq*:$fb_{\mathcal{F}}$ $F\ X = \{s.\ \forall y.\ y \in F\ s \longrightarrow y \in X\}$
  unfolding *ffb-def* **apply**(*simp add*: *kop-def klift-def map-dual-def*)
  unfolding *dual-set-def f2r-def r2f-def* **by** *auto*

lemma *ffb-eq-univD*:$fb_{\mathcal{F}}$ $F\ P = UNIV \Longrightarrow (\forall y.\ y \in (F\ x) \longrightarrow y \in P)$
proof
  fix *y* **assume** $fb_{\mathcal{F}}$ $F\ P = UNIV$
  hence $UNIV = \{s.\ \forall y.\ y \in (F\ s) \longrightarrow y \in P\}$
    by(*subst ffb-eq*[*THEN sym*], *simp*)
  hence $\bigwedge x.\ \{x\} = \{s.\ s = x \wedge (\forall y.\ y \in (F\ s) \longrightarrow y \in P)\}$
    by *auto*
  then show *s2p* $(F\ x)\ y \longrightarrow y \in P$
    by *auto*
qed

Next, we introduce assignments and their wlps.

abbreviation *vec-upd* :: $('a\hat{\ }'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a\hat{\ }'b$ (-(*2*[- :== -]) [*70*, *65*] *61*)
  where $x[i :== a] \equiv (\chi\ j.\ (\textit{if}\ j = i\ \textit{then}\ a\ \textit{else}\ (x\ \$\ j)))$

abbreviation *assign* :: $'b \Rightarrow ('a\hat{\ }'b \Rightarrow 'a) \Rightarrow ('a\hat{\ }'b) \Rightarrow ('a\hat{\ }'b)$ *set* ((*2*[- ::== -]) [*70*, *65*] *61*)
  where $[x ::== expr] \equiv (\lambda s.\ \{s[x :== expr\ s]\})$

lemma *ffb-assign*[*simp*]: $fb_{\mathcal{F}}$ $([x ::== expr])\ Q = \{s.\ (s[x :== expr\ s]) \in Q\}$

by(*subst ffb-eq*, *simp*)

The wlp of a (kleisli) composition is just the composition of the wlps.

lemma *ffb-kcomp*:$fb_{\mathcal{F}}$ $(G \circ_K F)$ P = $fb_{\mathcal{F}}$ G $(fb_{\mathcal{F}}$ F $P)$
unfolding *ffb-def* **apply**(*simp add*: *kop-def klift-def map-dual-def*)
unfolding *dual-set-def f2r-def r2f-def* **by**(*auto simp*: *kcomp-def*)

We also have an implementation of the conditional operator and its wlp.

definition *ifthenelse* :: $'a$ *pred* $\Rightarrow$ $('a \Rightarrow 'b$ *set*$)$ $\Rightarrow$ $('a \Rightarrow 'b$ *set*$)$ $\Rightarrow$ $('a \Rightarrow 'b$ *set*$)$
(*IF - THEN - ELSE - FI* [*64*,*64*,*64*] *63*) **where**
IF P THEN X ELSE Y FI $\equiv$ $(\lambda$ x. *if* P x *then* X x *else* Y $x)$

lemma *ffb-if-then-else*:
assumes $P \cap \{s.\ T\ s\} \leq fb_{\mathcal{F}}\ X\ Q$
and $P \cap \{s.\ \neg\ T\ s\} \leq fb_{\mathcal{F}}\ Y\ Q$
shows $P \leq fb_{\mathcal{F}}$ (*IF T THEN X ELSE Y FI*) Q
using *assms* **apply**(*subst ffb-eq*)
apply(*subst* (*asm*) *ffb-eq*)+
unfolding *ifthenelse-def* **by** *auto*

lemma *ffb-if-then-elseD*:
assumes $T\ x \longrightarrow x \in fb_{\mathcal{F}}\ X\ Q$
and $\neg\ T\ x \longrightarrow x \in fb_{\mathcal{F}}\ Y\ Q$
shows $x \in fb_{\mathcal{F}}$ (*IF T THEN X ELSE Y FI*) Q
using *assms* **apply**(*subst ffb-eq*)
apply(*subst* (*asm*) *ffb-eq*)+
unfolding *ifthenelse-def* **by** *auto*

The final wlp we add is that of the finite iteration.

lemma *kstar-inv*:$I \leq \{s.\ \forall y.\ y \in F\ s \longrightarrow y \in I\} \Longrightarrow I \leq \{s.\ \forall y.\ y \in ($*kpower* $F\ n\ s) \longrightarrow y \in I\}$
apply(*induct n*, *simp*)
by(*auto simp*: *kcomp-prop*)

lemma *ffb-star-induct-self*:$I \leq fb_{\mathcal{F}}\ F\ I \Longrightarrow I \subseteq fb_{\mathcal{F}}$ (*kstar* F) I
apply(*subst ffb-eq*, *subst* (*asm*) *ffb-eq*)
unfolding *kstar-def* **apply** *clarsimp*
using *kstar-inv* **by** *blast*

lemma *ffb-starI*:
assumes $P \leq I$ **and** $I \leq fb_{\mathcal{F}}\ F\ I$ **and** $I \leq Q$
shows $P \leq fb_{\mathcal{F}}$ (*kstar* F) Q
proof−
from *assms*(*2*) **have** $I \subseteq fb_{\mathcal{F}}$ (*kstar* F) I
using *ffb-star-induct-self* **by** *blast*
then have $P \leq fb_{\mathcal{F}}$ (*kstar* F) I
using *assms*(*1*) **by** *auto*
from *this* **and** *assms*(*3*) **show** *?thesis*
by(*subst ffb-eq*, *subst* (*asm*) *ffb-eq*, *auto*)

qed

3.2 Verification of hybrid programs

3.2.1 Verification by providing solutions

abbreviation *guards* :: $('a \Rightarrow bool) \Rightarrow (real \Rightarrow 'a) \Rightarrow (real\ set) \Rightarrow bool$ (- $\rhd$ - - [*70*, *65*] *61*)
  where $G \rhd x\ T \equiv \forall\ r \in T.\ G\ (x\ r)$

definition *ivp-sols* $f\ T\ t_0\ s = \{x\ |x.\ (D\ x = (f \circ x)\ on\ T) \wedge x\ t_0 = s \wedge t_0 \in T\}$

lemma *ivp-solsI*:
  assumes $D\ x = (f \circ x)\ on\ T\ x\ t_0 = s\ t_0 \in T$
  shows $x \in$ *ivp-sols* $f\ T\ t_0\ s$
  using *assms* **unfolding** *ivp-sols-def* **by** *blast*

lemma *ivp-solsD*:
  assumes $x \in$ *ivp-sols* $f\ T\ t_0\ s$
  shows $D\ x = (f \circ x)\ on\ T$
    and $x\ t_0 = s$ **and** $t_0 \in T$
  using *assms* **unfolding** *ivp-sols-def* **by** *auto*

We use closed segments instead of closed intervals for the following definition due to the following property.

lemma $(t::real) \in \{0{-}{-}t\}$
  by (*rule ends-in-segment*(*2*))

lemma $(t::real) \in \{0..t\}$
  apply *auto*
  oops

definition *g-orbital* $f\ T\ t_0\ G\ s = \bigcup\ \{\{x\ t|t.\ t \in T \wedge G \rhd x\ \{t_0{-}{-}t\}\ \}|x.\ x \in$ *ivp-sols* $f\ T\ t_0\ s\}$

lemma *g-orbital-eq*: *g-orbital* $f\ T\ t_0\ G\ s =$
  $\{x\ t\ |t\ x.\ t \in T \wedge (D\ x = (f \circ x)\ on\ T) \wedge x\ t_0 = s \wedge t_0 \in T \wedge G \rhd x\ \{t_0{-}{-}t\}\}$
  unfolding *g-orbital-def ivp-sols-def* **by** *auto*

lemma *g-orbital* $f\ T\ t_0\ G\ s = (\bigcup\ x \in$ *ivp-sols* $f\ T\ t_0\ s.\ \{x\ t|t.\ t \in T \wedge G \rhd x\ \{t_0{-}{-}t\}\})$
  unfolding *g-orbital-def ivp-sols-def* **by** *auto*

abbreviation *g-evol* $::(('a::banach) \Rightarrow 'a) \Rightarrow real\ set \Rightarrow 'a\ pred \Rightarrow 'a \Rightarrow 'a\ set$ ((*1*[x'=-]- & -))
  where $[x'{=}f]T\ \&\ G \equiv (\lambda\ s.$ *g-orbital* $f\ T\ 0\ G\ s)$

lemmas *g-evol-def* = *g-orbital-eq*[**where** $t_0{=}0$]

```
lemma g-evolI:
  assumes D x = (f ∘ x) on T x 0 = s
    and 0 ∈ T t ∈ T and G ▷ x {0−−t}
  shows x t ∈ ([x´=f] T & G) s
  using assms unfolding g-orbital-def ivp-sols-def by blast

lemma g-evolD:
  assumes s' ∈ ([x´=f] T & G) s
  obtains x and t where x ∈ ivp-sols f T 0 s
  and D x = (f ∘ x) on T x 0 = s
  and x t = s' and 0 ∈ T t ∈ T and G ▷ x {0−−t}
  using assms unfolding g-orbital-def ivp-sols-def by blast

context local-flow
begin

lemma in-ivp-sols: (λt. φ t s) ∈ ivp-sols f T 0 s
  by(auto intro: ivp-solsI simp: ivp init-time)

definition orbit s = g-orbital f T 0 (λs. True) s

lemma orbit-eq[simp]: orbit s = {φ t s| t. t ∈ T}
  unfolding orbit-def g-evol-def
  by(auto intro: usolves-ivp intro!: ivp simp: init-time)

lemma g-evol-collapses:
  shows ([x´=f] T & G) s = {φ t s| t. t ∈ T ∧ G ▷ (λr. φ r s) {0−−t}} (is -
= ?gorbit)
proof(rule subset-antisym, simp-all only: subset-eq)
  {fix s' assume s' ∈ ([x´=f] T & G) s
    then obtain x and t where x-ivp:D x = (f ∘ x) on T
      x 0 = s and x t = s' and t ∈ T and guard:G ▷ x {0−−t}
      unfolding g-orbital-eq by blast
    hence obs:∀ τ∈{0−−t}. x τ = φ τ s
      using usolves-ivp[of x s] closed-segment-subset-domainI init-time comp-def
      by (metis (mono-tags, lifting) has-vderiv-eq)
    hence G ▷ (λr. φ r s) {0−−t}
      using guard by simp
    hence s' ∈ ?gorbit
      using ‹x t = s'› ‹t ∈ T› obs by blast}
  thus ∀ s'∈([x´=f] T & G) s. s' ∈ ?gorbit
    by blast
next
  {fix s' assume s' ∈ ?gorbit
    then obtain t where G ▷ (λr. φ r s) {0−−t} and t ∈ T and φ t s = s'
      by blast
    hence s' ∈ ([x´=f] T & G) s
      by(auto intro: g-evolI simp: ivp init-time)}
  thus ∀ s'∈?gorbit. s' ∈ ([x´=f] T & G) s
```

by *blast*
qed

lemma *ffb-orbit*: $fb_{\mathcal{F}}$ (*orbit*) $Q = \{s.\ \forall\ t \in T.\ \varphi\ t\ s \in Q\}$
unfolding *orbit-eq ffb-eq* **by** *auto*

lemma *ffb-g-orbit*: $fb_{\mathcal{F}}$ ($[x'{=}f]\,T$ & G) $Q = \{s.\ \forall t{\in}T.\ (G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\})$ $\longrightarrow (\varphi\ t\ s) \in Q\}$
unfolding *g-evol-collapses ffb-eq* **by** *auto*

end

lemma (**in** *global-flow*) *ivp-sols-collapse*[*simp*]: *ivp-sols f UNIV 0 s* $= \{(\lambda t.\ \varphi\ t\ s)\}$
by(*auto intro*: *usolves-ivp simp*: *ivp-sols-def ivp*)

The previous lemma allows us to compute wlps for known systems of ODEs. We can also implement a version of it as an inference rule. A simple computation of a wlp is shown immmediately after.

lemma *dSolution*:
assumes *local-flow f T L* φ
and $\forall s.\ s \in P \longrightarrow (\forall\ t \in T.\ (G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0..t\}) \longrightarrow (\varphi\ t\ s) \in Q)$
shows $P \leq fb_{\mathcal{F}}$ ($[x'{=}f]\,T$ & G) Q
using *assms* **apply**(*subst local-flow.ffb-g-orbit*)
by (*auto simp*: *Starlike.closed-segment-eq-real-ivl*)

lemma *ffb-line*: $0 \leq t \Longrightarrow fb_{\mathcal{F}}$ ($[x'{=}\lambda t.\ c]\{0..t\}$ & G) $Q =$
$\{x.\ \forall\ \tau \in \{0..t\}.\ (G \rhd (\lambda r.\ x + r *_R c)\ \{0..\tau\}) \longrightarrow (x + \tau *_R c) \in Q\}$
apply(*subst local-flow.ffb-g-orbit*[*of* λ *t. c - 1/(t + 1)* (λ *t x. x + t* $*_R$ *c*)])
by(*auto simp*: *line-is-local-flow closed-segment-eq-real-ivl*)

3.2.2 Verification with differential invariants

We derive domain specific rules of differential dynamic logic (dL). In each subsubsection, we first derive the dL axioms (named below with two capital letters and "D" being the first one). This is done mainly to prove that there are minimal requirements in Isabelle to get the dL calculus. Then we prove the inference rules which are used in our verification proofs.

Differential Weakening

lemma *DW*:
shows $fb_{\mathcal{F}}$ ($[x'{=}f]\,T$ & G) $Q = fb_{\mathcal{F}}$ ($[x'{=}f]\,T$ & G) $\{s.\ G\ s \longrightarrow s \in Q\}$
by(*auto intro*: *g-evolD simp*: *ffb-eq*)

lemma *dWeakening*:
assumes $\{s.\ G\ s\} \leq Q$
shows $P \leq fb_{\mathcal{F}}$ ($[x'{=}f]\,T$ & G) Q

```
  using assms by(auto intro: g-evolD simp: le-fun-def g-evol-def ffb-eq)
```

Differential Cut

```
lemma ffb-g-orbit-eq-univD:
  assumes fbℱ ([x´=f]T & G) {s. C s} = UNIV
    and ∀ r∈{0−−t}. x r ∈ ([x´=f]T & G) s
  shows ∀r∈{0−−t}. C (x r)
proof
  fix r assume r ∈ {0−−t}
  then have x r ∈ ([x´=f]T & G) s
    using assms(2) by blast
  also have ∀y. y ∈ ([x´=f]T & G) s ⟶ C y
    using assms(1) ffb-eq-univD by fastforce
  ultimately show C (x r) by blast
qed

lemma DC:
  assumes interval T and fbℱ ([x´=f]T & G) {s. C s} = UNIV
  shows fbℱ ([x´=f]T & G) Q = fbℱ ([x´=f]T & (λs. G s ∧ C s)) Q
proof(rule-tac f=λ x. fbℱ x Q in HOL.arg-cong, rule ext, rule subset-antisym)
  fix s
  {fix s′ assume s′ ∈ ([x´=f]T & G) s
    then obtain t::real and x where x-ivp: D x = (f ∘ x) on T x 0 = s
      and guard-x:G ⊳ x {0−−t} and s′ = x t and 0 ∈ T t ∈ T
      using g-evolD[of s′ f T G s] by (metis (full-types))
    from guard-x have ∀r∈{0−−t}.∀ τ∈{0−−r}. G (x τ)
      by (metis contra-subsetD ends-in-segment(1) subset-segment(1))
    also have ∀τ∈{0−−t}. τ ∈ T
      using interval.closed-segment-subset-domain[OF assms(1) ‹0 ∈ T› ‹t ∈ T›]
by blast
    ultimately have ∀τ∈{0−−t}. x τ ∈ ([x´=f]T & G) s
      using g-evolI[OF x-ivp ‹0 ∈ T›] by blast
    hence C ⊳ x {0−−t}
      using ffb-g-orbit-eq-univD assms(2) by blast
    hence s′ ∈ ([x´=f]T & (λs. G s ∧ C s)) s
      using g-evolI[OF x-ivp ‹0 ∈ T› ‹t ∈ T›] guard-x ‹s′ = x t› by fastforce}
  thus ([x´=f]T & G) s ⊆ ([x´=f]T & (λs. G s ∧ C s)) s
    by blast
next show ⋀s. ([x´=f]T & (λs. G s ∧ C s)) s ⊆ ([x´=f]T & G) s
    by (auto simp: g-evol-def)
qed

lemma dCut:
  assumes ffb-C:P ≤ fbℱ ([x´=f]{0..t} & G) {s. C s}
    and ffb-Q:P ≤ fbℱ ([x´=f]{0..t} & (λ s. G s ∧ C s)) Q
  shows P ≤ fbℱ ([x´=f]{0..t} & G) Q
proof(subst ffb-eq, subst g-evol-def, clarsimp)
  fix τ::real and x::real ⇒ ′a assume (x 0) ∈ P and 0 ≤ τ and τ ≤ t
```

and *x-solves*:$D\ x = (\lambda t.\ f\ (x\ t))$ *on* $\{0..t\}$ **and** *guard-x*:$(\forall\ r \in \{0{-}{-}\tau\}.\ G\ (x\ r))$
hence $\forall r{\in}\{0{-}{-}\tau\}.\forall \tau{\in}\{0{-}{-}r\}.\ G\ (x\ \tau)$
using *closed-segment-closed-segment-subset* **by** *blast*
hence $\forall r{\in}\{0{-}{-}\tau\}.\ x\ r \in ([x{'}{=}f]\{0..t\}\ \&\ G)\ (x\ 0)$
using *g-evolI x-solves* $\langle 0 \le \tau\rangle$ $\langle \tau \le t\rangle$ *closed-segment-eq-real-ivl* **by** *fastforce*
hence $\forall r{\in}\{0{-}{-}\tau\}.\ C\ (x\ r)$
using *ffb-C* $\langle (x\ 0) \in P\rangle$ **by**(*subst* (*asm*) *ffb-eq*, *auto*)
hence $x\ \tau \in ([x{'}{=}f]\{0..t\}\ \&\ (\lambda\ s.\ G\ s \wedge C\ s))\ (x\ 0)$
using *g-evolI x-solves guard-x* $\langle 0 \le \tau\rangle$ $\langle \tau \le t\rangle$ **by** *fastforce*
from *this* $\langle (x\ 0) \in P\rangle$ **and** *ffb-Q* **show** $(x\ \tau) \in Q$
by(*subst* (*asm*) *ffb-eq*, *auto simp*: *closed-segment-eq-real-ivl*)
qed

Differential Invariant

lemma *DI-sufficiency*:
assumes $\forall s.\ \exists x.\ x \in$ *ivp-sols* $f\ T\ 0\ s$
shows $fb_{\mathcal{F}}\ ([x{'}{=}f]T\ \&\ G)\ Q \le fb_{\mathcal{F}}\ (\lambda\ x.\ \{s.\ s = x \wedge G\ s\})\ Q$
using *assms* **apply**(*subst ffb-eq*, *subst ffb-eq*, *clarsimp*)
apply(*rename-tac s*, *erule-tac x=s* **in** *allE*, *erule impE*)
apply(*simp add*: *g-evol-def ivp-sols-def*)
apply(*erule-tac x=s* **in** *allE*, *clarify*)
by(*rule-tac x=0* **in** *exI*, *rule-tac x=x* **in** *exI*, *auto*)

lemma (**in** *local-flow*) *DI-necessity*:
shows $fb_{\mathcal{F}}\ (\lambda\ x.\ \{s.\ s = x \wedge G\ s\})\ Q \le fb_{\mathcal{F}}\ ([x{'}{=}f]T\ \&\ G)\ Q$
unfolding *ffb-g-orbit* **apply**(*subst ffb-eq*, *clarsimp*, *safe*)
apply(*erule-tac x=0* **in** *ballE*)
apply(*simp add*: *ivp*, *simp*)
oops

definition *diff-invariant* :: $'a$ *pred* $\Rightarrow$ $(('a$::*real-normed-vector*$) \Rightarrow 'a) \Rightarrow$ *real set* $\Rightarrow$ *bool*
((-)/ *is'-diff'-invariant'-of* (-)/ *along* (-) [*70*,*65*]*61*)
where I *is-diff-invariant-of* f *along* $T \equiv$
$(\forall s.\ I\ s \longrightarrow (\forall\ x.\ x \in$ *ivp-sols* $f\ T\ 0\ s \longrightarrow (\forall\ t \in T.\ I\ (x\ t))))$

lemma *invariant-to-set*:
shows (I *is-diff-invariant-of* f *along* T) $\longleftrightarrow$ $(\forall s.\ I\ s \longrightarrow$ (*g-orbital* $f\ T\ 0\ (\lambda s.$ *True*$)\ s) \subseteq \{s.\ I\ s\})$
unfolding *diff-invariant-def ivp-sols-def g-orbital-eq* **apply** *safe*
apply(*erule-tac x=xa 0* **in** *allE*)
apply(*drule mp*, *simp-all*)
apply(*erule-tac x=xa 0* **in** *allE*)
apply(*drule mp*, *simp-all add*: *subset-eq*)
apply(*erule-tac x=xa t* **in** *allE*)
by(*drule mp*, *auto*)

context *local-flow*
begin

lemma *diff-invariant-eq-invariant-set*:
 $(I$ *is-diff-invariant-of f along* $T) = (\forall s.\ \forall t{\in}T.\ I\ s \longrightarrow I\ (\varphi\ t\ s))$
 by(*subst invariant-to-set*, *auto simp*: *g-evol-collapses*)

lemma *invariant-set-eq-dl-invariant*:
 shows $(\forall s.\ \forall t{\in}T.\ I\ s \longrightarrow I\ (\varphi\ t\ s)) = (\{s.\ I\ s\} = fb_{\mathcal{F}}\ (orbit)\ \{s.\ I\ s\})$
 apply(*safe*, *simp-all add*: *ffb-orbit*)
  apply(*erule-tac x=0* **in** *ballE*)
 by(*auto simp*: *ivp*(*2*) *init-time*)

end

lemma *dInvariant*:
 assumes *I is-diff-invariant-of f along T*
 shows $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ ([x'{=}f]T\ \&\ G)\ \{s.\ I\ s\}$
 using *assms* **by**(*auto simp*: *diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *dInvariant-converse*:
 assumes $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ ([x'{=}f]T\ \&\ (\lambda s.\ True))\ \{s.\ I\ s\}$
 shows *I is-diff-invariant-of f along T*
 using *assms* **unfolding** *invariant-to-set ffb-eq* **by** *auto*

lemma *ffb-g-evol-le-requires*:
 assumes $\forall s.\ \exists x.\ x \in (ivp\text{-}sols\ f\ T\ 0\ s) \wedge G\ s$
  shows $fb_{\mathcal{F}}\ ([x'{=}f]T\ \&\ G)\ \{s.\ I\ s\} \leq \{s.\ I\ s\}$
 apply(*simp add*: *ffb-eq g-orbital-eq*, *clarify*)
 apply(*erule-tac x=x* **in** *allE*, *erule impE*, *simp-all*)
 using *assms ivp-solsD*(*1*) **by**(*fastforce simp*: *ivp-sols-def*)

lemma *dI*:
assumes *I is-diff-invariant-of f along* $\{0..t\}$
  and $P \leq \{s.\ I\ s\}$ **and** $\{s.\ I\ s\} \leq Q$
 shows $P \leq fb_{\mathcal{F}}\ ([x'{=}f]\{0..t\}\ \&\ G)\ Q$
 apply(*rule-tac C=I* **in** *dCut*)
 using *dInvariant assms* **apply** *blast*
 apply(*rule dWeakening*)
 using *assms* **by** *auto*

Finally, we obtain some conditions to prove specific instances of differential invariants.

named-theorems *ode-invariant-rules compilation of rules for differential invariants*.

lemma [*ode-invariant-rules*]:
fixes $\vartheta{::}'a{::}banach \Rightarrow real$
assumes $\forall\ x.\ (D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}) \longrightarrow (\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0{-}{-}\tau\}.$

$((\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)\ )$ *has-derivative* $(\lambda\tau.\ \ \tau *_R 0))$ (*at r within* $\{0--\tau\}))$
shows $(\lambda s.\ \vartheta\ s = \nu\ s)$ *is-diff-invariant-of f along* $\{0..t\}$
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
  fix $x\ \tau$ **assume** *tHyp*:$0 \leq \tau\ \tau \leq t$
    and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))$ *on* $\{0..t\}\ \vartheta\ (x\ 0) = \nu\ (x\ 0)$
  hence $\forall\ r \in \{0--\tau\}.\ D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R 0)$ *at r within* $\{0--\tau\}$
    using *assms* **by** *auto*
  hence $\exists\, r\in\{0--\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) = (\lambda\tau.\ \tau *_R 0)\ (\tau - 0)$
    by(*rule-tac closed-segment-mvt*, *auto simp*: *tHyp*)
  thus $\vartheta\ (x\ \tau) = \nu\ (x\ \tau)$ **by** (*simp add*: *x-ivp*(2))
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (D\ x = (\lambda\tau.\ f\ (x\ \tau))$ *on* $\{0..t\}) \longrightarrow (\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0--\tau\}.\ \vartheta'\ (x\ r) \geq \nu'\ (x\ r)$
$\wedge\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) - \nu'\ (x\ r)))$ *at r within* $\{0--\tau\}))$
shows $(\lambda s.\ \nu\ s \leq \vartheta\ s)$ *is-diff-invariant-of f along* $\{0..t\}$
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
  fix $x\ \tau$ **assume** *tHyp*:$0 \leq \tau\ \tau \leq t$
    and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))$ *on* $\{0..t\}\ \nu\ (x\ 0) \leq \vartheta\ (x\ 0)$
  hence *primed*:$\forall\ r \in \{0--\tau\}.\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) - \nu'\ (x\ r)))$
  at r within $\{0--\tau\}) \wedge \nu'\ (x\ r) \leq \vartheta'\ (x\ r)$
    using *assms* **by** *auto*
  hence $\exists\, r\in\{0--\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) =$
  $(\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) - \ \nu'\ (x\ r)))\ (\tau - 0)$
    by(*rule-tac closed-segment-mvt*, *auto simp*: $\langle 0 \leq \tau\rangle$)
  then obtain r **where** $r \in \{0--\tau\}$
    and $\vartheta\ (x\ \tau) - \nu\ (x\ \tau) = (\tau - 0) *_R (\vartheta'\ (x\ r) - \ \nu'\ (x\ r)) + (\vartheta\ (x\ 0) - \nu\ (x\ 0))$
    by *force*
  also have ... ≥ 0
    using *tHyp*(1) *x-ivp*(2) *primed* **by** (*simp add*: *calculation*(1))
  ultimately show $\nu\ (x\ \tau) \leq \vartheta\ (x\ \tau)$
    by *simp*
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (D\ x = (\lambda\tau.\ f\ (x\ \tau))$ *on* $\{0..t\}) \longrightarrow (\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0--\tau\}.\ \vartheta'\ (x\ r) \geq \nu'\ (x\ r)$
$\wedge\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) - \nu'\ (x\ r)))$ *at r within* $\{0--\tau\}))$
shows $(\lambda s.\ \nu\ s < \vartheta\ s)$ *is-diff-invariant-of f along* $\{0..t\}$

proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
fix x τ **assume** *tHyp*:$0 \leq \tau$ $\tau \leq t$
and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))$ *on* $\{0..t\}$ $\nu\ (x\ 0) < \vartheta\ (x\ 0)$
hence *primed*:$\forall\ r \in \{0{-}{-}\tau\}.\ ((\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau))$ *has-derivative*
$(\lambda\tau.\ \tau *_R\ (\vartheta'\ (x\ r) -\ \nu'\ (x\ r))))$ $($*at* r *within* $\{0{-}{-}\tau\}) \wedge \vartheta'\ (x\ r) \geq \nu'\ (x\ r)$
using *assms* **by** *auto*
hence $\exists\, r\in\{0{-}{-}\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) =$
$(\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) -\ \nu'\ (x\ r)))\ (\tau - 0)$
by(*rule-tac closed-segment-mvt*, *auto simp*: ‹$0 \leq \tau$›)
then obtain r **where** $r \in \{0{-}{-}\tau\}$ **and**
$\vartheta\ (x\ \tau) - \nu\ (x\ \tau) = (\tau - 0) *_R (\vartheta'\ (x\ r) -\ \nu'\ (x\ r)) + (\vartheta\ (x\ 0) - \nu\ (x\ 0))$
by *force*
also have ... > 0
using *tHyp*(*1*) *x-ivp*(*2*) *primed* **by** (*metis* (*no-types*,*hide-lams*) *Groups.add-ac*(*2*)
add-sign-intros(*1*)
calculation(*1*) *diff-gt-0-iff-gt ge-iff-diff-ge-0 less-eq-real-def zero-le-scaleR-iff*)

ultimately show $\nu\ (x\ \tau) < \vartheta\ (x\ \tau)$
by *simp*
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes *I1 is-diff-invariant-of f along* $\{0..t\}$
and *I2 is-diff-invariant-of f along* $\{0..t\}$
shows (λs. *I1 s* $\wedge$ *I2 s*) *is-diff-invariant-of f along* $\{0..t\}$
using *assms* **unfolding** *diff-invariant-def* **by** *auto*

lemma [*ode-invariant-rules*]:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes *I1 is-diff-invariant-of f along* $\{0..t\}$
and *I2 is-diff-invariant-of f along* $\{0..t\}$
shows (λs. *I1 s* $\vee$ *I2 s*) *is-diff-invariant-of f along* $\{0..t\}$
using *assms* **unfolding** *diff-invariant-def* **by** *auto*

end
theory *cat2funcset-examples*
imports ../*hs-prelims-matrices cat2funcset*

begin

3.2.3 Examples

The examples in this subsection show different approaches for the verification of hybrid systems. However, the general approach can be outlined as follows: First, we select a finite type to model program variables $'n$. We use this to define a vector field f of type $('a,\ 'n)$ *vec* $\Rightarrow$ $('a,\ 'n)$ *vec* to model the dynamics of our system. Then we show a partial correctness specification involving the evolution command $[x' = f]\, T \ \&\ G$ either by finding a flow for the vector field or through differential invariants.

Single constantly accelerated evolution

The main characteristics distinguishing this example from the rest are:

1. We define the finite type of program variables with 2 Isabelle strings which make the final verification easier to parse.
2. We define the vector field (named K) to model a constantly accelerated object.
3. We define a local flow (φ_K) and use it to compute the wlp for this vector field.
4. The verification is only done on a single evolution command (not operated with any other hybrid program).

```
typedef program-vars ={''y'',''v''}
  morphisms to-str to-var
  apply(rule-tac x=''y'' in exI)
  by simp

notation to-var (↾V)

lemma number-of-program-vars:CARD(program-vars) = 2
  using type-definition.card type-definition-program-vars by fastforce

instance program-vars::finite
  apply(standard, subst bij-betw-finite[of to-str UNIV {''y'',''v''}])
   apply(rule bij-betwI')
     apply (simp add: to-str-inject)
  using to-str apply blast
   apply (metis to-var-inverse UNIV-I)
  by simp

lemma program-vars-univD:(UNIV::program-vars set) = {↾V ''y'', ↾V ''v''}
  apply auto by (metis to-str to-str-inverse insertE singletonD)

lemma program-vars-exhaust:∀ x::program-vars. x = ↾V ''y'' ∨ x = ↾V ''v''
  using program-vars-univD by auto

abbreviation constant-acceleration-kinematics g s ≡
  (χ i. if i=(↾V ''y'') then s $ (↾V ''v'') else g)

notation constant-acceleration-kinematics (K)
```

lemma *cnst-acc-continuous*:
fixes X::(*real^program-vars*) *set*
shows *continuous-on* X (K g)
apply(*rule continuous-on-vec-lambda*)
unfolding *continuous-on-def* **apply** *clarsimp*
by(*intro tendsto-intros*)

lemma *picard-lindeloef-cnst-acc*:
fixes g::*real* **assumes** $0 \le t$ **and** $t < 1$
shows *picard-lindeloef* ($\lambda t.\ K\ g$) $\{0..t\}$ 1 0
unfolding *picard-lindeloef-def* **apply**(*simp add*: *lipschitz-on-def assms*, *safe*)
apply(*rule-tac t=UNIV* **and** *f=snd* **in** *continuous-on-compose2*)
apply(*simp-all add*: *cnst-acc-continuous continuous-on-snd*)
apply(*simp add*: *dist-vec-def L2-set-def dist-real-def*)
apply(*subst program-vars-univD*, *subst program-vars-univD*)
apply(*simp-all add*: *to-var-inject*)
using *assms* **by** *linarith*

abbreviation *constant-acceleration-kinematics-flow* $g\ t\ s \equiv$
(χ i. *if* i=($\upharpoonleft_V$ $''y''$) *then* $g \cdot t$ ^ $2/2$ + s \$ ($\upharpoonleft_V$ $''v''$) $\cdot$ t + s \$ ($\upharpoonleft_V$ $''y''$)
else $g \cdot t$ + s \$ ($\upharpoonleft_V$ $''v''$))

notation *constant-acceleration-kinematics-flow* (φ_K)

lemma *local-flow-cnst-acc*:
assumes $0 \le t$ **and** $t < 1$
shows *local-flow* (K g) $\{0..t\}$ 1 (φ_K g)
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *assms picard-lindeloef-cnst-acc* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*)
using *poly-derivatives*(*3*,*4*) *program-vars-exhaust*
apply(*simp-all add*: *to-var-inject vec-eq-iff has-vderiv-on-def has-vector-derivative-def*)
using *program-vars-exhaust* **by** *blast*

lemma *ffb-cnst-acc*:
assumes $0 \le t$ **and** $t < 1$
shows $fb_{\mathcal{F}}$ ([x'=K g]$\{0..t\}$ & G) Q = $\{s.\ \forall \tau \in \{0..t\}.\ (G \rhd (\lambda r.\ \varphi_K\ g\ r$
$s)\{0{-}{-}\tau\}) \longrightarrow (\varphi_K\ g\ \tau\ s) \in Q\}$
apply(*subst local-flow.ffb-g-orbit*[*of K g - 1* (λ t x. φ_K g t x)])
using *local-flow-cnst-acc* **and** *assms* **by** *auto*

lemma *single-evolution-ball*:
fixes H::*real* **assumes** $0 \le t$ **and** $t < 1$ **and** $g < 0$
shows $\{s.\ 0 \le s$ \$ ($\upharpoonleft_V$ $''y''$) $\wedge$ s \$ ($\upharpoonleft_V$ $''y''$) = H $\wedge$ s \$ ($\upharpoonleft_V$ $''v''$) = $0\}$
$\le fb_{\mathcal{F}}$ ([x'=K g]$\{0..t\}$ & (λ s. s \$ ($\upharpoonleft_V$ $''y''$) ≥ 0))
$\{s.\ 0 \le s$ \$ ($\upharpoonleft_V$ $''y''$) $\wedge$ s \$ ($\upharpoonleft_V$ $''y''$) $\le H\}$
apply(*subst ffb-cnst-acc*)
using *assms* **by**(*auto simp*: *mult-nonpos-nonneg*)

no-notation *to-var* ($\restriction_V$)

no-notation *constant-acceleration-kinematics* (K)

no-notation *constant-acceleration-kinematics-flow* (φ_K)

Single evolution revisited.

We list again the characteristics that distinguish this example:

1. We employ an existing finite type of size 3 to model program variables.
2. We define a 3×3 matrix (named K) to denote the linear operator that models the constantly accelerated motion.
3. We define a local flow (φ_K) and use it to compute the wlp for this linear operator.
4. The verification is done equivalently to the above example.

term *x*::*2* — It turns out that there is already a 2-element type:

lemma *CARD*(*program-vars*) = *CARD*(*2*)
  unfolding *number-of-program-vars* **by** *simp*

In fact, for each natural number n there is already a corresponding n-element type in Isabelle. However, there are still lemmas to prove about them in order to do verification of hybrid systems in n-dimensional Euclidean spaces.

lemma *exhaust-5*: — The analogs for $1, 2$ and 3 have already been proven in Analysis.
  fixes *x*::*5*
  shows *x*=*1* ∨ *x*=*2* ∨ *x*=*3* ∨ *x*=*4* ∨ *x*=*5*
proof (*induct x*)
  case (*of-int z*)
  then have *0* ≤ *z* **and** *z* < *5* **by** *simp-all*
  then have *z* = *0* ∨ *z* = *1* ∨ *z* = *2* ∨ *z* = *3* ∨ *z* = *4* **by** *arith*
  then show *?case* **by** *auto*
qed

lemma *UNIV-3*:(*UNIV*::*3 set*) = {*0*, *1*, *2*}
  apply *safe* **using** *exhaust-3 three-eq-zero* **by**(*blast*, *auto*)

lemma *sum-axis-UNIV-3*[*simp*]:($\sum$*j*∈(*UNIV*::*3 set*). *axis i 1* \$ *j* · *f j*) = (*f*::*3* ⇒ *real*) *i*
  unfolding *axis-def UNIV-3* **apply** *simp*
  using *exhaust-3* **by** *force*

We can rewrite the original constant acceleration kinematics as a linear operator applied to a 3-dimensional vector. For that we take advantage of the following fact:

```
lemma e 1 = (χ j::3. if j= 0 then 0 else if j = 1 then 1 else 0)
  unfolding axis-def by(rule Cart-lambda-cong, simp)

abbreviation constant-acceleration-kinematics-matrix ≡
  (χ i. if i= (0::3) then axis (1::3) (1::real) else if i= 1 then axis 2 1 else 0)

abbreviation constant-acceleration-kinematics-matrix-flow t s ≡
  (χ i. if i= (0::3) then s $ 2 · t ^ 2/2 + s $ 1 · t + s $ 0
   else if i=1 then s $ 2 · t + s $ 1 else s $ 2)

notation constant-acceleration-kinematics-matrix (K)

notation constant-acceleration-kinematics-matrix-flow (φK)
```

With these 2 definitions and the proof that linear systems of ODEs are Picard-Lindeloef, we can show that they form a pair of vector-field and its flow.

```
lemma entries-cnst-acc-matrix: entries K = {0, 1}
  apply (simp-all add: axis-def, safe)
  by(rule-tac x=1 in exI, simp)+

lemma picard-lindeloef-cnst-acc-matrix:
  assumes 0 ≤ t and t < 1/9
  shows picard-lindeloef (λ t s. K *v s) {0..t} ((real CARD(3))² · (‖K‖max)) 0
  apply(rule picard-lindeloef-linear-system)
  unfolding entries-cnst-acc-matrix using assms by auto

lemma local-flow-cnst-acc-matrix:
  assumes 0 ≤ t and t < 1/9
  shows local-flow ((*v) K) {0..t} ((real CARD(3))² · (‖K‖max)) φK
  unfolding local-flow-def local-flow-axioms-def apply safe
  using picard-lindeloef-cnst-acc-matrix[OF assms] apply blast
   apply(rule has-vderiv-on-vec-lambda)
  using poly-derivatives(1,3, 4)
   apply(force simp: matrix-vector-mult-def)
  using exhaust-3 by(force simp: matrix-vector-mult-def vec-eq-iff)
```

Finally, we compute the wlp and use it to verify the single-evolution ball again.

```
lemma ffb-cnst-acc-mtx:
  assumes 0 ≤ t and t < 1/9
   shows fbF ([x´=(*v) K]{0..t} & G) Q = {s. ∀τ∈{0..t}. (G ▷ (λr. φK r
s){0−−τ}) ⟶ (φK τ s) ∈ Q}
   apply(subst local-flow.ffb-g-orbit[of (*v) K - ((real CARD(3))² · (‖K‖max))
φK])
  using local-flow-cnst-acc-matrix and assms by auto

lemma single-evolution-ball-matrix:
  assumes 0 ≤ t and t < 1/9
```

shows $\{s.\ 0 \leq s \$ 0 \wedge s \$ 0 = H \wedge s \$ 1 = 0 \wedge 0 > s \$ 2\}$
$\leq fb_{\mathcal{F}}$ $([x'{=}(*v)\ K]\{0..t\}\ \&\ (\lambda\ s.\ s \$ 0 \geq 0))$
$\{s.\ 0 \leq s \$ 0 \wedge s \$ 0 \leq H\}$
apply(*subst ffb-cnst-acc-mtx*)
using *assms* **by**(*auto simp*: *mult-nonneg-nonpos2*)

Circular Motion

The characteristics that distinguish this example are:

1. We employ an existing finite type of size 2 to model program variables.
2. We define a 2×2 matrix (named C) to denote the linear operator that models circular motion.
3. We show that the circle equation is a differential invariant for the linear operator.
4. We prove the partial correctness specification corresponding to the previous point.
5. For completeness, we define a local flow (φ_C) and use it to compute the wlp for the linear operator and the specification is proven again with this flow.

lemma *two-eq-zero*: $(2{::}2) = 0$
by *simp*

lemma [*simp*]:$i \neq (0{::}2) \longrightarrow i = 1$
using *exhaust-2* **by** *fastforce*

lemma *UNIV-2*:$(UNIV{::}2\ set) = \{0,\ 1\}$
apply *safe* **using** *exhaust-2 two-eq-zero* **by** *auto*

abbreviation *circular-motion-matrix* $\equiv$
$(\chi\ i.\ if\ i{=}\ (0{::}2)\ then\ axis\ (1{::}2)\ (-\ 1{::}real)\ else\ axis\ 0\ 1)$

notation *circular-motion-matrix* (C)

lemma *circle-invariant*:
assumes $0 < R$
shows $(\lambda s.\ R^2 = (s \$ 0)^2 + (s \$ 1)^2)$ *is-diff-invariant-of* $(*v)$ C *along* $\{0..t\}$
apply(*rule-tac ode-invariant-rules*, *clarsimp*)
apply(*frule-tac i=0* **in** *has-vderiv-on-vec-nth*, *drule-tac i=1* **in** *has-vderiv-on-vec-nth*)
apply(*unfold has-vderiv-on-def has-vector-derivative-def*, *clarsimp*)
apply(*erule-tac x=r* **in** *ballE*)+
apply(*simp add*: *matrix-vector-mult-def has-vderiv-on-vec-lambda*)
subgoal for $x\ \tau\ r$ **apply**(*rule-tac* $f'1{=}\lambda t.\ 0$ **and** $g'1{=}\lambda t.\ 0$ **in** *derivative-eq-intros*(*11*), *simp-all*)

apply(*rule-tac* $f'1=\lambda t. - 2 \cdot (x\ r\ \$\ 0) \cdot (t \cdot x\ r\ \$\ 1)$
and $g'1=\lambda t.\ 2 \cdot (x\ r\ \$\ 1) \cdot t \cdot x\ r\ \$\ 0$ **in** *derivative-eq-intros*(*8*), *simp-all*)
apply(*rule-tac* $f'1=\lambda t. - (t \cdot x\ r\ \$\ 1)$ **in** *derivative-eq-intros*(*15*))
apply(*rule-tac* $t=\{0--\tau\}$ **and** $s=\{0..t\}$ **in** *has-derivative-within-subset*)
apply(*simp*, *simp add*: *closed-segment-eq-real-ivl*, *force*)
apply(*rule-tac* $f'1=\lambda t.\ (t \cdot x\ r\ \$\ 0)$ **in** *derivative-eq-intros*(*15*))
apply(*rule-tac* $t=\{0--\tau\}$ **and** $s=\{0..t\}$ **in** *has-derivative-within-subset*)
by(*simp*, *simp add*: *closed-segment-eq-real-ivl*, *force*)
by(*auto simp*: *closed-segment-eq-real-ivl*)

lemma *circular-motion-invariants*:
assumes $(R::real) > 0$
shows$\{s.\ R^2 = (s\ \$\ (0::2))^2 + (s\ \$\ 1)^2\}$
$\leq fb_{\mathcal{F}}\ ([x\acute{}=(*v)\ C]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0))$
$\{s.\ R^2 = (s\ \$\ (0::2))^2 + (s\ \$\ 1)^2\}$
using *assms* **apply**(*rule-tac* $C=\lambda s.\ R^2 = (s\ \$\ (0::2))^2 + (s\ \$\ 1)^2$ **in** *dCut*)
apply(*rule-tac* $I=\lambda s.\ R^2 = (s\ \$\ (0::2))^2 + (s\ \$\ 1)^2$ **in** *dInvariant*)
using *circle-invariant* **apply** *blast*
by(*rule dWeakening*, *auto*)

— Proof of the same specification by providing solutions:

lemma *entries-circ-mtx*:*entries* $C = \{0, -1, 1\}$
apply (*simp-all add*: *axis-def*, *safe*)
subgoal by(*rule-tac* $x=0$ **in** *exI*, *simp*)+
subgoal by(*rule-tac* $x=0$ **in** *exI*, *simp*)+
by(*rule-tac* $x=1$ **in** *exI*, *simp*)+

lemma *picard-lindeloef-circ-mtx*:
assumes $0 \leq t$ **and** $t < 1/4$
shows *picard-lindeloef* $(\lambda t.\ (*v)\ C)\ \{0..t\}\ ((real\ CARD(2))^2 \cdot (\|C\|_{max}))\ 0$
apply(*rule picard-lindeloef-linear-system*)
unfolding *entries-circ-mtx* **using** *assms* **by** *auto*

abbreviation *circular-motion-matrix-flow* $t\ s \equiv (\chi\ i.\ if\ i= (0::2)\ then$
$s\$0 \cdot cos\ t - s\$1 \cdot sin\ t\ else\ s\$0 \cdot sin\ t + s\$1 \cdot cos\ t)$

notation *circular-motion-matrix-flow* (φ_C)

lemma *local-flow-circ-mtx*:
assumes $0 \leq t$ **and** $t < 1/4$
shows *local-flow* $((*v)\ C)\ \{0..t\}\ ((real\ CARD(2))^2 \cdot (\|C\|_{max}))\ \varphi_C$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-circ-mtx assms* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*)
apply(*simp add*: *matrix-vector-mult-def has-vderiv-on-def has-vector-derivative-def*, *safe*)
subgoal for $s\ i\ x$
apply(*rule-tac* $f'1=\lambda t. - s\$0 \cdot (t \cdot sin\ x)$ **and** $g'1=\lambda t.\ s\$1 \cdot (t \cdot cos\ x)$**in**

derivative-eq-intros(*11*))
apply(*rule derivative-eq-intros*(*6*)[*of cos* ($\lambda xa.\ -\ (xa \cdot sin\ x)$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*58*))
apply(*rule ssubst*[*of* (·) *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of sin* ($\lambda xa.\ (xa \cdot cos\ x)$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* (·) *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
subgoal for *s i x*
apply(*rule-tac* $f'1=\lambda t.\ s\$0 \cdot (t \cdot cos\ x)$ **and** $g'1=\lambda t.\ -\ s\$1 \cdot (t \cdot sin\ x)$**in** *derivative-eq-intros*(*8*))
apply(*rule derivative-eq-intros*(*6*)[*of sin* ($\lambda xa.\ xa \cdot cos\ x$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* (·) *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of cos* ($\lambda xa.\ -\ (xa \cdot sin\ x)$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*58*))
apply(*rule ssubst*[*of* (·) *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
using *exhaust-2 two-eq-zero* **by**(*force simp*: *vec-eq-iff*)

lemma *ffb-circ-mtx*:
assumes $0 \leq t$ **and** $t < 1/4$
shows $fb_{\mathcal{F}}$ ([$x' = \lambda s.\ C *v\ s$]{$0..t$} & G) Q =
{$x.\ \forall\ \tau \in \{0..t\}.\ (\forall r \in \{0--\tau\}.\ G\ (\varphi_C\ r\ x)) \longrightarrow (\varphi_C\ \tau\ x) \in Q$}
apply(*subst local-flow.ffb-g-orbit*[*of* $\lambda s.\ C *v\ s$ - ((*real CARD*(*2*))$^2 \cdot (\|C\|_{max})$) ($\lambda\ t\ x.\ \varphi_C\ t\ x$)])
using *local-flow-circ-mtx* **and** *assms* **by** *auto*

lemma *circular-motion*:
assumes $0 \leq t$ **and** $t < 1/4$ **and** (R::*real*) > 0
shows $\{s.\ R^2 = (s\ \$\ (0::2))^2 + (s\ \$\ 1)^2\} \leq fb_{\mathcal{F}}$
([$x' = \lambda s.\ C *v\ s$]{$0..t$} & ($\lambda\ s.\ s\ \$\ 0 \geq 0$))
$\{s.\ R^2 = (s\ \$\ (0::2))^2 + (s\ \$\ 1)^2\}$
apply(*subst ffb-circ-mtx*)
using *assms* **by** *auto*

no-notation *circular-motion-matrix* (C)

no-notation *circular-motion-matrix-flow* (φ_C)

Bouncing Ball with solution

We revisit the previous dynamics for a constantly accelerated object modelled with the matrix K. We compose the corresponding evolution command with an if-statement, and iterate this hybrid program to model a (completely elastic) "bouncing ball". Using the previously defined flow for this dynamics, proving a specification for this hybrid program is merely an exercise of real arithmetic.

named-theorems *bb-real-arith real arithmetic properties for the bouncing ball.*

lemma [*bb-real-arith*]:$0 \leq x \Longrightarrow 0 > g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v \Longrightarrow (x::real) \leq H$
proof−
  assume $0 \leq x$ **and** $0 > g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
  then have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot H \wedge 0 > g$ **by** *auto*
  hence $*{:}v \cdot v = 2 \cdot g \cdot (x - H) \wedge 0 > g \wedge v \cdot v \geq 0$
    using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
  from *this* **have** $(v \cdot v)/(2 \cdot g) = (x - H)$ **by** *auto*
  also from $*$ **have** $(v \cdot v)/(2 \cdot g) \leq 0$
    using *divide-nonneg-neg* **by** *fastforce*
  ultimately have $H - x \geq 0$ **by** *linarith*
  thus *?thesis* **by** *auto*
qed

lemma [*bb-real-arith*]:
  assumes *invar*:$2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
    and *pos*:$g \cdot \tau^2 \ / \ 2 + v \cdot \tau + (x::real) = 0$
  shows $2 \cdot g \cdot H + (-\ (g \cdot \tau) - v) \cdot (-\ (g \cdot \tau) - v) = 0$
    and $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
proof−
  from *pos* **have** $g \cdot \tau^2 \ + 2 \cdot v \cdot \tau + 2 \cdot x = 0$ **by** *auto*
  then have $g^2 \ \cdot \tau^2 \ + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$
    by (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(*1*,*3*) *mult-zero-right*
      monoid-mult-class.power2-eq-square semiring-class.distrib-left)
  hence $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot H = 0$
    using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
  from *this* **have** $*{:}(g \cdot \tau + v)^2 + 2 \cdot g \cdot H = 0$
   apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

      Groups.mult-ac(*2*, *3*) *monoid-mult-class.power2-eq-square nat-distrib*(*2*))
  hence $2 \cdot g \cdot H + (-\ ((g \cdot \tau) + v))^2 = 0$
    by (*metis Groups.add-ac*(*2*) *power2-minus*)
  thus $2 \cdot g \cdot H + (-\ (g \cdot \tau) - v) \cdot (-\ (g \cdot \tau) - v) = 0$
    by (*simp add*: *monoid-mult-class.power2-eq-square*)
  from $*$ **show** $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
    by (*simp add*: *monoid-mult-class.power2-eq-square*)
qed

lemma [*bb-real-arith*]:
  assumes *invar*:$2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
  shows $2 \cdot g \cdot (g \cdot \tau^2 \ / \ 2 + v \cdot \tau + (x::real)) =$
  $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**is** *?lhs* = *?rhs*)
proof−
  have $?lhs = g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
      apply(*subst Rat.sign-simps*(*18*))+
      by(*auto simp*: *semiring-normalization-rules*(*29*))
    also have $\ldots = g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot H + v \cdot v$ (**is** ... = *?middle*)

by(*subst invar*, *simp*)
finally have *?lhs* = *?middle*.
moreover
{**have** *?rhs* = $g \cdot g \cdot (\tau \cdot \tau) + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot H + v \cdot v$
by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
also have ... = *?middle*
by (*simp add*: *semiring-normalization-rules*(*29*))
finally have *?rhs* = *?middle*.}
ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball*:
assumes $0 \leq t$ **and** $t < 1/9$
shows $\{s.\ (0::real) \leq s\ \$\ (0::3) \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\} \leq fb_{\mathcal{F}}$
$(kstar\ (([x' {=} \lambda s.\ K * v\ s]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)) \circ_K$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s. - s\ \$\ 1)])\ ELSE\ \eta\ FI)))$
$\{s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H\}$
apply(*rule ffb-starI*[*of* - $\{s.\ 0 \leq s\ \$\ (0::3) \wedge 0 > s\ \$\ 2\ \wedge$
$2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 = 2 \cdot s\ \$\ 2 \cdot H + (s\ \$\ 1 \cdot s\ \$\ 1)\}$])
apply(*clarsimp*, *simp only*: *ffb-kcomp*)
apply(*subst ffb-cnst-acc-mtx*)
using *assms* **apply**(*simp*, *simp*, *clarsimp*)
apply(*rule ffb-if-then-elseD*)
by(*auto simp*: *bb-real-arith*)

Bouncing Ball with invariants

We prove again the bouncing ball but this time with differential invariants.

lemma *gravity-invariant*: $(\lambda s.\ s\ \$\ 2 < 0)$ *is-diff-invariant-of* $(*v)$ *K along* $\{0..t\}$
apply(*rule-tac* $\vartheta' {=} \lambda s.\ 0$ **and** $\nu' {=} \lambda s.\ 0$ **in** *ode-invariant-rules*(*3*), *clarsimp*)
apply(*drule-tac* $i{=}2$ **in** *has-vderiv-on-vec-nth*)
apply(*unfold has-vderiv-on-def has-vector-derivative-def*)
apply(*erule-tac* $x{=}r$ **in** *ballE*, *simp add*: *matrix-vector-mult-def*)
apply(*rule-tac* $f'1 {=} \lambda s.\ 0$ **in** *derivative-eq-intros*(*10*))
by(*auto simp*: *closed-segment-eq-real-ivl has-derivative-within-subset*)

lemma *energy-conservation-invariant*:
$(\lambda s.\ 2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 - 2 \cdot s\ \$\ 2 \cdot H - s\ \$\ 1 \cdot s\ \$\ 1 = 0)$ *is-diff-invariant-of*
$(*v)$ *K along* $\{0..t\}$
apply(*rule ode-invariant-rules*, *clarify*)
apply(*frule-tac* $i{=}2$ **in** *has-vderiv-on-vec-nth*)
apply(*frule-tac* $i{=}1$ **in** *has-vderiv-on-vec-nth*)
apply(*drule-tac* $i{=}0$ **in** *has-vderiv-on-vec-nth*)
apply(*unfold has-vderiv-on-def has-vector-derivative-def*)
apply(*erule-tac* $x{=}r$ **in** *ballE*, *simp-all add*: *matrix-vector-mult-def*)+
apply(*rule-tac* $f'1 {=} \lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$
and $g'1 {=} \lambda t.\ 2 \cdot (t \cdot (x\ r\ \$\ 1 \cdot x\ r\ \$\ 2))$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac* $f'1 {=} \lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$ **and** $g'1 {=} \lambda t.\ 0$ **in** *derivative-eq-intros*(*11*))

apply(*rule-tac* $f'1=\lambda t.\ 0$ **and** $g'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 1)$ **in** *derivative-eq-intros*(*12*))
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*))
apply(*simp-all add*: *has-derivative-within-subset closed-segment-eq-real-ivl*)
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*7*))
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*))
apply(*simp-all add*: *has-derivative-within-subset*)
apply(*rule-tac* $f'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **and** $g'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **in** *derivative-eq-intros*(*12*))
by(*simp-all add*: *has-derivative-within-subset*)

lemma *bouncing-ball-invariants*:
shows $\{s.\ (0::real) \leq s\ \$\ (0::3) \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\} \leq fb_{\mathcal{F}}$
$(kstar\ (([x' {=} \lambda s.\ K *v\ s]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)) \circ_K$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.\ -\ s\ \$\ 1)])\ ELSE\ \eta\ FI)))$
$\{s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H\}$
apply(*rule-tac* $I=\{s.\ 0 \leq s\$0 \wedge 0 > s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot H + (s\$1 \cdot s\$1)\}$ **in** *ffb-starI*)
apply(*clarsimp*, *simp only*: *ffb-kcomp*)
apply(*rule dCut*[**where** $C=\lambda\ s.\ s\ \$\ 2 < 0$])
apply(*rule-tac* $I=\lambda\ s.\ s\ \$\ 2 < 0$ **in** *dI*)
using *gravity-invariant* **apply**(*blast*, *force*, *force*)
apply(*rule-tac* $C=\lambda\ s.\ 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1 = 0$ **in** *dCut*)
apply(*rule-tac* $I=\lambda\ s.\ 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1 = 0$ **in** *dI*)
using *energy-conservation-invariant* **apply**(*blast*, *force*, *force*)
apply(*rule dWeakening*)
apply(*rule ffb-if-then-else*)
by(*auto simp*: *bb-real-arith le-fun-def*)

no-notation *constant-acceleration-kinematics-matrix* (K)

no-notation *constant-acceleration-kinematics-matrix-flow* (φ_K)

Bouncing Ball with exponential solution

In our final example, we prove again the bouncing ball specification but this time we do it with the general solution for linear systems.

abbreviation *constant-acceleration-kinematics-sq-mtx* $\equiv$ *sq-mtx-chi constant-acceleration-kinematics-ma*

notation *constant-acceleration-kinematics-sq-mtx* (K)

lemma *max-norm-cnst-acc-sq-mtx*: $\|to\text{-}vec\ K\|_{max} = 1$
proof–
have $\{to\text{-}vec\ K\ \$\ i\ \$\ j\ | i\ j.\ s2p\ UNIV\ i \wedge s2p\ UNIV\ j\} = \{0,\ 1\}$
apply (*simp-all add*: *axis-def*, *safe*)
by(*rule-tac* $x=1$ **in** *exI*, *simp*)+
thus *?thesis*
by *auto*
qed

lemma *ffb-cnst-acc-sq-mtx*:
assumes $0 \leq t$ **and** $t < 1/9$
shows $fb_{\mathcal{F}}$ $([x'=(*_V)\ K]\{0..t\}\ \&\ G)\ Q =$
$\{x.\ \forall\ \tau \in \{0..t\}.\ (\forall r\in\{0--\tau\}.\ G\ ((exp\ (r *_R K)) *_V x)) \longrightarrow ((exp\ (\tau *_R K)) *_V x) \in Q\}$
apply(*subst local-flow.ffb-g-orbit*[*of* $(*_V)$ K - $((real\ CARD(3))^2 \cdot (\|to\text{-}vec\ K\|_{max}))$

$(\lambda t\ x.\ (exp\ (t *_R K)) *_V x)$])
apply(*rule local-flow-exp*)
using *max-norm-cnst-acc-sq-mtx assms* **by** *auto*

lemma *exp-cnst-acc-sq-mtx-simps*:
$exp\ (\tau *_R K)\ \$\$\ 0\ \$\ 0 = 1$ $exp\ (\tau *_R K)\ \$\$\ 0\ \$\ 1 = \tau$ $exp\ (\tau *_R K)\ \$\$\ 0\ \$\ 2 = \tau\hat{\ }2/2$
$exp\ (\tau *_R K)\ \$\$\ 1\ \$\ 0 = 0$ $exp\ (\tau *_R K)\ \$\$\ 1\ \$\ 1 = 1$ $exp\ (\tau *_R K)\ \$\$\ 1\ \$\ 2 = \tau$
$exp\ (\tau *_R K)\ \$\$\ 2\ \$\ 0 = 0$ $exp\ (\tau *_R K)\ \$\$\ 2\ \$\ 1 = 0$ $exp\ (\tau *_R K)\ \$\$\ 2\ \$\ 2 = 1$
sorry

lemma *bouncing-ball-sq-mtx*:
assumes $0 \leq t$ **and** $t < 1/9$
shows $\{s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\} \leq fb_{\mathcal{F}}$
$(kstar\ (([x'=(*_V)\ K]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)) \circ_K$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s. - s\ \$\ 1)])\ ELSE\ \eta\ FI)))$
$\{s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H\}$
apply(*rule ffb-starI*[*of* - $\{s.\ 0 \leq s\ \$\ (0::3) \wedge 0 > s\ \$\ 2 \wedge$
$2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 = 2 \cdot s\ \$\ 2 \cdot H + (s\ \$\ 1 \cdot s\ \$\ 1)\}$])
apply(*clarsimp*, *simp only*: *ffb-kcomp*)
apply(*subst ffb-cnst-acc-sq-mtx*)
using *assms* **apply**(*simp*, *simp*, *clarify*)
apply(*rule ffb-if-then-elseD*, *clarsimp*)
apply(*simp-all add*: *sq-mtx-vec-prod-eq*)
unfolding *UNIV-3* **apply**(*simp-all add*: *exp-cnst-acc-sq-mtx-simps*)
subgoal for *x* **using** *bb-real-arith*(*3*)[*of x* $ *2*]
by (*simp add*: *add.commute mult.commute*)
subgoal for $x\ \tau$ **using** *bb-real-arith*(*4*)[**where** *g=x* $ *2* **and** *v=x* $ *1*]
by(*simp add*: *add.commute mult.commute*)
by (*force simp*: *bb-real-arith*)

end
theory *cat2rel*
imports
../hs-prelims-matrices
../../afpModified/VC-KAD

begin

Chapter 4

Hybrid System Verification with relations

— We start by deleting some conflicting notation.
no-notation *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor$-$\rfloor$)
and *Range-Semiring.antirange-semiring-class.ars-r* (r)
and *Relation.Domain* ($r2s$)

4.1 Verification of regular programs

Below we explore the behavior of the forward box operator from the antidomain kleene algebra with the lifting ($\lceil - \rceil^*$) operator from predicates to relations $\lceil P \rceil = \{(s,\ s)\ |s.\ P\ s\}$ and its dropping counterpart $\lfloor R \rfloor = (\lambda x.\ x \in \mathit{Domain}\ R)$.

lemma *p2r-IdD*:$\lceil P \rceil = Id \Longrightarrow P\ s$
by (*metis* (*full-types*) *UNIV-I impl-prop p2r-subid top-empty-eq*)

lemma *wp-rel*:$wp\ R\ \lceil P \rceil = \lceil \lambda\ x.\ \forall\ y.\ (x,y) \in R \longrightarrow P\ y \rceil$
proof−
have $\lfloor wp\ R\ \lceil P \rceil \rfloor = \lfloor \lceil \lambda\ x.\ \forall\ y.\ (x,y) \in R \longrightarrow P\ y \rceil \rfloor$
by (*simp add*: *wp-trafo pointfree-idE*)
thus $wp\ R\ \lceil P \rceil = \lceil \lambda\ x.\ \forall\ y.\ (x,y) \in R \longrightarrow P\ y \rceil$
by (*metis* (*no-types*, *lifting*) *wp-simp d-p2r pointfree-idE prp*)
qed

corollary *wp-relD*:$(x,x) \in wp\ R\ \lceil P \rceil \Longrightarrow \forall\ y.\ (x,y) \in R \longrightarrow P\ y$
proof−
assume $(x,x) \in wp\ R\ \lceil P \rceil$
hence $(x,x) \in \lceil \lambda\ x.\ \forall\ y.\ (x,y) \in R \longrightarrow P\ y \rceil$ **using** *wp-rel* **by** *auto*
thus $\forall\ y.\ (x,y) \in R \longrightarrow P\ y$ **by** (*simp add*: *p2r-def*)
qed

lemma *p2r-r2p-wp-sym*:$wp\ R\ P = \lceil \lfloor wp\ R\ P \rfloor \rceil$
 using *d-p2r wp-simp* **by** *blast*

lemma *p2r-r2p-wp*:$\lceil \lfloor wp\ R\ P \rfloor \rceil = wp\ R\ P$
 by(*rule sym*, *subst p2r-r2p-wp-sym*, *simp*)

Next, we introduce assignments and compute their *wp*.

abbreviation *vec-upd* :: $('a\hat{\ }'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a\hat{\ }'b$ (-(*2*[- :== -]) [*70*, *65*] *61*)
where
$x[i :== a] \equiv (\chi\ j.\ (\mathit{if}\ j = i\ \mathit{then}\ a\ \mathit{else}\ (x\ \$\ j)))$

abbreviation *assign* :: $'b \Rightarrow ('a\hat{\ }'b \Rightarrow 'a) \Rightarrow ('a\hat{\ }'b)\ \mathit{rel}$ ((*2*[- ::== -]) [*70*, *65*] *61*) **where**
$[x ::== expr] \equiv \{(s,\ s[x :== expr\ s]) |\ s.\ \mathit{True}\}$

lemma *wp-assign* [*simp*]: $wp\ ([x ::== expr])\ \lceil Q \rceil = \lceil \lambda s.\ Q\ (s[x :== expr\ s]) \rceil$
 by(*auto simp*: *rel-antidomain-kleene-algebra.fbox-def rel-ad-def p2r-def*)

lemma *wp-assign-var* [*simp*]: $\lfloor wp\ ([x ::== expr])\ \lceil Q \rceil \rfloor = (\lambda s.\ Q\ (s[x :== expr\ s]))$
 by(*subst wp-assign*, *simp add*: *pointfree-idE*)

The *wp* of the composition was already obtained in KAD.Antidomain_Semiring: $|x \cdot y]\ z = \ |x]\ \ |y]\ z$.

There is also already an implementation of the conditional operator *if p then x else y fi* $= d\ p \cdot x + ad\ p \cdot y$ and its *wp*: $|\mathit{if}\ p\ \mathit{then}\ x\ \mathit{else}\ y\ \mathit{fi}]\ q = d\ p \cdot |x]\ q + ad\ p \cdot\ |y]\ q$.

Finally, we add a wp-rule for a simple finite iteration.

lemma (**in** *antidomain-kleene-algebra*) *fbox-starI*:
assumes $d\ p \le d\ i$ **and** $d\ i \le |x]\ i$ **and** $d\ i \le d\ q$
shows $d\ p \le |x^\star]\ q$
proof−
from ‹$d\ i \le |x]\ i$› **have** $d\ i \le |x]\ (d\ i)$
 using *local.fbox-simp* **by** *auto*
hence $|1]\ p \le |x^\star]\ i$ **using** ‹$d\ p \le d\ i$› **by** (*metis* (*no-types*)
 local.dual-order.trans local.fbox-one local.fbox-simp local.fbox-star-induct-var)
thus *?thesis* **using** ‹$d\ i \le d\ q$› **by** (*metis* (*full-types*)
 local.fbox-mult local.fbox-one local.fbox-seq-var local.fbox-simp)
qed

lemma *rel-ad-mka-starI*:
assumes $P \subseteq I$ **and** $I \subseteq wp\ R\ I$ **and** $I \subseteq Q$
shows $P \subseteq wp\ (R^*)\ Q$
proof−
 have $wp\ R\ I \subseteq Id$
  by (*simp add*: *rel-antidomain-kleene-algebra.a-subid rel-antidomain-kleene-algebra.fbox-def*)
 hence $P \subseteq Id$ **using** *assms*(*1*,*2*) **by** *blast*

from *this* **have** *rdom* $P = P$ **by** (*metis d-p2r p2r-surj*)
also have *rdom* $P \subseteq$ *wp* (R^*) Q
by (*metis* ‹*wp* R $I \subseteq Id$› *assms d-p2r p2r-surj*
rel-antidomain-kleene-algebra.dka.dom-iso rel-antidomain-kleene-algebra.fbox-starI)
ultimately show *?thesis* **by** *blast*
qed

4.2 Verification of hybrid programs

4.2.1 Verification by providing solutions

abbreviation *guards* :: $('a \Rightarrow bool) \Rightarrow (real \Rightarrow 'a) \Rightarrow (real\ set) \Rightarrow bool$ (- $\rhd$ - - [*70*, *65*] *61*)
where $G \rhd x\ T \equiv \forall\ r \in T.\ G\ (x\ r)$

definition *ivp-sols* $f\ T\ t_0\ s = \{x\ |x.\ (D\ x = (f \circ x)\ on\ T) \wedge x\ t_0 = s \wedge t_0 \in T\}$

lemma *ivp-solsI*:
assumes $D\ x = (f \circ x)\ on\ T\ x\ t_0 = s\ t_0 \in T$
shows $x \in$ *ivp-sols* $f\ T\ t_0\ s$
using *assms* **unfolding** *ivp-sols-def* **by** *blast*

lemma *ivp-solsD*:
assumes $x \in$ *ivp-sols* $f\ T\ t_0\ s$
shows $D\ x = (f \circ x)\ on\ T$
and $x\ t_0 = s$ **and** $t_0 \in T$
using *assms* **unfolding** *ivp-sols-def* **by** *auto*

lemma $(t{::}real) \in \{0{-}{-}t\}$
by (*rule ends-in-segment*(*2*))

lemma $(t{::}real) \in \{0..t\}$
apply *auto*
oops

definition *g-orbital* $f\ T\ t_0\ G\ s = \bigcup\ \{\{x\ t|t.\ t \in T \wedge G \rhd x\ \{t_0{-}{-}t\}\ \}|x.\ x \in$ *ivp-sols* $f\ T\ t_0\ s\}$

lemma *g-orbital-eq*: *g-orbital* $f\ T\ t_0\ G\ s =$
$\{x\ t\ |t\ x.\ t \in T \wedge (D\ x = (f \circ x)\ on\ T) \wedge x\ t_0 = s \wedge t_0 \in T \wedge G \rhd x\ \{t_0{-}{-}t\}\}$
unfolding *g-orbital-def ivp-sols-def* **by** *auto*

lemma *g-orbital* $f\ T\ t_0\ G\ s = (\bigcup\ x \in$ *ivp-sols* $f\ T\ t_0\ s.\ \{x\ t|t.\ t \in T \wedge G \rhd x\ \{t_0{-}{-}t\}\})$
unfolding *g-orbital-def ivp-sols-def* **by** *auto*

lemma *g-orbitalI*:
assumes $D\ x = (f \circ x)\ on\ T\ x\ t_0 = s$
and $t_0 \in T\ t \in T$ **and** $G \rhd x\ \{t_0{-}{-}t\}$

shows $x\ t \in$ *g-orbital f T* t_0 *G s*
using *assms* **unfolding** *g-orbital-def ivp-sols-def* **by** *blast*

lemma *g-orbitalD*:
assumes $s' \in$ *g-orbital f T* t_0 *G s*
obtains *x* **and** *t* **where** $x \in$ *ivp-sols f T* t_0 *s*
and $D\ x = (f \circ x)$ *on T* $x\ t_0 = s$
and $x\ t = s'$ **and** $t_0 \in T\ t \in T$ **and** $G \triangleright x\ \{t_0{-}{-}t\}$
using *assms* **unfolding** *g-orbital-def ivp-sols-def* **by** *blast*

abbreviation *g-evol* :: $(('a{::}banach) \Rightarrow 'a) \Rightarrow$ *real set* $\Rightarrow 'a$ *pred* $\Rightarrow 'a$ *rel* ((*1*[*x*´=-]- & -))
where [*x*´=*f*]*T* & $G \equiv \{(s,s').\ s' \in$ *g-orbital f T 0 G s*$\}$

lemmas *g-evol-def* = *g-orbital-eq*[**where** t_0=*0*]

context *local-flow*
begin

lemma *in-ivp-sols*: $(\lambda t.\ \varphi\ t\ s) \in$ *ivp-sols f T 0 s*
by(*auto intro*: *ivp-solsI simp*: *ivp init-time*)

definition *orbit s* = *g-orbital f T 0* ($\lambda s.$ *True*) *s*

lemma *orbit-eq*[*simp*]: *orbit s* = $\{\varphi\ t\ s|\ t.\ t \in T\}$
unfolding *orbit-def g-evol-def*
by(*auto intro*: *usolves-ivp intro*!: *ivp simp*: *init-time*)

lemma *g-orbital-collapses*:
shows *g-orbital f T 0 G s* = $\{\varphi\ t\ s \mid t.\ t \in T \wedge G \triangleright (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}\}$ (**is** - = *?gorbit*)
proof(*rule subset-antisym*, *simp-all only*: *subset-eq*)
{**fix** s' **assume** $s' \in$ *g-orbital f T 0 G s*
then obtain *x* **and** *t* **where** *x-ivp*:$D\ x = (f \circ x)$ *on T*
$x\ 0 = s$ **and** $x\ t = s'$ **and** $t \in T$ **and** *guard*:$G \triangleright x\ \{0{-}{-}t\}$
unfolding *g-orbital-eq* **by** *blast*
hence *obs*:$\forall \tau \in \{0{-}{-}t\}.\ x\ \tau = \varphi\ \tau\ s$
using *usolves-ivp*[*of x s*] *closed-segment-subset-domainI init-time comp-def*
by (*metis* (*mono-tags*, *lifting*) *has-vderiv-eq*)
hence $G \triangleright (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}$
using *guard* **by** *simp*
hence $s' \in$ *?gorbit*
using ‹$x\ t = s'$› ‹$t \in T$› *obs* **by** *blast*}
thus $\forall s' \in$ *g-orbital f T 0 G s*. $s' \in$ *?gorbit*
by *blast*
next
{**fix** s' **assume** $s' \in$ *?gorbit*
then obtain *t* **where** $G \triangleright (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}$ **and** $t \in T$ **and** $\varphi\ t\ s = s'$
by *blast*

  hence $s' \in$ *g-orbital f T 0 G s*
    by(*auto intro*: *g-orbitalI simp*: *ivp init-time*)}
 thus $\forall s' \in ?gorbit.\ s' \in$ *g-orbital f T 0 G s*
  by *blast*
qed

lemma *g-evol-collapses*:
 shows $([x'{=}f]T\ \&\ G) = \{(s, \varphi\ t\ s) \mid t\ s.\ t \in T \wedge G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}\}$
 unfolding *g-orbital-collapses* **by** *auto*

lemma *wp-orbit*: $wp\ (\{(s,s') \mid s\ s'.\ s' \in orbit\ s\})\ \lceil Q \rceil = \lceil \lambda\ s.\ \forall\ t \in T.\ Q\ (\varphi\ t\ s) \rceil$
 unfolding *orbit-eq wp-rel* **by** *auto*

lemma *wp-g-orbit*: $wp\ ([x'{=}f]T\ \&\ G)\ \lceil Q \rceil = \lceil \lambda\ s.\ \forall t{\in}T.\ (G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}) \longrightarrow Q\ (\varphi\ t\ s) \rceil$
 unfolding *g-evol-collapses wp-rel* **by** *auto*

end

lemma (**in** *global-flow*) *ivp-sols-collapse*[*simp*]: *ivp-sols f UNIV 0 s* $= \{(\lambda t.\ \varphi\ t\ s)\}$
 by(*auto intro*: *usolves-ivp simp*: *ivp-sols-def ivp*)

The previous theorem allows us to compute wlps for known systems of ODEs. We can also implement a version of it as an inference rule. A simple computation of a wlp is shown immmediately after.

lemma *dSolution*:
 assumes *local-flow f T L* φ
  and $\forall s.\ P\ s \longrightarrow (\forall\ t \in T.\ (G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0..t\}) \longrightarrow Q\ (\varphi\ t\ s))$
 shows $\lceil P \rceil \le wp\ ([x'{=}f]T\ \&\ G)\ \lceil Q \rceil$
 using *assms* **apply**(*subst local-flow.wp-g-orbit*, *auto*)
 by (*simp add*: *Starlike.closed-segment-eq-real-ivl*)

lemma *line-DS*: $0 \le t \Longrightarrow wp\ ([x'{=}\lambda s.\ c]\{0..t\}\ \&\ G)\ \lceil Q \rceil =$
  $\lceil \lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (G \rhd (\lambda r.\ x + r *_R c)\ \{0..\tau\}) \longrightarrow Q\ (x + \tau *_R c) \rceil$
 apply(*subst local-flow.wp-g-orbit*[*of* $\lambda s.\ c$ - $1/(t + 1)$ $(\lambda\ t\ x.\ x + t *_R c)$])
 by(*auto simp*: *line-is-local-flow closed-segment-eq-real-ivl*)

4.2.2 Verification with differential invariants

We derive the domain specific rules of differential dynamic logic (dL). In each subsubsection, we first derive the dL axioms (named below with two capital letters and "D" being the first one). This is done mainly to prove that there are minimal requirements in Isabelle to get the dL calculus. Then we prove the inference rules which are used in verification proofs.

Differential Weakening

lemma *DW*: *wp* ([x´=f]T & G) ⌈Q⌉ = *wp* ([x´=f]T & G) ⌈λ *s*. G *s* ⟶ Q *s*⌉
  apply(*subst wp-rel*)+
  by(*auto simp*: *g-orbital-eq*)

lemma *dWeakening*:
  assumes ⌈G⌉ ≤ ⌈Q⌉
  shows ⌈P⌉ ≤ *wp* ([x´=f]T & G) ⌈Q⌉
  using *assms* **apply**(*subst wp-rel*)
  by(*auto simp*: *g-orbital-eq*)

Differential Cut

lemma *wp-g-orbit-IdD*:
  assumes *wp* ([x´=f]T & G) ⌈C⌉ = *Id* **and** ∀ *r*∈{*0*−−*t*}. (*s*, *x r*) ∈ ([x´=f]T & G)
  shows ∀ *r*∈{*0*−−*t*}. C (*x r*)
proof
  fix *r* **assume** *r* ∈ {*0*−−*t*}
  then have *x r* ∈ *g-orbital f T 0 G s*
    using *assms*(*2*) **by** *blast*
  also have ∀ *y*. *y* ∈ (*g-orbital f T 0 G s*) ⟶ *C y*
    using *assms*(*1*) **unfolding** *wp-rel* **by**(*auto simp*: *p2r-def*)
  ultimately show C (*x r*) **by** *blast*
qed

theorem *DC*:
  assumes *interval T* **and** *wp* ([x´=f]T & G) ⌈C⌉ = *Id*
  shows *wp* ([x´=f]T & G) ⌈Q⌉ = *wp* ([x´=f]T & (λ*s*. *G s* ∧ *C s*)) ⌈Q⌉
proof(*rule-tac f*=λ *x*. *wp x* ⌈Q⌉ **in** *HOL.arg-cong*, *rule subset-antisym*, *safe*)
  {fix *s* **and** *s*′ **assume** *s*′ ∈ *g-orbital f T 0 G s*
    then obtain *t*::*real* **and** *x* **where** *x-ivp*: *D x* = (*f* ∘ *x*) *on T x 0* = *s*
      and *guard-x*:*G* ⊳ *x* {*0*−−*t*} **and** *s*′ = *x t* **and** *0* ∈ *T t* ∈ *T*
      using *g-orbitalD*[*of s*′ *f T 0 G s*] **by** (*metis* (*full-types*))
    from *guard-x* **have** ∀ *r*∈{*0*−−*t*}.∀ τ∈{*0*−−*r*}. *G* (*x* τ)
      by (*metis contra-subsetD ends-in-segment*(*1*) *subset-segment*(*1*))
    also have ∀ τ∈{*0*−−*t*}. τ ∈ *T*
      using *interval.closed-segment-subset-domain*[*OF assms*(*1*) ‹*0* ∈ *T*› ‹*t* ∈ *T*›] **by** *blast*
    ultimately have ∀ τ∈{*0*−−*t*}. *x* τ ∈ *g-orbital f T 0 G s*
      using *g-orbitalI*[*OF x-ivp* ‹*0* ∈ *T*›] **by** *blast*
    hence ∀ τ∈{*0*−−*t*}. (*s*, *x* τ) ∈ [x´=f]T & G
      unfolding *wp-rel* **by**(*auto simp*: *p2r-def*)
    hence *C* ⊳ *x* {*0*−−*t*}
      using *wp-g-orbit-IdD*[*OF assms*(*2*)] **by** *blast*
    hence *s*′ ∈ *g-orbital f T 0* (λ*s*. *G s* ∧ *C s*) *s*
      using *g-orbitalI*[*OF x-ivp* ‹*0* ∈ *T*› ‹*t* ∈ *T*›] *guard-x* ‹*s*′ = *x t*› **by** *fastforce***}**
  thus ⋀*s s*′. *s*′ ∈ *g-orbital f T 0 G s* ⟹ *s*′ ∈ *g-orbital f T 0* (λ*s*. *G s* ∧ *C s*) *s*
    by *blast*

next show $\bigwedge s\ s'.\ s' \in$ *g-orbital f T 0* $(\lambda s.\ G\ s \wedge C\ s)\ s \Longrightarrow s' \in$ *g-orbital f T 0* $G\ s$
  by (*auto simp*: *g-evol-def*)
qed

theorem *dCut*:
  assumes *wp-C*:$\lceil P\rceil \leq wp\ ([x'{=}f]\{0..t\}\ \&\ G)\ \lceil C\rceil$
    and *wp-Q*:$\lceil P\rceil \subseteq wp\ ([x'{=}f]\{0..t\}\ \&\ (\lambda\ s.\ G\ s \wedge C\ s))\ \lceil Q\rceil$
  shows $\lceil P\rceil \subseteq wp\ ([x'{=}f]\{0..t\}\ \&\ G)\ \lceil Q\rceil$
proof(*subst wp-rel*, *simp add*: *g-orbital-eq p2r-def*, *clarsimp*)
  fix τ::*real* **and** x::*real* $\Rightarrow$ *'a* **assume** $P\ (x\ 0)$ **and** $0 \leq \tau$ **and** $\tau \leq t$
    and *x-solves*:$D\ x = (\lambda t.\ f\ (x\ t))$ *on* $\{0..t\}$ **and** *guard-x*:$(\forall\ r \in \{0{-}{-}\tau\}.\ G\ (x\ r))$
  hence $\forall r{\in}\{0{-}{-}\tau\}.\forall \tau{\in}\{0{-}{-}r\}.\ G\ (x\ \tau)$
    using *closed-segment-closed-segment-subset* **by** *blast*
  hence $\forall r{\in}\{0{-}{-}\tau\}.\ x\ r \in$ *g-orbital f* $\{0..t\}\ 0\ G\ (x\ 0)$
    using *g-orbitalI x-solves* ‹$0 \leq \tau$› ‹$\tau \leq t$› *closed-segment-eq-real-ivl* **by** *fastforce*
  hence $\forall r{\in}\{0{-}{-}\tau\}.\ C\ (x\ r)$
    using *wp-C* ‹$P\ (x\ 0)$› **by**(*subst* (*asm*) *wp-rel*, *auto*)
  hence $x\ \tau \in$ *g-orbital f* $\{0..t\}\ 0\ (\lambda s.\ G\ s \wedge C\ s)\ (x\ 0)$
    using *g-orbitalI x-solves guard-x* ‹$0 \leq \tau$› ‹$\tau \leq t$› **by** *fastforce*
  from *this* ‹$P\ (x\ 0)$› **and** *wp-Q* **show** $Q\ (x\ \tau)$
    by(*subst* (*asm*) *wp-rel*, *auto simp*: *closed-segment-eq-real-ivl*)
qed

Differential Invariant

lemma *DI-sufficiency*:
  assumes $\forall s.\ \exists x.\ x \in$ *ivp-sols f T 0 s*
  shows $wp\ ([x'{=}f]T\ \&\ G)\ \lceil Q\rceil \leq wp\ \lceil G\rceil\ \lceil Q\rceil$
  apply(*subst wp-rel*, *subst wp-rel*, *simp add*: *p2r-def*, *clarsimp*)
  using *assms* **apply**(*simp add*: *g-evol-def ivp-sols-def*)
  apply(*erule-tac x=s* **in** *allE*)+
  apply(*erule exE*, *erule impE*)
  by(*rule-tac x=0* **in** *exI*, *rule-tac x=x* **in** *exI*, *auto*)

lemma (**in** *local-flow*) *DI-necessity*:
  shows $wp\ \lceil G\rceil\ \lceil Q\rceil \leq wp\ ([x'{=}f]T\ \&\ G)\ \lceil Q\rceil$
  unfolding *wp-g-orbit* **apply**(*subst wp-rel*, *simp add*: *p2r-def*, *clarsimp*)
   apply(*erule-tac x=0* **in** *ballE*)
    apply(*simp-all add*: *ivp*)
  oops

definition *diff-invariant* :: *'a pred* $\Rightarrow$ ((*'a*::*real-normed-vector*) $\Rightarrow$ *'a*) $\Rightarrow$ *real set* $\Rightarrow$ *bool*
((-)/ *is'-diff'-invariant'-of* (-)/ *along* (-) [*70*,*65*]*61*)
where *I is-diff-invariant-of f along T* $\equiv$
  $(\forall s.\ I\ s \longrightarrow (\forall\ x.\ x \in$ *ivp-sols f T 0 s* $\longrightarrow (\forall\ t \in T.\ I\ (x\ t))))$

lemma *invariant-to-set*:
shows (I *is-diff-invariant-of* f *along* T) $\longleftrightarrow$ ($\forall s.\ I\ s \longrightarrow$ (*g-orbital* f T *0* ($\lambda s.$ *True*) s) $\subseteq \{s.\ I\ s\}$)
unfolding *diff-invariant-def ivp-sols-def g-orbital-eq* **apply** *safe*
apply(*erule-tac x=xa 0* **in** *allE*)
apply(*drule mp*, *simp-all*)
apply(*erule-tac x=xa 0* **in** *allE*)
apply(*drule mp*, *simp-all add*: *subset-eq*)
apply(*erule-tac x=xa t* **in** *allE*)
by(*drule mp*, *auto*)

lemma *dInvariant*:
assumes I *is-diff-invariant-of* f *along* T
shows $\lceil I \rceil \leq wp$ ($[x' = f]T$ & G) $\lceil I \rceil$
using *assms* **unfolding** *diff-invariant-def*
by(*auto simp*: *wp-rel g-evol-def ivp-sols-def*)

lemma *dI*:
assumes I *is-diff-invariant-of* f *along* $\{0..t\}$
and $\lceil P \rceil \leq \lceil I \rceil$ **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp$ ($[x' = f]\{0..t\}$ & G) $\lceil Q \rceil$
using *assms*(*1*) **apply**(*rule-tac C=I* **in** *dCut*)
apply(*drule-tac G=G* **in** *dInvariant*)
using *assms*(*2*) *dual-order.trans* **apply** *blast*
apply(*rule dWeakening*)
using *assms* **by** *auto*

Finally, we obtain some conditions to prove specific instances of differential invariants.

named-theorems *ode-invariant-rules compilation of rules for differential invariants*.

lemma [*ode-invariant-rules*]:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.$ ($D\ x = (\lambda\tau.\ f\ (x\ \tau))$ *on* $\{0..t\}$) $\longrightarrow$ ($\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0{-}{-}\tau\}.$

(($\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)$) *has-derivative* ($\lambda\tau.\ \tau *_R 0$)) (*at* r *within* $\{0{-}{-}\tau\}$))
shows ($\lambda s.\ \vartheta\ s = \nu\ s$) *is-diff-invariant-of* f *along* $\{0..t\}$
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
fix $x\ \tau$ **assume** *tHyp*:$0 \leq \tau\ \tau \leq t$
and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))$ *on* $\{0..t\}$ $\vartheta\ (x\ 0) = \nu\ (x\ 0)$
hence $\forall\ r \in \{0{-}{-}\tau\}.\ D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R 0)$ *at* r *within* $\{0{-}{-}\tau\}$
using *assms* **by** *auto*
hence $\exists\, r{\in}\{0{-}{-}\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) = (\lambda\tau.\ \tau *_R 0)\ (\tau - 0)$
by(*rule-tac closed-segment-mvt*, *auto simp*: *tHyp*)
thus $\vartheta\ (x\ \tau) = \nu\ (x\ \tau)$ **by** (*simp add*: *x-ivp*(*2*))

qed

lemma [*ode-invariant-rules*]:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}) \longrightarrow (\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0{-}{-}\tau\}.$
$\vartheta'\ (x\ r) \geq \nu'\ (x\ r)$
$\wedge\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau)\ -\ \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r)\ -\ \nu'\ (x\ r)))\ at\ r\ within$
$\{0{-}{-}\tau\}))$
shows $(\lambda s.\ \nu\ s \leq \vartheta\ s)$ *is-diff-invariant-of f along* $\{0..t\}$
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
  fix $x\ \tau$ **assume** *tHyp*:$0 \leq \tau\ \tau \leq t$
    and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}\ \nu\ (x\ 0) \leq \vartheta\ (x\ 0)$
  hence *primed*:$\forall\ r \in \{0{-}{-}\tau\}.\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x$
$r) - \nu'\ (x\ r)))$
  $at\ r\ within\ \{0{-}{-}\tau\}) \wedge \nu'\ (x\ r) \leq \vartheta'\ (x\ r)$
    using *assms* **by** *auto*
  hence $\exists\, r{\in}\{0{-}{-}\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) =$
  $(\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) -\ \nu'\ (x\ r)))\ (\tau - 0)$
    by(*rule-tac closed-segment-mvt*, *auto simp*: ‹$0 \leq \tau$›)
  then obtain r **where** $r \in \{0{-}{-}\tau\}$
    and $\vartheta\ (x\ \tau) - \nu\ (x\ \tau) = (\tau - 0) *_R (\vartheta'\ (x\ r) -\ \nu'\ (x\ r)) + (\vartheta\ (x\ 0) - \nu\ (x$
$0))$
    by *force*
  also have ... ≥ 0
    using *tHyp*(*1*) *x-ivp*(*2*) *primed* **by** (*simp add*: *calculation*(*1*))
  ultimately show $\nu\ (x\ \tau) \leq \vartheta\ (x\ \tau)$
    by *simp*
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::$'a$::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}) \longrightarrow (\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0{-}{-}\tau\}.$
$\vartheta'\ (x\ r) \geq \nu'\ (x\ r)$
$\wedge\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau)\ -\ \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r)\ -\ \nu'\ (x\ r)))\ at\ r\ within$
$\{0{-}{-}\tau\}))$
shows $(\lambda s.\ \nu\ s < \vartheta\ s)$ *is-diff-invariant-of f along* $\{0..t\}$
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
  fix $x\ \tau$ **assume** *tHyp*:$0 \leq \tau\ \tau \leq t$
    and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}\ \nu\ (x\ 0) < \vartheta\ (x\ 0)$
  hence *primed*:$\forall\ r \in \{0{-}{-}\tau\}.\ ((\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau))$ *has-derivative*
$(\lambda\tau.\ \tau *_R\ (\vartheta'\ (x\ r) -\ \nu'\ (x\ r))))\ (at\ r\ within\ \{0{-}{-}\tau\}) \wedge \vartheta'\ (x\ r) \geq \nu'\ (x\ r)$
    using *assms* **by** *auto*
  hence $\exists\, r{\in}\{0{-}{-}\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) =$
  $(\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) -\ \nu'\ (x\ r)))\ (\tau - 0)$
    by(*rule-tac closed-segment-mvt*, *auto simp*: ‹$0 \leq \tau$›)
  then obtain r **where** $r \in \{0{-}{-}\tau\}$ **and**
    $\vartheta\ (x\ \tau) - \nu\ (x\ \tau) = (\tau - 0) *_R (\vartheta'\ (x\ r) -\ \nu'\ (x\ r)) + (\vartheta\ (x\ 0) - \nu\ (x\ 0))$
    by *force*
  also have ... > 0

using *tHyp(1) x-ivp(2) primed* **by** (*metis* (*no-types,hide-lams*) *Groups.add-ac(2)* *add-sign-intros(1)*
calculation(1) diff-gt-0-iff-gt ge-iff-diff-ge-0 less-eq-real-def zero-le-scaleR-iff)

ultimately show ν (x τ) < ϑ (x τ)
by *simp*
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::'*a*::*banach* ⇒ *real*
assumes *I1 is-diff-invariant-of f along* {*0..t*}
and *I2 is-diff-invariant-of f along* {*0..t*}
shows (λs. *I1 s* ∧ *I2 s*) *is-diff-invariant-of f along* {*0..t*}
using *assms* **unfolding** *diff-invariant-def* **by** *auto*

lemma [*ode-invariant-rules*]:
fixes ϑ::'*a*::*banach* ⇒ *real*
assumes *I1 is-diff-invariant-of f along* {*0..t*}
and *I2 is-diff-invariant-of f along* {*0..t*}
shows (λs. *I1 s* ∨ *I2 s*) *is-diff-invariant-of f along* {*0..t*}
using *assms* **unfolding** *diff-invariant-def* **by** *auto*

end
theory *cat2rel-examples*
imports *cat2rel*

begin

4.2.3 Examples

The examples in this subsection show different approaches for the verification of hybrid systems. However, the general approach can be outlined as follows: First, we select a finite type to model program variables $'n$. We use this to define a vector field f of type $('a,\ 'n)$ *vec* ⇒ $('a,\ 'n)$ *vec* to model the dynamics of our system. Then we show a partial correctness specification involving the evolution command $[x' {=} f]\, T$ & G either by finding a flow for the vector field or through differential invariants.

Single constantly accelerated evolution

The main characteristics distinguishing this example from the rest are:

1. We define the finite type of program variables with 2 Isabelle strings which make the final verification easier to parse.

2. We define the vector field (named K) to model a constantly accelerated object.

3. We define a local flow (φ_K) and use it to compute the wlp for this vector field.

4. The verification is only done on a single evolution command (not operated with any other hybrid program).

typedef *program-vars* $=\{''y'',''v''\}$
 morphisms *to-str to-var*
 apply(*rule-tac* $x=''y''$ **in** *exI*)
 by *simp*

notation *to-var* ($\restriction_V$)

lemma *number-of-program-vars*: $CARD(program\text{-}vars) = 2$
 using *type-definition.card type-definition-program-vars* **by** *fastforce*

instance *program-vars*::*finite*
 apply(*standard*, *subst bij-betw-finite*[*of to-str UNIV* $\{''y'',''v''\}$])
  apply(*rule bij-betwI'*)
    apply (*simp add*: *to-str-inject*)
 using *to-str* **apply** *blast*
  apply (*metis to-var-inverse UNIV-I*)
 by *simp*

lemma *program-vars-univD*:(*UNIV*::*program-vars set*) $= \{\restriction_V ''y'', \restriction_V ''v''\}$
 apply *auto* **by** (*metis to-str to-str-inverse insertE singletonD*)

lemma *program-vars-exhaust*: $\forall\, x$::*program-vars*. $x = \restriction_V ''y'' \lor x = \restriction_V ''v''$
 using *program-vars-univD* **by** *auto*

abbreviation *constant-acceleration-kinematics g s* $\equiv$
 (χ *i*. *if* $i=(\restriction_V ''y'')$ *then s* \$ ($\restriction_V ''v''$) *else g*)

notation *constant-acceleration-kinematics* (K)

lemma *cnst-acc-continuous*:
 fixes X::(*real*^*program-vars*) *set*
 shows *continuous-on X* (K g)
 apply(*rule continuous-on-vec-lambda*)
 unfolding *continuous-on-def* **apply** *clarsimp*
 by(*intro tendsto-intros*)

lemma *picard-lindeloef-cnst-acc*:
 fixes *g*::*real* **assumes** $0 \le t$ **and** $t < 1$
 shows *picard-lindeloef* (λt. K g) $\{0..t\}$ 1 0
 unfolding *picard-lindeloef-def* **apply**(*simp add*: *lipschitz-on-def assms*, *safe*)
 apply(*rule-tac* $t=UNIV$ **and** $f=snd$ **in** *continuous-on-compose2*)
 apply(*simp-all add*: *cnst-acc-continuous continuous-on-snd*)
  apply(*simp add*: *dist-vec-def L2-set-def dist-real-def*)

apply(*subst program-vars-univD*, *subst program-vars-univD*)
apply(*simp-all add*: *to-var-inject*)
using *assms* **by** *linarith*

abbreviation *constant-acceleration-kinematics-flow* $g\ t\ s \equiv$
$(\chi\ i.\ \textit{if}\ i{=}(\restriction_V\ ''y'')\ \textit{then}\ g \cdot t\ \hat{}\ 2/2 + s\ \$\ (\restriction_V\ ''v'') \cdot t + s\ \$\ (\restriction_V\ ''y'')$
$\textit{else}\ g \cdot t + s\ \$\ (\restriction_V\ ''v''))$

notation *constant-acceleration-kinematics-flow* (φ_K)

lemma *local-flow-cnst-acc*:
assumes $0 \le t$ **and** $t < 1$
shows *local-flow* $(K\ g)\ \{0..t\}\ 1\ (\varphi_K\ g)$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *assms picard-lindeloef-cnst-acc* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*)
using *poly-derivatives*(*3*,*4*) *program-vars-exhaust*
apply(*simp-all add*: *to-var-inject vec-eq-iff has-vderiv-on-def has-vector-derivative-def*)
using *program-vars-exhaust* **by** *blast*

lemma *wp-cnst-acc*:
assumes $0 \le t$ **and** $t < 1$
shows $wp\ ([x' {=} K\ g]\{0..t\}\ \&\ G)\ \lceil Q \rceil =$
$\lceil \lambda\ s.\ \forall\ \tau \in \{0..t\}.\ (G \rhd (\lambda r.\ \varphi_K\ g\ r\ s)\{0{-}{-}\tau\}) \longrightarrow Q\ (\varphi_K\ g\ \tau\ s) \rceil$
apply(*subst local-flow.wp-g-orbit*[*of* $K\ g$ - 1 $(\lambda\ t\ x.\ \varphi_K\ g\ t\ x)$])
using *local-flow-cnst-acc* **and** *assms* **by**(*auto simp*: *p2r-def*)

lemma *single-evolution-ball*:
fixes H::*real* **assumes** $0 \le t$ **and** $t < 1$ **and** $g < 0$
shows $\lceil \lambda s.\ 0 \le s\ \$\ (\restriction_V\ ''y'') \wedge s\ \$\ (\restriction_V\ ''y'') = H \wedge s\ \$\ (\restriction_V\ ''v'') = 0 \rceil$
$\le wp\ ([x' {=} K\ g]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ (\restriction_V\ ''y'') \ge 0))$
$\lceil \lambda s.\ 0 \le s\ \$\ (\restriction_V\ ''y'') \wedge s\ \$\ (\restriction_V\ ''y'') \le H \rceil$
apply(*subst wp-cnst-acc*)
using *assms* **by**(*auto simp*: *mult-nonpos-nonneg*)

no-notation *to-var* $(\restriction_V)$

no-notation *constant-acceleration-kinematics* (K)

no-notation *constant-acceleration-kinematics-flow* (φ_K)

Single evolution revisited.

We list again the characteristics that distinguish this example:

1. We employ an existing finite type of size 3 to model program variables.

2. We define a 3×3 matrix (named K) to denote the linear operator that models the constantly accelerated motion.

3. We define a local flow (φ_K) and use it to compute the wlp for this linear operator.

4. The verification is done equivalently to the above example.

term *x::2* — It turns out that there is already a 2-element type:

lemma $CARD(program\text{-}vars) = CARD(2)$
  unfolding *number-of-program-vars* **by** *simp*

In fact, for each natural number n there is already a corresponding n-element type in Isabelle. However, there are still lemmas to prove about them in order to do verification of hybrid systems in n-dimensional Euclidean spaces.

lemma *exhaust-5*: — The analogs for 1, 2 and 3 have already been proven in Analysis.
  fixes *x::5*
  shows $x{=}1 \lor x{=}2 \lor x{=}3 \lor x{=}4 \lor x{=}5$
proof (*induct x*)
  case (*of-int z*)
  then have $0 \leq z$ **and** $z < 5$ **by** *simp-all*
  then have $z = 0 \lor z = 1 \lor z = 2 \lor z = 3 \lor z = 4$ **by** *arith*
  then show *?case* **by** *auto*
qed

lemma *UNIV-3*:(*UNIV::3 set*) = $\{0,\ 1,\ 2\}$
  apply *safe* **using** *exhaust-3 three-eq-zero* **by**(*blast*, *auto*)

lemma *sum-axis-UNIV-3*[*simp*]:($\sum j{\in}(UNIV{::}3\ set).\ axis\ i\ 1\ \$\ j \cdot f\ j) = (f{::}3 \Rightarrow real)\ i$
  unfolding *axis-def UNIV-3* **apply** *simp*
  using *exhaust-3* **by** *force*

We can rewrite the original constant acceleration kinematics as a linear operator applied to a 3-dimensional vector. For that we take advantage of the following fact:

lemma e $1 = (\chi\ j{::}3.\ if\ j{=}\ 0\ then\ 0\ else\ if\ j = 1\ then\ 1\ else\ 0)$
  unfolding *axis-def* **by**(*rule Cart-lambda-cong*, *simp*)

abbreviation *constant-acceleration-kinematics-matrix* $\equiv$
  $(\chi\ i.\ if\ i{=}\ (0{::}3)\ then\ axis\ (1{::}3)\ (1{::}real)\ else\ if\ i{=}\ 1\ then\ axis\ 2\ 1\ else\ 0)$

abbreviation *constant-acceleration-kinematics-matrix-flow t s* $\equiv$
  $(\chi\ i.\ if\ i{=}\ (0{::}3)\ then\ s\ \$\ 2 \cdot t\ \hat{}\ 2/2 + s\ \$\ 1 \cdot t + s\ \$\ 0$
   $else\ if\ i{=}1\ then\ s\ \$\ 2 \cdot t + s\ \$\ 1\ else\ s\ \$\ 2)$

notation *constant-acceleration-kinematics-matrix* (K)

notation *constant-acceleration-kinematics-matrix-flow* (φ_K)

With these 2 definitions and the proof that linear systems of ODEs are Picard-Lindeloef, we can show that they form a pair of vector-field and its flow.

lemma *entries-cnst-acc-matrix*: *entries K* = {*0*, *1*}
 apply (*simp-all add*: *axis-def*, *safe*)
 by(*rule-tac x=1* **in** *exI*, *simp*)+

lemma *picard-lindeloef-cnst-acc-matrix*:
 assumes $0 \leq t$ **and** $t < 1/9$
 shows *picard-lindeloef* (λ *t s. K* $*v$ *s*) {*0..t*} ((*real CARD*(*3*))^2 $\cdot$ ($\|K\|_{max}$)) *0*
 apply(*rule picard-lindeloef-linear-system*)
 unfolding *entries-cnst-acc-matrix* **using** *assms* **by** *auto*

lemma *local-flow-cnst-acc-matrix*:
 assumes $0 \leq t$ **and** $t < 1/9$
 shows *local-flow* (($*v$) *K*) {*0..t*} ((*real CARD*(*3*))^2 $\cdot$ ($\|K\|_{max}$)) φ_K
 unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
 using *picard-lindeloef-cnst-acc-matrix*[*OF assms*] **apply** *blast*
  apply(*rule has-vderiv-on-vec-lambda*)
 using *poly-derivatives*(*1*,*3*, *4*)
  apply(*force simp*: *matrix-vector-mult-def*)
 using *exhaust-3* **by**(*force simp*: *matrix-vector-mult-def vec-eq-iff*)

Finally, we compute the wlp of this example and use it to verify the single-evolution ball again.

lemma *wp-cnst-acc-matrix*:
 assumes $0 \leq t$ **and** $t < 1/9$
 shows *wp* ([*x*´=($*v$) *K*]{*0..t*} & *G*) $\lceil Q \rceil$ = $\lceil \lambda$ *s*. $\forall \tau \in$ {*0..t*}. (*G* $\rhd$ (λr. φ_K *r s*){*0--*τ}) $\longrightarrow$ *Q* (φ_K τ *s*)$\rceil$
  apply(*subst local-flow.wp-g-orbit*[*of* ($*v$) *K* - ((*real CARD*(*3*))^2 $\cdot$ ($\|K\|_{max}$)) φ_K])
 using *local-flow-cnst-acc-matrix* **and** *assms* **by** *auto*

lemma *single-evolution-ball-K*:
 assumes $0 \leq t$ **and** $t < 1/9$
 shows $\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2 \rceil$
 $\leq$ *wp* ([*x*´=($*v$) *K*]{*0..t*} & ($\lambda s.\ s\ \$\ 0 \geq 0$)) $\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H \rceil$
 apply(*subst wp-cnst-acc-matrix*)
 using *assms* **by**(*auto simp*: *mult-nonneg-nonpos2*)

Circular Motion

The characteristics that distinguish this example are:

1. We employ an existing finite type of size 2 to model program variables.

2. We define a 2×2 matrix (named C) to denote the linear operator that models circular motion.

3. We show that the circle equation is a differential invariant for the linear operator.

4. We prove the partial correctness specification corresponding to the previous point.

5. For completeness, we define a local flow (φ_C) and use it to compute the wlp for the linear operator and the specification is proven again with this flow.

lemma *two-eq-zero*: $(2::2) = 0$
 by *simp*

lemma [*simp*]:$i \neq (0::2) \longrightarrow i = 1$
 using *exhaust-2* **by** *fastforce*

lemma *UNIV-2*:($UNIV$::*2 set*) $= \{0,\ 1\}$
 apply *safe* **using** *exhaust-2 two-eq-zero* **by** *auto*

abbreviation *circular-motion-matrix* $\equiv$
 (χ *i. if i*= (*0*::*2*) *then axis* (*1*::*2*) ($-$ *1*::*real*) *else axis 0 1*)

notation *circular-motion-matrix* (C)

lemma *circle-invariant*:
 assumes $0 < R$
 shows ($\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2$) *is-diff-invariant-of* ($*v$) C *along* $\{0..t\}$
 apply(*rule-tac ode-invariant-rules*, *clarsimp*)
 apply(*frule-tac i=0* **in** *has-vderiv-on-vec-nth*, *drule-tac i=1* **in** *has-vderiv-on-vec-nth*)
 apply(*unfold has-vderiv-on-def has-vector-derivative-def*, *clarsimp*)
 apply(*erule-tac x=r* **in** *ballE*)+
 apply(*simp add*: *matrix-vector-mult-def has-vderiv-on-vec-lambda*)
 subgoal for $x\ \tau\ r$ **apply**(*rule-tac f'1*=$\lambda t.\ 0$ **and** *g'1*=$\lambda t.\ 0$ **in** *derivative-eq-intros*(*11*), *simp-all*)
 apply(*rule-tac f'1*=$\lambda t.\ -\ 2 \cdot (x\ r\ \$\ 0) \cdot (t \cdot x\ r\ \$\ 1)$
 and *g'1*=$\lambda t.\ 2 \cdot (x\ r\ \$\ 1) \cdot t \cdot x\ r\ \$\ 0$ **in** *derivative-eq-intros*(*8*), *simp-all*)
 apply(*rule-tac f'1*=$\lambda t.\ -\ (t \cdot x\ r\ \$\ 1)$ **in** *derivative-eq-intros*(*15*))
 apply(*rule-tac t*=$\{0--\tau\}$ **and** *s*=$\{0..t\}$ **in** *has-derivative-within-subset*)
 apply(*simp*, *simp add*: *closed-segment-eq-real-ivl*, *force*)
 apply(*rule-tac f'1*=$\lambda t.\ (t \cdot x\ r\ \$\ 0)$ **in** *derivative-eq-intros*(*15*))
 apply(*rule-tac t*=$\{0--\tau\}$ **and** *s*=$\{0..t\}$ **in** *has-derivative-within-subset*)
 by(*simp*, *simp add*: *closed-segment-eq-real-ivl*, *force*)
 by(*auto simp*: *closed-segment-eq-real-ivl*)

lemma *circular-motion-invariants*:
 assumes (R::*real*) > 0
 shows $\lceil\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\rceil \leq wp\ ([x'{=}(*v)\ C]\{0..t\}\ \&\ G)\ \lceil\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\rceil$
 using *assms*(*1*) **apply**(*rule-tac* C=$\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2$ **in** *dCut*)

apply(*rule-tac* $I{=}\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2$ **in** *dI*)
using *circle-invariant* ‹$R > 0$› **apply**(*blast*, *force*, *force*)
by(*rule dWeakening*, *auto*)

— Proof of the same specification by providing solutions:

lemma *entries-circ-matrix*:*entries* $C = \{0,\ -\ 1,\ 1\}$
apply (*simp-all add*: *axis-def*, *safe*)
subgoal by(*rule-tac* $x{=}0$ **in** *exI*, *simp*)+
subgoal by(*rule-tac* $x{=}0$ **in** *exI*, *simp*)+
by(*rule-tac* $x{=}1$ **in** *exI*, *simp*)+

lemma *picard-lindeloef-circ-matrix*:
assumes $0 \leq t$ **and** $t < 1/4$
shows *picard-lindeloef* $(\lambda t.\ (*v)\ C)\ \{0..t\}\ ((real\ CARD(2))^2 \cdot (\|C\|_{max}))\ 0$
apply(*rule picard-lindeloef-linear-system*)
unfolding *entries-circ-matrix* **using** *assms* **by** *auto*

abbreviation *circular-motion-matrix-flow* $t\ s \equiv (\chi\ i.\ if\ i{=}\ (0{::}2)\ then$
$s\$0 \cdot cos\ t - s\$1 \cdot sin\ t\ else\ s\$0 \cdot sin\ t + s\$1 \cdot cos\ t)$

notation *circular-motion-matrix-flow* (φ_C)

lemma *local-flow-circ-mtx*:
assumes $0 \leq t$ **and** $t < 1/4$
shows *local-flow* $((*v)\ C)\ \{0..t\}\ ((real\ CARD(2))^2 \cdot (\|C\|_{max}))\ \varphi_C$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-circ-matrix assms* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*)
apply(*simp add*: *matrix-vector-mult-def has-vderiv-on-def has-vector-derivative-def*, *safe*)
subgoal for $s\ i\ x$
apply(*rule-tac* $f'1{=}\lambda t.\ -\ s\$0 \cdot (t \cdot sin\ x)$ **and** $g'1{=}\lambda t.\ s\$1 \cdot (t \cdot cos\ x)$**in** *derivative-eq-intros*(*11*))
apply(*rule derivative-eq-intros*(*6*)[*of cos* $(\lambda xa.\ -\ (xa \cdot sin\ x))$])
apply(*rule-tac* $Db1{=}1$ **in** *derivative-eq-intros*(*58*))
apply(*rule ssubst*[*of* $(\cdot)$ *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of sin* $(\lambda xa.\ (xa \cdot cos\ x))$])
apply(*rule-tac* $Db1{=}1$ **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* $(\cdot)$ *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
subgoal for $s\ i\ x$
apply(*rule-tac* $f'1{=}\lambda t.\ s\$0 \cdot (t \cdot cos\ x)$ **and** $g'1{=}\lambda t.\ -\ s\$1 \cdot (t \cdot sin\ x)$**in** *derivative-eq-intros*(*8*))
apply(*rule derivative-eq-intros*(*6*)[*of sin* $(\lambda xa.\ xa \cdot cos\ x)$])
apply(*rule-tac* $Db1{=}1$ **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* $(\cdot)$ *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of cos* $(\lambda xa.\ -\ (xa \cdot sin\ x))$])
apply(*rule-tac* $Db1{=}1$ **in** *derivative-eq-intros*(*58*))

apply(*rule ssubst*[*of* (·) *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
using *exhaust-2 two-eq-zero* **by**(*force simp*: *vec-eq-iff*)

lemma *flow-for-Circ-DS*:
assumes $0 \leq t$ **and** $t < 1/4$
shows *wp* ([$x' = (*v)$ C]{$0..t$} & G) $\lceil Q \rceil$ =
$\lceil \lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (\forall r \in \{0--\tau\}.\ G\ (\varphi_C\ r\ x)) \longrightarrow Q\ (\varphi_C\ \tau\ x) \rceil$
apply(*subst local-flow.wp-g-orbit*[*of* $(*v)$ C - $((real\ CARD(2))^2 \cdot (\|C\|_{max}))$ φ_C])
using *local-flow-circ-mtx* **and** *assms* **by** *auto*

lemma *circular-motion*:
assumes $0 \leq t$ **and** $t < 1/4$ **and** $R > 0$
shows $\lceil \lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2 \rceil \leq$ *wp* ([$x' = (*v)$ C]{$0..t$} & G) $\lceil \lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2 \rceil$
apply(*subst flow-for-Circ-DS*)
using *assms* **by** *simp-all*

no-notation *circular-motion-matrix* (C)

no-notation *circular-motion-matrix-flow* (φ_C)

Bouncing Ball with solution

We revisit the previous dynamics for a constantly accelerated object modelled with the matrix K. We compose the corresponding evolution command with an if-statement, and iterate this hybrid program to model a (completely elastic) "bouncing ball". Using the previously defined flow for this dynamics, proving a specification for this hybrid program is merely an exercise of real arithmetic.

named-theorems *bb-real-arith real arithmetic properties for the bouncing ball.*

lemma [*bb-real-arith*]:$0 \leq x \Longrightarrow 0 > g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v \Longrightarrow (x::real) \leq H$
proof−
assume $0 \leq x$ **and** $0 > g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
then have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot H \wedge 0 > g$ **by** *auto*
hence $*$:$v \cdot v = 2 \cdot g \cdot (x - H) \wedge 0 > g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
from *this* **have** $(v \cdot v)/(2 \cdot g) = (x - H)$ **by** *auto*
also from $*$ **have** $(v \cdot v)/(2 \cdot g) \leq 0$
using *divide-nonneg-neg* **by** *fastforce*
ultimately have $H - x \geq 0$ **by** *linarith*
thus *?thesis* **by** *auto*
qed

lemma [*bb-real-arith*]:

assumes *invar*:$2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
and *pos*:$g \cdot \tau^2 / 2 + v \cdot \tau + (x::real) = 0$
shows $2 \cdot g \cdot H + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
and $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
proof−
from *pos* **have** $g \cdot \tau^2 + 2 \cdot v \cdot \tau + 2 \cdot x = 0$ **by** *auto*
then have $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$
by (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(*1*,*3*) *mult-zero-right*
monoid-mult-class.power2-eq-square semiring-class.distrib-left)
hence $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot H = 0$
using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
from *this* **have** $*$:$(g \cdot \tau + v)^2 + 2 \cdot g \cdot H = 0$
apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

Groups.mult-ac(*2*, *3*) *monoid-mult-class.power2-eq-square nat-distrib*(*2*))
hence $2 \cdot g \cdot H + (- ((g \cdot \tau) + v))^2 = 0$
by (*metis Groups.add-ac*(*2*) *power2-minus*)
thus $2 \cdot g \cdot H + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
by (*simp add*: *monoid-mult-class.power2-eq-square*)
from $*$ **show** $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
by (*simp add*: *monoid-mult-class.power2-eq-square*)
qed

lemma [*bb-real-arith*]:
assumes *invar*:$2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
shows $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x::real)) =$
$2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**is** *?lhs* = *?rhs*)
proof−
have *?lhs* $= g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
apply(*subst Rat.sign-simps*(*18*))+
by(*auto simp*: *semiring-normalization-rules*(*29*))
also have ... $= g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot H + v \cdot v$ (**is** ... = *?middle*)
by(*subst invar*, *simp*)
finally have *?lhs* = *?middle***.**
moreover
{have *?rhs* $= g \cdot g \cdot (\tau \cdot \tau) + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot H + v \cdot v$
by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
also have ... = *?middle*
by (*simp add*: *semiring-normalization-rules*(*29*))
finally have *?rhs* = *?middle***.}**
ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball*:
assumes $0 \leq t$ **and** $t < 1/9$
shows $\lceil \lambda s.\ (0::real) \leq s \$ (0::3) \wedge s \$ 0 = H \wedge s \$ 1 = 0 \wedge 0 > s \$ 2 \rceil \subseteq wp$
$((([x{\acute{}}{=}\lambda s.\ K *v\ s]\{0..t\}\ \&\ (\lambda\ s.\ s \$ 0 \geq 0));$
$(IF\ (\lambda\ s.\ s \$ 0 = 0)\ THEN\ ([1 ::== (\lambda s.\ - s \$ 1)])\ ELSE\ Id\ FI))^*)$
$\lceil \lambda s.\ 0 \leq s \$ 0 \wedge s \$ 0 \leq H \rceil$

apply(*rule rel-ad-mka-starI* [*of* - $\lceil\lambda s.\ 0 \le s\ \$\ (0::3) \wedge 0 > s\ \$\ 2 \wedge$
$2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 = 2 \cdot s\ \$\ 2 \cdot H + (s\ \$\ 1 \cdot s\ \$\ 1)\rceil$]])
apply(*simp*, *simp only*: *rel-antidomain-kleene-algebra.fbox-seq*)
apply(*subst p2r-r2p-wp-sym*[*of* (*IF* ($\lambda s.\ s\ \$\ 0 = 0$) *THEN* ([*1* ::== ($\lambda s.\ - s\ \$\ 1$)]) *ELSE Id FI*)])
apply(*subst wp-cnst-acc-matrix*) **using** *assms* **apply**(*simp*, *simp*) **apply**(*subst wp-trafo*)
unfolding *rel-antidomain-kleene-algebra.cond-def rel-antidomain-kleene-algebra.ads-d-def*

by(*auto simp*: *p2r-def rel-ad-def bb-real-arith*)

Bouncing Ball with invariants

We prove again the bouncing ball but this time with differential invariants.

lemma *gravity-invariant*: ($\lambda s.\ s\ \$\ 2 < 0$) *is-diff-invariant-of* ($*v$) *K along* $\{0..t\}$
apply(*rule-tac* $\vartheta'=\lambda s.\ 0$ **and** $\nu'=\lambda s.\ 0$ **in** *ode-invariant-rules*(*3*), *clarsimp*)
apply(*drule-tac* $i=2$ **in** *has-vderiv-on-vec-nth*)
apply(*unfold has-vderiv-on-def has-vector-derivative-def*)
apply(*erule-tac* $x=r$ **in** *ballE*, *simp add*: *matrix-vector-mult-def*)
apply(*rule-tac* $f'1=\lambda s.\ 0$ **in** *derivative-eq-intros*(*10*))
by(*auto simp*: *closed-segment-eq-real-ivl has-derivative-within-subset*)

lemma *energy-conservation-invariant*:
($\lambda s.\ 2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 - 2 \cdot s\ \$\ 2 \cdot H - s\ \$\ 1 \cdot s\ \$\ 1 = 0$) *is-diff-invariant-of* ($*v$) *K along* $\{0..t\}$
apply(*rule ode-invariant-rules*, *clarify*)
apply(*frule-tac* $i=2$ **in** *has-vderiv-on-vec-nth*)
apply(*frule-tac* $i=1$ **in** *has-vderiv-on-vec-nth*)
apply(*drule-tac* $i=0$ **in** *has-vderiv-on-vec-nth*)
apply(*unfold has-vderiv-on-def has-vector-derivative-def*)
apply(*erule-tac* $x=r$ **in** *ballE*, *simp-all add*: *matrix-vector-mult-def*)+
apply(*rule-tac* $f'1=\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$
and $g'1=\lambda t.\ 2 \cdot (t \cdot (x\ r\ \$\ 1 \cdot x\ r\ \$\ 2))$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac* $f'1=\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$ **and** $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac* $f'1=\lambda t.\ 0$ **and** $g'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 1)$ **in** *derivative-eq-intros*(*12*))
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*))
apply(*simp-all add*: *has-derivative-within-subset closed-segment-eq-real-ivl*)
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*7*))
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*))
apply(*simp-all add*: *has-derivative-within-subset*)
apply(*rule-tac* $f'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **and** $g'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **in** *derivative-eq-intros*(*12*))
by(*simp-all add*: *has-derivative-within-subset*)

lemma *bouncing-ball-invariants*:
$\lceil\lambda s.\ (0::real) \le s\ \$\ (0::3) \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\rceil \subseteq wp$
((([$x' = \lambda s.\ K *v\ s$]$\{0..t\}$ & ($\lambda\ s.\ s\ \$\ 0 \ge 0$));
(*IF* ($\lambda\ s.\ s\ \$\ 0 = 0$) *THEN* ([*1* ::== ($\lambda s.\ - s\ \$\ 1$)]) *ELSE Id FI*))*)

$\lceil\lambda s.\ 0 \le s\ \$\ 0 \wedge s\ \$\ 0 \le H\rceil$
apply(*rule-tac* $I{=}\lceil\lambda s.\ 0 \le s\$0 \wedge 0 > s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot H + (s\$1 \cdot s\$1)\rceil$ **in** *rel-ad-mka-starI*)
apply(*simp*, *simp only*: *rel-antidomain-kleene-algebra.fbox-seq*)
apply(*subst p2r-r2p-wp-sym*[*of* (*IF* ($\lambda s.\ s\ \$\ 0 = 0$) *THEN* ([*1* ::== ($\lambda s.\ -\ s\ \$\ 1$)]) *ELSE Id FI*)])
apply(*rule dCut*[**where** $C{=}\lambda\ s.\ s\ \$\ 2 < 0$])
apply(*rule-tac* $I{=}\lambda\ s.\ s\ \$\ 2 < 0$ **in** *dI*)
using *gravity-invariant* **apply**(*blast*, *force simp*: *p2r-def*, *force simp*: *p2r-def*)
apply(*rule-tac* $C{=}\lambda\ s.\ 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1 = 0$ **in** *dCut*)
apply(*rule-tac* $I{=}\lambda\ s.\ 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1 = 0$ **in** *dI*)
using *energy-conservation-invariant* **apply**(*blast*, *force simp*: *p2r-def*, *force simp*: *p2r-def*)
apply(*rule dWeakening*, *subst p2r-r2p-wp*)
apply(*simp add*: *rel-antidomain-kleene-algebra.fbox-def*)
unfolding *rel-antidomain-kleene-algebra.cond-def p2r-def*
by(*auto simp*: *bb-real-arith rel-ad-def rel-antidomain-kleene-algebra.ads-d-def*)

end
theory *cat2ndfun*
imports *../hs-prelims-matrices Transformer-Semantics.Kleisli-Quantale KAD.Modal-Kleene-Algebra*

begin

Chapter 5

Hybrid System Verification with nondeterministic functions

— We start by deleting some conflicting notation and introducing some new.
no-notation *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
  and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor$-$\rfloor$)
  and *Range-Semiring.antirange-semiring-class.ars-r* (r)
  and *Isotone-Transformers.bqtran* ($\lfloor$-$\rfloor$)

type-synonym $'a$ *pred* $=$ $'a \Rightarrow$ *bool*

notation *Abs-nd-fun* ($-^{\bullet}$ [*101*] *100*) **and** *Rep-nd-fun* ($-_{\bullet}$ [*101*] *100*)

5.1 Nondeterministic Functions

Our semantics correspond now to nondeterministic functions $'a$ *nd-fun*. Below we prove some auxiliary lemmas for them and show that they form an antidomain kleene algebra. The proof just extends the results on the Transformer_Semantics.Kleisli_Quantale theory.

— Analog of already existing $(x_{\bullet})^{\bullet} = x$.
lemma *Abs-nd-fun-inverse2*[*simp*]:$(f^{\bullet})_{\bullet} = f$
 by(*simp add*: *Abs-nd-fun-inverse*)

— Analog of already existing $(x_{\bullet})^{\bullet} = x$.
lemma *nd-fun-ext*:$(\bigwedge x.\ (f_{\bullet})\ x = (g_{\bullet})\ x) \Longrightarrow f = g$
 apply(*subgoal-tac Rep-nd-fun f* = *Rep-nd-fun g*)
 using *Rep-nd-fun-inject* **apply** *blast*
 by(*rule ext*, *simp*)

instantiation *nd-fun* :: (*type*) *antidomain-kleene-algebra*
begin

lift-definition *antidomain-op-nd-fun* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun*
is $\lambda f.\ (\lambda x.\ \textit{if}\ \ ((f_\bullet)\ x = \{\})\ \textit{then}\ \{x\}\ \textit{else}\ \{\})^\bullet$.
lift-definition *zero-nd-fun* :: $'a$ *nd-fun*
is $\zeta^\bullet$.
lift-definition *star-nd-fun* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun*
is $\lambda(f{::}'a\ \textit{nd-fun}).\textit{qstar}\ f$.
lift-definition *plus-nd-fun* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun*
is $\lambda f\ g.((f_\bullet) \sqcup (g_\bullet))^\bullet$.

named-theorems *nd-fun-aka antidomain kleene algebra properties for nondeterministic functions*.

lemma *nd-fun-assoc*[*nd-fun-aka*]:$(a{::}'a\ \textit{nd-fun}) + b + c = a + (b + c)$
by(*transfer*, *simp add*: *ksup-assoc*)

lemma *nd-fun-comm*[*nd-fun-aka*]:$(a{::}'a\ \textit{nd-fun}) + b = b + a$
by(*transfer*, *simp add*: *ksup-comm*)

lemma *nd-fun-distr*[*nd-fun-aka*]:$((x{::}'a\ \textit{nd-fun}) + y) \cdot z = x \cdot z + y \cdot z$
and *nd-fun-distl*[*nd-fun-aka*]:$x \cdot (y + z) = x \cdot y + x \cdot z$
by(*transfer*, *simp add*: *kcomp-distr*, *transfer*, *simp add*: *kcomp-distl*)

lemma *nd-fun-zero-sum*[*nd-fun-aka*]:$\mathit{0} + (x{::}'a\ \textit{nd-fun}) = x$
and *nd-fun-zero-dot*[*nd-fun-aka*]:$\mathit{0} \cdot x = \mathit{0}$
by(*transfer*, *simp*, *transfer*, *auto*)

lemma *nd-fun-leq*[*nd-fun-aka*]:$((x{::}'a\ \textit{nd-fun}) \leq y) = (x + y = y)$
and *nd-fun-leq-add*[*nd-fun-aka*]: $z \cdot x \leq z \cdot (x + y)$
apply(*transfer*, *metis Abs-nd-fun-inverse2 Rep-nd-fun-inverse le-iff-sup*)
by(*transfer*, *simp add*: *kcomp-isol*)

lemma *nd-fun-ad-zero*[*nd-fun-aka*]: $\textit{ad}\ (x{::}'a\ \textit{nd-fun}) \cdot x = \mathit{0}$
and *nd-fun-ad*[*nd-fun-aka*]: $\textit{ad}\ (x \cdot y) + \textit{ad}\ (x \cdot \textit{ad}\ (\textit{ad}\ y)) = \textit{ad}\ (x \cdot \textit{ad}\ (\textit{ad}\ y))$
and *nd-fun-ad-one*[*nd-fun-aka*]: $\textit{ad}\ (\textit{ad}\ x) + \textit{ad}\ x = \mathit{1}$
apply(*transfer*, *rule nd-fun-ext*, *simp add*: *kcomp-def*)
apply(*transfer*, *rule nd-fun-ext*, *simp*, *simp add*: *kcomp-def*)
by(*transfer*, *simp*, *rule nd-fun-ext*, *simp add*: *kcomp-def*)

lemma *nd-star-one*[*nd-fun-aka*]:$\mathit{1} + (x{::}'a\ \textit{nd-fun}) \cdot x^\star \leq x^\star$
and *nd-star-unfoldl*[*nd-fun-aka*]:$z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
and *nd-star-unfoldr*[*nd-fun-aka*]:$z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
apply(*transfer*, *metis Abs-nd-fun-inverse Rep-comp-hom UNIV-I fun-star-unfoldr*

le-sup-iff less-eq-nd-fun.abs-eq mem-Collect-eq one-nd-fun.abs-eq qstar-comm)
apply(*transfer*, *metis* (*no-types*, *lifting*) *Abs-comp-hom Rep-nd-fun-inverse fun-star-inductl less-eq-nd-fun.transfer sup-nd-fun.transfer*)
by(*transfer*, *metis qstar-inductr Rep-comp-hom Rep-nd-fun-inverse less-eq-nd-fun.abs-eq sup-nd-fun.transfer*)

instance
  apply *intro-classes* **apply** *auto*
  using *nd-fun-aka* **apply** *simp-all*
  by(*transfer*; *auto*)+
end

Now that we know that nondeterministic functions form an Antidomain Kleene Algebra, we give a lifting operation from predicates to *'a nd-fun* and prove some useful results for them. Then we add an operation that does the opposite and prove the relationship between both of these.

abbreviation *p2ndf* :: *'a pred* $\Rightarrow$ *'a nd-fun* ((*1*$\lceil$-$\rceil$))
  where $\lceil Q \rceil \equiv (\lambda\ x{::}'a.\ \{s{::}'a.\ s = x \wedge Q\ s\})^{\bullet}$

lemma *le-nd-fun-def*:$F^{\bullet} \leq G^{\bullet} = (\forall s.\ F\ s \subseteq G\ s)$
  by(*transfer*, *auto simp*: *le-fun-def*)

lemma *le-p2ndf-iff* [*simp*]:$\lceil P \rceil \leq \lceil Q \rceil = (\forall s.\ P\ s \longrightarrow Q\ s)$
  by(*transfer*, *auto simp*: *le-fun-def*)

lemma *eq-p2ndf-iff*:$(\lceil P \rceil = \lceil Q \rceil) = (P = Q)$
proof(*safe*)
  assume $\lceil P \rceil = \lceil Q \rceil$
  hence $\lceil P \rceil \leq \lceil Q \rceil \wedge \lceil Q \rceil \leq \lceil P \rceil$ **by** *simp*
  then have $(\forall s.\ P\ s \longrightarrow Q\ s) \wedge (\forall s.\ Q\ s \longrightarrow P\ s)$ **by** *simp*
  thus $P = Q$ **by** *auto*
qed

lemma *p2ndf-le-eta*[*simp*]:$\lceil P \rceil \leq \eta^{\bullet}$
  by(*transfer*, *simp add*: *le-fun-def*, *clarify*)

lemma *ads-d-p2ndf* [*simp*]:$d\ \lceil P \rceil = \lceil P \rceil$
  unfolding *ads-d-def antidomain-op-nd-fun-def* **by**(*rule nd-fun-ext*, *auto*)

lemma *ad-p2ndf* [*simp*]:$ad\ \lceil P \rceil = \lceil \lambda s.\ \neg\ P\ s \rceil$
  unfolding *antidomain-op-nd-fun-def* **by**(*rule nd-fun-ext*, *auto*)

abbreviation *ndf2p* :: *'a nd-fun* $\Rightarrow$ *'a* $\Rightarrow$ *bool* ((*1*$\lfloor$-$\rfloor$))
  where $\lfloor f \rfloor \equiv (\lambda x.\ x \in Domain\ (\mathcal{R}\ (f_{\bullet})))$

lemma *p2ndf-ndf2p-id*:$F \leq \eta^{\bullet} \Longrightarrow \lceil \lfloor F \rfloor \rceil = F$
  unfolding *f2r-def* **apply**(*rule nd-fun-ext*)
  apply(*subgoal-tac* $\forall x.\ (F_{\bullet})\ x \subseteq \{x\}$, *simp*)
  by(*blast*, *simp add*: *le-fun-def less-eq-nd-fun.rep-eq*)

lemma *ndf2p-p2ndf-id*:$\lfloor \lceil P \rceil \rfloor = P$
  by(*simp add*: *f2r-def*)

5.2 Verification of regular programs

As expected, the weakest precondition is just the forward box operator from the KAD. Below we explore its behavior with the previously defined lifting ($\lceil - \rceil^*$) and dropping ($\lfloor - \rfloor^*$) operators

abbreviation *wp f* $\equiv$ *fbox* (*f*::*'a nd-fun*)

lemma *wp-eta*[*simp*]:*wp* ($\eta^\bullet$) $\lceil P \rceil = \lceil P \rceil$
 apply(*simp add*: *fbox-def*, *transfer*, *simp*)
 by(*rule nd-fun-ext*, *auto simp*: *kcomp-def*)

lemma *wp-nd-fun*:*wp* ($F^\bullet$) $\lceil P \rceil = \lceil \lambda\ x.\ \forall\ \ y.\ y \in (F\ x) \longrightarrow P\ y \rceil$
 apply(*simp add*: *fbox-def*, *transfer*, *simp*)
 by(*rule nd-fun-ext*, *auto simp*: *kcomp-def*)

lemma *wp-nd-fun2*:*wp F* $\lceil P \rceil = \lceil \lambda\ x.\ \forall\ \ y.\ y \in ((F_\bullet)\ x) \longrightarrow P\ y \rceil$
 apply(*simp add*: *fbox-def antidomain-op-nd-fun-def*)
 by(*rule nd-fun-ext*, *auto simp*: *Rep-comp-hom kcomp-prop*)

lemma *wp-nd-fun-etaD*:*wp* ($F^\bullet$) $\lceil P \rceil = \eta^\bullet \Longrightarrow\ (\forall y.\ y \in (F\ x) \longrightarrow P\ y)$
proof
 fix *y* **assume** *wp* ($F^\bullet$) $\lceil P \rceil = (\eta^\bullet)$
 from *this* **have** $\eta^\bullet = \lceil \lambda s.\ \forall y.\ s2p\ (F\ s)\ y \longrightarrow P\ y \rceil$
  by(*subst wp-nd-fun*[*THEN sym*], *simp*)
 hence $\bigwedge x.\ \{x\} = \{s.\ s = x \wedge (\forall y.\ s2p\ (F\ s)\ y \longrightarrow P\ y)\}$
  apply(*subst* (*asm*) *Abs-nd-fun-inject*, *simp-all*)
  by(*drule-tac x=x* **in** *fun-cong*, *simp*)
 then show *s2p* (*F x*) *y* $\longrightarrow$ *P y* **by** *auto*
qed

lemma *p2ndf-ndf2p-wp*:$\lceil \lfloor wp\ R\ P \rfloor \rceil = wp\ R\ P$
 apply(*rule p2ndf-ndf2p-id*)
 by (*simp add*: *a-subid fbox-def one-nd-fun.transfer*)

lemma *p2ndf-ndf2p-wp-sym*:*wp R P* $= \lceil \lfloor wp\ R\ P \rfloor \rceil$
 by(*rule sym*, *simp add*: *p2ndf-ndf2p-wp*)

lemma *ndf2p-wpD*: $\lfloor wp\ F\ \lceil Q \rceil \rfloor\ s = (\forall s'.\ s' \in (F_\bullet)\ s \longrightarrow Q\ s')$
 apply(*subgoal-tac* $F = (F_\bullet)^\bullet$)
 apply(*rule ssubst*[*of F* $(F_\bullet)^\bullet$], *simp*)
 apply(*subst wp-nd-fun*)
 by(*simp-all add*: *f2r-def*)

We can verify that our introduction of *wp* coincides with another definition of the forward box operator $fb_{\mathcal{F}} = \partial_F \circ bd_{\mathcal{F}} \circ op_K$ with the following characterization lemmas.

lemma *ffb-is-wp*:$fb_{\mathcal{F}}\ (F_\bullet)\ \{x.\ P\ x\} = \{s.\ \lfloor wp\ F\ \lceil P \rceil \rfloor\ s\}$
 unfolding *ffb-def* **unfolding** *map-dual-def klift-def kop-def fbox-def*

unfolding *r2f-def f2r-def* **apply** *clarsimp*
unfolding *antidomain-op-nd-fun-def* **unfolding** *dual-set-def*
unfolding *times-nd-fun-def kcomp-def* **by** *force*

lemma *wp-is-ffb*:$wp\ F\ P = (\lambda x.\ \{x\} \cap fb_{\mathcal{F}}\ (F_{\bullet})\ \{s.\ \lfloor P \rfloor\ s\})^{\bullet}$
apply(*rule nd-fun-ext*, *simp*)
unfolding *ffb-def* **unfolding** *map-dual-def klift-def kop-def fbox-def*
unfolding *r2f-def f2r-def* **apply** *clarsimp*
unfolding *antidomain-op-nd-fun-def* **unfolding** *dual-set-def*
unfolding *times-nd-fun-def* **apply** *auto*
unfolding *kcomp-prop* **by** *auto*

Next, we introduce assignments and compute their *wp*.

abbreviation *vec-upd* :: $('a\hat{\ }'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a\hat{\ }'b$ (-(*2*[- :== -]) [*70*, *65*] *61*)
where
$x[i :== a] \equiv (\chi\ j.\ (\mathit{if}\ j = i\ \mathit{then}\ a\ \mathit{else}\ (x\ \$\ j)))$

abbreviation *assign* :: $'b \Rightarrow ('a\hat{\ }'b \Rightarrow 'a) \Rightarrow ('a\hat{\ }'b)$ *nd-fun* ((*2*[- ::== -]) [*70*, *65*] *61*) **where**
$[x ::== expr] \equiv (\lambda s.\ \{s[x :== expr\ s]\})^{\bullet}$

lemma *wp-assign*[*simp*]: $wp\ ([x ::== expr])\ \lceil Q \rceil = \lceil \lambda s.\ Q\ (s[x :== expr\ s]) \rceil$
by(*subst wp-nd-fun*, *rule nd-fun-ext*, *simp*)

The *wp* of the composition was already obtained in KAD.Antidomain_Semiring:
$|x \cdot y]\ z = \ |x]\ \ |y]\ z.$

We also have an implementation of the conditional operator and its *wp*.

definition (**in** *antidomain-kleene-algebra*) *cond* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
(*if - then - else - fi* [*64*,*64*,*64*] *63*) **where** $\mathit{if}\ p\ \mathit{then}\ x\ \mathit{else}\ y\ \mathit{fi} = d\ p \cdot x + ad\ p \cdot y$

abbreviation *cond-sugar* :: $'a\ pred \Rightarrow 'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun \Rightarrow 'a\ nd\text{-}fun$
(*IF - THEN - ELSE - FI* [*64*,*64*,*64*] *63*) **where**
IF P THEN X ELSE Y FI $\equiv$ *cond* $\lceil P \rceil\ X\ Y$

lemma *wp-if-then-else*:
assumes $\lceil \lambda s.\ P\ s \wedge T\ s \rceil \le wp\ X\ \lceil Q \rceil$
and $\lceil \lambda s.\ P\ s \wedge \neg\ T\ s \rceil \le wp\ Y\ \lceil Q \rceil$
shows $\lceil P \rceil \le wp$ (*IF T THEN X ELSE Y FI*) $\lceil Q \rceil$
using *assms* **apply**(*subst wp-nd-fun2*)
apply(*subst* (*asm*) *wp-nd-fun2*)+
unfolding *cond-def* **apply**(*clarsimp*, *transfer*)
by(*auto simp*: *kcomp-prop*)

Finally we also deal with finite iteration.

lemma (**in** *antidomain-kleene-algebra*) *fbox-starI*:
assumes $d\ p \le d\ i$ **and** $d\ i \le |x]\ i$ **and** $d\ i \le d\ q$
shows $d\ p \le |x^{\star}]\ q$
by (*meson assms local.dual-order.trans local.fbox-iso local.fbox-star-induct-var*)

lemma *nd-fun-ads-d-def*:$d\ (f{::}'a\ nd\text{-}fun) = (\lambda x.\ if\ (f_\bullet)\ x = \{\}\ then\ \{\}\ else\ \eta\ x\ )^\bullet$
unfolding *ads-d-def* **apply**(*rule nd-fun-ext*, *simp*)
apply *transfer* **by** *auto*

lemma *ads-d-mono*: $x \le y \Longrightarrow d\ x \le d\ y$
by (*metis ads-d-def fbox-antitone-var fbox-dom*)

lemma *nd-fun-top-ads-d*:$(x{::}'a\ nd\text{-}fun) \le 1 \Longrightarrow d\ x = x$
apply(*simp add*: *ads-d-def*, *transfer*, *simp*)
apply(*rule nd-fun-ext*, *simp*)
apply(*subst* (*asm*) *le-fun-def*)
by *auto*

lemma *wp-starI*:
assumes $P \le I$ **and** $I \le wp\ F\ I$ **and** $I \le Q$
shows $P \le wp\ (qstar\ F)\ Q$
proof−
from *assms*(*1*,*2*) **have** $P \le 1$
by (*metis a-subid basic-trans-rules*(*23*) *fbox-def*)
hence $d\ P = P$ **using** *nd-fun-top-ads-d* **by** *blast*
have $\bigwedge x\ y.\ d\ (wp\ x\ y) = wp\ x\ y$
by(*metis ds.ddual.mult-oner fbox-mult fbox-one*)
from *this* **and** *assms* **have** $d\ P \le d\ I \wedge d\ I \le wp\ F\ I \wedge d\ I \le d\ Q$
by (*metis* (*no-types*) *ads-d-mono assms*)
hence $d\ P \le wp\ (F^\star)\ Q$
by(*simp add*: *fbox-starI*[*of - I*])
then show $P \le wp\ (qstar\ F)\ Q$
using ‹$d\ P = P$› **by** (*transfer*, *simp*)
qed

5.3 Verification of hybrid programs

5.3.1 Verification by providing solutions

abbreviation *guards* :: $('a \Rightarrow bool) \Rightarrow (real \Rightarrow 'a) \Rightarrow (real\ set) \Rightarrow bool$ (- ⊳ - - [*70*, *65*] *61*)
where $G \rhd x\ T \equiv \forall\ r \in T.\ G\ (x\ r)$

definition *ivp-sols* $f\ T\ t_0\ s = \{x\ |x.\ (D\ x = (f \circ x)\ on\ T) \wedge x\ t_0 = s \wedge t_0 \in T\}$

lemma *ivp-solsI*:
assumes $D\ x = (f \circ x)\ on\ T\ x\ t_0 = s\ t_0 \in T$
shows $x \in ivp\text{-}sols\ f\ T\ t_0\ s$
using *assms* **unfolding** *ivp-sols-def* **by** *blast*

lemma *ivp-solsD*:
assumes $x \in ivp\text{-}sols\ f\ T\ t_0\ s$

shows $D\ x = (f \circ x)$ *on* T
and $x\ t_0 = s$ **and** $t_0 \in T$
using *assms* **unfolding** *ivp-sols-def* **by** *auto*

lemma $(t::real) \in \{0--t\}$
by (*rule ends-in-segment*(*2*))

lemma $(t::real) \in \{0..t\}$
apply *auto*
oops

definition *g-orbital* $f\ T\ t_0\ G\ s = \bigcup\ \{\{x\ t|t.\ t \in T \wedge G \rhd x\ \{t_0--t\}\ \}|x.\ x \in$ *ivp-sols* $f\ T\ t_0\ s\}$

lemma *g-orbital-eq*: *g-orbital* $f\ T\ t_0\ G\ s =$
$\{x\ t\ |t\ x.\ t \in T \wedge (D\ x = (f \circ x)\ on\ T) \wedge x\ t_0 = s \wedge t_0 \in T \wedge G \rhd x\ \{t_0--t\}\}$
unfolding *g-orbital-def ivp-sols-def* **by** *auto*

lemma *g-orbital* $f\ T\ t_0\ G\ s = (\bigcup\ x \in$ *ivp-sols* $f\ T\ t_0\ s.\ \{x\ t|t.\ t \in T \wedge G \rhd x\ \{t_0--t\}\})$
unfolding *g-orbital-def ivp-sols-def* **by** *auto*

lemma *g-orbitalI*:
assumes $D\ x = (f \circ x)$ *on* $T\ x\ t_0 = s$
and $t_0 \in T\ t \in T$ **and** $G \rhd x\ \{t_0--t\}$
shows $x\ t \in$ *g-orbital* $f\ T\ t_0\ G\ s$
using *assms* **unfolding** *g-orbital-def ivp-sols-def* **by** *blast*

lemma *g-orbitalD*:
assumes $s' \in$ *g-orbital* $f\ T\ t_0\ G\ s$
obtains x **and** t **where** $x \in$ *ivp-sols* $f\ T\ t_0\ s$
and $D\ x = (f \circ x)$ *on* $T\ x\ t_0 = s$
and $x\ t = s'$ **and** $t_0 \in T\ t \in T$ **and** $G \rhd x\ \{t_0--t\}$
using *assms* **unfolding** *g-orbital-def ivp-sols-def* **by** *blast*

abbreviation *g-evol* ::(('a::banach)⇒'a) ⇒ *real set* ⇒ *'a pred* ⇒ *'a nd-fun* ((*1*[x´=-]- & -))
where [x´=f]T & G ≡ (λ s. *g-orbital* f T *0* G s)$^\bullet$

lemmas *g-evol-def* = *g-orbital-eq*[**where** t_0=*0*]

context *local-flow*
begin

lemma *in-ivp-sols*: $(\lambda t.\ \varphi\ t\ s) \in$ *ivp-sols* $f\ T\ 0\ s$
by(*auto intro*: *ivp-solsI simp*: *ivp init-time*)

definition *orbit* $s =$ *g-orbital* $f\ T\ 0\ (\lambda s.\ True)\ s$

lemma *orbit-eq*[*simp*]: *orbit* $s = \{\varphi\ t\ s |\ t.\ t \in T\}$
  unfolding *orbit-def g-evol-def*
  by(*auto intro*: *usolves-ivp intro*!: *ivp simp*: *init-time*)

lemma *g-evol-collapses*:
  shows $([x\acute{}=f]T\ \&\ G) = (\lambda s.\ \{\varphi\ t\ s |\ t.\ t \in T \wedge G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}\})^\bullet$
proof(*rule nd-fun-ext*, *rule subset-antisym*, *simp-all add*: *subset-eq*)
  fix s
  let $?P\ s\ s' = \exists t.\ s' = \varphi\ t\ s \wedge s2p\ T\ t \wedge (\forall r {\in} \{0{-}{-}t\}.\ G\ (\varphi\ r\ s))$
  {fix s' **assume** $s' \in$ *g-orbital* $f\ T\ 0\ G\ s$
    then obtain x **and** t **where** *x-ivp*:$D\ x = (f \circ x)$ *on* T
      $x\ 0 = s$ **and** $x\ t = s'$ **and** $t \in T$ **and** *guard*:$G \rhd x\ \{0{-}{-}t\}$
      unfolding *g-orbital-eq* **by** *blast*
    hence *obs*:$\forall \tau {\in} \{0{-}{-}t\}.\ x\ \tau = \varphi\ \tau\ s$
      using *usolves-ivp*[*of x s*] *closed-segment-subset-domainI init-time comp-def*
      by (*metis* (*mono-tags*, *lifting*) *has-vderiv-eq*)
    hence $G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}$
      using *guard* **by** *simp*
    hence $\exists t.\ s' = \varphi\ t\ s \wedge s2p\ T\ t \wedge (\forall r {\in} \{0{-}{-}t\}.\ G\ (\varphi\ r\ s))$
      using $\langle x\ t = s' \rangle\ \langle t \in T \rangle$ *obs* **by** *blast***}**
  thus $\forall s' {\in}$*g-orbital* $f\ T\ 0\ G\ s.\ ?P\ s\ s'$
    by *blast*
  {fix s' **assume** $\exists t.\ s' = \varphi\ t\ s \wedge s2p\ T\ t \wedge (\forall r {\in} \{0{-}{-}t\}.\ G\ (\varphi\ r\ s))$
    then obtain t **where** $G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}$ **and** $t \in T$ **and** $\varphi\ t\ s = s'$
      by *blast*
    hence $s' \in$ *g-orbital* $f\ T\ 0\ G\ s$
      by(*auto intro*: *g-orbitalI simp*: *ivp init-time*)**}**
  thus $\forall s'.\ ?P\ s\ s' \longrightarrow s' \in$ (*g-orbital* $f\ T\ 0\ G\ s$)
    by *blast*
qed

lemma *wp-orbit*: $wp\ ((\lambda\ s.\ orbit\ s)^\bullet)\ \lceil Q \rceil = \lceil \lambda\ s.\ \forall\ t \in T.\ Q\ (\varphi\ t\ s) \rceil$
  unfolding *orbit-eq wp-nd-fun* **apply**(*rule nd-fun-ext*) **by** *auto*

lemma *wp-g-orbit*: $wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil Q \rceil = \lceil \lambda\ s.\ \forall t {\in} T.\ (G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0{-}{-}t\}) \longrightarrow Q\ (\varphi\ t\ s) \rceil$
  unfolding *g-evol-collapses wp-nd-fun* **apply**(*rule nd-fun-ext*) **by** *auto*

end

lemma (**in** *global-flow*) *ivp-sols-collapse*[*simp*]: *ivp-sols* f *UNIV* $0\ s = \{(\lambda t.\ \varphi\ t\ s)\}$
  by(*auto intro*: *usolves-ivp simp*: *ivp-sols-def ivp*)

The previous lemma allows us to compute wlps for known systems of ODEs. We can also implement a version of it as an inference rule. A simple computation of a wlp is shown immmediately after.

lemma *dSolution*:
  assumes *local-flow* $f\ T\ L\ \varphi$

and $\forall s.\ P\ s \longrightarrow (\forall\ t \in T.\ (G \rhd (\lambda r.\ \varphi\ r\ s)\ \{0..t\}) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\lceil P \rceil \leq wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil Q \rceil$
using *assms* **apply**(*subst local-flow.wp-g-orbit*, *auto*)
by (*simp add*: *Starlike.closed-segment-eq-real-ivl*)

lemma *line-DS*: $0 \leq t \Longrightarrow wp\ ([x\acute{}=\lambda s.\ c]\{0..t\}\ \&\ G)\ \lceil Q \rceil =$
$\lceil \lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (G \rhd (\lambda r.\ x + r *_R c)\ \{0..\tau\}) \longrightarrow Q\ (x + \tau *_R c) \rceil$
apply(*subst local-flow.wp-g-orbit*[*of* $\lambda s.\ c$ - $1/(t + 1)$ $(\lambda\ t\ x.\ x + t *_R c)$])
by(*auto simp*: *line-is-local-flow closed-segment-eq-real-ivl*)

5.3.2 Verification with differential invariants

We derive the domain specific rules of differential dynamic logic (dL). In each subsubsection, we first derive the dL axioms (named below with two capital letters and "D" being the first one). This is done mainly to prove that there are minimal requirements in Isabelle to get the dL calculus. Then we prove the inference rules which are used in verification proofs.

Differential Weakening

lemma *DW*: $wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil Q \rceil = wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil \lambda\ s.\ G\ s \longrightarrow Q\ s \rceil$
apply(*subst wp-nd-fun*)+
apply(*rule nd-fun-ext*)
by(*auto simp*: *g-orbital-eq*)

lemma *dWeakening*:
assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil Q \rceil$
using *assms* **apply**(*subst wp-nd-fun*)
by(*auto simp*: *le-fun-def g-orbital-eq*)

Differential Cut

lemma *wp-g-orbit-etaD*:
assumes $wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil C \rceil = \eta^\bullet$ **and** $\forall\ r{\in}\{0{-}{-}t\}.\ x\ r \in$ *g-orbital* $f\ T\ 0$ $G\ s$
shows $\forall r{\in}\{0{-}{-}t\}.\ C\ (x\ r)$
proof
fix r **assume** $r \in \{0{-}{-}t\}$
then have $x\ r \in$ *g-orbital* $f\ T\ 0\ G\ s$
using *assms*(*2*) **by** *blast*
also have $\forall y.\ y \in ($*g-orbital* $f\ T\ 0\ G\ s) \longrightarrow C\ y$
using *assms*(*1*) *wp-nd-fun-etaD* **by** *fastforce*
ultimately show $C\ (x\ r)$ **by** *blast*
qed

lemma *DC*:
assumes *interval* T **and** $wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil C \rceil = \eta^\bullet$
shows $wp\ ([x\acute{}=f]T\ \&\ G)\ \lceil Q \rceil = wp\ ([x\acute{}=f]T\ \&\ (\lambda s.\ G\ s \wedge C\ s))\ \lceil Q \rceil$

proof(*rule-tac f*=λ *x*. *wp x* $\lceil Q \rceil$ **in** *HOL.arg-cong*, *rule nd-fun-ext*, *rule subset-antisym*, *simp-all*)
  fix *s*
  {**fix** *s'* **assume** $s' \in$ *g-orbital f T 0 G s*
    then obtain *t*::*real* **and** *x* **where** *x-ivp*: $D\ x = (f \circ x)$ *on* $T\ x\ 0 = s$
      and *guard-x*:$G \rhd x\ \{0{-}{-}t\}$ **and** $s' = x\ t$ **and** $0 \in T\ t \in T$
      using *g-orbitalD*[*of s' f T 0 G s*] **by** (*metis* (*full-types*))
    from *guard-x* **have** $\forall\, r\in\{0{-}{-}t\}.\forall\ \tau\in\{0{-}{-}r\}.\ G\ (x\ \tau)$
      by (*metis contra-subsetD ends-in-segment*(*1*) *subset-segment*(*1*))
    also have $\forall\,\tau\in\{0{-}{-}t\}.\ \tau \in T$
      using *interval.closed-segment-subset-domain*[*OF assms*(*1*) ‹$0 \in T$› ‹$t \in T$›]
by *blast*
    ultimately have $\forall\,\tau\in\{0{-}{-}t\}.\ x\ \tau \in$ *g-orbital f T 0 G s*
      using *g-orbitalI*[*OF x-ivp* ‹$0 \in T$›] **by** *blast*
    hence $C \rhd x\ \{0{-}{-}t\}$
      using *wp-g-orbit-etaD assms*(*2*) **by** *blast*
    hence $s' \in$ *g-orbital f T 0* ($\lambda s.\ G\ s \wedge C\ s$) *s*
      using *g-orbitalI*[*OF x-ivp* ‹$0 \in T$› ‹$t \in T$›] *guard-x* ‹$s' = x\ t$› **by** *fastforce*}
  thus *g-orbital f T 0 G s* $\subseteq$ *g-orbital f T 0* ($\lambda s.\ G\ s \wedge C\ s$) *s*
    by *blast*
next show $\bigwedge s.$ *g-orbital f T 0* ($\lambda s.\ G\ s \wedge C\ s$) *s* $\subseteq$ *g-orbital f T 0 G s*
    by (*auto simp*: *g-evol-def*)
qed

lemma *dCut*:
  assumes *wp-C*:$\lceil P \rceil \le$ *wp* ([*x´=f*]{*0..t*} & *G*) $\lceil C \rceil$
    and *wp-Q*:$\lceil P \rceil \le$ *wp* ([*x´=f*]{*0..t*} & ($\lambda\ s.\ G\ s \wedge C\ s$)) $\lceil Q \rceil$
  shows $\lceil P \rceil \le$ *wp* ([*x´=f*]{*0..t*} & *G*) $\lceil Q \rceil$
proof(*simp add*: *wp-nd-fun g-orbital-eq*, *clarsimp*)
  fix τ::*real* **and** *x*::*real* $\Rightarrow$ *'a* **assume** $P\ (x\ 0)$ **and** $0 \le \tau$ **and** $\tau \le t$
    and *x-solves*:$D\ x = (\lambda t.\ f\ (x\ t))$ *on* $\{0..t\}$ **and** *guard-x*:($\forall\ r \in \{0{-}{-}\tau\}.\ G$ (*x*
r))
  hence $\forall\, r\in\{0{-}{-}\tau\}.\forall\,\tau\in\{0{-}{-}r\}.\ G\ (x\ \tau)$
    using *closed-segment-closed-segment-subset* **by** *blast*
  hence $\forall\, r\in\{0{-}{-}\tau\}.\ x\ r \in$ *g-orbital f* $\{0..t\}$ *0 G* ($x\ 0$)
    using *g-orbitalI x-solves* ‹$0 \le \tau$› ‹$\tau \le t$› *closed-segment-eq-real-ivl* **by** *fastforce*
  hence $\forall\, r\in\{0{-}{-}\tau\}.\ C\ (x\ r)$
    using *wp-C* ‹$P\ (x\ 0)$› **by**(*subst* (*asm*) *wp-nd-fun*, *auto*)
  hence $x\ \tau \in$ *g-orbital f* $\{0..t\}$ *0* ($\lambda s.\ G\ s \wedge C\ s$) ($x\ 0$)
    using *g-orbitalI x-solves guard-x* ‹$0 \le \tau$› ‹$\tau \le t$› **by** *fastforce*
  from *this* ‹$P\ (x\ 0)$› **and** *wp-Q* **show** $Q\ (x\ \tau)$
    by(*subst* (*asm*) *wp-nd-fun*, *auto simp*: *closed-segment-eq-real-ivl*)
qed

Differential Invariant

lemma *DI-sufficiency*:
  assumes $\forall\, s.\ \exists\, x.\ x \in$ *ivp-sols f T 0 s*
  shows *wp* ([*x´=f*]*T* & *G*) $\lceil Q \rceil \le$ *wp* $\lceil G \rceil$ $\lceil Q \rceil$

using *assms* **apply**(*subst wp-nd-fun*, *subst wp-nd-fun*, *clarsimp*)
apply(*rename-tac s*, *erule-tac x=s* **in** *allE*, *erule impE*)
apply(*simp add*: *g-evol-def ivp-sols-def*)
apply(*erule-tac x=s* **in** *allE*, *clarify*)
by(*rule-tac x=0* **in** *exI*, *rule-tac x=x* **in** *exI*, *auto*)

lemma (**in** *local-flow*) *DI-necessity*:
shows *wp* $\lceil G\rceil$ $\lceil Q\rceil$ $\leq$ *wp* ([*x´=f*]*T* & *G*) $\lceil Q\rceil$
unfolding *wp-g-orbit* **apply**(*subst wp-nd-fun*, *clarsimp*, *safe*)
apply(*erule-tac x=0* **in** *ballE*)
apply(*simp add*: *ivp*, *simp*)
oops

definition *diff-invariant* :: *'a pred* $\Rightarrow$ ((*'a*::*real-normed-vector*) $\Rightarrow$ *'a*) $\Rightarrow$ *real set* $\Rightarrow$ *bool*
((-)/ *is'-diff'-invariant'-of* (-)/ *along* (-) [*70*,*65*]*61*)
where *I is-diff-invariant-of f along T* $\equiv$
($\forall$ *s*. *I s* $\longrightarrow$ ($\forall$ *x*. *x* $\in$ *ivp-sols f T 0 s* $\longrightarrow$ ($\forall$ *t* $\in$ *T*. *I* (*x t*))))

lemma *invariant-to-set*:
shows (*I is-diff-invariant-of f along T*) $\longleftrightarrow$ ($\forall$ *s*. *I s* $\longrightarrow$ (*g-orbital f T 0* (λ*s*. *True*) *s*) $\subseteq$ {*s*. *I s*})
unfolding *diff-invariant-def ivp-sols-def g-orbital-eq* **apply** *safe*
apply(*erule-tac x=xa 0* **in** *allE*)
apply(*drule mp*, *simp-all*)
apply(*erule-tac x=xa 0* **in** *allE*)
apply(*drule mp*, *simp-all add*: *subset-eq*)
apply(*erule-tac x=xa t* **in** *allE*)
by(*drule mp*, *auto*)

lemma *dInvariant*:
assumes *I is-diff-invariant-of f along T*
shows $\lceil I\rceil$ $\leq$ *wp* ([*x´=f*]*T* & *G*) $\lceil I\rceil$
using *assms* **unfolding** *diff-invariant-def* **apply**(*subst wp-nd-fun*)
apply(*subst le-p2ndf-iff*, *clarify*)
apply(*erule-tac x=s* **in** *allE*)
unfolding *g-orbital-def* **by** *auto*

lemma *dI*:
assumes *I is-diff-invariant-of f along* {*0*..*t*}
and $\lceil P\rceil$ $\leq$ $\lceil I\rceil$ **and** $\lceil I\rceil$ $\leq$ $\lceil Q\rceil$
shows $\lceil P\rceil$ $\leq$ *wp* ([*x´=f*]{*0*..*t*} & *G*) $\lceil Q\rceil$
using *assms*(*1*) **apply**(*rule-tac C=I* **in** *dCut*)
apply(*drule-tac G=G* **in** *dInvariant*)
using *assms*(*2*) *dual-order.trans* **apply** *blast*
apply(*rule dWeakening*)
using *assms* **by** *auto*

Finally, we obtain some conditions to prove specific instances of differential invariants.

named-theorems *ode-invariant-rules compilation of rules for differential invariants.*

lemma [*ode-invariant-rules*]:
fixes ϑ::'*a*::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}) \longrightarrow (\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0{-}{-}\tau\}.$

$((\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)\ )\ has\text{-}derivative\ (\lambda\tau.\ \ \tau *_R 0))\ (at\ r\ within\ \{0{-}{-}\tau\}))$
shows $(\lambda s.\ \vartheta\ s = \nu\ s)$ *is-diff-invariant-of f along* $\{0..t\}$
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
  fix $x\ \tau$ **assume** *tHyp*:$0 \leq \tau\ \tau \leq t$
    and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}\ \vartheta\ (x\ 0) = \nu\ (x\ 0)$
  hence $\forall\ r \in \{0{-}{-}\tau\}.\ D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R 0)\ at\ r\ within\ \{0{-}{-}\tau\}$
    using *assms* **by** *auto*
  hence $\exists\, r{\in}\{0{-}{-}\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) = (\lambda\tau.\ \tau *_R 0)\ (\tau - 0)$
    by(*rule-tac closed-segment-mvt*, *auto simp*: *tHyp*)
  thus $\vartheta\ (x\ \tau) = \nu\ (x\ \tau)$ **by** (*simp add*: *x-ivp*(2))
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::'*a*::*banach* $\Rightarrow$ *real*
assumes $\forall\ x.\ (D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}) \longrightarrow (\forall\ \tau \in \{0..t\}.\ \forall\ r \in \{0{-}{-}\tau\}.$
$\vartheta'\ (x\ r) \geq \nu'\ (x\ r)$
$\wedge\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) - \nu'\ (x\ r)))\ at\ r\ within\ \{0{-}{-}\tau\}))$
shows $(\lambda s.\ \nu\ s \leq \vartheta\ s)$ *is-diff-invariant-of f along* $\{0..t\}$
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
  fix $x\ \tau$ **assume** *tHyp*:$0 \leq \tau\ \tau \leq t$
    and *x-ivp*:$D\ x = (\lambda\tau.\ f\ (x\ \tau))\ on\ \{0..t\}\ \nu\ (x\ 0) \leq \vartheta\ (x\ 0)$
  hence *primed*:$\forall\ r \in \{0{-}{-}\tau\}.\ (D\ (\lambda\tau.\ \vartheta\ (x\ \tau) - \nu\ (x\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) - \nu'\ (x\ r)))$
  $at\ r\ within\ \{0{-}{-}\tau\}) \wedge \nu'\ (x\ r) \leq \vartheta'\ (x\ r)$
    using *assms* **by** *auto*
  hence $\exists\, r{\in}\{0{-}{-}\tau\}.\ (\vartheta\ (x\ \tau) - \nu\ (x\ \tau)) - (\vartheta\ (x\ 0) - \nu\ (x\ 0)) =$
  $(\lambda\tau.\ \tau *_R (\vartheta'\ (x\ r) - \ \nu'\ (x\ r)))\ (\tau - 0)$
    by(*rule-tac closed-segment-mvt*, *auto simp*: ‹$0 \leq \tau$›)
  then obtain r **where** $r \in \{0{-}{-}\tau\}$
    and $\vartheta\ (x\ \tau) - \nu\ (x\ \tau) = (\tau - 0) *_R (\vartheta'\ (x\ r) - \ \nu'\ (x\ r)) + (\vartheta\ (x\ 0) - \nu\ (x\ 0))$
    by *force*
  also have ... ≥ 0
    using *tHyp*(1) *x-ivp*(2) *primed* **by** (*simp add*: *calculation*(1))
  ultimately show $\nu\ (x\ \tau) \leq \vartheta\ (x\ \tau)$
    by *simp*
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::'*a*::*banach* ⇒ *real*
assumes ∀ *x*. (*D x* = (λτ. *f* (*x* τ)) *on* {*0*..*t*}) ⟶ (∀ τ ∈ {*0*..*t*}. ∀ *r* ∈ {*0*−−τ}.
ϑ′ (*x r*) ≥ ν′ (*x r*)
∧ (*D* (λτ. ϑ (*x* τ) − ν (*x* τ)) ↦ (λτ. τ ∗$_R$ (ϑ′ (*x r*) − ν′ (*x r*))) *at r within*
{*0*−−τ}))
shows (λ*s*. ν *s* < ϑ *s*) *is-diff-invariant-of f along* {*0*..*t*}
proof(*simp add*: *diff-invariant-def ivp-sols-def*, *clarsimp*)
  fix *x* τ **assume** *tHyp*:*0* ≤ τ τ ≤ *t*
    and *x-ivp*:*D x* = (λτ. *f* (*x* τ)) *on* {*0*..*t*} ν (*x 0*) < ϑ (*x 0*)
  hence *primed*:∀ *r* ∈ {*0*−−τ}. ((λτ. ϑ (*x* τ) − ν (*x* τ)) *has-derivative*
(λτ. τ ∗$_R$ (ϑ′ (*x r*) − ν′ (*x r*)))) (*at r within* {*0*−−τ}) ∧ ϑ′ (*x r*) ≥ ν′ (*x r*)
    using *assms* **by** *auto*
  hence ∃ *r*∈{*0*−−τ}. (ϑ (*x* τ) − ν (*x* τ)) − (ϑ (*x 0*) − ν (*x 0*)) =
  (λτ. τ ∗$_R$ (ϑ′ (*x r*) − ν′ (*x r*))) (τ − *0*)
    by(*rule-tac closed-segment-mvt*, *auto simp*: ‹*0* ≤ τ›)
  then obtain *r* **where** *r* ∈ {*0*−−τ} **and**
    ϑ (*x* τ) − ν (*x* τ) = (τ − *0*) ∗$_R$ (ϑ′ (*x r*) − ν′ (*x r*)) + (ϑ (*x 0*) − ν (*x 0*))
    by *force*
  also have ... > *0*
   using *tHyp*(*1*) *x-ivp*(*2*) *primed* **by** (*metis* (*no-types*,*hide-lams*) *Groups*.*add-ac*(*2*)
add-sign-intros(*1*)
       calculation(*1*) *diff-gt-0-iff-gt ge-iff-diff-ge-0 less-eq-real-def zero-le-scaleR-iff*)

  ultimately show ν (*x* τ) < ϑ (*x* τ)
    by *simp*
qed

lemma [*ode-invariant-rules*]:
fixes ϑ::'*a*::*banach* ⇒ *real*
assumes *I1 is-diff-invariant-of f along* {*0*..*t*}
    and *I2 is-diff-invariant-of f along* {*0*..*t*}
shows (λ*s*. *I1 s* ∧ *I2 s*) *is-diff-invariant-of f along* {*0*..*t*}
  using *assms* **unfolding** *diff-invariant-def* **by** *auto*

lemma [*ode-invariant-rules*]:
fixes ϑ::'*a*::*banach* ⇒ *real*
assumes *I1 is-diff-invariant-of f along* {*0*..*t*}
    and *I2 is-diff-invariant-of f along* {*0*..*t*}
shows (λ*s*. *I1 s* ∨ *I2 s*) *is-diff-invariant-of f along* {*0*..*t*}
  using *assms* **unfolding** *diff-invariant-def* **by** *auto*

end
theory *cat2ndfun-examples*
  imports *cat2ndfun*

begin

5.3.3 Examples

The examples in this subsection show different approaches for the verification of hybrid systems. However, the general approach can be outlined as follows: First, we select a finite type to model program variables $'n$. We use this to define a vector field f of type $('a,\ 'n)\ vec \Rightarrow ('a,\ 'n)\ vec$ to model the dynamics of our system. Then we show a partial correctness specification involving the evolution command $[x\acute{}=f]\,T\ \&\ G$ either by finding a flow for the vector field or through differential invariants.

Single constantly accelerated evolution

The main characteristics distinguishing this example from the rest are:

1. We define the finite type of program variables with 2 Isabelle strings which make the final verification easier to parse.
2. We define the vector field (named K) to model a constantly accelerated object.
3. We define a local flow (φ_K) and use it to compute the wlp for this vector field.
4. The verification is only done on a single evolution command (not operated with any other hybrid program).

typedef *program-vars* $=\{''y'',''v''\}$
  morphisms *to-str to-var*
  apply(*rule-tac* $x=''y''$ **in** *exI*)
  by *simp*

notation *to-var* ($\upharpoonright_V$)

lemma *number-of-program-vars*: $CARD(program\text{-}vars) = 2$
  using *type-definition.card type-definition-program-vars* **by** *fastforce*

instance *program-vars*::*finite*
  apply(*standard, subst bij-betw-finite*[*of to-str UNIV* $\{''y'',''v''\}$])
   apply(*rule bij-betwI'*)
    apply (*simp add*: *to-str-inject*)
  using *to-str* **apply** *blast*
   apply (*metis to-var-inverse UNIV-I*)
  by *simp*

lemma *program-vars-univD*:(*UNIV*::*program-vars set*) $= \{\upharpoonright_V\ ''y'',\ \upharpoonright_V\ ''v''\}$
  apply *auto* **by** (*metis to-str to-str-inverse insertE singletonD*)

lemma *program-vars-exhaust*: $\forall\, x$::*program-vars*. $x = \upharpoonright_V\ ''y'' \vee x = \upharpoonright_V\ ''v''$

```
  using program-vars-univD by auto

abbreviation constant-acceleration-kinematics g s ≡
  (χ i. if i=(↾V ''y'') then s $ (↾V ''v'') else g)

notation constant-acceleration-kinematics (K)

lemma cnst-acc-continuous:
  fixes X::(real^program-vars) set
  shows continuous-on X (K g)
  apply(rule continuous-on-vec-lambda)
  unfolding continuous-on-def apply clarsimp
  by(intro tendsto-intros)

lemma picard-lindeloef-cnst-acc:
  fixes g::real assumes 0 ≤ t and t < 1
  shows picard-lindeloef (λt. K g) {0..t} 1 0
  unfolding picard-lindeloef-def apply(simp add: lipschitz-on-def assms, safe)
  apply(rule-tac t=UNIV and f=snd in continuous-on-compose2)
  apply(simp-all add: cnst-acc-continuous continuous-on-snd)
   apply(simp add: dist-vec-def L2-set-def dist-real-def)
   apply(subst program-vars-univD, subst program-vars-univD)
   apply(simp-all add: to-var-inject)
  using assms by linarith

abbreviation constant-acceleration-kinematics-flow g t s ≡
  (χ i. if i=(↾V ''y'') then g · t ^ 2/2 + s $ (↾V ''v'') · t + s $ (↾V ''y'')
        else g · t + s $ (↾V ''v''))

notation constant-acceleration-kinematics-flow (φK)

lemma local-flow-cnst-acc:
  assumes 0 ≤ t and t < 1
  shows local-flow (K g) {0..t} 1 (φK g)
  unfolding local-flow-def local-flow-axioms-def apply safe
  using assms picard-lindeloef-cnst-acc apply blast
   apply(rule has-vderiv-on-vec-lambda)
  using poly-derivatives(3,4) program-vars-exhaust
  apply(simp-all add: to-var-inject vec-eq-iff has-vderiv-on-def has-vector-derivative-def)
  using program-vars-exhaust by blast

lemma wp-cnst-acc:
  assumes 0 ≤ t and t < 1
  shows  wp ([x´=K g]{0..t} & G) ⌈Q⌉ =
    ⌈λ s. ∀ τ ∈ {0..t}. (G ▷ (λr. φK g r s){0−−τ}) ⟶ Q (φK g τ s)⌉
  apply(subst local-flow.wp-g-orbit[of K g - 1 (λ t x. φK g t x)])
  using local-flow-cnst-acc and assms by(auto simp: nd-fun-ext)

lemma single-evolution-ball:
```

fixes H::*real* **assumes** $0 \leq t$ **and** $t < 1$ **and** $g < 0$
shows $\lceil\lambda s.\ 0 \leq s\ \$\ (\restriction_V\ ''y'') \wedge s\ \$\ (\restriction_V\ ''y'') = H \wedge s\ \$\ (\restriction_V\ ''v'') = 0\rceil$
$\leq wp\ ([x\acute{}=K\ g]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ (\restriction_V\ ''y'') \geq 0))$
$\lceil\lambda s.\ 0 \leq s\ \$\ (\restriction_V\ ''y'') \wedge s\ \$\ (\restriction_V\ ''y'') \leq H\rceil$
apply(*subst wp-cnst-acc*)
using *assms* **by**(*auto simp*: *mult-nonpos-nonneg*)

no-notation *to-var* ($\restriction_V$)

no-notation *constant-acceleration-kinematics* (K)

no-notation *constant-acceleration-kinematics-flow* (φ_K)

Single evolution revisited.

We list again the characteristics that distinguish this example:

1. We employ an existing finite type of size 3 to model program variables.
2. We define a 3×3 matrix (named K) to denote the linear operator that models the constantly accelerated motion.
3. We define a local flow (φ_K) and use it to compute the wlp for this linear operator.
4. The verification is done equivalently to the above example.

term x::*2* — It turns out that there is already a 2-element type:

lemma *CARD*(*program-vars*) = *CARD*(*2*)
unfolding *number-of-program-vars* **by** *simp*

In fact, for each natural number n there is already a corresponding n-element type in Isabelle. However, there are still lemmas to prove about them in order to do verification of hybrid systems in n-dimensional Euclidean spaces.

lemma *exhaust-5*: — The analogs for $1, 2$ and 3 have already been proven in Analysis.
fixes x::*5*
shows $x=1 \vee x=2 \vee x=3 \vee x=4 \vee x=5$
proof (*induct x*)
case (*of-int z*)
then have $0 \leq z$ **and** $z < 5$ **by** *simp-all*
then have $z = 0 \vee z = 1 \vee z = 2 \vee z = 3 \vee z = 4$ **by** *arith*
then show *?case* **by** *auto*
qed

lemma *UNIV-3*:(*UNIV*::*3 set*) = $\{0,\ 1,\ 2\}$
apply *safe* **using** *exhaust-3 three-eq-zero* **by**(*blast*, *auto*)

lemma *sum-axis-UNIV-3*[*simp*]:($\sum j \in (UNIV::3\ set).\ axis\ i\ 1\ \$\ j \cdot f\ j) = (f::3 \Rightarrow real)\ i$
unfolding *axis-def UNIV-3* **apply** *simp*
using *exhaust-3* **by** *force*

We can rewrite the original constant acceleration kinematics as a linear operator applied to a 3-dimensional vector. For that we take advantage of the following fact:

lemma e $1 = (\chi\ j::3.\ if\ j= 0\ then\ 0\ else\ if\ j = 1\ then\ 1\ else\ 0)$
unfolding *axis-def* **by**(*rule Cart-lambda-cong*, *simp*)

abbreviation *constant-acceleration-kinematics-matrix* $\equiv$
$(\chi\ i.\ if\ i= (0::3)\ then\ axis\ (1::3)\ (1::real)\ else\ if\ i= 1\ then\ axis\ 2\ 1\ else\ 0)$

abbreviation *constant-acceleration-kinematics-matrix-flow t s* $\equiv$
$(\chi\ i.\ if\ i= (0::3)\ then\ s\ \$\ 2 \cdot t\ \hat{}\ 2/2 + s\ \$\ 1 \cdot t + s\ \$\ 0$
$else\ if\ i=1\ then\ s\ \$\ 2 \cdot t + s\ \$\ 1\ else\ s\ \$\ 2)$

notation *constant-acceleration-kinematics-matrix* (K)

notation *constant-acceleration-kinematics-matrix-flow* (φ_K)

With these 2 definitions and the proof that linear systems of ODEs are Picard-Lindeloef, we can show that they form a pair of vector-field and its flow.

lemma *entries-cnst-acc-matrix*: $entries\ K = \{0,\ 1\}$
apply (*simp-all add*: *axis-def*, *safe*)
by(*rule-tac x=1* **in** *exI*, *simp*)+

lemma *picard-lindeloef-cnst-acc-matrix*:
assumes $0 \le t$ **and** $t < 1/9$
shows *picard-lindeloef* $(\lambda\ t\ s.\ K *v\ s)\ \{0..t\}\ ((real\ CARD(3))^2 \cdot (\|K\|_{max}))\ 0$
apply(*rule picard-lindeloef-linear-system*)
unfolding *entries-cnst-acc-matrix* **using** *assms* **by** *auto*

lemma *local-flow-cnst-acc-matrix*:
assumes $0 \le t$ **and** $t < 1/9$
shows *local-flow* $((*v)\ K)\ \{0..t\}\ ((real\ CARD(3))^2 \cdot (\|K\|_{max}))\ \varphi_K$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-cnst-acc-matrix*[*OF assms*] **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*)
using *poly-derivatives*(*1*,*3*, *4*)
apply(*force simp*: *matrix-vector-mult-def*)
using *exhaust-3* **by**(*force simp*: *matrix-vector-mult-def vec-eq-iff*)

Finally, we compute the wlp of this example and use it to verify the single-evolution ball again.

lemma *wp-cnst-acc-matrix*:

assumes $0 \le t$ **and** $t < 1/9$
shows $wp\ ([x\acute{}=(*v)\ K]\{0..t\}\ \&\ G)\ \lceil Q\rceil = \lceil\lambda\ s.\ \forall\tau\in\{0..t\}.\ (G \rhd (\lambda r.\ \varphi_K\ r\ s)\{0--\tau\}) \longrightarrow Q\ (\varphi_K\ \tau\ s)\rceil$
apply(*subst local-flow.wp-g-orbit*[*of* $(*v)\ K$ - $((real\ CARD(3))^2 \cdot (\|K\|_{max}))\ \varphi_K$])
using *local-flow-cnst-acc-matrix* **and** *assms* **by** *auto*

lemma *single-evolution-ball-K*:
assumes $0 \le t$ **and** $t < 1/9$
shows $\lceil\lambda s.\ 0 \le s\ \$\ 0 \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\rceil$
$\le wp\ ([x\acute{}=(*v)\ K]\{0..t\}\ \&\ (\lambda s.\ s\ \$\ 0 \ge 0))\ \lceil\lambda s.\ 0 \le s\ \$\ 0 \wedge s\ \$\ 0 \le H\rceil$
apply(*subst wp-cnst-acc-matrix*)
using *assms* **by**(*auto simp*: *mult-nonneg-nonpos2*)

Circular Motion

The characteristics that distinguish this example are:

1. We employ an existing finite type of size 2 to model program variables.
2. We define a 2×2 matrix (named C) to denote the linear operator that models circular motion.
3. We show that the circle equation is a differential invariant for the linear operator.
4. We prove the partial correctness specification corresponding to the previous point.
5. For completeness, we define a local flow (φ_C) and use it to compute the wlp for the linear operator and the specification is proven again with this flow.

lemma *two-eq-zero*: $(2::2) = 0$
by *simp*

lemma [*simp*]:$i \neq (0::2) \longrightarrow i = 1$
using *exhaust-2* **by** *fastforce*

lemma *UNIV-2*:$(UNIV::2\ set) = \{0,\ 1\}$
apply *safe* **using** *exhaust-2 two-eq-zero* **by** *auto*

abbreviation *circular-motion-matrix* $\equiv$
$(\chi\ i.\ if\ i= (0::2)\ then\ axis\ (1::2)\ (-\ 1::real)\ else\ axis\ 0\ 1)$

notation *circular-motion-matrix* (C)

lemma *circle-invariant*:
assumes $0 < R$

shows $(\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2)$ *is-diff-invariant-of* $(*v)$ C *along* $\{0..t\}$
apply(*rule-tac ode-invariant-rules*, *clarsimp*)
apply(*frule-tac i=0* **in** *has-vderiv-on-vec-nth*, *drule-tac i=1* **in** *has-vderiv-on-vec-nth*)
apply(*unfold has-vderiv-on-def has-vector-derivative-def*, *clarsimp*)
apply(*erule-tac x=r* **in** *ballE*)+
apply(*simp add*: *matrix-vector-mult-def has-vderiv-on-vec-lambda*)
subgoal for $x\ \tau\ r$ **apply**(*rule-tac* $f'1=\lambda t.\ 0$ **and** $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*11*), *simp-all*)
apply(*rule-tac* $f'1=\lambda t. - 2 \cdot (x\ r\ \$\ 0) \cdot (t \cdot x\ r\ \$\ 1)$
and $g'1=\lambda t.\ 2 \cdot (x\ r\ \$\ 1) \cdot t \cdot x\ r\ \$\ 0$ **in** *derivative-eq-intros*(*8*), *simp-all*)
apply(*rule-tac* $f'1=\lambda t. - (t \cdot x\ r\ \$\ 1)$ **in** *derivative-eq-intros*(*15*))
apply(*rule-tac* $t=\{0--\tau\}$ **and** $s=\{0..t\}$ **in** *has-derivative-within-subset*)
apply(*simp*, *simp add*: *closed-segment-eq-real-ivl*, *force*)
apply(*rule-tac* $f'1=\lambda t.\ (t \cdot x\ r\ \$\ 0)$ **in** *derivative-eq-intros*(*15*))
apply(*rule-tac* $t=\{0--\tau\}$ **and** $s=\{0..t\}$ **in** *has-derivative-within-subset*)
by(*simp*, *simp add*: *closed-segment-eq-real-ivl*, *force*)
by(*auto simp*: *closed-segment-eq-real-ivl*)

lemma *circular-motion-invariants*:
assumes $(R{::}real) > 0$
shows $\lceil\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\rceil \le wp\ ([x' {=} (*v)\ C]\{0..t\}\ \&\ G)\ \lceil\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\rceil$
using *assms*(*1*) **apply**(*rule-tac* $C=\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2$ **in** *dCut*)
apply(*rule-tac* $I=\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2$ **in** *dI*)
using *circle-invariant* $\langle R > 0\rangle$ **apply**(*blast*, *force*, *force*)
by(*rule dWeakening*, *auto*)

— Proof of the same specification by providing solutions:

lemma *entries-circ-matrix*:*entries* $C = \{0, - 1, 1\}$
apply (*simp-all add*: *axis-def*, *safe*)
subgoal by(*rule-tac x=0* **in** *exI*, *simp*)+
subgoal by(*rule-tac x=0* **in** *exI*, *simp*)+
by(*rule-tac x=1* **in** *exI*, *simp*)+

lemma *picard-lindeloef-circ-matrix*:
assumes $0 \le t$ **and** $t < 1/4$
shows *picard-lindeloef* $(\lambda t.\ (*v)\ C)\ \{0..t\}\ ((real\ CARD(2))^2 \cdot (\|C\|_{max}))\ 0$
apply(*rule picard-lindeloef-linear-system*)
unfolding *entries-circ-matrix* **using** *assms* **by** *auto*

abbreviation *circular-motion-matrix-flow* $t\ s \equiv (\chi\ i.\ if\ i{=}\ (0{::}2)\ then$ $s\$0 \cdot cos\ t - s\$1 \cdot sin\ t\ else\ s\$0 \cdot sin\ t + s\$1 \cdot cos\ t)$

notation *circular-motion-matrix-flow* (φ_C)

lemma *local-flow-circ-mtx*:
assumes $0 \le t$ **and** $t < 1/4$
shows *local-flow* $((*v)\ C)\ \{0..t\}\ ((real\ CARD(2))^2 \cdot (\|C\|_{max}))\ \varphi_C$

unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-circ-matrix assms* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*)
apply(*simp add*: *matrix-vector-mult-def has-vderiv-on-def has-vector-derivative-def*, *safe*)
subgoal for *s i x*
apply(*rule-tac* $f'1{=}\lambda t.\ -\ s\$0\ \cdot\ (t\ \cdot\ sin\ x)$ **and** $g'1{=}\lambda t.\ s\$1\ \cdot\ (t\ \cdot\ cos\ x)$**in** *derivative-eq-intros*(*11*))
apply(*rule derivative-eq-intros*(*6*)[*of cos* ($\lambda xa.\ -\ (xa\ \cdot\ sin\ x)$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*58*))
apply(*rule ssubst*[*of* ($\cdot$) *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of sin* ($\lambda xa.\ (xa\ \cdot\ cos\ x)$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* ($\cdot$) *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
subgoal for *s i x*
apply(*rule-tac* $f'1{=}\lambda t.\ s\$0\ \cdot\ (t\ \cdot\ cos\ x)$ **and** $g'1{=}\lambda t.\ -\ s\$1\ \cdot\ (t\ \cdot\ sin\ x)$**in** *derivative-eq-intros*(*8*))
apply(*rule derivative-eq-intros*(*6*)[*of sin* ($\lambda xa.\ xa\ \cdot\ cos\ x$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*55*))
apply(*rule ssubst*[*of* ($\cdot$) *1 id*], *force*, *simp*, *force*, *force*)
apply(*rule derivative-eq-intros*(*6*)[*of cos* ($\lambda xa.\ -\ (xa\ \cdot\ sin\ x)$)])
apply(*rule-tac Db1=1* **in** *derivative-eq-intros*(*58*))
apply(*rule ssubst*[*of* ($\cdot$) *1 id*], *force*, *simp*, *force*, *force*)
by (*simp add*: *Groups.mult-ac*(*3*) *Rings.ring-distribs*(*4*))
using *exhaust-2 two-eq-zero* **by**(*force simp*: *vec-eq-iff*)

lemma *flow-for-Circ-DS*:
assumes $0 \le t$ **and** $t < 1/4$
shows $wp\ ([x' {=} (*v)\ C]\{0..t\}\ \&\ G)\ \lceil Q\rceil =$
$\lceil\lambda\ x.\ \forall\ \tau \in \{0..t\}.\ (\forall r{\in}\{0{-}{-}\tau\}.\ G\ (\varphi_C\ r\ x)) \longrightarrow Q\ (\varphi_C\ \tau\ x)\rceil$
apply(*subst local-flow.wp-g-orbit*[*of* $(*v)\ C$ - $((real\ CARD(2))^2\ \cdot\ (\|C\|_{max}))$ φ_C])
using *local-flow-circ-mtx* **and** *assms* **by** *auto*

lemma *circular-motion*:
assumes $0 \le t$ **and** $t < 1/4$ **and** $R > 0$
shows $\lceil\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\rceil \le wp\ ([x' {=} (*v)\ C]\{0..t\}\ \&\ G)\ \lceil\lambda s.\ R^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\rceil$
apply(*subst flow-for-Circ-DS*)
using *assms* **by** *simp-all*

no-notation *circular-motion-matrix* (C)

no-notation *circular-motion-matrix-flow* (φ_C)

Bouncing Ball with solution

We revisit the previous dynamics for a constantly accelerated object modelled with the matrix K. We compose the corresponding evolution command with an if-statement, and iterate this hybrid program to model a (completely elastic) "bouncing ball". Using the previously defined flow for this dynamics, proving a specification for this hybrid program is merely an exercise of real arithmetic.

named-theorems *bb-real-arith real arithmetic properties for the bouncing ball.*

lemma [*bb-real-arith*]:$0 \leq x \Longrightarrow 0 > g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v \Longrightarrow$ $(x{::}real) \leq H$
proof−
  assume $0 \leq x$ **and** $0 > g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
  then have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot H \wedge 0 > g$ **by** *auto*
  hence $*{:}v \cdot v = 2 \cdot g \cdot (x - H) \wedge 0 > g \wedge v \cdot v \geq 0$
    using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
  from *this* **have** $(v \cdot v)/(2 \cdot g) = (x - H)$ **by** *auto*
  also from $*$ **have** $(v \cdot v)/(2 \cdot g) \leq 0$
    using *divide-nonneg-neg* **by** *fastforce*
  ultimately have $H - x \geq 0$ **by** *linarith*
  thus *?thesis* **by** *auto*
qed

lemma [*bb-real-arith*]:
  assumes *invar*:$2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
    and *pos*:$g \cdot \tau^2 \;/\; 2 + v \cdot \tau + (x{::}real) = 0$
  shows $2 \cdot g \cdot H + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
and $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
proof−
  from *pos* **have** $g \cdot \tau^2 \; + 2 \cdot v \cdot \tau + 2 \cdot x = 0$ **by** *auto*
  then have $g^2 \; \cdot \tau^2 \; + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x = 0$
    by (*metis* (*mono-tags*, *hide-lams*) *Groups.mult-ac*(*1*,*3*) *mult-zero-right*
        monoid-mult-class.power2-eq-square semiring-class.distrib-left)
  hence $g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + v^2 + 2 \cdot g \cdot H = 0$
    using *invar* **by** (*simp add*: *monoid-mult-class.power2-eq-square*)
  from *this* **have** $*{:}(g \cdot \tau + v)^2 + 2 \cdot g \cdot H = 0$
   apply(*subst power2-sum*) **by** (*metis* (*no-types*, *hide-lams*) *Groups.add-ac*(*2*, *3*)

        Groups.mult-ac(*2*, *3*) *monoid-mult-class.power2-eq-square nat-distrib*(*2*))
  hence $2 \cdot g \cdot H + (- ((g \cdot \tau) + v))^2 = 0$
    by (*metis Groups.add-ac*(*2*) *power2-minus*)
  thus $2 \cdot g \cdot H + (- (g \cdot \tau) - v) \cdot (- (g \cdot \tau) - v) = 0$
    by (*simp add*: *monoid-mult-class.power2-eq-square*)
  from $*$ **show** $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
    by (*simp add*: *monoid-mult-class.power2-eq-square*)
qed

lemma [*bb-real-arith*]:
 assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot H + v \cdot v$
 shows $2 \cdot g \cdot (g \cdot \tau^2 / 2 + v \cdot \tau + (x::real)) =$
 $2 \cdot g \cdot H + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**is** *?lhs* = *?rhs*)
proof−
 have *?lhs* $= g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot x$
  apply(*subst Rat.sign-simps*(*18*))+
  by(*auto simp*: *semiring-normalization-rules*(*29*))
 also have ... $= g^2 \cdot \tau^2 + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot H + v \cdot v$ (**is** ... = *?middle*)
  by(*subst invar*, *simp*)
 finally have *?lhs* = *?middle*.
 moreover
 {**have** *?rhs* $= g \cdot g \cdot (\tau \cdot \tau) + 2 \cdot g \cdot v \cdot \tau + 2 \cdot g \cdot H + v \cdot v$
  by (*simp add*: *Groups.mult-ac*(*2*,*3*) *semiring-class.distrib-left*)
 also have ... = *?middle*
  by (*simp add*: *semiring-normalization-rules*(*29*))
 finally have *?rhs* = *?middle*.}
 ultimately show *?thesis* **by** *auto*
qed

lemma *bouncing-ball*:
 assumes $0 \leq t$ **and** $t < 1/9$
 shows $\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2 \rceil \leq wp$
 $((([x' = (*v)\ K]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0)) \cdot$
 $(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.\ -\ s\ \$\ 1)])\ ELSE\ \eta^\bullet\ FI))^\star)$
 $\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq H \rceil$
 apply(*subst star-nd-fun.abs-eq*)
 apply(*rule wp-starI*[*of* - $\lceil \lambda s.\ 0 \leq s\ \$\ 0 \wedge 0 > s\ \$\ 2\ \wedge$
 $2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 = 2 \cdot s\ \$\ 2 \cdot H + (s\ \$\ 1 \cdot s\ \$\ 1)\rceil$])
  apply(*simp*, *simp only*: *fbox-mult*)
  apply(*subst p2ndf-ndf2p-wp-sym*[*of* $(IF\ (\lambda s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.$
$-\ s\ \$\ 1)])\ ELSE\ \eta^\bullet\ FI)$])
  apply(*subst wp-cnst-acc-matrix*)
 using *assms* **apply**(*simp*, *simp*)
  apply(*subst ndf2p-wpD*)
 unfolding *cond-def* **apply** *clarsimp*
  apply(*transfer*, *simp add*: *kcomp-def*)
 by(*auto simp*: *bb-real-arith*)

Bouncing Ball with invariants

We prove again the bouncing ball but this time with differential invariants.

lemma *gravity-invariant*: $(\lambda s.\ s\ \$\ 2 < 0)$ *is-diff-invariant-of* $(*v)\ K$ *along* $\{0..t\}$
 apply(*rule-tac* $\vartheta' = \lambda s.\ 0$ **and** $\nu' = \lambda s.\ 0$ **in** *ode-invariant-rules*(*3*), *clarsimp*)
 apply(*drule-tac* $i=2$ **in** *has-vderiv-on-vec-nth*)
 apply(*unfold has-vderiv-on-def has-vector-derivative-def*)
 apply(*erule-tac* $x=r$ **in** *ballE*, *simp add*: *matrix-vector-mult-def*)
  apply(*rule-tac* $f'1 = \lambda s.\ 0$ **in** *derivative-eq-intros*(*10*))
 by(*auto simp*: *closed-segment-eq-real-ivl has-derivative-within-subset*)

lemma *energy-conservation-invariant*:
$(\lambda s.\ 2 \cdot s\ \$\ 2 \cdot s\ \$\ 0 - 2 \cdot s\ \$\ 2 \cdot H - s\ \$\ 1 \cdot s\ \$\ 1 = 0)$ *is-diff-invariant-of* $(*v)$ K *along* $\{0..t\}$
apply(*rule ode-invariant-rules*, *clarify*)
apply(*frule-tac* $i=2$ **in** *has-vderiv-on-vec-nth*)
apply(*frule-tac* $i=1$ **in** *has-vderiv-on-vec-nth*)
apply(*drule-tac* $i=0$ **in** *has-vderiv-on-vec-nth*)
apply(*unfold has-vderiv-on-def has-vector-derivative-def*)
apply(*erule-tac* $x=r$ **in** *ballE*, *simp-all add*: *matrix-vector-mult-def*)+
apply(*rule-tac* $f'1=\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$
and $g'1=\lambda t.\ 2 \cdot (t \cdot (x\ r\ \$\ 1 \cdot x\ r\ \$\ 2))$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac* $f'1=\lambda t.\ 2 \cdot x\ r\ \$\ 2 \cdot (t \cdot x\ r\ \$\ 1)$ **and** $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*11*))
apply(*rule-tac* $f'1=\lambda t.\ 0$ **and** $g'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 1)$ **in** *derivative-eq-intros*(*12*))
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*))
apply(*simp-all add*: *has-derivative-within-subset closed-segment-eq-real-ivl*)
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*7*))
apply(*rule-tac* $g'1=\lambda t.\ 0$ **in** *derivative-eq-intros*(*6*))
apply(*simp-all add*: *has-derivative-within-subset*)
apply(*rule-tac* $f'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **and** $g'1=(\lambda xa.\ xa \cdot x\ r\ \$\ 2)$ **in** *derivative-eq-intros*(*12*))
by(*simp-all add*: *has-derivative-within-subset*)

lemma *bouncing-ball-invariants*:
$\lceil \lambda s.\ 0 \le s\ \$\ 0 \wedge s\ \$\ 0 = H \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2 \rceil \le$
wp $((([x' = (\lambda s.\ K *v\ s)]\{0..t\}\ \&\ (\lambda\ s.\ s\ \$\ 0 \ge 0)) \cdot$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.\ - s\ \$\ 1)])\ ELSE\ \eta^{\bullet}\ FI))^{\star})$
$\lceil \lambda s.\ 0 \le s\ \$\ 0 \wedge s\ \$\ 0 \le H \rceil$
apply(*subst star-nd-fun.abs-eq*)
apply(*rule-tac* $I=\lceil \lambda s.\ 0 \le s\$0 \wedge 0 > s\$2 \wedge 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot H + (s\$1 \cdot s\$1) \rceil$ **in** *wp-starI*)
apply(*simp*, *simp only*: *fbox-mult*)
apply(*subst p2ndf-ndf2p-wp-sym*[*of* $(IF\ (\lambda s.\ s\ \$\ 0 = 0)\ THEN\ ([1 ::== (\lambda s.\ - s\ \$\ 1)])\ ELSE\ \eta^{\bullet}\ FI)$])
apply(*rule dCut*[**where** $C=\lambda\ s.\ s\ \$\ 2 < 0$])
apply(*rule-tac* $I=\lambda\ s.\ s\ \$\ 2 < 0$ **in** *dI*)
using *gravity-invariant* **apply**(*blast*, *force*, *force*)
apply(*rule-tac* $C=\lambda\ s.\ 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1 = 0$ **in** *dCut*)
apply(*rule-tac* $I=\lambda\ s.\ 2 \cdot s\$2 \cdot s\$0 - 2 \cdot s\$2 \cdot H - s\$1 \cdot s\$1 = 0$ **in** *dI*)
using *energy-conservation-invariant* **apply**(*blast*, *force*, *force*)
apply(*rule dWeakening*, *subst p2ndf-ndf2p-wp*)
apply(*rule wp-if-then-else*)
by(*auto simp*: *bb-real-arith le-fun-def*)

end

5.4 VC_diffKAD

theory *VC-diffKAD-auxiliarities*
imports
Main
../afpModified/VC-KAD
Ordinary-Differential-Equations.ODE-Analysis

begin

5.4.1 Stack Theories Preliminaries: VC_KAD and ODEs

To make our notation less code-like and more mathematical we declare:

no-notation *Archimedean-Field.ceiling* (⌈-⌉)
  and *Archimedean-Field.floor* (⌊-⌋)
  and *Set.image* (')
  and *Range-Semiring.antirange-semiring-class.ars-r* (r)

notation *p2r* (⌈-⌉)
  and *r2p* (⌊-⌋)
  and *Set.image* (-(|-|))
  and *Product-Type.prod.fst* (π_1)
  and *Product-Type.prod.snd* (π_2)
  and *List.zip* (**infixl** ⊗ *63*)
  and *rel-ad* ($\Delta^c{}_1$)

This and more notation is explained by the following lemmata.

lemma shows $\lceil P\rceil = \{(s,\ s)\ |s.\ P\ s\}$
  and $\lfloor R\rfloor = (\lambda x.\ x \in \mathit{r2s}\ R)$
  and $\mathit{r2s}\ R = \{x\ |x.\ \exists\ y.\ (x,y) \in R\}$
  and $\pi_1\ (x,y) = x \wedge \pi_2\ (x,y) = y$
  and $\Delta^c{}_1\ R = \{(x,\ x)\ |x.\ \nexists y.\ (x,\ y) \in R\}$
  and $\mathit{wp}\ R\ Q = \Delta^c{}_1\ (R\ ;\ \Delta^c{}_1\ Q)$
  and $[\mathit{x1},\mathit{x2},\mathit{x3},\mathit{x4}] \otimes [\mathit{y1},\mathit{y2}] = [(\mathit{x1},\mathit{y1}),(\mathit{x2},\mathit{y2})]$
  and $\{a..b\} = \{x.\ a \leq x \wedge x \leq b\}$
  and $\{a<..<b\} = \{x.\ a < x \wedge x < b\}$
  and $(x\ \mathit{solves\text{-}ode}\ f)\ \{0..t\}\ R = ((x\ \mathit{has\text{-}vderiv\text{-}on}\ (\lambda t.\ f\ t\ (x\ t)))\ \{0..t\} \wedge x \in \{0..t\} \rightarrow R)$
  and $f \in A \rightarrow B = (f \in \{f.\ \forall\ x.\ x \in A \longrightarrow (f\ x) \in B\})$
  and $(x\ \mathit{has\text{-}vderiv\text{-}on}\ x')\{0..t\} =$
   $(\forall\, r{\in}\{0..t\}.\ (x\ \mathit{has\text{-}vector\text{-}derivative}\ x'\ r)\ (\mathit{at}\ r\ \mathit{within}\ \{0..t\}))$
  and $(x\ \mathit{has\text{-}vector\text{-}derivative}\ x'\ r)\ (\mathit{at}\ r\ \mathit{within}\ \{0..t\}) =$
   $(x\ \mathit{has\text{-}derivative}\ (\lambda x.\ x *_R x'\ r))\ (\mathit{at}\ r\ \mathit{within}\ \{0..t\})$
apply(*simp-all add*: *p2r-def r2p-def rel-ad-def rel-antidomain-kleene-algebra.fbox-def*

 solves-ode-def has-vderiv-on-def)
apply(*blast*, *fastforce*, *fastforce*)
using *has-vector-derivative-def* **by** *auto*

Observe also, the following consequences and facts:

proposition $\pi_1(\!|R|\!) = \textit{r2s}\ R$
by (*simp add*: *fst-eq-Domain*)

proposition $\Delta^c{}_1\ R = \textit{Id} - \{(s,\ s)\ |s.\ s \in (\pi_1(\!|R|\!))\}$
by(*simp add*: *image-def rel-ad-def*, *fastforce*)

proposition $P \subseteq Q \Longrightarrow \textit{wp}\ R\ P \subseteq \textit{wp}\ R\ Q$
by(*simp add*: *rel-antidomain-kleene-algebra.dka.dom-iso rel-antidomain-kleene-algebra.fbox-iso*)

proposition *boxProgrPred-IsProp*: $\textit{wp}\ R\ \lceil P\rceil \subseteq \textit{Id}$
by(*simp add*: *rel-antidomain-kleene-algebra.a-subid′ rel-antidomain-kleene-algebra.addual.bbox-def*)

proposition *rdom-p2r-contents*:$(a,\ b) \in \textit{rdom}\ \lceil P\rceil = ((a = b) \wedge P\ a)$
proof−
have $(a,\ b) \in \textit{rdom}\ \lceil P\rceil = ((a = b) \wedge (a,\ a) \in \textit{rdom}\ \lceil P\rceil)$ **using** *p2r-subid* **by** *fastforce*
also have ... $= ((a = b) \wedge (a,\ a) \in \lceil P\rceil)$ **by** *simp*
also have ... $= ((a = b) \wedge P\ a)$ **by** (*simp add*: *p2r-def*)
ultimately show *?thesis* **by** *simp*
qed

~~*..Should not add these complement-rule's to simp..*~~
proposition *rel-ad-rule1*: $(x,x) \notin \Delta^c{}_1\ \lceil P\rceil \Longrightarrow P\ x$
by(*auto simp*: *rel-ad-def p2r-subid p2r-def*)

proposition *rel-ad-rule2*: $(x,x) \in \Delta^c{}_1\ \lceil P\rceil \Longrightarrow \neg\ P\ x$
by(*metis ComplD VC-KAD.p2r-neg-hom rel-ad-rule1 empty-iff mem-Collect-eq p2s-neg-hom*

rel-antidomain-kleene-algebra.a-one rel-antidomain-kleene-algebra.am1 relcomp.relcompI)

proposition *rel-ad-rule3*: $R \subseteq \textit{Id} \Longrightarrow (x,x) \notin R \Longrightarrow (x,x) \in \Delta^c{}_1\ R$
by(*metis IdI Un-iff d-p2r rel-antidomain-kleene-algebra.addual.ars3 rel-antidomain-kleene-algebra.addual.ars-r-def rpr*)

proposition *rel-ad-rule4*: $(x,x) \in R \Longrightarrow (x,x) \notin \Delta^c{}_1\ R$
by(*metis empty-iff rel-antidomain-kleene-algebra.addual.ars1 relcomp.relcompI*)

proposition *boxProgrPred-chrctrztn*:$(x,x) \in \textit{wp}\ R\ \lceil P\rceil = (\forall\ y.\ (x,y) \in R \longrightarrow P\ y)$
by(*metis boxProgrPred-IsProp rel-ad-rule1 rel-ad-rule2 rel-ad-rule3 rel-ad-rule4 d-p2r wp-simp wp-trafo*)

lemma (**in** *antidomain-kleene-algebra*) *fbox-starI*:
assumes $d\ p \leq d\ i$ **and** $d\ i \leq |x]\ i$ **and** $d\ i \leq d\ q$
shows $d\ p \leq |x^\star]\ q$
proof−
from ‹$d\ i \leq |x]\ i$› **have** $d\ i \leq |x]\ (d\ i)$
  using *local.fbox-simp* **by** *auto*

hence $\lfloor 1 \rfloor\ p \leq \lfloor x^\star \rfloor\ i$ **using** ‹$d\ p \leq d\ i$› **by** (*metis* (*no-types*)
local.dual-order.trans local.fbox-one local.fbox-simp local.fbox-star-induct-var)
thus *?thesis* **using** ‹$d\ i \leq d\ q$› **by** (*metis* (*full-types*)
local.fbox-mult local.fbox-one local.fbox-seq-var local.fbox-simp)
qed

proposition *cons-eq-zipE*:
$(x,\ y)\ \#\ tail = xList \otimes yList \Longrightarrow \exists xTail\ yTail.\ x\ \#\ xTail = xList \wedge y\ \#\ yTail = yList$
by(*induction xList*, *simp-all*, *induction yList*, *simp-all*)

proposition *set-zip-left-rightD*:
$(x,\ y) \in set\ (xList \otimes yList) \Longrightarrow x \in set\ xList \wedge y \in set\ yList$
apply(*rule conjI*)
apply(*rule-tac y=y* **and** *ys=yList* **in** *set-zip-leftD*, *simp*)
apply(*rule-tac x=x* **and** *xs=xList* **in** *set-zip-rightD*, *simp*)
done

declare *zip-map-fst-snd* [*simp*]

5.4.2 VC_diffKAD Preliminaries

In dL, the set of possible program variables is split in two, the set of variables V and their primed counterparts V'. To implement this, we use Isabelle's string-type and define a function that primes a given string. We then define the set of primed-strings based on it.

definition *vdiff* ::*string* $\Rightarrow$ *string* (∂ - [55] 70) **where**
$(\partial\ x) = ''d[''@x@'']''$

definition *varDiffs* :: *string set* **where**
$varDiffs = \{y.\ \exists\ x.\ y = \partial\ x\}$

proposition *vdiff-inj*:$(\partial\ x) = (\partial\ y) \Longrightarrow x = y$
by(*simp add*: *vdiff-def*)

proposition *vdiff-noFixPoints*:$x \neq (\partial\ x)$
by(*simp add*: *vdiff-def*)

lemma *varDiffsI*:$x = (\partial\ z) \Longrightarrow x \in varDiffs$
by(*simp add*: *varDiffs-def vdiff-def*)

lemma *varDiffsE*:
assumes $x \in varDiffs$
obtains y **where** $x = ''d[''@y@'']''$
using *assms* **unfolding** *varDiffs-def vdiff-def* **by** *auto*

proposition *vdiff-invarDiffs*:$(\partial\ x) \in varDiffs$
by (*simp add*: *varDiffsI*)

(primed) dSolve preliminaries

This subsubsection is to define a function that takes a system of ODEs (expressed as a list $xfList$), a presumed solution $uInput = [u_1, \ldots, u_n]$, a state s and a time t, and outputs the induced flow $sol\, s[xfList \leftarrow uInput]\, t$.

abbreviation *varDiffs-to-zero* ::*real store* ⇒ *real store* (*sol*) **where**
sol a ≡ (*override-on a* (λ *x*. *0*) *varDiffs*)

proposition *varDiffs-to-zero-vdiff* [*simp*]: (*sol s*) (∂ *x*) = *0*
apply(*simp add*: *override-on-def varDiffs-def*)
by *auto*

proposition *varDiffs-to-zero-beginning*[*simp*]: *take 2 x* ≠ ''d['' ⟹ (*sol s*) *x* = *s* *x*
apply(*simp add*: *varDiffs-def override-on-def vdiff-def*)
by *fastforce*

— Next, for each entry of the input-list, we update the state using said entry.

definition *vderiv-of f S* = (*SOME f'*. (*f has-vderiv-on f'*) *S*)

primrec *state-list-upd* :: ((*real* ⇒ *real store* ⇒ *real*) × *string* × (*real store* ⇒ *real*)) *list* ⇒
real ⇒ *real store* ⇒ *real store* **where**
state-list-upd [] *t s* = *s*|
state-list-upd (*uxf* # *tail*) *t s* = (*state-list-upd tail t s*)
($(\pi_1\ (\pi_2\ uxf))$:= $(\pi_1\ uxf)$ *t s*,
∂ $(\pi_1\ (\pi_2\ uxf))$:= (*if t* = *0 then* $(\pi_2\ (\pi_2\ uxf))$ *s*
else vderiv-of (λ *r*. $(\pi_1\ uxf)$ *r s*) {*0*<..< (*2* $*_R$ *t*)} *t*))

abbreviation *state-list-cross-upd* ::*real store* ⇒ (*string* × (*real store* ⇒ *real*)) *list* ⇒
(*real* ⇒ *real store* ⇒ *real*) *list* ⇒ *real* ⇒ (*char list* ⇒ *real*) (-[-←-] - [*64*,*64*,*64*] *63*) **where**
s[*xfList*←*uInput*] *t* ≡ *state-list-upd* (*uInput* ⊗ *xfList*) *t s*

proposition *state-list-cross-upd-empty*[*simp*]: (*s*[[]←*list*] *t*) = *s*
by(*induction list*, *simp-all*)

lemma *inductive-state-list-cross-upd-its-vars*:
assumes *distHyp*:*distinct* (*map* π_1 ((*y*, *g*) # *xftail*))
and *varHyp*:∀ *xf*∈*set*((*y*, *g*) # *xftail*). π_1 *xf* ∉ *varDiffs*
and *indHyp*:(*u*, *x*, *f*) ∈ *set* (*utail* ⊗ *xftail*) ⟹ (*s*[*xftail*←*utail*] *t*) *x* = *u t s*
and *disjHyp*:(*u*, *x*, *f*) = (*v*, *y*, *g*) ∨ (*u*, *x*, *f*) ∈ *set* (*utail* ⊗ *xftail*)
shows (*s*[(*y*, *g*) # *xftail*←*v* # *utail*] *t*) *x* = *u t s*
using *disjHyp* **proof**
assume (*u*, *x*, *f*) = (*v*, *y*, *g*)
hence (*s*[(*y*, *g*) # *xftail*←*v* # *utail*] *t*) *x* = ((*s*[*xftail*←*utail*] *t*)(*x* := *u t s*,
∂ *x* := *if t* = *0 then f s else vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* $*_R$ *t*)} *t*)) *x* **by**

simp
also have ... = *u t s* **by** (*simp add*: *vdiff-def*)
ultimately show *?thesis* **by** *simp*
next
assume *yTailHyp*:$(u, x, f) \in$ *set* (*utail* $\otimes$ *xftail*)
from *this* **and** *indHyp* **have** *3*:(*s*[*xftail*$\leftarrow$*utail*] *t*) $x = u\ t\ s$ **by** *fastforce*
from *yTailHyp* **and** *distHyp* **have** *2*:$y \neq x$ **using** *set-zip-left-rightD* **by** *force*
from *yTailHyp* **and** *varHyp* **have** *1*:$x \neq \partial\ y$
using *set-zip-left-rightD vdiff-invarDiffs* **by** *fastforce*
from *1* **and** *2* **have** (*s*[(*y*, *g*) # *xftail*$\leftarrow$*v* # *utail*] *t*) *x* = (*s*[*xftail*$\leftarrow$*utail*] *t*) *x* **by** *simp*
thus *?thesis* **using** *3* **by** *simp*
qed

theorem *state-list-cross-upd-its-vars*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *its-var*: (*u*,*x*,*f*) $\in$ *set* (*uInput* $\otimes$ *xfList*)
shows (*s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = *u t s*
using *assms* **apply**(*induct xfList uInput arbitrary*: *x rule*: *list-induct2'*, *simp*, *simp*, *simp*)
by(*clarify*, *rule inductive-state-list-cross-upd-its-vars*, *simp-all*)

lemma *override-on-upd*:$x \in X \Longrightarrow$ (*override-on f g X*)($x := z$) = (*override-on f* ($g(x := z)$) *X*)
by (*rule ext*, *simp add*: *override-on-def*)

lemma *inductive-state-list-cross-upd-its-dvars*:
assumes $\exists\, g$. (*s*[*xfTail*$\leftarrow$*uTail*] *0*) = *override-on s g varDiffs*
and $\forall$ *xf*$\in$*set* (*xf* # *xfTail*). π_1 *xf* $\notin$ *varDiffs*
and $\forall$ *uxf*$\in$*set* (*u* # *uTail* $\otimes$ *xf* # *xfTail*). π_1 *uxf 0 s* = *s* (π_1 (π_2 *uxf*))
shows $\exists\, g$. (*s*[*xf* # *xfTail*$\leftarrow$*u* # *uTail*] *0*) = *override-on s g varDiffs*
proof−
let *?gLHS*=(*s*[(*xf* # *xfTail*)$\leftarrow$(*u* # *uTail*)] *0*)
have *observ*:∂ (π_1 *xf*) $\in$ *varDiffs* **by** (*auto simp*: *varDiffs-def*)
from *assms*(*1*) **obtain** *g* **where** (*s*[*xfTail*$\leftarrow$*uTail*] *0*) = *override-on s g varDiffs* **by** *force*
then have *?gLHS* = (*override-on s g varDiffs*)(π_1 *xf* := *u 0 s*, ∂ (π_1 *xf*) := π_2 *xf s*) **by** *simp*
also have . . . = (*override-on s g varDiffs*)(∂ (π_1 *xf*) := π_2 *xf s*)
using *override-on-def varDiffs-def assms* **by** *auto*
also have ... = (*override-on s* (g(∂ (π_1 *xf*) := π_2 *xf s*)) *varDiffs*)
using *observ* **and** *override-on-upd* **by** *force*
ultimately show *?thesis* **by** *auto*
qed

theorem *state-list-cross-upd-its-dvars*:
assumes *lengthHyp*:*length xfList* = *length uInput*

and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0 s* = *s* (π_1 (π_2 *uxf*))
shows $\exists$ *g*. (*s*[*xfList*$\leftarrow$*uInput*] *0*) = (*override-on s g varDiffs*)
using *assms* **proof**(*induct xfList uInput rule*: *list-induct2'*)
case *1*
  have (*s*[[]$\leftarrow$[]] *0*) = *override-on s s varDiffs*
  unfolding *override-on-def* **by** *simp*
  thus *?case* **by** *metis*
next
  case (*2 xf xfTail*)
  have (*s*[(*xf* # *xfTail*)$\leftarrow$[]] *0*) = *override-on s s varDiffs*
  unfolding *override-on-def* **by** *simp*
  thus *?case* **by** *metis*
next
  case (*3 u utail*)
  have (*s*[[]$\leftarrow$*utail*] *0*) = *override-on s s varDiffs*
  unfolding *override-on-def* **by** *simp*
  thus *?case* **by** *force*
next
  case (*4 xf xfTail u uTail*)
  then have $\exists$ *g*. (*s*[*xfTail*$\leftarrow$*uTail*] *0*) = *override-on s g varDiffs* **by** *simp*
  thus *?case* **using** *inductive-state-list-cross-upd-its-dvars 4.prems* **by** *blast*
qed

lemma *vderiv-unique-within-open-interval*:
assumes (*f has-vderiv-on f'*) {*0*<..< *t*} **and** *t*>*0*
    and (*f has-vderiv-on f''*){*0*<..< *t*} **and** *tauHyp*:τ $\in$ {*0*<..< *t*}
shows *f'* τ = *f''* τ
using *assms* **apply**(*simp add*: *has-vderiv-on-def has-vector-derivative-def*)
using *frechet-derivative-unique-within-open-interval* **by** (*metis box-real*(*1*) *scaleR-one tauHyp*)

lemma *has-vderiv-on-cong-open-interval*:
assumes *gHyp*:$\forall$ τ > *0*. *f* τ = *g* τ **and** *tHyp*: *t*>*0*
and *fHyp*:(*f has-vderiv-on f'*) {*0*<..<*t*}
shows (*g has-vderiv-on f'*) {*0*<..<*t*}
proof−
from *gHyp* **have** $\bigwedge\tau$. τ $\in$ {*0*<..< *t*} $\Longrightarrow$ *f* τ = *g* τ **using** *tHyp* **by** *force*
hence *eqDs*:(*f has-vderiv-on f'*) {*0*<..<*t*} = (*g has-vderiv-on f'*) {*0*<..<*t*}
apply(*rule-tac has-vderiv-on-cong*) **by** *auto*
thus (*g has-vderiv-on f'*) {*0*<..<*t*} **using** *eqDs fHyp* **by** *simp*
qed

lemma *closed-vderiv-on-cong-to-open-vderiv*:
assumes *gHyp*:$\forall$ τ > *0*. *f* τ = *g* τ
and *fHyp*:$\forall$ *t*$\geq$*0*. (*f has-vderiv-on f'*) {*0*..*t*}
and *tHyp*: *t*>*0* **and** *cHyp*: *c* > *1*
shows *vderiv-of g* {*0*<..< (*c* $*_R$ *t*)} *t* = *f'* *t*
proof−

have *ctHyp*:$c \cdot t > 0$ **using** *tHyp* **and** *cHyp* **by** *auto*
from *fHyp* **have** (f *has-vderiv-on* f') $\{0<..<c \cdot t\}$ **using** *has-vderiv-on-subset*
by (*metis greaterThanLessThan-subseteq-atLeastAtMost-iff less-eq-real-def*)
then have *derivHyp*:(g *has-vderiv-on* f') $\{0<..<c \cdot t\}$
using *gHyp ctHyp* **and** *has-vderiv-on-cong-open-interval* **by** *blast*
hence f'*Hyp*:$\forall\ f''.$ (g *has-vderiv-on* f'') $\{0<..<c \cdot t\} \longrightarrow (\forall\ \tau \in \{0<..< c \cdot t\}.$
$f'\ \tau = f''\ \tau)$
using *vderiv-unique-within-open-interval ctHyp* **by** *blast*
also have (g *has-vderiv-on* (*vderiv-of* g $\{0<..< (c *_R t)\}$)) $\{0<..<c \cdot t\}$
by(*simp add*: *vderiv-of-def*, *metis derivHyp someI-ex*)
ultimately show*vderiv-of* g $\{0<..<c *_R t\}$ $t = f'\ t$ **using** *tHyp cHyp* **by** *force*
qed

lemma *vderiv-of-to-sol-its-vars*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp2*:$\forall\, t{\geq}0.$ (($\lambda\tau.$ (*sol s*[*xfList*$\leftarrow$*uInput*] τ) x)
has-vderiv-on ($\lambda\tau.$ f (*sol s*[*xfList*$\leftarrow$*uInput*] τ))) $\{0..t\}$
and *tHyp*: $t{>}0$ **and** *uxfHyp*:(u, x, f) $\in$ *set* (*uInput* $\otimes$ *xfList*)
shows *vderiv-of* ($\lambda\tau.$ u τ (*sol s*)) $\{0<..< (2 *_R t)\}$ $t = f$ (*sol s*[*xfList*$\leftarrow$*uInput*]
t)
apply(*rule-tac f*=($\lambda\tau.$ (*sol s*[*xfList*$\leftarrow$*uInput*] τ) x) **in** *closed-vderiv-on-cong-to-open-vderiv*)
subgoal using *assms* **and** *state-list-cross-upd-its-vars* **by** *metis*
by(*simp-all add*: *solHyp2 tHyp*)

lemma *inductive-to-sol-zero-its-dvars*:
assumes *eqFuncs*:$\forall\ s.\ \forall\, g.\ \forall$ *xf*$\in$*set* ((x, f) # *xfs*). π_2 *xf* (*override-on s g varDiffs*)
= π_2 *xf s*
and *eqLengths*:*length* ((x, f) # *xfs*) = *length* (u # *us*)
and *distinct*:*distinct* (*map* π_1 ((x, f) # *xfs*))
and *vars*:$\forall$ *xf*$\in$*set* ((x, f) # *xfs*). π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* ((u # *us*) $\otimes$ ((x, f) # *xfs*)). π_1 *uxf* 0 (*sol s*) = *sol s* (π_1
(π_2 *uxf*))
and *disjHyp*:(y, g) = (x, f) $\vee$ (y, g) $\in$ *set xfs*
and *indHyp*:(y, g) $\in$ *set xfs* $\Longrightarrow$ (*sol s*[*xfs*$\leftarrow$*us*] 0) (∂ y) = g (*sol s*[*xfs*$\leftarrow$*us*] 0)
shows (*sol s*[(x, f) # *xfs*$\leftarrow$$u$ # *us*] 0) (∂ y) = g (*sol s*[(x, f) # *xfs*$\leftarrow$$u$ # *us*] 0)
proof−
from *assms* **obtain** *h1* **where** *h1Def*:(*sol s*[((x, f) # *xfs*)$\leftarrow$(u # *us*)] 0) =
(*override-on* (*sol s*) *h1 varDiffs*) **using** *state-list-cross-upd-its-dvars* **by** *blast*
from *disjHyp* **show** (*sol s*[(x, f) # *xfs*$\leftarrow$$u$ # *us*] 0) (∂ y) = g (*sol s*[(x, f) #
xfs$\leftarrow$$u$ # *us*] 0)
proof
  assume *eqHeads*:(y, g) = (x, f)
  then have g (*sol s*[(x, f) # *xfs*$\leftarrow$$u$ # *us*] 0) = f (*sol s*) **using** *h1Def eqFuncs*
by *simp*
  also have ... = (*sol s*[(x, f) # *xfs*$\leftarrow$$u$ # *us*] 0) (∂ y) **using** *eqHeads* **by** *auto*
  ultimately show *?thesis* **by** *linarith*
next

assume *tailHyp*:$(y,\ g) \in$ *set xfs*
then have $y \neq x$ **using** *distinct set-zip-left-rightD* **by** *force*
hence $\partial\ x \neq \partial\ y$ **by**(*simp add*: *vdiff-def*)
have $x \neq \partial\ y$ **using** *vars vdiff-invarDiffs* **by** *auto*
obtain *h2* **where** *h2Def*:(*sol s*[*xfs*←*us*] *0*) = *override-on* (*sol s*) *h2 varDiffs*
using *state-list-cross-upd-its-dvars eqLengths distinct vars* **and** *solHyp1* **by** *force*
have (*sol s*[(x, f) # *xfs*←*u* # *us*] *0*) $(\partial\ y) = g$ (*sol s*[*xfs*←*us*] *0*)
using *tailHyp indHyp* ‹$x \neq \partial\ y$› **and** ‹$\partial\ x \neq \partial\ y$› **by** *simp*
also have ... = *g* (*override-on* (*sol s*) *h2 varDiffs*) **using** *h2Def* **by** *simp*
also have ... = *g* (*sol s*) **using** *eqFuncs* **and** *tailHyp* **by** *force*
also have ... = *g* (*sol s*[(x, f) # *xfs*←*u* # *us*] *0*)
using *eqFuncs h1Def tailHyp* **and** *eq-snd-iff* **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed
qed

lemma *to-sol-zero-its-dvars*:
assumes *funcsHyp*:∀ *s*. ∀ *g*. ∀ *xf* ∈ *set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
and *solHyp1*:∀ *uxf* ∈ *set* (*uInput* ⊗ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *ygHyp*:$(y,\ g) \in$ *set xfList*
shows (*sol s*[*xfList*←*uInput*] *0*)$(\partial\ y) = g$ (*sol s*[*xfList*←*uInput*] *0*)
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2′*, *simp*, *simp*, *simp*, *clarify*)
by(*rule inductive-to-sol-zero-its-dvars*, *simp-all*)

lemma *inductive-to-sol-greater-than-zero-its-dvars*:
assumes *lengthHyp*:*length* ((y, g) # *xfs*) = *length* (*v* # *vs*)
and *distHyp*:*distinct* (*map* π_1 ((y, g) # *xfs*))
and *varHyp*: ∀ *xf*∈*set* ((y, g) # *xfs*). π_1 *xf* ∉ *varDiffs*
and *indHyp*:(*u*,*x*,*f*) ∈ *set* (*vs* ⊗ *xfs*) ⟹ (*s*[*xfs*←*vs*]*t*)$(\partial\ x)$ = *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2*$*_R$*t*} *t*
and *disjHyp*:$(v,\ y,\ g) = (u,\ x,\ f) \vee (u,\ x,\ f) \in$ *set* (*vs* ⊗ *xfs*) **and** *tHyp*:$t > 0$
shows (*s*[(y, g) # *xfs*←*v* # *vs*] *t*) $(\partial\ x)$ = *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2* $*_R$ *t*} *t*
proof−
let *?lhs* = ((*s*[*xfs*←*vs*] *t*)(*y* := *v t s*, ∂ *y* := *vderiv-of* (λ *r*. *v r s*) {*0*<..< (*2* · *t*)} *t*)) $(\partial\ x)$
let *?rhs* = *vderiv-of* (λ*r*. *u r s*) {*0*<..< (*2* · *t*)} *t*
have (*s*[(y, g) # *xfs*←*v* # *vs*] *t*) $(\partial\ x)$ = *?lhs* **using** *tHyp* **by** *simp*
also have *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2* $*_R$ *t*} *t* =*?rhs* **by** *simp*
ultimately have *obs*:*?thesis* = (*?lhs* = *?rhs*) **by** *simp*
from *disjHyp* **have** *?lhs* = *?rhs*
proof
assume *uxfEq*:$(v,\ y,\ g) = (u,\ x,\ f)$
then have *?lhs* = *vderiv-of* (λ *r*. *u r s*) {*0*<..< (*2* · *t*)} *t* **by** *simp*

  also have *vderiv-of* $(\lambda\ r.\ u\ r\ s)\ \{0{<}..{<}\ (2 \cdot t)\}\ t = ?rhs$ **using** *uxfEq* **by** *simp*
  ultimately show $?lhs = ?rhs$ **by** *simp*
next
  assume *sygTail*:$(u, x, f) \in set\ (vs \otimes xfs)$
  from *this* **have** $y \neq x$ **using** *distHyp set-zip-left-rightD* **by** *force*
  hence $\partial\ x \neq \partial\ y$ **by**(*simp add*: *vdiff-def*)
  have $y \neq \partial\ x$ **using** *varHyp* **using** *vdiff-invarDiffs* **by** *auto*
  then have $?lhs = (s[xfs{\leftarrow}vs]\ t)\ (\partial\ x)$ **using** $\langle y \neq \partial\ x\rangle$ **and** $\langle \partial\ x \neq \partial\ y\rangle$ **by** *simp*
  also have $(s[xfs{\leftarrow}vs]\ t)\ (\partial\ x) = ?rhs$ **using** *indHyp sygTail* **by** *simp*
  ultimately show $?lhs = ?rhs$ **by** *simp*
qed
from *this* **and** *obs* **show** *?thesis* **by** *simp*
qed

lemma *to-sol-greater-than-zero-its-dvars*:
assumes *distinctHyp*:*distinct* $(map\ \pi_1\ xfList)$
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall xf \in set\ xfList.\ \pi_1\ xf \notin varDiffs$
and *uxfHyp*:$(u, x, f) \in set\ (uInput \otimes xfList)$ **and** *tHyp*:$t > 0$
shows $(s[xfList{\leftarrow}uInput]\ t)\ (\partial\ x) = vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0{<}..{<}\ (2 *_R t)\}\ t$
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2′*, *simp*, *simp*, *simp*, *clarify*)
by(*rule-tac f=f* **in** *inductive-to-sol-greater-than-zero-its-dvars*, *auto*)

dInv preliminaries

Here, we introduce syntactic notation to talk about differential invariants.

no-notation *Antidomain-Semiring.antidomain-left-monoid-class.am-add-op* (**infixl** $\oplus$ *65*)
no-notation *Dioid.times-class.opp-mult* (**infixl** $\odot$ *70*)
no-notation *Lattices.inf-class.inf* (**infixl** $\sqcap$ *70*)
no-notation *Lattices.sup-class.sup* (**infixl** $\sqcup$ *65*)

datatype *trms* = *Const real* (t_C - [*54*] *70*) | *Var string* (t_V - [*54*] *70*) |
    Mns trms ($\ominus$ - [*54*] *65*) | *Sum trms trms* (**infixl** $\oplus$ *65*) |
    Mult trms trms (**infixl** $\odot$ *68*)

primrec *tval* ::*trms* $\Rightarrow$ (*real store* $\Rightarrow$ *real*) $((1[\![\ \text{-}\ ]\!]_t))$ **where**
$[\![t_C\ r]\!]_t = (\lambda\ s.\ r)|$
$[\![t_V\ x]\!]_t = (\lambda\ s.\ s\ x)|$
$[\![\ominus\ \vartheta]\!]_t = (\lambda\ s.\ -\ ([\![\vartheta]\!]_t)\ s)|$
$[\![\vartheta \oplus \eta]\!]_t = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s + ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \odot \eta]\!]_t = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s \cdot ([\![\eta]\!]_t)\ s)$

datatype *props* = *Eq trms trms* (**infixr** $\doteq$ *60*) | *Less trms trms* (**infixr** $\prec$ *62*) |
    Leq trms trms (**infixr** $\preceq$ *61*) | *And props props* (**infixl** $\sqcap$ *63*) |
    Or props props (**infixl** $\sqcup$ *64*)

primrec *pval* ::*props* $\Rightarrow$ (*real store* $\Rightarrow$ *bool*) $((1[\![\text{-}]\!]_P))$ **where**

$[\![\vartheta \doteq \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s = ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \prec \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s < ([\![\eta]\!]_t)\ s)|$
$[\![\vartheta \preceq \eta]\!]_P = (\lambda\ s.\ ([\![\vartheta]\!]_t)\ s \leq ([\![\eta]\!]_t)\ s)|$
$[\![\varphi \sqcap \psi]\!]_P = (\lambda\ s.\ ([\![\varphi]\!]_P)\ s \wedge ([\![\psi]\!]_P)\ s)|$
$[\![\varphi \sqcup \psi]\!]_P = (\lambda\ s.\ ([\![\varphi]\!]_P)\ s \vee ([\![\psi]\!]_P)\ s)$

primrec *tdiff* ::*trms* $\Rightarrow$ *trms* (∂_t - [*54*] *70*) **where**
$(\partial_t\ t_C\ r) = t_C\ 0|$
$(\partial_t\ t_V\ x) = t_V\ (\partial\ x)|$
$(\partial_t \ominus \vartheta) = \ominus\ (\partial_t\ \vartheta)|$
$(\partial_t\ (\vartheta \oplus \eta)) = (\partial_t\ \vartheta) \oplus (\partial_t\ \eta)|$
$(\partial_t\ (\vartheta \odot \eta)) = ((\partial_t\ \vartheta) \odot \eta) \oplus (\vartheta \odot (\partial_t\ \eta))$

primrec *pdiff* ::*props* $\Rightarrow$ *props* (∂_P - [*54*] *70*) **where**
$(\partial_P\ (\vartheta \doteq \eta)) = ((\partial_t\ \vartheta) \doteq (\partial_t\ \eta))|$
$(\partial_P\ (\vartheta \prec \eta)) = ((\partial_t\ \vartheta) \preceq (\partial_t\ \eta))|$
$(\partial_P\ (\vartheta \preceq \eta)) = ((\partial_t\ \vartheta) \preceq (\partial_t\ \eta))|$
$(\partial_P\ (\varphi \sqcap \psi)) = (\partial_P\ \varphi) \sqcap (\partial_P\ \psi)|$
$(\partial_P\ (\varphi \sqcup \psi)) = (\partial_P\ \varphi) \sqcap (\partial_P\ \psi)$

primrec *trmVars* :: *trms* $\Rightarrow$ *string set* **where**
trmVars $(t_C\ r) = \{\}|$
trmVars $(t_V\ x) = \{x\}|$
trmVars $(\ominus\ \vartheta)$ = *trmVars* $\vartheta|$
trmVars $(\vartheta \oplus \eta)$ = *trmVars* $\vartheta \cup$ *trmVars* $\eta|$
trmVars $(\vartheta \odot \eta)$ = *trmVars* $\vartheta \cup$ *trmVars* η

fun *substList* ::(*string* $\times$ *trms*) *list* $\Rightarrow$ *trms* $\Rightarrow$ *trms* (-$\langle$-$\rangle$ [*54*] *80*) **where**
xtList$\langle t_C\ r\rangle = t_C\ r|$
$[]\langle t_V\ x\rangle = t_V\ x|$
$((y,\xi)$ # *xtTail*)$\langle$*Var x*$\rangle$ = (*if* $x = y$ *then* ξ *else xtTail*$\langle$*Var x*$\rangle$)|
xtList$\langle\ominus\ \vartheta\rangle = \ominus$ (*xtList*$\langle\vartheta\rangle$)|
xtList$\langle\vartheta \oplus \eta\rangle$ = (*xtList*$\langle\vartheta\rangle$) $\oplus$ (*xtList*$\langle\eta\rangle$)|
xtList$\langle\vartheta \odot \eta\rangle$ = (*xtList*$\langle\vartheta\rangle$) $\odot$ (*xtList*$\langle\eta\rangle$)

proposition *substList-on-compl-of-varDiffs*:
assumes *trmVars* $\eta \subseteq$ (*UNIV* $-$ *varDiffs*)
and *set* (*map* π_1 *xtList*) $\subseteq$ *varDiffs*
shows *xtList*$\langle\eta\rangle = \eta$
using *assms* **apply**(*induction* η, *simp-all add*: *varDiffs-def*)
by(*induction xtList*, *auto*)

lemma *substList-help1*:*set* (*map* π_1 ((*map* (*vdiff* $\circ$ π_1) *xfList*) $\otimes$ *uInput*)) $\subseteq$ *varDiffs*
apply(*induct xfList uInput rule*: *list-induct2'*, *simp-all add*: *varDiffs-def*)
by *auto*

lemma *substList-help2*:
assumes *trmVars* $\eta \subseteq$ (*UNIV* $-$ *varDiffs*)

shows $((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\eta\rangle = \eta$
using *assms substList-help1 substList-on-compl-of-varDiffs* **by** *blast*

lemma *substList-cross-vdiff-on-non-ocurring-var*:
assumes $x \notin set\ list1$
shows $((map\ vdiff\ list1) \otimes list2)\langle t_V\ (\partial\ x)\rangle = t_V\ (\partial\ x)$
using *assms* **apply**(*induct list1 list2 rule*: *list-induct2'*, *simp*, *simp*, *clarsimp*)
by(*simp add*: *vdiff-def*)

primrec *propVars* :: *props* $\Rightarrow$ *string set* **where**
$propVars\ (\vartheta \doteq \eta) = trmVars\ \vartheta \cup trmVars\ \eta|$
$propVars\ (\vartheta \prec \eta) = trmVars\ \vartheta \cup trmVars\ \eta|$
$propVars\ (\vartheta \preceq \eta) = trmVars\ \vartheta \cup trmVars\ \eta|$
$propVars\ (\varphi \sqcap \psi) = propVars\ \varphi \cup propVars\ \psi|$
$propVars\ (\varphi \sqcup \psi) = propVars\ \varphi \cup propVars\ \psi$

primrec *subspList* :: (*string* $\times$ *trms*) *list* $\Rightarrow$ *props* $\Rightarrow$ *props* (-$\upharpoonleft$-$\upharpoonleft$ [*54*] *80*) **where**
$xtList\upharpoonleft\vartheta \doteq \eta\upharpoonleft = ((xtList\langle\vartheta\rangle) \doteq (xtList\langle\eta\rangle))|$
$xtList\upharpoonleft\vartheta \prec \eta\upharpoonleft = ((xtList\langle\vartheta\rangle) \prec (xtList\langle\eta\rangle))|$
$xtList\upharpoonleft\vartheta \preceq \eta\upharpoonleft = ((xtList\langle\vartheta\rangle) \preceq (xtList\langle\eta\rangle))|$
$xtList\upharpoonleft\varphi \sqcap \psi\upharpoonleft = ((xtList\upharpoonleft\varphi\upharpoonleft) \sqcap (xtList\upharpoonleft\psi\upharpoonleft))|$
$xtList\upharpoonleft\varphi \sqcup \psi\upharpoonleft = ((xtList\upharpoonleft\varphi\upharpoonleft) \sqcup (xtList\upharpoonleft\psi\upharpoonleft))$

ODE Extras

For exemplification purposes, we compile some concrete derivatives used commonly in classical mechanics. A more general approach should be taken that generates this theorems as instantiations.

named-theorems *ubc-definitions definitions used in the locale unique-on-bounded-closed*

declare *unique-on-bounded-closed-def* [*ubc-definitions*]
  and *unique-on-bounded-closed-axioms-def* [*ubc-definitions*]
  and *unique-on-closed-def* [*ubc-definitions*]
  and *compact-interval-def* [*ubc-definitions*]
  and *compact-interval-axioms-def* [*ubc-definitions*]
  and *self-mapping-def* [*ubc-definitions*]
  and *self-mapping-axioms-def* [*ubc-definitions*]
  and *continuous-rhs-def* [*ubc-definitions*]
  and *closed-domain-def* [*ubc-definitions*]
  and *global-lipschitz-def* [*ubc-definitions*]
  and *interval-def* [*ubc-definitions*]
  and *nonempty-set-def* [*ubc-definitions*]
  and *lipschitz-on-def* [*ubc-definitions*]

named-theorems *poly-deriv temporal compilation of derivatives representing galilean transformations*
named-theorems *galilean-transform temporal compilation of vderivs representing galilean transformations*
named-theorems *galilean-transform-eq the equational version of galilean*$-$*transform*

lemma *vector-derivative-line-at-origin*:$((\cdot)\ a$ *has-vector-derivative* $a)$ (*at x within* T)
by (*auto intro*: *derivative-eq-intros*)

lemma [*poly-deriv*]:$((\cdot)\ a$ *has-derivative* $(\lambda x.\ x *_R a))$ (*at x within* T)
using *vector-derivative-line-at-origin* **unfolding** *has-vector-derivative-def* **by** *simp*

lemma *quadratic-monomial-derivative*:
$((\lambda t{::}real.\ a \cdot t^2)$ *has-derivative* $(\lambda t.\ a \cdot (2 \cdot x \cdot t)))$ (*at x within* T)
apply(*rule-tac* $g'1{=}\lambda\ t.\ 2 \cdot x \cdot t$ **in** *derivative-eq-intros*(*6*))
apply(*rule-tac* $f'1{=}\lambda\ t.\ t$ **in** *derivative-eq-intros*(*15*))
by (*auto intro*: *derivative-eq-intros*)

lemma *quadratic-monomial-derivative2*:
$((\lambda t{::}real.\ a \cdot t^2\ /\ 2)$ *has-derivative* $(\lambda t.\ a \cdot x \cdot t))$ (*at x within* T)
apply(*rule-tac* $f'1{=}\lambda t.\ a \cdot (2 \cdot x \cdot t)$ **and** $g'1{=}\lambda\ x.\ 0$ **in** *derivative-eq-intros*(*18*))
using *quadratic-monomial-derivative* **by** *auto*

lemma *quadratic-monomial-vderiv*[*poly-deriv*]:$((\lambda t.\ a \cdot t^2\ /\ 2)$ *has-vderiv-on* $(\cdot)$ $a)$ T
apply(*simp add*: *has-vderiv-on-def has-vector-derivative-def*, *clarify*)
using *quadratic-monomial-derivative2* **by** (*simp add*: *mult-commute-abs*)

lemma *galilean-position*[*galilean-transform*]:
$((\lambda t.\ a \cdot t^2\ /\ 2 + v \cdot t + x)$ *has-vderiv-on* $(\lambda t.\ a \cdot t + v))$ T
apply(*rule-tac* $f'{=}\lambda\ x.\ a \cdot x + v$ **and** $g'1{=}\lambda\ x.\ 0$ **in** *derivative-intros*(*191*))
apply(*rule-tac* $f'1{=}\lambda\ x.\ a \cdot x$ **and** $g'1{=}\lambda\ x.\ v$ **in** *derivative-intros*(*191*))
using *poly-deriv*(*2*) **by**(*auto intro*: *derivative-intros*)

lemma [*poly-deriv*]:
$t \in T \Longrightarrow ((\lambda\tau.\ a \cdot \tau^2\ /\ 2 + v \cdot \tau + x)$ *has-derivative* $(\lambda x.\ x *_R (a \cdot t + v)))$ (*at t within* T)
using *galilean-position* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **by** *simp*

lemma [*galilean-transform-eq*]:
$t > 0 \Longrightarrow$ *vderiv-of* $(\lambda t.\ a \cdot t\hat{}2\ /\ 2 + v \cdot t + x)\ \{0{<}..{<}2 \cdot t\}\ t = a \cdot t + v$
proof−
let $?f$ = *vderiv-of* $(\lambda t.\ a \cdot t\hat{}2\ /\ 2 + v \cdot t + x)\ \{0{<}..{<}2 \cdot t\}$
assume $t > 0$ **hence** $t \in \{0{<}..{<}2 \cdot t\}$ **by** *auto*
have $\exists\ f.\ ((\lambda t.\ a \cdot t^2\ /\ 2 + v \cdot t + x)$ *has-vderiv-on* $f)\ \{0{<}..{<}2 \cdot t\}$
using *galilean-position* **by** *blast*
hence $((\lambda t.\ a \cdot t\hat{}2\ /\ 2 + v \cdot t + x)$ *has-vderiv-on* $?f)\ \{0{<}..{<}2 \cdot t\}$
unfolding *vderiv-of-def* **by** (*metis* (*mono-tags*, *lifting*) *someI-ex*)
also have $((\lambda t.\ a \cdot t^2\ /\ 2 + v \cdot t + x)$ *has-vderiv-on* $(\lambda t.\ a \cdot t + v))\ \{0{<}..{<}2 \cdot t\}$
using *galilean-position* **by** *simp*
ultimately show (*vderiv-of* $(\lambda t.\ a \cdot t\hat{}2\ /\ 2 + v \cdot t + x)\ \{0{<}..{<}2 \cdot t\})\ t = a \cdot$

$t + v$
apply(*rule-tac f'=?f* **and** τ=*t* **and** *t*=*2*·*t* **in** *vderiv-unique-within-open-interval*)
using ‹$t \in \{0<..<2 \cdot t\}$› **by** *auto*
qed

lemma $t > 0 \Longrightarrow$ *vderiv-of* ($\lambda t.\ a \cdot t\hat{}2\ /\ 2 + v \cdot t + x$) $\{0<..<2 \cdot t\}$ $t = a \cdot t$ $+ v$
unfolding *vderiv-of-def* **apply**(*subst some1-equality*[*of* - ($\lambda t.\ a \cdot t + v$)])
apply(*rule-tac a*=$\lambda t.\ a \cdot t + v$ **in** *ex1I*)
apply(*simp-all add*: *galilean-position*)
apply(*rule ext*, *rename-tac f* τ)
apply(*rule-tac f*=$\lambda t.\ a \cdot t^2\ /\ 2 + v \cdot t + x$ **and** *t*=*2* · *t* **and** *f'*=*f* **in** *vderiv-unique-within-open-interval*)
apply(*simp-all add*: *galilean-position*)
oops

lemma *galilean-velocity*[*galilean-transform*]:(($\lambda r.\ a \cdot r + v$) *has-vderiv-on* ($\lambda t.\ a$)) *T*
apply(*rule-tac f'1*=λ *x. a* **and** *g'1*=λ *x. 0* **in** *derivative-intros*(*191*))
unfolding *has-vderiv-on-def* **by**(*auto intro*: *derivative-eq-intros*)

lemma [*galilean-transform-eq*]:
$t > 0 \Longrightarrow$ *vderiv-of* ($\lambda r.\ a \cdot r + v$) $\{0<..<2 \cdot t\}$ $t = a$
proof−
let *?f* = *vderiv-of* ($\lambda r.\ a \cdot r + v$) $\{0<..<2 \cdot t\}$
assume $t > 0$ **hence** $t \in \{0<..<2 \cdot t\}$ **by** *auto*
have $\exists$ *f*. (($\lambda r.\ a \cdot r + v$) *has-vderiv-on f*) $\{0<..<2 \cdot t\}$
using *galilean-velocity* **by** *blast*
hence (($\lambda r.\ a \cdot r + v$) *has-vderiv-on ?f*) $\{0<..<2 \cdot t\}$
unfolding *vderiv-of-def* **by** (*metis* (*mono-tags*, *lifting*) *someI-ex*)
also have (($\lambda r.\ a \cdot r + v$) *has-vderiv-on* ($\lambda t.\ a$)) $\{0<..<2 \cdot t\}$
using *galilean-velocity* **by** *simp*
ultimately show (*vderiv-of* ($\lambda r.\ a \cdot r + v$) $\{0<..<2 \cdot t\}$) $t = a$
apply(*rule-tac f'=?f* **and** τ=*t* **and** *t*=*2*·*t* **in** *vderiv-unique-within-open-interval*)
using ‹$t \in \{0<..<2 \cdot t\}$› **by** *auto*
qed

lemma [*galilean-transform*]:
(($\lambda t.\ v \cdot t - a \cdot t^2\ /\ 2 + x$) *has-vderiv-on* ($\lambda x.\ v - a \cdot x$)) $\{0..t\}$
apply(*subgoal-tac* (($\lambda t.\ -\ a \cdot t^2\ /\ 2 + v \cdot t\ + x$) *has-vderiv-on* ($\lambda x.\ -\ a \cdot x +$ v)) $\{0..t\}$, *simp*)
by(*rule galilean-transform*)

lemma [*galilean-transform-eq*]:$t > 0 \Longrightarrow$ *vderiv-of* ($\lambda t.\ v \cdot t - a \cdot t\hat{}2\ /\ 2 + x$) $\{0<..<2 \cdot t\}$ $t = v - a \cdot t$
apply(*subgoal-tac vderiv-of* ($\lambda t.\ -\ a \cdot t^2\ /\ 2 + v \cdot t + x$) $\{0<..<2 \cdot t\}$ $t = -\ a$ $\cdot\ t + v$, *simp*)
by(*rule galilean-transform-eq*)

lemma [*galilean-transform*]:
$((\lambda t.\ v - a \cdot t)$ *has-vderiv-on* $(\lambda x.\ - a))\ \{0..t\}$
apply(*subgoal-tac* $((\lambda t.\ - a \cdot t + v)$ *has-vderiv-on* $(\lambda x.\ - a))\ \{0..t\}$, *simp*)
by(*rule galilean-transform*)

lemma [*galilean-transform-eq*]:$t > 0 \Longrightarrow$ *vderiv-of* $(\lambda r.\ v - a \cdot r)\ \{0<..<2 \cdot t\}$
$t = - a$
apply(*subgoal-tac vderiv-of* $(\lambda t.\ - a \cdot t + v)\ \{0<..<2 \cdot t\}\ t = - a$, *simp*)
by(*rule galilean-transform-eq*)

lemma [*simp*]:$(\lambda x.$ *case* x *of* $(t, x) \Rightarrow f\ t) = (\lambda\ x.\ (f \circ \pi_1)\ x)$
by *auto*

end
theory *VC-diffKAD*
imports *VC-diffKAD-auxiliarities*

begin

5.4.3 Phase Space Relational Semantics

definition *solvesStoreIVP* :: (*real* $\Rightarrow$ *real store*) $\Rightarrow$ (*string* $\times$ (*real store* $\Rightarrow$ *real*)) *list* $\Rightarrow$
real store $\Rightarrow$ *bool*
((- *solvesTheStoreIVP* - *withInitState* -) [*70*, *70*, *70*] *68*) **where**
solvesStoreIVP φ_S *xfList* $s \equiv$
— F sends vdiffs-in-list to derivs.
$(\forall\ t \geq 0.\ (\forall\ xf \in$ *set xfList*. $\varphi_S\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (\varphi_S\ t))\ \wedge$
— F preserves the rest of the variables and F sends derivs of constants to 0.
$(\forall\ y.\ (y \notin (\pi_1 (\!|$*set xfList*$|\!)) \cup$ *varDiffs* $\longrightarrow \varphi_S\ t\ y = s\ y)\ \wedge$
$(y \notin (\pi_1 (\!|$*set xfList*$|\!)) \longrightarrow \varphi_S\ t\ (\partial\ y) = 0))\ \wedge$
— F solves the induced IVP.
$(\forall\ xf \in$ *set xfList*. $((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))$ *solves-ode* $(\lambda\ t.\lambda\ r.(\pi_2\ xf)\ (\varphi_S\ t)))\ \{0..t\}$
UNIV $\wedge$
$\varphi_S\ 0\ (\pi_1\ xf) = s(\pi_1\ xf)))$

lemma *solves-store-ivpI*:
assumes $\forall\ t \geq 0.\forall\ xf \in$ *set xfList*. $(\varphi_S\ t\ (\partial\ (\pi_1\ xf))) = (\pi_2\ xf)\ (\varphi_S\ t)$
and $\forall\ t \geq 0.\forall\ y.\ y \notin (\pi_1 (\!|$*set xfList*$|\!))\cup$*varDiffs* $\longrightarrow \varphi_S\ t\ y = s\ y$
and $\forall\ t \geq 0.\forall\ y.\ y \notin (\pi_1 (\!|$*set xfList*$|\!)) \longrightarrow \varphi_S\ t\ (\partial\ y) = 0$
and $\forall\ t \geq 0.\ \forall\ xf \in$ *set xfList*. $((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))$ *solves-ode* $(\lambda\ t.\lambda\ r.(\pi_2\ xf)$
$(\varphi_S\ t)))\ \{0..t\}$ *UNIV*
and $\forall\ xf \in$ *set xfList*. $\varphi_S\ 0\ (\pi_1\ xf) = s(\pi_1\ xf)$
shows φ_S *solvesTheStoreIVP xfList withInitState s*
apply(*simp add*: *solvesStoreIVP-def*, *safe*)
using *assms* **apply** *simp-all*
by(*force*,*force*,*force*)

named-theorems *solves-store-ivpE elimination rules for solvesStoreIVP*

lemma [*solves-store-ivpE*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
shows $\forall\ t \geq 0.\forall\ y.\ y \notin (\pi_1 (\!|set\ xfList|\!)) \cup varDiffs \longrightarrow \varphi_S\ t\ y = s\ y$
and $\forall\ t \geq 0.\forall\ y.\ y \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow \varphi_S\ t\ (\partial\ y) = 0$
and $\forall\ t \geq 0.\forall\ xf \in set\ xfList.\ (\varphi_S\ t\ (\partial\ (\pi_1\ xf))) = (\pi_2\ xf)\ (\varphi_S\ t)$
and $\forall\ t \geq 0.\ \forall\ xf \in set\ xfList.\ ((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))$ *solves-ode* $(\lambda\ t.\lambda\ r.(\pi_2\ xf)\ (\varphi_S\ t)))\ \{0..t\}\ UNIV$
and $\forall\ xf \in set\ xfList.\ \varphi_S\ 0\ (\pi_1\ xf) = s(\pi_1\ xf)$
using *assms solvesStoreIVP-def* **by** *auto*

lemma [*solves-store-ivpE*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
shows $\forall\ y.\ y \notin varDiffs \longrightarrow \varphi_S\ 0\ y = s\ y$
proof(*clarify*, *rename-tac x*)
fix x **assume** $x \notin varDiffs$
from *assms* **and** *solves-store-ivpE*(*5*) **have** $x \in (\pi_1 (\!|set\ xfList|\!)) \Longrightarrow \varphi_S\ 0\ x = s\ x$ **by** *fastforce*
also have $x \notin (\pi_1 (\!|set\ xfList|\!)) \cup varDiffs \Longrightarrow \varphi_S\ 0\ x = s\ x$
using *assms* **and** *solves-store-ivpE*(*1*) **by** *simp*
ultimately show $\varphi_S\ 0\ x = s\ x$ **using** ‹$x \notin varDiffs$› **by** *auto*
qed

named-theorems *solves-store-ivpD computation rules for solvesStoreIVP*

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $t \geq 0$
and $y \notin (\pi_1 (\!|set\ xfList|\!)) \cup varDiffs$
shows $\varphi_S\ t\ y = s\ y$
using *assms solves-store-ivpE*(*1*) **by** *simp*

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $t \geq 0$
and $y \notin (\pi_1 (\!|set\ xfList|\!))$
shows $\varphi_S\ t\ (\partial\ y) = 0$
using *assms solves-store-ivpE*(*2*) **by** *simp*

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $t \geq 0$
and $xf \in set\ xfList$
shows $(\varphi_S\ t\ (\partial\ (\pi_1\ xf))) = (\pi_2\ xf)\ (\varphi_S\ t)$
using *assms solves-store-ivpE*(*3*) **by** *simp*

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $t \geq 0$

and $xf \in$ *set xfList*
shows $((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))$ *solves-ode* $(\lambda\ t.\lambda\ r.(\pi_2\ xf)\ (\varphi_S\ t)))\ \{0..t\}$ *UNIV*
using *assms solves-store-ivpE*(4) **by** *simp*

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $(x,f) \in$ *set xfList*
shows $\varphi_S\ 0\ x = s\ x$
using *assms solves-store-ivpE*(5) **by** *fastforce*

lemma [*solves-store-ivpD*]:
assumes φ_S *solvesTheStoreIVP xfList withInitState s*
and $y \notin$ *varDiffs*
shows $\varphi_S\ 0\ y = s\ y$
using *assms solves-store-ivpE*(6) **by** *simp*

definition *guarDiffEqtn* :: (*string* $\times$ (*real store* $\Rightarrow$ *real*)) *list* $\Rightarrow$ (*real store pred*) $\Rightarrow$
real store rel (*ODEsystem - with -* [70, 70] 61) **where**
ODEsystem xfList with $G = \{(s,\varphi_S\ t)\ |s\ t\ \varphi_S.\ t \geq 0 \wedge (\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r))$
$\wedge$ *solvesStoreIVP* φ_S *xfList* $s\}$

5.4.4 Derivation of Differential Dynamic Logic Rules

"Differential Weakening"

lemma *wlp-evol-guard*:*Id* $\subseteq$ *wp* (*ODEsystem xfList with G*) $\lceil G \rceil$
by(*simp add*: *rel-antidomain-kleene-algebra.fbox-def rel-ad-def guarDiffEqtn-def p2r-def*, *force*)

theorem *dWeakening*:
assumes *guardImpliesPost*: $\lceil G \rceil \subseteq \lceil Q \rceil$
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
using *assms* **and** *wlp-evol-guard* **by** (*metis* (*no-types*, *hide-lams*) *d-p2r order-trans p2r-subid rel-antidomain-kleene-algebra.fbox-iso*)

theorem *dW*: *wp* (*ODEsystem xfList with G*) $\lceil Q \rceil$ = *wp* (*ODEsystem xfList with G*) $\lceil \lambda s.\ G\ s \longrightarrow Q\ s \rceil$
unfolding *rel-antidomain-kleene-algebra.fbox-def rel-ad-def guarDiffEqtn-def*
by(*simp add*: *relcomp.simps p2r-def*, *fastforce*)

"Differential Cut"

lemma *all-interval-guarDiffEqtn*:
assumes *solvesStoreIVP* φ_S *xfList* $s \wedge (\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r)) \wedge 0 \leq t$
shows $\forall\ r \in \{0..t\}.\ (s,\ \varphi_S\ r) \in$ (*ODEsystem xfList with G*)
unfolding *guarDiffEqtn-def* **using** *atLeastAtMost-iff* **apply** *clarsimp*
apply(*rule-tac x=r* **in** *exI*, *rule-tac* $x{=}\varphi_S$ **in** *exI*) **using** *assms* **by** *simp*

lemma *condAfterEvol-remainsAlongEvol*:

assumes *boxDiffC*:$(s, s) \in wp$ (*ODEsystem xfList with G*) $\lceil C \rceil$
and *FisSol*:*solvesStoreIVP* φ_S *xfList s* $\wedge$ ($\forall\ r \in \{0..t\}$. *G* (φ_S *r*)) $\wedge$ $0 \leq t$
shows $\forall\ r \in \{0..t\}$. *G* (φ_S *r*) $\wedge$ *C* (φ_S *r*)
proof−
from *boxDiffC* **have** $\forall\ c.\ (s,c) \in$ (*ODEsystem xfList with G*) $\longrightarrow$ *C c*
  by (*simp add*: *boxProgrPred-chrctrztn*)
also from *FisSol* **have** $\forall\ r \in \{0..t\}$. (s, φ_S *r*) $\in$ (*ODEsystem xfList with G*)
  using *all-interval-guarDiffEqtn* **by** *blast*
ultimately show *?thesis*
  using *FisSol atLeastAtMost-iff guarDiffEqtn-def* **by** *fastforce*
qed

theorem *dCut*:
assumes *pBoxDiffCut*:(*PRE P* (*ODEsystem xfList with G*) *POST C*)
assumes *pBoxCutQ*:(*PRE P* (*ODEsystem xfList with* (λ *s. G s* $\wedge$ *C s*)) *POST Q*)
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*clarify*, *subgoal-tac a* = *b*) **defer**
proof(*metis d-p2r rdom-p2r-contents*, *simp*, *subst boxProgrPred-chrctrztn*, *clarify*)
fix *b y* **assume** $(b, b) \in \lceil P \rceil$ **and** $(b, y) \in$ *ODEsystem xfList with G*
then obtain φ_S *t* **where** $*$:*solvesStoreIVP* φ_S *xfList b* $\wedge$ ($\forall\ r \in \{0..t\}$. *G* (φ_S *r*)) $\wedge$ $0 \leq t \wedge \varphi_S\ t = y$
  using *guarDiffEqtn-def* **by** *auto*
hence $\forall\ r \in \{0..t\}$. (b, φ_S *r*) $\in$ (*ODEsystem xfList with G*)
  using *all-interval-guarDiffEqtn* **by** *blast*
from *this* **and** *pBoxDiffCut* **have** $\forall\ r \in \{0..t\}$. *C* (φ_S *r*)
  using *boxProgrPred-chrctrztn* ‹$(b, b) \in \lceil P \rceil$› **by** (*metis* (*no-types*, *lifting*) *d-p2r subsetCE*)
then have $\forall\ r \in \{0..t\}$. (b, φ_S *r*) $\in$ (*ODEsystem xfList with* (λ *s. G s* $\wedge$ *C s*))
  using $*$ *all-interval-guarDiffEqtn* **by** (*metis* (*mono-tags*, *lifting*))
from *this* **and** *pBoxCutQ* **have** $\forall\ r \in \{0..t\}$. *Q* (φ_S *r*)
  using *boxProgrPred-chrctrztn* ‹$(b, b) \in \lceil P \rceil$› **by** (*metis* (*no-types*, *lifting*) *d-p2r subsetCE*)
thus *Q y* **using** $*$ **by** *auto*
qed

theorem *dC*:
assumes *Id* $\subseteq$ *wp* (*ODEsystem xfList with G*) $\lceil C \rceil$
shows *wp* (*ODEsystem xfList with G*) $\lceil Q \rceil$ = *wp* (*ODEsystem xfList with* (λ *s. G s* $\wedge$ *C s*)) $\lceil Q \rceil$
proof(*rule-tac f*=λ *x. wp x* $\lceil Q \rceil$ **in** *HOL.arg-cong*, *safe*)
  fix *a b* **assume** $(a, b) \in$ *ODEsystem xfList with G*
  then obtain φ_S *t* **where** $*$:*solvesStoreIVP* φ_S *xfList a* $\wedge$ ($\forall\ r \in \{0..t\}$. *G* (φ_S *r*)) $\wedge$ $0 \leq t \wedge \varphi_S\ t = b$
    using *guarDiffEqtn-def* **by** *auto*
  hence *1*:$\forall\ r \in \{0..t\}$. (a, φ_S *r*) $\in$ *ODEsystem xfList with G*
    by (*meson all-interval-guarDiffEqtn*)
  from *this* **have** $\forall\ r \in \{0..t\}$. *C* (φ_S *r*) **using** *assms boxProgrPred-chrctrztn*
    by (*metis IdI boxProgrPred-IsProp subset-antisym*)
  thus $(a, b) \in$ *ODEsystem xfList with* (λ*s. G s* $\wedge$ *C s*)

using $*$ *guarDiffEqtn-def* **by** *blast*
next
fix *a b* **assume** $(a,\ b) \in$ *ODEsystem xfList with* $(\lambda s.\ G\ s \wedge C\ s)$
then show $(a,\ b) \in$ *ODEsystem xfList with G*
unfolding *guarDiffEqtn-def* **by**(*clarsimp*, *rule-tac* $x=t$ **in** *exI*, *rule-tac* $x=\varphi_S$ **in** *exI*, *simp*)
qed

Solve Differential Equation

lemma *prelim-dSolve*:
assumes *solHyp*:$(\lambda t.\ sol\ s[xfList \leftarrow uInput]\ t)$ *solvesTheStoreIVP xfList withInitState s*
and *uniqHyp*:$\forall\ X.$ *solvesStoreIVP X xfList s* $\longrightarrow (\forall\ t \geq 0.\ (sol\ s[xfList \leftarrow uInput]\ t) = X\ t)$
and *diffAssgn*: $\forall\, t \geq 0.\ G\ (sol\ s[xfList \leftarrow uInput]\ t) \longrightarrow Q\ (sol\ s[xfList \leftarrow uInput]\ t)$
shows $\forall\ c.\ (s,c) \in$ (*ODEsystem xfList with G*) $\longrightarrow Q\ c$
proof(*clarify*)
fix *c* **assume** $(s,c) \in$ (*ODEsystem xfList with G*)
from *this* **obtain** *t*::*real* **and** φ_S::*real* $\Rightarrow$ *real store*
where *FHyp*:$t \geq 0 \wedge \varphi_S\ t = c \wedge$ *solvesStoreIVP* φ_S *xfList s* $\wedge (\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r))$
using *guarDiffEqtn-def* **by** *auto*
from *this* **and** *uniqHyp* **have** $(sol\ s[xfList \leftarrow uInput]\ t) = \varphi_S\ t$ **by** *blast*
then have *cHyp*:$c = (sol\ s[xfList \leftarrow uInput]\ t)$ **using** *FHyp* **by** *simp*
from *this* **have** $G\ (sol\ s[xfList \leftarrow uInput]\ t)$ **using** *FHyp* **by** *force*
then show *Q c* **using** *diffAssgn FHyp cHyp* **by** *auto*
qed

theorem *dS*:
assumes *solHyp*:$\forall\ s.$ *solvesStoreIVP* $(\lambda t.\ sol\ s[xfList \leftarrow uInput]\ t)$ *xfList s*
and *uniqHyp*:$\forall\ s\ X.$ *solvesStoreIVP X xfList s* $\longrightarrow (\forall\ t \geq 0.\ (sol\ s[xfList \leftarrow uInput]\ t) = X\ t)$
shows *wp* (*ODEsystem xfList with G*) $\lceil Q \rceil =$
$\lceil \lambda\ s.\ \forall\, t \geq 0.\ (\forall\, r \in \{0..t\}.\ G\ (sol\ s[xfList \leftarrow uInput]\ r)) \longrightarrow Q\ (sol\ s[xfList \leftarrow uInput]\ t) \rceil$
apply(*simp add*: *p2r-def*, *rule subset-antisym*)
unfolding *guarDiffEqtn-def rel-antidomain-kleene-algebra.fbox-def rel-ad-def*
using *solHyp* **apply**(*simp add*: *relcomp.simps*) **apply** *clarify*
apply(*rule-tac* $x=x$ **in** *exI*, *clarsimp*)
apply(*erule-tac* $x=sol\ x[xfList \leftarrow uInput]\ t$ **in** *allE*, *erule disjE*)
apply(*erule-tac* $x=x$ **in** *allE*, *erule-tac* $x=t$ **in** *allE*)
apply(*erule impE*, *simp*, *erule-tac* $x=\lambda t.\ sol\ x[xfList \leftarrow uInput]\ t$ **in** *allE*)
apply(*simp-all*, *clarify*, *rule-tac* $x=s$ **in** *exI*, *simp add*: *relcomp.simps*)
using *uniqHyp* **by** *fastforce*

theorem *dSolve*:
assumes *solHyp*:$\forall\, s.$ *solvesStoreIVP* $(\lambda t.\ sol\ s[xfList \leftarrow uInput]\ t)$ *xfList s*
and *uniqHyp*:$\forall\ s.\ \forall\ X.$ *solvesStoreIVP X xfList s* $\longrightarrow (\forall\ t \geq 0.(sol\ s[xfList \leftarrow uInput]$

t) = *X t*)
and *diffAssgn*: ∀ *s*. *P s* ⟶ (∀ *t*≥*0*. *G* (*sol s*[*xfList*←*uInput*] *t*) ⟶ *Q* (*sol s*[*xfList*←*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*clarsimp*, *subgoal-tac a*=*b*)
apply(*clarify*, *subst boxProgrPred-chrctrztn*)
apply(*simp-all add*: *p2r-def*)
apply(*rule-tac uInput*=*uInput* **in** *prelim-dSolve*)
apply(*simp add*: *solHyp*, *simp add*: *uniqHyp*)
by (*metis* (*no-types*, *lifting*) *diffAssgn*)

— We proceed to refine the previous rule by finding the necessary restrictions on varFunList and uInput so that the solution to the store-IVP is guaranteed.

lemma *conds4vdiffs-prelim*:
assumes *funcsHyp*:∀ *s g*. ∀ *xf*∈*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*:∀ *xf* ∈ *set xfList*. π_1 *xf* ∉ *varDiffs*
and *lengthHyp*:*length xfList* = *length uInput*
and *solHyp1*:∀ *uxf* ∈ *set* (*uInput* ⊗ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:∀ *t*≥*0*. ((λτ. (*sol s*[*xfList*←*uInput*] τ) *x*)
has-vderiv-on (λτ. *f* (*sol s*[*xfList*←*uInput*] τ))) {*0*..*t*}
and *xfHyp*:(*x*, *f*) ∈ *set xfList* **and** *tHyp*:*t* ≥ *0*
shows (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *f* (*sol s*[*xfList*←*uInput*] *t*)
proof−
from *xfHyp* **obtain** *u* **where** *xfuHyp*: (*u*,*x*,*f*) ∈ *set* (*uInput* ⊗ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
show (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) =*f* (*sol s*[*xfList*←*uInput*] *t*)
  proof(*cases t*=*0*)
  case *True*
    have (*sol s*[*xfList*←*uInput*] *0*) (∂ *x*) = *f* (*sol s*[*xfList*←*uInput*] *0*)
    using *assms* **and** *to-sol-zero-its-dvars* **by** *blast*
    then show *?thesis* **using** *True* **by** *blast*
  next
    case *False*
    from *this* **have** *t* > *0* **using** *tHyp* **by** *simp*
    hence (*sol s*[*xfList*←*uInput*] *t*) (∂ *x*) = *vderiv-of* (λ *r*. *u r* (*sol s*)) {*0*<..< (*2* $*_R$ *t*)} *t*
    using *xfuHyp assms to-sol-greater-than-zero-its-dvars* **by** *blast*
   also have *vderiv-of* (λ*r*. *u r* (*sol s*)) {*0*<..< (*2* $*_R$ *t*)} *t* = *f* (*sol s*[*xfList*←*uInput*] *t*)
    using *assms xfuHyp* ‹*t* > *0*› **and** *vderiv-of-to-sol-its-vars* **by** *blast*
    ultimately show *?thesis* **by** *simp*
  qed
qed

lemma *conds4vdiffs*:

assumes *funcsHyp*:$\forall$ *s g*. $\forall$ *xf*$\in$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *lengthHyp*:*length xfList* = *length uInput*
and *solHyp1*:$\forall$ *uxf* $\in$ *set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ *t*$\geq$*0*. $\forall$ *xf* $\in$ *set xfList*. (($\lambda\tau$. (*sol s*[*xfList*$\leftarrow$*uInput*] τ) (π_1 *xf*)) *has-vderiv-on* ($\lambda\tau$. (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] τ))) {*0..t*}
shows $\forall$ *t* $\geq$ *0*. $\forall$ *xf* $\in$ *set xfList*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (∂ (π_1 *xf*)) = (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *t*)
apply(*rule allI*, *rule impI*, *rule ballI*, *rule conds4vdiffs-prelim*)
using *assms* **by** *simp-all*

lemma *conds4Consts*:
assumes *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
shows $\forall$ *x*. *x* $\notin$ (π_1(|*set xfList*|)) $\longrightarrow$ (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (∂ *x*) = *0*
using *varsHyp* **apply**(*induct xfList uInput rule*: *list-induct2'*)
apply(*simp-all add*: *override-on-def varDiffs-def vdiff-def*)
by *clarsimp*

lemma *conds4InitState*:
assumes *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *xfHyp*:(*x*, *f*) $\in$ *set xfList*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) *x* = *s x*
proof−
from *xfHyp* **obtain** *u* **where** *uxfHyp*:(*u*, *x*, *f*) $\in$ *set* (*uInput* $\otimes$ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
from *varsHyp* **have** *toZeroHyp*:(*sol s*) *x* = *s x* **using** *override-on-def xfHyp* **by** *auto*
from *uxfHyp* **and** *solHyp1* **have** *u 0* (*sol s*) = (*sol s*) *x* **by** *fastforce*
also have (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) *x* = *u 0* (*sol s*)
using *state-list-cross-upd-its-vars uxfHyp* **and** *assms* **by** *blast*
ultimately show (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) *x* = *s x* **using** *toZeroHyp* **by** *simp*
qed

lemma *conds4RestOfStrings*:
assumes *x* $\notin$ (π_1(|*set xfList*|)) $\cup$ *varDiffs*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = *s x*
using *assms* **apply**(*induct xfList uInput rule*: *list-induct2'*)
by(*auto simp*: *varDiffs-def*)

lemma *conds4storeIVP-on-toSol*:
assumes *funcsHyp*:$\forall$ *s g*. $\forall$ *xf*$\in$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*

and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ *t* $\geq$ *0*. $\forall$ *xf* $\in$ *set xfList*.
((λt. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λt. π_2 *xf* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))) {*0*..*t*}
shows *solvesStoreIVP* (λ *t*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*)) *xfList s*
apply(*rule solves-store-ivpI*)
subgoal using *conds4vdiffs assms* **by** *blast*
subgoal using *conds4RestOfStrings* **by** *blast*
subgoal using *conds4Consts varsHyp* **by** *blast*
subgoal apply(*rule allI*, *rule impI*, *rule ballI*, *rule solves-odeI*)
 using *solHyp2* **by** *simp-all*
subgoal using *conds4InitState* **and** *assms* **by** *force*
done

theorem *dSolve-toSolve*:
assumes *funcsHyp*:$\forall$ *s g*. $\forall$ *xf*$\in$*set xfList*. π_2 *xf* (*override-on s g varDiffs*) = π_2 *xf s*
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *solHyp1*:$\forall$ *s*.$\forall$ *uxf*$\in$*set* (*uInput* $\otimes$ *xfList*). (π_1 *uxf*) *0* (*sol s*) = (*sol s*) (π_1 (π_2 *uxf*))
and *solHyp2*:$\forall$ *s*.$\forall$ *t* $\geq$ *0*. $\forall$ *xf* $\in$ *set xfList*.
((λt. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λt. π_2 *xf* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))) {*0*..*t*}
and *uniqHyp*:$\forall$ *s*.$\forall$ *X*. *solvesStoreIVP X xfList s* $\longrightarrow$ ($\forall$ *t* $\geq$ *0*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = *X t*)
and *postCondHyp*:$\forall$ *s*. *P s* $\longrightarrow$ ($\forall$ *t*$\geq$*0*. *Q* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput*=*uInput* **in** *dSolve*)
subgoal using *assms* **and** *conds4storeIVP-on-toSol* **by** *simp*
subgoal by (*simp add*: *uniqHyp*)
using *postCondHyp postCondHyp* **by** *simp*

— As before, we keep refining the rule dSolve. This time we find the necessary restrictions to attain uniqueness.

lemma *conds4UniqSol*:
fixes *f*::*real store* $\Rightarrow$ *real*
assumes *tHyp*:*t* $\geq$ *0*
and *contHyp*:*continuous-on* ({*0*..*t*} $\times$ *UNIV*) (λ(*t*, (*r*::*real*)). *f* (φ_s *t*))
shows *unique-on-bounded-closed 0* {*0*..*t*} τ (λ*t r*. *f* (φ_s *t*)) *UNIV* (*if t* = *0 then 1 else 1*/(*t*+*1*))
apply(*simp add*: *ubc-definitions*, *rule conjI*)
subgoal using *contHyp continuous-rhs-def* **by** *fastforce*

subgoal using *assms continuous-rhs-def* **by** *fastforce*
done

lemma *solves-store-ivp-at-beginning-overrides*:
assumes *solvesStoreIVP* φ_s *xfList a*
shows φ_s *0* = *override-on a* (φ_s *0*) *varDiffs*
apply(*rule ext*, *subgoal-tac x* $\notin$ *varDiffs* $\longrightarrow$ φ_s *0 x* = *a x*)
subgoal by (*simp add*: *override-on-def*)
using *assms* **and** *solves-store-ivpD*(*6*) **by** *simp*

lemma *ubcStoreUniqueSol*:
assumes *tHyp*:*t* $\geq$ *0*
assumes *contHyp*:$\forall$ *xf* $\in$ *set xfList*. *continuous-on* ({*0..t*} $\times$ *UNIV*)
(λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] *t*))
and *eqDerivs*:$\forall$ *xf* $\in$ *set xfList*. $\forall$ τ $\in$ {*0..t*}. (π_2 *xf*) (φ_s τ) = (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] τ)
and *Fsolves*:*solvesStoreIVP* φ_s *xfList s*
and *solHyp*:*solvesStoreIVP* (λ τ. (*sol s*[*xfList*$\leftarrow$*uInput*] τ)) *xfList s*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = φ_s *t*
proof
  fix *x*::*string* **show** (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = φ_s *t x*
  proof(*cases x* $\in$ (π_1(|*set xfList*|)) $\cup$ *varDiffs*)
  case *False*
    then have *notInVars*:*x* $\notin$ (π_1(|*set xfList*|)) $\cup$ *varDiffs* **by** *simp*
    from *solHyp* **have** (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = *s x*
    using *tHyp notInVars solves-store-ivpD*(*1*) **by** *blast*
   also from *Fsolves* **have** φ_s *t x* = *s x* **using** *tHyp notInVars solves-store-ivpD*(*1*) **by** *blast*
    ultimately show (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) *x* = φ_s *t x* **by** *simp*
  next case *True*
    then have *x* $\in$ (π_1(|*set xfList*|)) $\vee$ *x* $\in$ *varDiffs* **by** *simp*
    from *this* **show** *?thesis*
    proof
      assume *x* $\in$ (π_1(|*set xfList*|))
      from *this* **obtain** *f* **where** *xfHyp*:(*x*, *f*) $\in$ *set xfList* **by** *fastforce*

      then have *expand1*:$\forall$ *xf* $\in$ *set xfList*.(($\lambda\tau$. φ_s τ (π_1 *xf*)) *solves-ode*
      ($\lambda\tau$ *r*. (π_2 *xf*) (φ_s τ))){*0..t*} *UNIV* $\wedge$ φ_s *0* (π_1 *xf*) = *s* (π_1 *xf*)
      using *Fsolves tHyp* **by** (*simp add*:*solvesStoreIVP-def*)
      hence *expand2*:$\forall$ *xf* $\in$ *set xfList*. $\forall$ τ $\in$ {*0..t*}. ((λ*r*. φ_s *r* (π_1 *xf*))
       has-vector-derivative (λ*r*. (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] τ)) τ) (*at* τ *within* {*0..t*})
      using *eqDerivs* **by** (*simp add*: *solves-ode-def has-vderiv-on-def*)

      then have $\forall$ *xf* $\in$ *set xfList*. (($\lambda\tau$. φ_s τ (π_1 *xf*)) *solves-ode*
       ($\lambda\tau$ *r*. (π_2 *xf*) (*sol s*[*xfList*$\leftarrow$*uInput*] τ))){*0..t*} *UNIV* $\wedge$ φ_s *0* (π_1 *xf*) = *s* (π_1 *xf*)
      by (*simp add*: *has-vderiv-on-def solves-ode-def expand1 expand2*)
    then have *1*:(($\lambda\tau$. φ_s τ *x*) *solves-ode* ($\lambda\tau$ *r*. *f* (*sol s*[*xfList*$\leftarrow$*uInput*] τ))){*0..t*}

UNIV ∧
φ_s *0 x = s x* **using** *xfHyp* **by** *fastforce*

from *solHyp* **and** *xfHyp* **have** *2*:(($\lambda\ \tau$. (*sol s*[*xfList*←*uInput*] τ) *x*) *solves-ode*

($\lambda\tau\ r$. *f* (*sol s*[*xfList*←*uInput*] τ))) {*0..t*} *UNIV* ∧ (*sol s*[*xfList*←*uInput*] *0*) *x = s x*
using *solvesStoreIVP-def tHyp* **by** *fastforce*

from *tHyp* **and** *contHyp* **have** ∀ *xf* ∈ *set xfList*. *unique-on-bounded-closed 0* {*0..t*} (*s* (π_1 *xf*))
($\lambda\tau\ r$. (π_2 *xf*) (*sol s*[*xfList*←*uInput*] τ)) *UNIV* (*if t = 0 then 1 else 1*/(*t*+*1*))

apply(*clarify*) **apply**(*rule conds4UniqSol*) **by**(*auto*)
from *this* **have** *3*:*unique-on-bounded-closed 0* {*0..t*} (*s x*) ($\lambda\tau\ r$. *f* (*sol s*[*xfList*←*uInput*] τ))
UNIV (*if t = 0 then 1 else 1*/(*t*+*1*)) **using** *xfHyp* **by** *fastforce*
from *1 2* **and** *3* **show** (*sol s*[*xfList*←*uInput*] *t*) *x* = φ_s *t x*
using *unique-on-bounded-closed.unique-solution* **using** *real-Icc-closed-segment tHyp* **by** *blast*
next
assume *x* ∈ *varDiffs*
then obtain *y* **where** *xDef*:*x* = ∂ *y* **by** (*auto simp*: *varDiffs-def*)
show (*sol s*[*xfList*←*uInput*] *t*) *x* = φ_s *t x*
proof(*cases y* ∈ *set* (*map* π_1 *xfList*))
case *True*
then obtain *f* **where** *xfHyp*:(*y*, *f*) ∈ *set xfList* **by** *fastforce*
from *tHyp* **and** *Fsolves* **have** φ_s *t x = f* (φ_s *t*)
using *solves-store-ivpD*(*3*) *xfHyp xDef* **by** *force*
also have (*sol s*[*xfList*←*uInput*] *t*) *x = f* (*sol s*[*xfList*←*uInput*] *t*)
using *solves-store-ivpD*(*3*) *xfHyp xDef solHyp tHyp* **by** *force*
ultimately show *?thesis* **using** *eqDerivs xfHyp tHyp* **by** *auto*
next case *False*
then have φ_s *t x = 0*
using *xDef solves-store-ivpD*(*2*) *Fsolves tHyp* **by** *simp*
also have (*sol s*[*xfList*←*uInput*] *t*) *x = 0*
using *False solHyp tHyp solves-store-ivpD*(*2*) *xDef* **by** *fastforce*
ultimately show *?thesis* **by** *simp*
qed
qed
qed
qed

theorem *dSolveUBC*:
assumes *contHyp*:∀ *s*. ∀ *t*≥*0*. ∀ *xf* ∈ *set xfList*. *continuous-on* ({*0..t*} × *UNIV*)

(λ(*t*, (*r*::*real*)). (π_2 *xf*) (*sol s*[*xfList*←*uInput*] *t*))
and *solHyp*:∀ *s*. *solvesStoreIVP* (λ *t*. (*sol s*[*xfList*←*uInput*] *t*)) *xfList s*
and *uniqHyp*:∀ *s*. ∀ φ_s. φ_s *solvesTheStoreIVP xfList withInitState s* ⟶

($\forall$ $t \geq 0$. $\forall$ $xf \in set\ xfList$. $\forall$ $r \in \{0..t\}$. (π_2 xf) (φ_s r) = (π_2 xf) (sol $s[xfList \leftarrow uInput]$ r))
and *diffAssgn*: $\forall$ s. P s $\longrightarrow$ ($\forall$ $t\geq 0$. G (sol $s[xfList \leftarrow uInput]$ t) $\longrightarrow$ Q (sol $s[xfList \leftarrow uInput]$ t))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput=uInput* **in** *dSolve*)
prefer *2* **subgoal proof**(*clarify*)
fix s::*real store* **and** φ_s::*real* $\Rightarrow$ *real store* **and** t::*real*
assume *isSol*:*solvesStoreIVP* φ_s *xfList s* **and** *sHyp*:$0 \leq t$
from *this* **and** *uniqHyp* **have** $\forall$ $xf \in set\ xfList$. $\forall$ $t \in \{0..t\}$.
(π_2 xf) (φ_s t) = (π_2 xf) (sol $s[xfList \leftarrow uInput]$ t) **by** *auto*
also have $\forall$ $xf \in set\ xfList$. *continuous-on* ($\{0..t\}$ $\times$ *UNIV*)
(λ(t, (r::*real*)). (π_2 xf) (sol $s[xfList \leftarrow uInput]$ t)) **using** *contHyp sHyp* **by** *blast*
ultimately show (sol $s[xfList \leftarrow uInput]$ t) = φ_s t
using *sHyp isSol ubcStoreUniqueSol solHyp* **by** *simp*
qed using *assms* **by** *simp-all*

theorem *dSolve-toSolveUBC*:
assumes *funcsHyp*:$\forall$ s g. $\forall$ $xf \in set\ xfList$. π_2 xf (*override-on s g varDiffs*) = π_2 xf s
and *distinctHyp*:*distinct* (*map* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ $xf \in set\ xfList$. π_1 xf $\notin$ *varDiffs*
and *solHyp1*:$\forall$ s. $\forall$ $uxf \in set$ (*uInput* $\otimes$ *xfList*). π_1 uxf 0 (sol s) = sol s (π_1 (π_2 uxf))
and *solHyp2*:$\forall$ s. $\forall$ $t\geq 0$. $\forall$ $xf \in set\ xfList$. ((λt. (sol $s[xfList \leftarrow uInput]$ t) (π_1 xf)) *has-vderiv-on*
(λt. π_2 xf (sol $s[xfList \leftarrow uInput]$ t))) $\{0..t\}$
and *contHyp*:$\forall$ s. $\forall$ $t\geq 0$. $\forall$ $xf \in set\ xfList$. *continuous-on* ($\{0..t\}$ $\times$ *UNIV*)
(λ(t, (r::*real*)). (π_2 xf) (sol $s[xfList \leftarrow uInput]$ t))
and *uniqHyp*:$\forall$ s. $\forall$ φ_s. φ_s *solvesTheStoreIVP xfList withInitState s* $\longrightarrow$
($\forall$ $t \geq 0$. $\forall$ $xf \in set\ xfList$. $\forall$ $r \in \{0..t\}$. (π_2 xf) (φ_s r) = (π_2 xf) (sol $s[xfList \leftarrow uInput]$ r))
and *postCondHyp*:$\forall$ s. P s $\longrightarrow$ ($\forall$ $t\geq 0$. Q (sol $s[xfList \leftarrow uInput]$ t))
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac uInput=uInput* **in** *dSolveUBC*)
using *contHyp* **apply** *simp*
apply(*rule allI*, *rule-tac uInput=uInput* **in** *conds4storeIVP-on-toSol*)
using *assms* **by** *auto*

"Differential Invariant."

lemma *solvesStoreIVP-couldBeModified*:
fixes F::*real* $\Rightarrow$ *real store*
assumes *vars*:$\forall$ $t\geq 0$. $\forall$ $xf \in set\ xfList$. ((λt. F t (π_1 xf)) *solves-ode* (λt r. π_2 xf (F t))) $\{0..t\}$ *UNIV*
and *dvars*:$\forall$ $t \geq 0$. $\forall$ $xf \in set\ xfList$. (F t (∂ (π_1 xf))) = (π_2 xf) (F t)
shows $\forall$ $t \geq 0$. $\forall$ $r \in \{0..t\}$. $\forall$ $xf \in set\ xfList$.
((λ t. F t (π_1 xf)) *has-vector-derivative* F r (∂ (π_1 xf))) (*at r within* $\{0..t\}$)

```
proof(clarify, rename-tac t r x f)
fix x f and t r::real
assume tHyp:0 ≤ t and xfHyp:(x, f) ∈ set xfList and rHyp:r ∈ {0..t}
from this and vars have ((λt. F t x) solves-ode (λt r. f (F t))) {0..t} UNIV
using tHyp by fastforce
hence *:∀ r∈{0..t}. ((λ t. F t x) has-vector-derivative (λ t. f (F t)) r) (at r within
{0..t})
by (simp add: solves-ode-def has-vderiv-on-def tHyp)
have ∀ t ≥ 0. ∀ r∈{0..t}. ∀ xf ∈ set xfList. (F r (∂ (π_1 xf))) = (π_2 xf) (F r)
using assms by auto
from this rHyp and xfHyp have (F r (∂ x)) = f (F r) by force
then show ((λt. F t (π_1 (x, f))) has-vector-derivative F r (∂ (π_1 (x, f)))) (at r
within {0..t})
using * rHyp by auto
qed

lemma derivationLemma-baseCase:
fixes F::real ⇒ real store
assumes solves:solvesStoreIVP F xfList a
shows ∀ x ∈ (UNIV − varDiffs). ∀ t ≥ 0. ∀ r∈{0..t}.
((λ t. F t x) has-vector-derivative F r (∂ x)) (at r within {0..t})
proof
fix x
assume x ∈ UNIV − varDiffs
then have notVarDiff:∀ z. x ≠ ∂ z using varDiffs-def by fastforce
 show ∀ t≥0. ∀ r∈{0..t}. ((λt. F t x) has-vector-derivative F r (∂ x)) (at r within
{0..t})
  proof(cases x ∈ set (map π_1 xfList))
    case True
    from this and solves have ∀ t ≥ 0. ∀ r∈{0..t}. ∀ xf ∈ set xfList.
    ((λ t. F t (π_1 xf)) has-vector-derivative F r (∂ (π_1 xf))) (at r within {0..t})
   apply(rule-tac solvesStoreIVP-couldBeModified) using solves solves-store-ivpD
by auto
    from this show ?thesis using True by auto
  next
    case False
    from this notVarDiff and solves have const:∀ t ≥ 0. F t x = a x
    using solves-store-ivpD(1) by (simp add: varDiffs-def)
     have constD:∀ t ≥ 0. ∀ r∈{0..t}. ((λ r. a x) has-vector-derivative 0) (at r
within {0..t})
    by (auto intro: derivative-eq-intros)
    {fix t r::real
      assume t≥0 and r ∈ {0..t}
      hence ((λ s. a x) has-vector-derivative 0) (at r within {0..t}) by (simp add:
constD)
      moreover have ⋀s. s ∈ {0..t} ⟹ (λ r. F r x) s = (λ r. a x) s
      using const by (simp add: ‹0 ≤ t›)
      ultimately have ((λ s. F s x) has-vector-derivative 0) (at r within {0..t})
      using has-vector-derivative-transform by (metis ‹r ∈ {0..t}›)}
```

  hence *isZero*:$\forall\, t{\geq}0.\forall\, r{\in}\{0..t\}.((\lambda\ t.\ F\ t\ x)$*has-vector-derivative* $0)($*at r within* $\{0..t\})$**by** *blast*
  from *False solves* **and** *notVarDiff* **have** $\forall\ \ t \geq 0.\ F\ t\ (\partial\ x) = 0$
  using *solves-store-ivpD*(2) **by** *simp*
  then show *?thesis* **using** *isZero* **by** *simp*
 qed
qed

lemma *derivationLemma*:
assumes *solvesStoreIVP F xfList a*
and *tHyp*:$t \geq 0$
and *termVarsHyp*:$\forall\ \ x \in trmVars\ \eta.\ x \in (UNIV\ -\ varDiffs)$
shows $\forall\, r{\in}\{0..t\}.\ ((\lambda\ s.\ [\![\eta]\!]_t\ (F\ s))$*has-vector-derivative* $[\![\partial_t\ \eta]\!]_t\ (F\ r))$ $($*at r within* $\{0..t\})$
using *termVarsHyp* **proof**$($*induction* $\eta)$
 case $(Const\ r)$
 then show *?case* **by** *simp*
next
 case $(Var\ y)$
 then have *yHyp*:$y \in UNIV\ -\ varDiffs$ **by** *auto*
 from *this tHyp* **and** *assms*(1) **show** *?case*
 using *derivationLemma-baseCase* **by** *auto*
next
 case $(Mns\ \eta)$
 then show *?case*
 apply$($*clarsimp*$)$
 by$($*rule derivative-intros*, *simp*$)$
next
 case $(Sum\ \eta 1\ \eta 2)$
 then show *?case*
 apply$($*clarsimp*$)$
 by$($*rule derivative-intros*, *simp-all*$)$
next
 case $(Mult\ \eta 1\ \eta 2)$
 then show *?case*
 apply$($*clarsimp*$)$
 apply$($*subgoal-tac* $((\lambda s.\ [\![\eta 1]\!]_t\ (F\ s) *_R [\![\eta 2]\!]_t\ (F\ s))$ *has-vector-derivative*
  $[\![\partial_t\ \eta 1]\!]_t\ (F\ r) \cdot [\![\eta 2]\!]_t\ (F\ r) + [\![\eta 1]\!]_t\ (F\ r) \cdot [\![\partial_t\ \eta 2]\!]_t\ (F\ r))$ $($*at r within* $\{0..t\})$,*simp*$)$
 apply$($*rule-tac* $f'1{=}[\![\partial_t\ \eta 1]\!]_t\ (F\ r)$ **and** $g'1{=}[\![\partial_t\ \eta 2]\!]_t\ (F\ r)$ **in** *derivative-eq-intros*$(25))$
 by $($*simp-all add*: *has-field-derivative-iff-has-vector-derivative*$)$
qed

lemma *diff-subst-prprty-4terms*:
assumes *solves*:$\forall\ \ xf \in set\ xfList.\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ \ (F\ t)$
and *tHyp*:$(t{::}real) \geq 0$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
and *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV\ -\ varDiffs)$
shows $[\![\partial_t\ \eta]\!]_t\ (F\ t) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\partial_t\ \eta\rangle]\!]_t\ (F\ t)$

using *termVarsHyp* **apply**(*induction* η) **apply**(*simp-all add*: *substList-help2*)
using *listsHyp* **and** *solves* **apply**(*induct xfList uInput rule*: *list-induct2′*, *simp*, *simp*, *simp*)
proof(*clarify*, *rename-tac y g xfTail* ϑ *trmTail x*)
fix *x y*::*string* **and** ϑ::*trms* **and** *g* **and** *xfTail*::((*string* $\times$ (*real store* $\Rightarrow$ *real*)) *list*)
and *trmTail*
assume *IH*:$\bigwedge x.\ x \notin varDiffs \Longrightarrow map\ \pi_2\ xfTail = map\ tval\ trmTail \Longrightarrow$
$\forall xf{\in}set\ xfTail.\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ \ (F\ t) \Longrightarrow$
$F\ t\ (\partial\ x) = [\![(map\ \ (vdiff \circ \pi_1)\ xfTail \otimes trmTail)\langle t_V\ \ (\partial\ x)\rangle]\!]_t\ \ (F\ t)$
and *1*:$x \notin varDiffs$ **and** *2*:$map\ \pi_2\ ((y,\ g)\ \#\ xfTail) = map\ tval\ (\vartheta\ \#\ trmTail)$
and *3*:$\forall xf{\in}set\ ((y,\ g)\ \#\ xfTail).\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ \ (F\ t)$
hence $*$:$[\![(map\ \ (vdiff\ \circ \pi_1)\ \ xfTail\ \otimes\ trmTail)\langle Var\ \ (\partial\ x)\rangle]\!]_t\ \ (F\ t)\ =\ F\ t\ (\partial\ x)$
using *tHyp* **by** *auto*
show $F\ t\ (\partial\ x) = [\![((map\ \ (vdiff\ \circ \pi_1)\ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\ \langle t_V$
$(\partial\ x)\rangle]\!]_t\ \ (F\ t)$
  proof(*cases* $x \in set\ \ (map\ \pi_1\ \ ((y,\ g)\ \#\ xfTail))$)
    case *True*
    then have $x = y \vee (x \neq y \wedge x \in set\ \ (map\ \pi_1\ \ xfTail))$ **by** *auto*
    moreover
    {**assume** $x = y$
      from *this* **have** $((map\ \ (vdiff\ \circ \pi_1)\ \ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\langle t_V$
$(\partial\ x)\rangle = \vartheta$ **by** *simp*
      also from *3 tHyp* **have** $F\ t\ (\partial\ y) = g\ \ (F\ t)$ **by** *simp*
      moreover from *2* **have** $[\![\vartheta]\!]_t\ \ (F\ t) = g\ \ (F\ t)$ **by** *simp*
      ultimately have *?thesis* **by** (*simp add*: ‹$x = y$›)}
    moreover
    {**assume** $x \neq y \wedge x \in set\ \ (map\ \pi_1\ \ xfTail)$
      then have $\partial\ x \neq \partial\ y$ **using** *vdiff-inj* **by** *auto*
      from *this* **have** $((map\ \ (vdiff\ \circ \pi_1)\ \ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\ \langle t_V$
$(\partial\ x)\rangle =$
      $((map\ \ (vdiff\ \circ \pi_1)\ \ xfTail) \otimes trmTail)\ \langle t_V\ \ (\partial\ x)\rangle$ **by** *simp*
      hence *?thesis* **using** $*$ **by** *simp*}
    ultimately show *?thesis* **by** *blast*
  next
    case *False*
    then have $((map\ \ (vdiff\ \circ \pi_1)\ \ ((y,\ g)\ \#\ xfTail)) \otimes (\vartheta\ \#\ trmTail))\ \langle t_V\ \ (\partial\ x)\rangle$
$= t_V\ \ (\partial\ x)$
   using *substList-cross-vdiff-on-non-ocurring-var* **by**(*metis*(*no-types*, *lifting*) *List.map.compositionality*)
    thus *?thesis* **by** *simp*
  qed
qed

lemma *eqInVars-impl-eqInTrms*:
assumes *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV\ -\ varDiffs)$
and *initHyp*:$\forall x.\ x \notin varDiffs \longrightarrow b\ x = a\ x$
shows $[\![\eta]\!]_t\ \ a = [\![\eta]\!]_t\ \ b$
using *assms* **by**(*induction* η, *simp-all*)

lemma *non-empty-funList-implies-non-empty-trmList*:

shows $\forall\ list.(x,f) \in set\ list \wedge map\ \pi_2\ list = map\ tval\ tList \longrightarrow (\exists\ \vartheta.[\![\vartheta]\!]_t = f \wedge \vartheta \in set\ tList)$
by(*induction tList*, *auto*)

lemma *dInvForTrms-prelim*:
assumes *substHyp*:
$\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1(\![set\ xfList]\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ st = 0$
and *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\eta]\!]_t\ a = 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c = 0)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\eta]\!]_t\ a = 0$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
from *this* **obtain** t::*real* **and** F::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t \geq 0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a \wedge (\forall r \in \{0..t\}.\ G\ (F\ r))$

using *guarDiffEqtn-def* **by** *auto*
then have $\forall x.\ x \notin varDiffs \longrightarrow F\ 0\ x = a\ x$ **using** *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$ **using** *termVarsHyp eqInVars-impl-eqInTrms*
by *blast*
hence *obs1*:$[\![\eta]\!]_t\ (F\ 0) = 0$ **using** *aHyp* **by** *simp*
from *tcHyp* **have** *obs2*:$\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative*
$[\![\partial_t\ \eta]\!]_t\ (F\ r))$ (*at r within* $\{0..t\}$) **using** *derivationLemma termVarsHyp* **by** *blast*
have $\forall r \in \{0..t\}.\ \forall\ xf \in set\ xfList.\ F\ r\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ r)$
using *tcHyp solves-store-ivpD*(*3*) **by** *fastforce*
hence $\forall r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t$
$(F\ r)$
using *tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall r \in \{0..t\}.\ [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\partial_t$
$\eta\rangle]\!]_t\ (F\ r) = 0$
using *solves-store-ivpD*(*2*) *tcHyp* **by** *fastforce*
ultimately have $\forall r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative* $0)$ (*at r within*
$\{0..t\}$)
using *obs2* **by** *auto*
from *this* **and** *tcHyp* **have** $\forall s \in \{0..t\}.\ ((\lambda x.\ [\![\eta]\!]_t\ (F\ x))$ *has-derivative* $(\lambda x.\ x *_R$
$0))$
(*at s within* $\{0..t\}$) **by** (*metis has-vector-derivative-def*)
hence $[\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = (\lambda x.\ x *_R\ 0)\ (t - 0)$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then show $[\![\eta]\!]_t\ c = 0$ **using** *obs1 tcHyp* **by** *auto*
qed

theorem *dInvForTrms*:
assumes $\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1(\![set\ xfList]\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ st = 0$
and *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
and *eta-f*:$f = [\![\eta]\!]_t$
shows $PRE\ (\lambda\ s.\ f\ s = 0)\ (ODEsystem\ xfList\ with\ G)\ POST\ (\lambda\ s.\ f\ s = 0)$

using *eta-f* **proof**(*clarsimp*)
fix *a b*
assume $(a,\ b) \in \lceil\lambda s.\ [\![\eta]\!]_t\ s = 0\rceil$ **and** $f = [\![\eta]\!]_t$
from *this* **have** *aHyp*:$a = b \wedge [\![\eta]\!]_t\ a = 0$ **by** (*metis* (*full-types*) *d-p2r rdom-p2r-contents*)
have $[\![\eta]\!]_t\ a = 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c = 0)$
using *assms dInvForTrms-prelim* **by** *metis*
from *this* **and** *aHyp* **have** $\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c = 0$ **by** *blast*
thus $(a,\ b) \in wp\ (ODEsystem\ xfList\ with\ G\ )\ \lceil\lambda s.\ [\![\eta]\!]_t\ s = 0\rceil$
using *aHyp* **by** (*simp add*: *boxProgrPred-chrctrztn*)
qed

lemma *diff-subst-prprty-4props*:
assumes *solves*:$\forall\ xf \in set\ xfList.\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t)$
and *tHyp*:$t \geq 0$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
and *propVarsHyp*:$propVars\ \varphi \subseteq (UNIV - varDiffs)$
shows $[\![\partial_P\ \varphi]\!]_P\ (F\ t) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\upharpoonright\partial_P\ \varphi\upharpoonright]\!]_P\ (F\ t)$
using *propVarsHyp* **apply**(*induction* φ, *simp-all*)
using *assms diff-subst-prprty-4terms* **apply** *fastforce*
using *assms diff-subst-prprty-4terms* **apply** *fastforce*
using *assms diff-subst-prprty-4terms* **by** *fastforce*

lemma *dInvForProps-prelim*:
assumes *substHyp*:
$\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ st \geq 0$
and *termVarsHyp*:$trmVars\ \eta \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\eta]\!]_t\ a > 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c > 0)$
and $[\![\eta]\!]_t\ a \geq 0 \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\eta]\!]_t\ c \geq 0)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\eta]\!]_t\ a > 0$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
from *this* **obtain** $t::real$ **and** $F::real \Rightarrow real\ store$
where *tcHyp*:$t{\geq}0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a \wedge (\forall r{\in}\{0..t\}.\ G\ (F\ r))$

using *guarDiffEqtn-def* **by** *auto*
then have $\forall x.\ x \notin varDiffs \longrightarrow F\ 0\ x = a\ x$ **using** *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$ **using** *termVarsHyp eqInVars-impl-eqInTrms* **by** *blast*
hence *obs1*:$[\![\eta]\!]_t\ (F\ 0) > 0$ **using** *aHyp tcHyp* **by** *simp*
from *tcHyp* **have** *obs2*:$\forall r{\in}\{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative* $[\![\partial_t\ \eta]\!]_t\ (F\ r))\ (at\ r\ within\ \{0..t\})$ **using** *derivationLemma termVarsHyp* **by** *blast*
have $(\forall t{\geq}0.\ \forall\ xf \in set\ xfList.\ F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t))$
using *tcHyp solves-store-ivpD*(*3*) **by** *blast*
hence $\forall r{\in}\{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r)$
using *diff-subst-prprty-4terms termVarsHyp tcHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall r{\in}\{0..t\}.\ [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t$

$\eta\rangle]\!]_t\ (F\ r) \geq 0$
using *solves-store-ivpD*(*2*) *tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $*$:$\forall\, r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0$ **by** (*simp*)
from *obs2* **and** *tcHyp* **have** $\forall\, r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-derivative*
$(\lambda x.\ x *_R ([\![\partial_t\ \eta]\!]_t\ (F\ r))))$ (*at r within* $\{0..t\}$) **by** (*simp add*: *has-vector-derivative-def*)

hence $\exists\, r \in \{0..t\}.\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![(\partial_t\ \eta)]\!]_t)\ (F\ r)$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain r **where** $[\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge [\![\partial_t\ \eta]\!]_t\ (F\ t) \geq 0$
$\wedge\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using $*$ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t\ c > 0$
using *obs1 tcHyp* **by** (*metis cancel-comm-monoid-add-class.diff-cancel diff-ge-0-iff-ge*

diff-strict-mono linorder-neqE-linordered-idom linordered-field-class.sign-simps(*45*)
not-le)
next
show $0 \leq [\![\eta]\!]_t\ a \longrightarrow (\forall\, c.\ (a,\ c) \in$ *ODEsystem xfList with G* $\longrightarrow 0 \leq [\![\eta]\!]_t\ c)$
proof(*clarify*)
fix c **assume** *aHyp*:$[\![\eta]\!]_t\ a \geq 0$ **and** *cHyp*:$(a,\ c) \in$ *ODEsystem xfList with G*
from *this* **obtain** t::*real* **and** F::*real* $\Rightarrow$ *real store*
where *tcHyp*:$t \geq 0 \wedge F\ t = c \wedge$ *solvesStoreIVP F xfList a* $\wedge\ (\forall\, r \in \{0..t\}.\ G\ (F\ r))$

using *guarDiffEqtn-def* **by** *auto*
then have $\forall\, x.\ x \notin$ *varDiffs* $\longrightarrow F\ 0\ x = a\ x$ **using** *solves-store-ivpD*(*6*) **by** *blast*
from *this* **have** $[\![\eta]\!]_t\ a = [\![\eta]\!]_t\ (F\ 0)$ **using** *termVarsHyp eqInVars-impl-eqInTrms*
by *blast*
hence *obs1*:$[\![\eta]\!]_t\ (F\ 0) \geq 0$ **using** *aHyp tcHyp* **by** *simp*
from *tcHyp* **have** *obs2*:$\forall\, r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-vector-derivative*
$[\![\partial_t\ \eta]\!]_t\ (F\ r))$ (*at r within* $\{0..t\}$) **using** *derivationLemma termVarsHyp* **by** *blast*
have $(\forall\, t \geq 0.\ \forall\ xf \in$ *set xfList*. $F\ t\ (\partial\ (\pi_1\ xf)) = \pi_2\ xf\ (F\ t))$
using *tcHyp solves-store-ivpD*(*3*) **by** *blast*
from *this* **and** *tcHyp* **have** $\forall\, r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) =$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r)$
using *diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall\, r \in \{0..t\}.\ [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\ \langle\partial_t$
$\eta\rangle]\!]_t\ (F\ r) \geq 0$
using *solves-store-ivpD*(*2*) *tcHyp* **by** (*metis atLeastAtMost-iff*)
ultimately have $*$:$\forall\, r \in \{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0$ **by** (*simp*)
from *obs2* **and** *tcHyp* **have** $\forall\, r \in \{0..t\}.\ ((\lambda s.\ [\![\eta]\!]_t\ (F\ s))$ *has-derivative*
$(\lambda x.\ x *_R ([\![\partial_t\ \eta]\!]_t\ (F\ r))))$ (*at r within* $\{0..t\}$) **by** (*simp add*: *has-vector-derivative-def*)

hence $\exists\, r \in \{0..t\}.\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using *mvt-very-simple* **and** *tcHyp* **by** *fastforce*
then obtain r **where** $[\![\partial_t\ \eta]\!]_t\ (F\ r) \geq 0 \wedge 0 \leq r \wedge r \leq t \wedge [\![\partial_t\ \eta]\!]_t\ (F\ t) \geq 0$
$\wedge\ [\![\eta]\!]_t\ (F\ t) - [\![\eta]\!]_t\ (F\ 0) = t \cdot ([\![\partial_t\ \eta]\!]_t\ (F\ r))$
using $*$ *tcHyp* **by** (*meson atLeastAtMost-iff order-refl*)
thus $[\![\eta]\!]_t\ c \geq 0$
using *obs1 tcHyp* **by** (*metis cancel-comm-monoid-add-class.diff-cancel diff-ge-0-iff-ge*

diff-strict-mono linorder-neqE-linordered-idom linordered-field-class.sign-simps(45) not-le)
qed
qed

lemma *less-pval-to-tval*:
assumes $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ (\vartheta \prec \eta)\restriction]\!]_P\ st$
shows $[\![((map\ (vdiff\circ\pi_1)\ xfList) \otimes uInput)\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t\ st \geq 0$
using *assms* **by**(*auto*)

lemma *leq-pval-to-tval*:
assumes $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ (\vartheta \preceq \eta)\restriction]\!]_P\ st$
shows $[\![((map\ (vdiff\circ\pi_1)\ xfList) \otimes uInput)\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t\ st \geq 0$
using *assms* **by**(*auto*)

lemma *dInv-prelim*:
assumes *substHyp*:$\forall\ st.\ G\ st \longrightarrow\ (\forall str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ \varphi\restriction]\!]_P\ st$
and *propVarsHyp*:$propVars\ \varphi \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
shows $[\![\varphi]\!]_P\ a \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow [\![\varphi]\!]_P\ c)$
proof(*clarify*)
fix *c* **assume** *aHyp*:$[\![\varphi]\!]_P\ a$ **and** *cHyp*:$(a,\ c) \in ODEsystem\ xfList\ with\ G$
from *this* **obtain** t::*real* **and** F::$real \Rightarrow real\ store$
where *tcHyp*:$t{\geq}0 \wedge F\ t = c \wedge solvesStoreIVP\ F\ xfList\ a$ **using** *guarDiffEqtn-def*
by *auto*
from *aHyp propVarsHyp* **and** *substHyp* **show** $[\![\varphi]\!]_P\ c$
proof(*induction* φ)
case $(Eq\ \vartheta\ \eta)$
hence *hyp*:$\forall st.\ G\ st \longrightarrow\ (\forall str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ (\vartheta \doteq \eta)\restriction]\!]_P\ st$ **by** *blast*
then have $\forall st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\langle\partial_t\ (\vartheta \oplus (\ominus\ \eta))\rangle]\!]_t\ st = 0$ **by** *simp*
also have $trmVars\ (\vartheta \oplus (\ominus\ \eta)) \subseteq UNIV - varDiffs$ **using** *Eq.prems(2)* **by** *simp*
moreover have $[\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ a = 0$ **using** *Eq.prems(1)* **by** *simp*
ultimately have $(\forall c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G\ \longrightarrow [\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ c = 0)$
using *dInvForTrms-prelim listsHyp* **by** *blast*
hence $[\![\vartheta \oplus (\ominus\ \eta)]\!]_t\ (F\ t) = 0$ **using** *tcHyp cHyp* **by** *simp*
from *this* **have** $[\![\vartheta]\!]_t\ (F\ t) = [\![\eta]\!]_t\ (F\ t)$ **by** *simp*
also have $([\![\vartheta \doteq \eta]\!]_P)\ c = ([\![\vartheta]\!]_t\ (F\ t) = [\![\eta]\!]_t\ (F\ t))$ **using** *tcHyp* **by** *simp*
ultimately show *?case* **by** *simp*
next
case $(Less\ \vartheta\ \eta)$
hence $\forall st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$0 \leq ([\![(map\ (vdiff \circ \pi_1)\ xfList \otimes uInput)\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t)\ st$
using *less-pval-to-tval* **by** *metis*

also from *Less.prems(2)***have** $trmVars\ (\eta \oplus (\ominus\ \vartheta)) \subseteq UNIV - varDiffs$ **by** *simp*
moreover have $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ a > 0$ **using** *Less.prems(1)* **by** *simp*
ultimately have $(\forall c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ c > 0)$
using *dInvForProps-prelim(1) listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) > 0$ **using** *tcHyp cHyp* **by** *simp*
from *this* **have** $[\![\eta]\!]_t\ (F\ t) > [\![\vartheta]\!]_t\ (F\ t)$ **by** *simp*
also have $[\![\vartheta \prec \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) < [\![\eta]\!]_t\ (F\ t))$ **using** *tcHyp* **by** *simp*
ultimately show *?case* **by** *simp*
next
case $(Leq\ \vartheta\ \eta)$
hence $\forall st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow 0 \leq ([\![(map\ (vdiff \circ \pi_1)\ xfList \otimes uInput)\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t)\ st$ **using** *leq-pval-to-tval*
by *metis*
also from *Leq.prems(2)***have** $trmVars\ (\eta \oplus (\ominus\ \vartheta)) \subseteq UNIV - varDiffs$ **by** *simp*
moreover have $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ a \geq 0$ **using** *Leq.prems(1)* **by** *simp*
ultimately have $(\forall c.\ (a,\ c) \in ODEsystem\ xfList\ with\ G \longrightarrow [\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ c \geq 0)$
using *dInvForProps-prelim(2) listsHyp* **by** *blast*
hence $[\![\eta \oplus (\ominus\ \vartheta)]\!]_t\ (F\ t) \geq 0$ **using** *tcHyp cHyp* **by** *simp*
from *this* **have** $([\![\eta]\!]_t\ (F\ t) \geq [\![\vartheta]\!]_t\ (F\ t))$ **by** *simp*
also have $[\![\vartheta \preceq \eta]\!]_P\ c = ([\![\vartheta]\!]_t\ (F\ t) \leq [\![\eta]\!]_t\ (F\ t))$ **using** *tcHyp* **by** *simp*
ultimately show *?case* **by** *simp*
next
case $(And\ \varphi 1\ \varphi 2)$
then show *?case* **by**(*simp*)
next
case $(Or\ \varphi 1\ \varphi 2)$
from *this* **show** *?case* **by** *auto*
qed
qed

theorem *dInv*:
assumes $\forall\ st.\ G\ st \longrightarrow (\forall str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow [\![((map\ (vdiff\ \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ \varphi\restriction]\!]_P\ st$
and *termVarsHyp*:$propVars\ \varphi \subseteq (UNIV - varDiffs)$
and *listsHyp*:$map\ \pi_2\ xfList = map\ tval\ uInput$
and *phi-p*:$P = [\![\varphi]\!]_P$
shows *PRE P (ODEsystem xfList with G) POST P*
proof(*clarsimp*)
fix *a b*
assume $(a,\ b) \in \lceil P\rceil$
from *this* **have** *aHyp*:$a = b \wedge P\ a$ **by** (*metis (full-types) d-p2r rdom-p2r-contents*)
have $P\ a \longrightarrow (\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow P\ c)$
using *assms dInv-prelim* **by** *metis*
from *this* **and** *aHyp* **have** $\forall\ c.\ (a,c) \in (ODEsystem\ xfList\ with\ G) \longrightarrow P\ c$ **by** *blast*
thus $(a,\ b) \in wp\ (ODEsystem\ xfList\ with\ G\ )\ \lceil P\rceil$
using *aHyp* **by** (*simp add*: *boxProgrPred-chrctrztn*)

qed

theorem *dInvFinal*:
assumes $\forall$ *st*. *G st* $\longrightarrow$ ($\forall$ *str*. *str* $\notin$ (π_1(|*set xfList*|)) $\longrightarrow$ *st* (∂ *str*) = *0*) $\longrightarrow$
$[\![$((*map* (*vdiff* $\circ$ π_1) *xfList*) $\otimes$ *uInput*)$\lceil\partial_P$ $\varphi\rceil]\!]_P$ *st*
and *termVarsHyp*:*propVars* $\varphi \subseteq$ (*UNIV* $-$ *varDiffs*)
and *listsHyp*:*map* π_2 *xfList* = *map tval uInput*
and *impls*:$\lceil P\rceil \subseteq \lceil F\rceil \wedge \lceil F\rceil \subseteq \lceil Q\rceil$
and *phi-f*:$F = [\![\varphi]\!]_P$
shows *PRE P* (*ODEsystem xfList with G*) *POST Q*
apply(*rule-tac* C=$[\![\varphi]\!]_P$ **in** *dCut*)
apply(*subgoal-tac* $\lceil F\rceil \subseteq$ *wp* (*ODEsystem xfList with G*) $\lceil F\rceil$, *simp*)
using *impls* **and** *phi-f* **apply** *blast*
apply(*subgoal-tac PRE F* (*ODEsystem xfList with G*) *POST F*, *simp*)
apply(*rule-tac* φ=φ **and** *uInput*=*uInput* **in** *dInv*)
prefer *5* **apply**(*subgoal-tac PRE P* (*ODEsystem xfList with* (λs. *G s* $\wedge$ *F s*)) *POST Q*, *simp add*: *phi-f*)
apply(*rule dWeakening*)
using *impls* **apply** *simp*
using *assms* **by** *simp-all*

end
theory *VC-diffKAD-examples*
imports *VC-diffKAD*

begin

5.4.5 Rules Testing

In this section we test the recently developed rules with simple dynamical systems.

— Example of hybrid program verified with the rule dSolve and a single differential equation: $x' = v$.
lemma *motion-with-constant-velocity*:
  PRE (λ *s*. *s* ''*y*'' < *s* ''*x*'' $\wedge$ *s* ''*v*'' > *0*)
  (*ODEsystem* [(''*x*'',(λ *s*. *s* ''*v*''))] *with* (λ *s*. *True*))
  POST (λ *s*. (*s* ''*y*'' < *s* ''*x*''))
apply(*rule-tac uInput*=[λ *t s*. *s* ''*v*'' $\cdot$ *t* + *s* ''*x*''] **in** *dSolve-toSolveUBC*)
prefer *9* **subgoal by**(*simp add*: *wp-trafo vdiff-def add-strict-increasing2*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
prefer *2* **apply**(*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*)
apply(*clarify*, *rule-tac f'1*=λ *x*. *s* ''*v*'' **and** *g'1*=λ *x*. *0* **in** *derivative-intros*(*191*))
apply(*rule-tac f'1*=λ *x*.*0* **and** *g'1*=λ *x*.*1* **in** *derivative-intros*(*194*))
by(*auto intro*: *derivative-intros*)

Same hybrid program verified with dSolve and the system of ODEs: $x' = v, v' = a$. The uniqueness part of the proof requires a preliminary lemma.

lemma *flow-vel-is-galilean-vel*:

assumes *solHyp*:φ_s *solvesTheStoreIVP* $[(x,\ \lambda s.\ s\ v),\ (v,\ \lambda s.\ s\ a)]$ *withInitState* s
and *tHyp*:$r \leq t$ **and** *rHyp*:$0 \leq r$ **and** *distinct*:$x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin$ *varDiffs*
shows $\varphi_s\ r\ v = s\ a \cdot r + s\ v$
proof−
from *assms* **have** *1*:$((\lambda t.\ \varphi_s\ t\ v)$ *solves-ode* $(\lambda t\ r.\ \varphi_s\ t\ a))\ \{0..t\}$ *UNIV* $\wedge\ \varphi_s\ 0\ v = s\ v$
by (*simp add*: *solvesStoreIVP-def*)
from *assms* **have** *obs*:$\forall\ r \in \{0..t\}.\ \varphi_s\ r\ a = s\ a$
by(*auto simp*: *solvesStoreIVP-def varDiffs-def*)
have *2*:$((\lambda t.\ s\ a \cdot t + s\ v)$ *solves-ode* $(\lambda t\ r.\ \varphi_s\ t\ a))\ \{0..t\}$ *UNIV*
unfolding *solves-ode-def* **apply**(*subgoal-tac* $((\lambda x.\ s\ a \cdot x + s\ v)$ *has-vderiv-on* $(\lambda x.\ s\ a))\ \{0..t\})$
using *obs* **apply** (*simp add*: *has-vderiv-on-def*) **by**(*rule galilean-transform*)
have *3*:*unique-on-bounded-closed* $0\ \{0..t\}\ (s\ v)\ (\lambda t\ r.\ \varphi_s\ t\ a)$ *UNIV* (*if* $t = 0$ *then* 1 *else* $1/(t+1)$)
apply(*simp add*: *ubc-definitions del*: *comp-apply*, *rule conjI*)
using *rHyp tHyp obs* **apply**(*simp-all del*: *comp-apply*)
apply(*clarify*, *rule continuous-intros*) **prefer** *3* **apply** *safe*
apply(*rule continuous-intros*)
apply(*auto intro*: *continuous-intros*)
by (*metis continuous-on-const continuous-on-eq*)
thus $\varphi_s\ r\ v = s\ a \cdot r + s\ v$
apply(*rule-tac unique-on-bounded-closed.unique-solution*[*of* $0\ \{0..t\}\ s\ v$ $(\lambda t\ r.\ \varphi_s\ t\ a)$ *UNIV* (*if* $t = 0$ *then* 1 *else* $1\ /\ (t + 1)$) $(\lambda t.\ \varphi_s\ t\ v)$])
using *rHyp tHyp 1 2* **and** *3* **by** *auto*
qed

lemma *motion-with-constant-acceleration*:
PRE $(\lambda\ s.\ s\ ''y'' < s\ ''x''\ \wedge s\ ''v'' \geq 0 \wedge s\ ''a'' > 0)$
(*ODEsystem* $[(''x'',(\lambda\ s.\ s\ ''v'')),(''v'',(\lambda\ s.\ s\ ''a''))]$ *with* $(\lambda\ s.$ *True*$)$)
POST $(\lambda\ s.\ (s\ ''y'' < s\ ''x''))$
apply(*rule-tac uInput*=$[\lambda\ t\ s.\ s\ ''a'' \cdot t\ \hat{}\ 2/2 + s\ ''v'' \cdot t + s\ ''x'',$
$\lambda\ t\ s.\ s\ ''a'' \cdot t + s\ ''v'']$ **in** *dSolve-toSolveUBC*)
prefer *9* **subgoal** **by**(*simp add*: *wp-trafo vdiff-def add-strict-increasing2*)
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *clarify*, *rule conjI*)
by(*rule galilean-transform*)+
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *safe*)
by(*rule continuous-intros*)+
prefer *6* **subgoal**
apply(*simp add*: *vdiff-def*, *safe*)
subgoal for $s\ \varphi_s\ t\ r$ **apply**(*rule flow-vel-is-galilean-vel*[*of* $\varphi_s\ ''x''$ - - - - t])
by(*simp-all add*: *varDiffs-def vdiff-def*)
apply(*simp add*: *solvesStoreIVP-def vdiff-def varDiffs-def*) **done**
by(*auto simp*: *varDiffs-def vdiff-def*)

Example of a hybrid system with two modes verified with the equality dS.

We also need to provide a previous (similar) lemma.

lemma *flow-vel-is-galilean-vel2*:
assumes *solHyp*:φ_s *solvesTheStoreIVP* $[(x, \lambda s.\ s\ v), (v, \lambda s. -\ s\ a)]$ *withInitState* s
and *tHyp*:$r \leq t$ **and** *rHyp*:$0 \leq r$ **and** *distinct*:$x \neq v \wedge v \neq a \wedge x \neq a \wedge a \notin$ *varDiffs*
shows $\varphi_s\ r\ v = s\ v - s\ a \cdot r$
proof−
from *assms* **have** *1*:$((\lambda t.\ \varphi_s\ t\ v)$ *solves-ode* $(\lambda t\ r. -\ \varphi_s\ t\ a))\ \{0..t\}$ *UNIV* $\wedge\ \varphi_s\ 0\ v = s\ v$
by (*simp add*: *solvesStoreIVP-def*)
from *assms* **have** *obs*:$\forall\ r \in \{0..t\}.\ \varphi_s\ r\ a = s\ a$
by(*auto simp*: *solvesStoreIVP-def varDiffs-def*)
have *2*:$((\lambda t. -\ s\ a \cdot t + s\ v)$ *solves-ode* $(\lambda t\ r. -\ \varphi_s\ t\ a))\ \{0..t\}$ *UNIV*
unfolding *solves-ode-def* **apply**(*subgoal-tac* $((\lambda x. -\ s\ a \cdot x + s\ v)$ *has-vderiv-on* $(\lambda x. -\ s\ a))\ \{0..t\})$
using *obs* **apply** (*simp add*: *has-vderiv-on-def*) **by**(*rule galilean-transform*)
have *3*:*unique-on-bounded-closed* $0\ \{0..t\}\ (s\ v)\ (\lambda t\ r. -\ \varphi_s\ t\ a)$ *UNIV* (*if* $t = 0$ *then* 1 *else* $1/(t+1)$)
apply(*simp add*: *ubc-definitions del*: *comp-apply*, *rule conjI*)
using *rHyp tHyp obs* **apply**(*simp-all del*: *comp-apply*)
apply(*clarify*, *rule continuous-intros*) **prefer** *3* **apply** *safe*
apply(*rule continuous-intros*)+
apply(*auto intro*: *continuous-intros*)
by (*metis continuous-on-const continuous-on-eq*)
thus $\varphi_s\ r\ v = s\ v - s\ a \cdot r$
apply(*rule-tac unique-on-bounded-closed.unique-solution*[*of* $0\ \{0..t\}\ s\ v$ $(\lambda t\ r. -\ \varphi_s\ t\ a)$ *UNIV* (*if* $t = 0$ *then* 1 *else* $1\ /\ (t + 1)$) $(\lambda t.\ \varphi_s\ t\ v)$])
using *rHyp tHyp 1 2* **and** *3* **by** *auto*
qed

lemma *single-hop-ball*:
PRE $(\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge s\ ''g'' > 0 \wedge 1 \geq c \wedge c \geq 0)$
(((*ODEsystem* $[(''x'', \lambda\ s.\ s\ ''v''),(''v'',\lambda\ s. -\ s\ ''g'')]$ *with* $(\lambda\ s.\ 0 \leq s\ ''x'')$));
(*IF* $(\lambda\ s.\ s\ ''x'' = 0)$ *THEN* $(''v'' ::= (\lambda\ s. -\ c \cdot s\ ''v''))$ *ELSE* $(''v'' ::= (\lambda\ s.\ s\ ''v''))$ *FI*))
POST $(\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H)$
apply(*simp*, *subst dS*[*of* $[\lambda\ t\ s. -\ s\ ''g'' \cdot t\ \hat{}\ 2/2 + s\ ''v'' \cdot t + s\ ''x'', \lambda\ t\ s. -\ s\ ''g'' \cdot t + s\ ''v'']$])
— Given solution is actually a solution.
apply(*simp add*: *vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def has-vderiv-on-singleton*, *safe*)
apply(*rule galilean-transform-eq*, *simp*)+
apply(*rule galilean-transform*)+
— Uniqueness of the flow.
apply(*rule ubcStoreUniqueSol*, *simp*)
apply(*simp add*: *vdiff-def del*: *comp-apply*)
apply(*auto intro*: *continuous-intros del*: *comp-apply*)[*1*]

apply(*rule continuous-intros*)+
apply(*simp add*: *vdiff-def*, *safe*)
apply(*clarsimp*) **subgoal for** $s\ X\ t\ \tau$
apply(*rule flow-vel-is-galilean-vel2*[*of X ''x''*])
by(*simp-all add*: *varDiffs-def vdiff-def*)
apply(*simp add*: *vdiff-def varDiffs-def solvesStoreIVP-def*)
apply(*simp add*: *vdiff-def varDiffs-def solvesStoreIVP-def solves-ode-def has-vderiv-on-singleton galilean-transform-eq galilean-transform*)
— Relation Between the guard and the postcondition.
by(*auto simp*: *vdiff-def p2r-def*)

— Example of hybrid program verified with differential weakening.
lemma *system-where-the-guard-implies-the-postcondition*:
$PRE\ (\lambda\ s.\ s\ ''x'' = 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''x'' + 1))]\ with\ (\lambda\ s.\ s\ ''x'' \geq 0))$
$POST\ (\lambda\ s.\ s\ ''x'' \geq 0)$
using *dWeakening* **by** *blast*

lemma *system-where-the-guard-implies-the-postcondition2*:
$PRE\ (\lambda\ s.\ s\ ''x'' = 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''x'' + 1))]\ with\ (\lambda\ s.\ s\ ''x'' \geq 0))$
$POST\ (\lambda\ s.\ s\ ''x'' \geq 0)$
apply(*clarify*, *simp add*: *p2r-def*)
apply(*simp add*: *rel-ad-def rel-antidomain-kleene-algebra.addual.ars-r-def*)
apply(*simp add*: *rel-antidomain-kleene-algebra.fbox-def*)
apply(*simp add*: *relcomp-def rel-ad-def guarDiffEqtn-def solvesStoreIVP-def*)
by *auto*

— Example of system proved with a differential invariant.
lemma *circular-motion*:
$PRE\ (\lambda\ s.\ (s\ ''x'') \cdot (s\ ''x'') + (s\ ''y'') \cdot (s\ ''y'') - (s\ ''r'') \cdot (s\ ''r'') = 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''y'')),(''y'',(\lambda\ s.\ -\ s\ ''x''))]\ with\ G)$
$POST\ (\lambda\ s.\ (s\ ''x'') \cdot (s\ ''x'') + (s\ ''y'') \cdot (s\ ''y'') - (s\ ''r'') \cdot (s\ ''r'') = 0)$
apply(*rule-tac* η=$(t_V\ ''x'')\odot(t_V\ ''x'') \oplus (t_V\ ''y'')\odot(t_V\ ''y'') \oplus (\ominus(t_V\ ''r'')\odot(t_V\ ''r''))$
and *uInput*=$[t_V\ ''y'', \ominus\ (t_V\ ''x'')]$ **in** *dInvForTrms*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*clarsimp*, *erule-tac* x=$''r''$ **in** *allE*)
by *simp*

— Example of systems proved with differential invariants, cuts and weakenings.
declare *d-p2r* [*simp del*]
lemma *motion-with-constant-velocity-and-invariants*:
$PRE\ (\lambda\ s.\ s\ ''x'' > s\ ''y'' \wedge s\ ''v'' > 0)$
$(ODEsystem\ [(''x'',\ \lambda\ s.\ s\ ''v'')]\ with\ (\lambda\ s.\ True))$
$POST\ (\lambda\ s.\ s\ ''x'' >\ s\ ''y'')$
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''v'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 0) \prec (t_V\ ''v'')$ **and** *uInput*=$[t_V\ ''v'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* x=$''v''$ **in** *allE*, *simp*)

apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''x'' > s\ ''y''$ **in** *dCut*)
apply(*rule-tac* $\varphi{=}(t_V\ \ ''y'') \prec (t_V\ \ ''x'')$ **and** *uInput*=$[t_V\ \ ''v'']$ **and**
$F{=}\lambda\ s.\ \ s\ ''x'' > s\ ''y''$ **in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''y''$ **in** *allE*, *simp*)
using *dWeakening* **by** *simp*

lemma *motion-with-constant-acceleration-and-invariants*:
$PRE\ (\lambda\ s.\ s\ ''y'' < s\ ''x''\ \ \wedge s\ ''v'' \geq 0 \wedge s\ ''a'' > 0)$
$(ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v'')),(''v'',(\lambda\ s.\ s\ ''a''))]\ with\ (\lambda\ s.\ True))$
$POST\ (\lambda\ s.\ (s\ ''y'' < s\ ''x''))$
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''a'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 0) \prec (t_V\ ''a'')$ **and** *uInput*=$[t_V\ ''v'',\ t_V\ ''a'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''a''$ **in** *allE*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''v'' \geq 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ \ 0) \preceq (t_V\ \ ''v'')$ **and** *uInput*=$[t_V\ \ ''v'',\ t_V\ \ ''a'']$ **in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''x'' >\ \ s\ ''y''$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V\ \ ''y'') \prec (t_V\ \ ''x'')$ **and** *uInput*=$[t_V\ \ ''v'',\ t_V\ \ ''a'']$**in** *dInvFinal*)
apply(*simp-all add*: *varDiffs-def vdiff-def*, *clarify*, *erule-tac* $x{=}''y''$ **in** *allE*, *simp*)
using *dWeakening* **by** *simp*

— We revisit the two modes example from before, and prove it with invariants.
lemma *single-hop-ball-and-invariants*:
$PRE\ (\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge s\ ''g'' > 0 \wedge 1 \geq c \wedge c \geq 0)$
$(((ODEsystem\ [(''x'',\ \lambda\ s.\ s\ ''v''),(''v'',\lambda\ s.\ -\ s\ ''g'')]\ with\ (\lambda\ s.\ 0 \leq s\ ''x'')));$
$(IF\ (\lambda\ s.\ s\ ''x'' = 0)\ THEN\ (''v''\ ::= (\lambda\ s.\ -\ c \cdot s\ ''v''))\ ELSE\ (''v''\ ::= (\lambda\ s.\ s\ ''v''))\ FI))$
$POST\ (\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H)$
apply(*simp add*: *d-p2r*, *subgoal-tac rdom* $\lceil \lambda s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge 0 < s\ ''g'' \wedge c \leq 1 \wedge 0 \leq c\rceil$
$\subseteq wp\ (ODEsystem\ [(''x'',\ \lambda s.\ s\ ''v''),\ (''v'',\ \lambda s.\ -\ s\ ''g'')]\ with\ (\lambda s.\ 0 \leq s\ ''x''))$
$\lceil inf\ (sup\ (-\ (\lambda s.\ s\ ''x'' = 0))\ (\lambda s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H))\ (sup\ (\lambda s.\ s\ ''x'' = 0)\ (\lambda s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H))\rceil$)
apply(*simp add*: *d-p2r*, *rule-tac* $C = \lambda\ s.\ \ s\ ''g'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ \ 0) \prec (t_V\ \ ''g'')$ **and** *uInput*=$[t_V\ \ ''v'',\ \ominus\ t_V\ \ ''g'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''g''$ **in** *allE*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''v'' \leq 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V\ \ ''v'') \preceq (t_C\ \ 0)$ **and** *uInput*=$[t_V\ \ ''v'',\ \ominus\ t_V\ \ ''g'']$ **in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''x'' \leq\ \ H$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_V\ \ ''x'') \preceq (t_C\ \ H)$ **and** *uInput*=$[t_V\ \ ''v'',\ \ominus\ t_V\ \ ''g'']$**in** *dInvFinal*)

apply(*simp-all add*: *varDiffs-def vdiff-def*)
using *dWeakening* **by** *simp*

— Finally, we add a well known example in the hybrid systems community, the bouncing ball.
lemma *bouncing-ball-invariant*:$0 \leq x \Longrightarrow 0 < g \Longrightarrow 2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v \Longrightarrow (x{::}real) \leq H$
proof−
assume $0 \leq x$ **and** $0 < g$ **and** $2 \cdot g \cdot x = 2 \cdot g \cdot H - v \cdot v$
then have $v \cdot v = 2 \cdot g \cdot H - 2 \cdot g \cdot x \wedge 0 < g$ **by** *auto*
hence $*$:$v \cdot v = 2 \cdot g \cdot (H - x) \wedge 0 < g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
from *this* **have** $(v \cdot v)/(2 \cdot g) = (H - x)$ **by** *auto*
also from $*$ **have** $(v \cdot v)/(2 \cdot g) \geq 0$
by (*meson divide-nonneg-pos linordered-field-class.sign-simps*(*44*) *zero-less-numeral*)

ultimately have $H - x \geq 0$ **by** *linarith*
thus *?thesis* **by** *auto*
qed

lemma *bouncing-ball*:
$PRE\ (\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge s\ ''g'' > 0)$
$((ODEsystem\ [(''x'', \lambda\ s.\ s\ ''v''),(''v'',\lambda\ s.\ -\ s\ ''g'')]\ with\ (\lambda\ s.\ 0 \leq s\ ''x''));$
$(IF\ (\lambda\ s.\ s\ ''x'' = 0)\ THEN\ (''v'' ::= (\lambda\ s.\ -\ s\ ''v''))\ ELSE\ (Id)\ FI))^*$
$POST\ (\lambda\ s.\ 0 \leq s\ ''x'' \wedge s\ ''x'' \leq H)$
apply(*rule rel-antidomain-kleene-algebra.fbox-starI*[*of* - $\lceil\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge$
$2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - (s\ ''v'' \cdot s\ ''v'')\rceil$])
apply(*simp*, *simp add*: *d-p2r*)
apply(*subgoal-tac*
rdom $\lceil\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''\rceil$
$\subseteq wp\ (ODEsystem\ [(''x'', \lambda s.\ s\ ''v''), (''v'', \lambda s.\ -\ s\ ''g'')]\ with\ (\lambda s.\ 0 \leq s\ ''x''))$
$\lceil inf\ \ (sup\ (-\ (\lambda s.\ s\ ''x'' = 0))\ (\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x'' =$
$2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''))$
$(sup\ (\lambda s.\ s\ ''x'' = 0)\ (\lambda s.\ 0 \leq s\ ''x'' \wedge 0 < s\ ''g'' \wedge 2 \cdot s\ ''g'' \cdot s\ ''x'' =$
$2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''))\rceil$)
apply(*simp add*: *d-p2r*)
apply(*rule-tac* $C = \lambda\ s.\ \ s\ ''g'' > 0$ **in** *dCut*)
apply(*rule-tac* $\varphi = ((t_C\ \ 0) \prec (t_V\ \ ''g''))$ **and** *uInput*=$[t_V\ \ ''v'', \ominus\ t_V\ \ ''g'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''g''$ **in** *allE*, *simp*)
apply(*rule-tac* $C = \lambda\ s.\ 2 \cdot s\ ''g'' \cdot s\ ''x'' = 2 \cdot s\ ''g'' \cdot H - s\ ''v'' \cdot s\ ''v''$ **in** *dCut*)
apply(*rule-tac* $\varphi = (t_C\ 2) \odot (t_V\ ''g'') \odot (t_C\ H) \oplus (\ominus\ ((t_V\ ''v'') \odot (t_V\ ''v'')))$
$\doteq (t_C\ 2) \odot (t_V\ ''g'') \odot (t_V\ ''x'')$ **and** *uInput*=$[t_V\ ''v'', \ominus\ t_V\ ''g'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''g''$ **in** *allE*, *simp*)

apply(*rule dWeakening*, *clarsimp*)
using *bouncing-ball-invariant* **by** *auto*

declare *d-p2r* [*simp*]

end