CPSVerification

CPSVerification

August 14, 2019

Contents

1 Hybrid Systems Preliminaries **5**
- 1.1 Miscellaneous . . . 5
  - 1.1.1 Functions . . . 5
  - 1.1.2 Limits . . . 5
  - 1.1.3 Real numbers . . . 6
- 1.2 Analisys . . . 8
  - 1.2.1 Single variable derivatives . . . 8
  - 1.2.2 Filters . . . 11
  - 1.2.3 Multivariable derivatives . . . 12
- 1.3 Dynamical Systems . . . 14
  - 1.3.1 Initial value problems and orbits . . . 14
  - 1.3.2 Differential Invariants . . . 15
  - 1.3.3 Picard-Lindeloef . . . 18
  - 1.3.4 Flows for ODEs . . . 20

2 Linear Algebra for Hybrid Systems **27**
- 2.1 Vector operations . . . 27
- 2.2 Matrix norms . . . 29
  - 2.2.1 Matrix operator norm . . . 29
  - 2.2.2 Matrix maximum norm . . . 33
- 2.3 Picard Lindeloef for linear systems . . . 33
- 2.4 Matrix Exponential . . . 34
  - 2.4.1 Squared matrices operations . . . 34
  - 2.4.2 Squared matrices form Banach space . . . 36
- 2.5 Flow for squared matrix systems . . . 40

3 Hybrid System Verification **43**
- 3.1 Verification of regular programs . . . 43
- 3.2 Verification of hybrid programs . . . 45
  - 3.2.1 Verification by providing solutions . . . 45
  - 3.2.2 Verification with differential invariants . . . 46
  - 3.2.3 Derivation of the rules of dL . . . 47
  - 3.2.4 Examples . . . 50

4 Hybrid System Verification with relations **59**
4.1 Verification of regular programs 59
4.2 Verification of hybrid programs 61
4.2.1 Verification by providing solutions 61
4.2.2 Verification with differential invariants 61
4.2.3 Derivation of the rules of dL 62
4.2.4 Examples 65

5 Hybrid System Verification with relations **73**
5.1 Verification of regular programs 73
5.2 Verification of hybrid programs 74
5.2.1 Verification by providing solutions 75
5.2.2 Verification with differential invariants 75
5.2.3 Derivation of the rules of dL 76
5.2.4 Examples 77

6 Hybrid System Verification with nondeterministic functions **85**
6.1 Nondeterministic Functions 85
6.2 Verification of regular programs 88
6.3 Verification of hybrid programs 90
6.3.1 Verification by providing solutions 90
6.3.2 Verification with differential invariants 91
6.3.3 Derivation of the rules of dL 91
6.3.4 Examples 94
6.4 VC_diffKAD 101
6.4.1 Stack Theories Preliminaries: VC_KAD and ODEs . . 101
6.4.2 VC_diffKAD Preliminaries 104
6.4.3 Phase Space Relational Semantics 115
6.4.4 Derivation of Differential Dynamic Logic Rules 117
6.4.5 Rules Testing 134

theory *hs-prelims*
imports *Ordinary-Differential-Equations.Picard-Lindeloef-Qualitative*

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 Limits

lemma *cSup-eq-linorder*:
  fixes $c::'a::conditionally\text{-}complete\text{-}linorder$
  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 $c::'a::conditionally\text{-}complete\text{-}lattice$
  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-ring-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 ‹$n \geq$ *?N*› **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

lemma *suminf-eq-sum*:
fixes f :: $nat \Rightarrow ('a$::*real-normed-vector*)
assumes $\bigwedge n.\ n > m \Longrightarrow f\ n = 0$
shows $(\sum n.\ f\ n) = (\sum n \leq m.\ f\ n)$
using *assms* **by** (*meson atMost-iff finite-atMost not-le suminf-finite*)

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*:$sqrt\ (real\ n) \le real\ n$
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 $(a{::}real) \ge 0 \Longrightarrow b \ge 0 \Longrightarrow a\hat{\ }2 \le b * a \Longrightarrow a \le b$
and $(a{::}real) \ge 0 \Longrightarrow b \ge 0 \Longrightarrow a\hat{\ }2 \le a * b \Longrightarrow a \le b$
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*))

lemma *abs-le-eq*:
shows $(r{::}real) > 0 \Longrightarrow (|x| < r) = (-r < x \wedge x < r)$
and $(r{::}real) > 0 \Longrightarrow (|x| \le r) = (-r \le x \wedge x \le r)$
by *linarith linarith*

lemma *real-ivl-eqs*:
assumes $0 < r$
shows $ball\ x\ r = \{x-r<--< x+r\}$ **and** $\{x-r<--< x+r\} = \{x-r<..< x+r\}$
and $ball\ (r\ /\ 2)\ (r\ /\ 2) = \{0<--<r\}$ **and** $\{0<--<r\} = \{0<..<r\}$
and $ball\ 0\ r = \{-r<--<r\}$ **and** $\{-r<--<r\} = \{-r<..<r\}$
and $cball\ x\ r = \{x-r--x+r\}$ **and** $\{x-r--x+r\} = \{x-r..x+r\}$
and $cball\ (r\ /\ 2)\ (r\ /\ 2) = \{0--r\}$ **and** $\{0--r\} = \{0..r\}$
and $cball\ 0\ r = \{-r--r\}$ **and** $\{-r--r\} = \{-r..r\}$
unfolding *open-segment-eq-real-ivl closed-segment-eq-real-ivl*
using *assms* **apply**(*auto simp*: *cball-def ball-def dist-norm*)
by(*simp-all add*: *field-simps*)

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

lemma [*trig-simps*, *simp*]:
fixes x :: $'a$:: {*banach*,*real-normed-field*}
shows $(x * cos\ t + y * sin\ t)^2 + (y * cos\ t - x * sin\ t)^2 = x^2 + y^2$
using *trig-simps*(*10*)[*of y t x*] **by** (*simp add*: *add.commute*)

thm *trig-simps*

1.2 Analisys

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 *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*]
and *has-vderiv-on-id* [*poly-derivatives*]
and *derivative-intros*(*191*) [*poly-derivatives*]
and *derivative-intros*(*192*) [*poly-derivatives*]

and *derivative-intros*(*194*) [*poly-derivatives*]

lemma *has-vector-derivative-mult-const* [*derivative-intros*]:
((∗) *a* *has-vector-derivative* *a*) *F*
by (*auto intro*: *derivative-eq-intros*)

lemma *has-derivative-mult-const* [*derivative-intros*]: $D\ (*)\ a \mapsto (\lambda x.\ x *_R a)\ F$
using *has-vector-derivative-mult-const* **unfolding** *has-vector-derivative-def* **by** *simp*

lemma *has-vderiv-on-mult-const* [*derivative-intros*]: $D\ (*)\ a = (\lambda x.\ a)$ *on* T
using *has-vector-derivative-mult-const* **unfolding** *has-vderiv-on-def* **by** *auto*

lemma *has-vderiv-on-power2* [*derivative-intros*]: D *power2* $= (*)\ 2$ *on* T
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
by(*rule-tac* $f'1{=}\lambda\ t.\ t$ **in** *derivative-eq-intros*(*15*)) *auto*

lemma *has-vderiv-on-divide-cnst* [*derivative-intros*]: $a \neq 0 \Longrightarrow D\ (\lambda t.\ t/a) = (\lambda t.\ 1/a)$ *on* T
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
apply(*rule-tac* $f'1{=}\lambda t.\ t$ **and** $g'1{=}\lambda\ x.\ 0$ **in** *derivative-eq-intros*(*18*))
by(*auto intro*: *derivative-eq-intros*)

lemma [*poly-derivatives*]: $g = (*)\ 2 \Longrightarrow D$ *power2* $= g$ *on* T
using *has-vderiv-on-power2* **by** *auto*

lemma [*poly-derivatives*]: $D\ f = f'$ *on* $T \Longrightarrow g = (\lambda t.\ -\ f'\ t) \Longrightarrow D\ (\lambda t.\ -\ f\ t) = g$ *on* T
using *has-vderiv-on-uminus* **by** *auto*

lemma [*poly-derivatives*]: $a \neq 0 \Longrightarrow g = (\lambda t.\ 1/a) \Longrightarrow D\ (\lambda t.\ t/a) = g$ *on* T
using *has-vderiv-on-divide-cnst* **by** *auto*

lemma *has-vderiv-on-compose-eq*:
assumes $D\ f = f'$ *on* g ‘ T
and $D\ g = g'$ *on* T
and $h = (\lambda x.\ g'\ x *_R f'\ (g\ x))$
shows $D\ (\lambda t.\ f\ (g\ t)) = h$ *on* T
apply(*subst ssubst*[*of h*], *simp*)
using *assms has-vderiv-on-compose* **by** *auto*

lemma *vderiv-on-compose-add* [*derivative-intros*]:
assumes $D\ x = x'$ *on* $(\lambda\tau.\ \tau + t)$ ‘ T
shows $D\ (\lambda\tau.\ x\ (\tau + t)) = (\lambda\tau.\ x'\ (\tau + t))$ *on* T
apply(*rule has-vderiv-on-compose-eq*[*OF assms*])
by(*auto intro*: *derivative-intros*)

lemma [*poly-derivatives*]:
assumes $(a{::}real) \neq 0$ **and** $D\ f = f'$ *on* T **and** $g = (\lambda t.\ (f'\ t)/a)$

shows $D\ (\lambda t.\ (f\ t)/a) = g\ on\ T$
apply(*rule has-vderiv-on-compose-eq*[*of* $\lambda t.\ t/a\ \lambda t.\ 1/a$])
using *assms* **by**(*auto intro*: *poly-derivatives*)

lemma [*poly-derivatives*]:
fixes f::*real* $\Rightarrow$ *real*
assumes $D\,f = f'\ on\ T$ **and** $g = (\lambda t.\ 2 *_R (f\ t) * (f'\ t))$
shows $D\ (\lambda t.\ (f\ t)\hat{}2) = g\ on\ T$
apply(*rule has-vderiv-on-compose-eq*[*of* $\lambda t.\ t\hat{}2$])
using *assms* **by**(*auto intro*!: *poly-derivatives*)

lemma *has-vderiv-on-cos*: $D\,f = f'\ on\ T \Longrightarrow D\ (\lambda t.\ cos\ (f\ t)) = (\lambda t.\ -\ sin\ (f\ t)$ $*_R\ (f'\ t))\ on\ T$
apply(*rule has-vderiv-on-compose-eq*[*of* $\lambda t.\ cos\ t$])
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
by(*auto intro*!: *derivative-eq-intros simp*: *fun-eq-iff*)

lemma *has-vderiv-on-sin*: $D\ f = f'\ on\ T \Longrightarrow D\ (\lambda t.\ sin\ (f\ t)) = (\lambda t.\ cos\ (f\ t)$ $*_R\ (f'\ t))\ on\ T$
apply(*rule has-vderiv-on-compose-eq*[*of* $\lambda t.\ sin\ t$])
unfolding *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
by(*auto intro*!: *derivative-eq-intros simp*: *fun-eq-iff*)

lemma [*poly-derivatives*]:
assumes $D\,f = f'\ on\ T$ **and** $g = (\lambda t.\ -\ sin\ (f\ t) *_R (f'\ t))$
shows $D\ (\lambda t.\ cos\ (f\ t)) = g\ on\ T$
using *assms* **and** *has-vderiv-on-cos* **by** *auto*

lemma [*poly-derivatives*]:
assumes $D\,f = f'\ on\ T$ **and** $g = (\lambda t.\ cos\ (f\ t) *_R (f'\ t))$
shows $D\ (\lambda t.\ sin\ (f\ t)) = g\ on\ T$
using *assms* **and** *has-vderiv-on-sin* **by** *auto*

lemma $D\ (\lambda t.\ a * t^2\ /\ 2) = (*)\ a\ on\ T$
by(*auto intro*!: *poly-derivatives*)

lemma $D\ (\lambda t.\ a * t^2\ /\ 2 + v * t + x) = (\lambda t.\ a * t + v)\ on\ T$
by(*auto intro*!: *poly-derivatives*)

lemma $D\ (\lambda r.\ a * r + v) = (\lambda t.\ a)\ on\ T$
by(*auto intro*!: *poly-derivatives*)

lemma $D\ (\lambda t.\ v * t - a * t^2\ /\ 2 + x) = (\lambda x.\ v - a * x)\ on\ T$
by(*auto intro*!: *poly-derivatives*)

lemma $D\ (\lambda t.\ v - a * t) = (\lambda x.\ -\ a)\ on\ T$
by(*auto intro*!: *poly-derivatives*)

thm *poly-derivatives*

1.2.2 Filters

lemma *eventually-at-within-mono*:
  assumes $t \in$ *interior* T **and** $T \subseteq S$
    and *eventually* P (*at* t *within* T)
  shows *eventually* P (*at* t *within* S)
  by (*meson assms eventually-within-interior interior-mono subsetD*)

lemma *netlimit-at-within-mono*:
  fixes t::$'a$::{*perfect-space*,*t2-space*}
  assumes $t \in$ *interior* T **and** $T \subseteq S$
  shows *netlimit* (*at* t *within* S) = t
  using *assms*(*1*) *interior-mono*[*OF* ‹$T \subseteq S$›] *netlimit-within-interior* **by** *auto*

lemma *has-derivative-at-within-mono*:
  assumes (t::*real*) $\in$ *interior* T **and** $T \subseteq S$
    and $D\ f \mapsto f'$ *at* t *within* T
  shows $D\ f \mapsto f'$ *at* t *within* S
  using *assms*(*3*) **apply**(*unfold has-derivative-def tendsto-iff*, *safe*)
  unfolding *netlimit-at-within-mono*[*OF assms*(*1*,*2*)] *netlimit-within-interior*[*OF assms*(*1*)]
  by (*rule eventually-at-within-mono*[*OF assms*(*1*,*2*)]) *simp*

lemma *eventually-all-finite2*:
  fixes P :: ($'a$::*finite*) $\Rightarrow$ $'b$ $\Rightarrow$ *bool*
  assumes h:$\forall i$. *eventually* ($P\ i$) F
  shows *eventually* (λx. $\forall i$. $P\ i\ x$) F
proof(*unfold eventually-def*)
  let *?F* = *Rep-filter* F
  have *obs*: $\forall i$. *?F* ($P\ i$)
    using h **by** *auto*
  have *?F* (λx. $\forall i \in$ *UNIV*. $P\ i\ x$)
    apply(*rule finite-induct*)
    by(*auto intro*: *eventually-conj simp*: *obs h*)
  thus *?F* (λx. $\forall i$. $P\ i\ x$)
    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* (λx. $\forall i$. $P\ i\ x$) F
    using *h1 eventually-all-finite2* **by** *blast*
  thus *eventually* Q F
    unfolding *eventually-def*
    using *h2 eventually-mono* **by** *auto*
qed

1.2.3 Multivariable derivatives

lemma *frechet-vec-lambda*:
fixes $f::real \Rightarrow ('a::banach)\hat{}('m::finite)$ **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 $\varepsilon::real$ **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 ‹$0 < \varepsilon$› **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 $sqrt\ (?c^2 * (\sum i{\in}UNIV.\ (\|?u\ t\ i\|)^2)) < sqrt\ (\varepsilon^2)$
using *real-sqrt-less-iff* **by** *blast*
also have ... $= \varepsilon$
using ‹$0 < \varepsilon$› **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 ‹$0 < \varepsilon$› *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 $\|(?\Delta f\ y - ?\Delta\ y *_R f'\ x)\ \$\ i\| \leq \|?\Delta f\ y - ?\Delta\ y *_R f'\ x\|$
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 $\|?u\ y\ i\| \leq \|?\Delta f\ y - ?\Delta\ y *_R f'\ x\|$
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*)) ⇒ ('*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*)

end
theory *hs-prelims-dyn-sys*
imports *hs-prelims*

begin

1.3 Dynamical Systems

1.3.1 Initial value problems and orbits

notation *image* ($\mathcal{P}$)

lemma *image-le-pred*: $(\mathcal{P}\ f\ A \subseteq \{s.\ G\ s\}) = (\forall x{\in}A.\ G\ (f\ x))$
unfolding *image-def* **by** *force*

definition *ivp-sols* $f\ T\ S\ t_0\ s = \{X\ |X.\ (D\ X = (\lambda t.\ f\ t\ (X\ t))\ on\ T) \wedge X\ t_0 = s \wedge X \in T \to S\}$

lemma *ivp-solsI*:
assumes $D\ X = (\lambda t.\ f\ t\ (X\ t))$ *on* $T\ X\ t_0 = s\ X \in T \to S$
shows $X \in$ *ivp-sols* $f\ T\ S\ t_0\ s$
using *assms* **unfolding** *ivp-sols-def* **by** *blast*

lemma *ivp-solsD*:
assumes $X \in$ *ivp-sols* $f\ T\ S\ t_0\ s$
shows $D\ X = (\lambda t.\ f\ t\ (X\ t))$ *on T*
and $X\ t_0 = s$ **and** $X \in T \to S$
using *assms* **unfolding** *ivp-sols-def* **by** *auto*

abbreviation *down* $T\ t \equiv \{\tau{\in}T.\ \tau{\leq}\ t\}$

definition *g-orbit* :: (*real* ⇒ '*a*) ⇒ ('*a* ⇒ *bool*) ⇒ *real set* ⇒ '*a set* (γ_{Guard})
where $\gamma_{Guard}\ X\ G\ T = \bigcup\ \{\mathcal{P}\ X\ (down\ T\ t)\ |t.\ \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}\}$

lemma *g-orbit-eq*: $\gamma_{Guard}\ X\ G\ T = \{X\ t\ |t.\ t \in T \wedge (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau))\}$
unfolding *g-orbit-def* **by** *safe* (*auto simp*: *subset-eq*)

lemma $\gamma_{Guard}\ X\ (\lambda s.\ True)\ T = \{X\ t\ |t.\ t \in T\}$
  unfolding *g-orbit-eq* **by** *simp*

definition *g-orbital* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow bool) \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow real \Rightarrow ('a::real\text{-}normed\text{-}vector) \Rightarrow 'a\ set$
  where $g\text{-}orbital\ f\ G\ T\ S\ t_0\ s = \bigcup\{\gamma_{Guard}\ X\ G\ T\ |X.\ X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s\}$

lemma *g-orbital-eq*: $g\text{-}orbital\ f\ G\ T\ S\ t_0\ s = \{X\ t|t\ X.\ t \in T \wedge (\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \wedge X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s\ \}$
  unfolding *g-orbital-def ivp-sols-def g-orbit-eq image-le-pred* **by** *auto*

lemma $g\text{-}orbital\ f\ G\ T\ S\ t_0\ s = \{X\ t|t\ X.\ t \in T \wedge (D\ X = (f \circ X)\ on\ T) \wedge X\ t_0 = s \wedge X \in T \rightarrow S \wedge (\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\})\}$
  unfolding *g-orbital-eq ivp-sols-def* **by** *auto*

lemma $g\text{-}orbital\ f\ G\ T\ S\ t_0\ s = (\bigcup\ X{\in}ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \gamma_{Guard}\ X\ G\ T)$
  unfolding *g-orbital-def ivp-sols-def g-orbit-eq* **by** *auto*

lemma *g-orbitalI*:
  assumes $X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s$
    and $t \in T$ **and** $(\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\})$
  shows $X\ t \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
  using *assms* **unfolding** *g-orbital-eq(1)* **by** *auto*

lemma *g-orbitalD*:
  assumes $s' \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
  obtains X **and** t **where** $X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s$
  and $X\ t = s'$ **and** $t \in T$ **and** $(\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\})$
  using *assms* **unfolding** *g-orbital-def g-orbit-eq* **by** *auto*

1.3.2 Differential Invariants

definition *diff-invariant* :: $('a \Rightarrow bool) \Rightarrow (('a::real\text{-}normed\text{-}vector) \Rightarrow 'a) \Rightarrow real\ set \Rightarrow$
  $'a\ set \Rightarrow real \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$
  where $diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G \equiv (\bigcup \circ (\mathcal{P}\ (g\text{-}orbital\ f\ G\ T\ S\ t_0)))\ \{s.\ I\ s\} \subseteq \{s.\ I\ s\}$

lemma *diff-invariant-eq*: $diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G =$
  $(\forall\, s.\ I\ s \longrightarrow (\forall\, X{\in}ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ (\forall\, t{\in}T.(\forall\, \tau{\in}(down\ T\ t).\ G\ (X\ \tau)) \longrightarrow I\ (X\ t))))$
  unfolding *diff-invariant-def g-orbital-eq image-le-pred* **by** *auto*

lemma *diff-inv-eq-inv-set*:
  $diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G = (\forall\, s.\ I\ s \longrightarrow (g\text{-}orbital\ f\ G\ T\ S\ t_0\ s) \subseteq \{s.\ I\ s\})$

unfolding *diff-invariant-eq g-orbital-eq image-le-pred* **by** *auto*

named-theorems *diff-invariant-rules rules for obtainin differential invariants.*

lemma [*diff-invariant-rules*]:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $\forall X.\ (D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T) \longrightarrow (D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = ((*_R)\ 0)\ on\ T)$
shows *diff-invariant* $(\lambda s.\ \mu\ s = \nu\ s)\ f\ T\ S\ t_0\ G$
proof(*simp add*: *diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix X τ **assume** *tHyp*:$\tau \in T$ **and** *x-ivp*:$D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T\ \mu\ (X\ t_0) = \nu\ (X\ t_0)$
hence *obs1*: $\forall t{\in}T.\ D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) \mapsto (\lambda\tau.\ \tau *_R 0)\ at\ t\ within\ T$
using *assms* **by** (*auto simp*: *has-vderiv-on-def has-vector-derivative-def*)
have *obs2*: $\{t_0{-}{-}\tau\} \subseteq T$
using *closed-segment-subset-interval tHyp Thyp* **by** *blast*
hence $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \tau *_R 0)\ on\ \{t_0{-}{-}\tau\}$
using *obs1 x-ivp* **by** (*auto intro*!: *has-derivative-subset*[*OF* - *obs2*]
simp: *has-vderiv-on-def has-vector-derivative-def*)
then obtain t **where** $t \in \{t_0{-}{-}\tau\}$ **and** $\mu\ (X\ \tau) - \nu\ (X\ \tau) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (\tau - t_0) * t *_R 0$
using *mvt-very-simple-closed-segmentE* **by** *blast*
thus $\mu\ (X\ \tau) = \nu\ (X\ \tau)$
by (*simp add*: *x-ivp*(*2*))
qed

lemma [*diff-invariant-rules*]:
fixes μ::$'a$::*banach* $\Rightarrow$ *real*
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $\forall X.\ (D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T) \longrightarrow (\forall\tau{\in}T.\ (\tau > t_0 \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)) \wedge$
$(\tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))) \wedge (D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ T)$
shows *diff-invariant* $(\lambda s.\ \nu\ s \leq \mu\ s)\ f\ T\ S\ t_0\ G$
proof(*simp add*: *diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix X τ **assume** $\tau \in T$ **and** *x-ivp*:$D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T\ \nu\ (X\ t_0) \leq \mu\ (X\ t_0)$
{**assume** $\tau \neq t_0$
hence *primed*: $\bigwedge\tau.\ \tau \in T \Longrightarrow \tau > t_0 \Longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$
$\bigwedge\tau.\ \tau \in T \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$
using *x-ivp assms* **by** *auto*
have *obs1*: $\forall t{\in}T.\ D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\mu'\ (X\ t) - \nu'\ (X\ t)))\ at\ t\ within\ T$
using *assms x-ivp* **by** (*auto simp*: *has-vderiv-on-def has-vector-derivative-def*)
have *obs2*: $\{t_0{<}{-}{-}{<}\tau\} \subseteq T$ $\{t_0{-}{-}\tau\} \subseteq T$
using $\langle\tau \in T\rangle$ *Thyp* $\langle\tau \neq t_0\rangle$ **by** (*auto simp*: *convex-contains-open-segment*
is-interval-convex-1 closed-segment-subset-interval)
hence $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ \{t_0{-}{-}\tau\}$
using *obs1 x-ivp* **by** (*auto intro*!: *has-derivative-subset*[*OF* - *obs2*(*2*)]

simp: *has-vderiv-on-def has-vector-derivative-def*)
then obtain t **where** $t \in \{t_0<-\!-<\tau\}$ **and**
$(\mu\ (X\ \tau) - \nu\ (X\ \tau)) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (\lambda\tau.\ \tau * (\mu'\ (X\ t) - \ \nu'\ (X\ t)))\ (\tau - t_0)$
using *mvt-simple-closed-segmentE* $\langle\tau \neq t_0\rangle$ **by** *blast*
hence *mvt*: $\mu\ (X\ \tau) - \nu\ (X\ \tau) = (\tau - t_0) * (\mu'\ (X\ t) - \ \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0))$
by *force*
have $\tau > t_0 \Longrightarrow t > t_0$ $\neg\ t_0 \leq \tau \Longrightarrow t < t_0$ $t \in T$
using $\langle t \in \{t_0<-\!-<\tau\}\rangle$ *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
moreover have $t > t_0 \Longrightarrow (\mu'\ (X\ t) - \ \nu'\ (X\ t)) \geq 0$ $t < t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \leq 0$
using *primed*(*1*,*2*)[*OF* $\langle t \in T\rangle$] **by** *auto*
ultimately have $(\tau - t_0) * (\mu'\ (X\ t) - \ \nu'\ (X\ t)) \geq 0$
apply(*case-tac* $\tau \geq t_0$) **by** (*force*, *auto simp*: *split-mult-pos-le*)
hence $(\tau - t_0) * (\mu'\ (X\ t) - \ \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0)) \geq 0$
using *x-ivp*(*2*) **by** *auto*
hence $\nu\ (X\ \tau) \leq \mu\ (X\ \tau)$
using *mvt* **by** *simp*}
thus $\nu\ (X\ \tau) \leq \mu\ (X\ \tau)$
using *x-ivp* **by** *blast*
qed

lemma [*diff-invariant-rules*]:
fixes μ::'*a*::*banach* $\Rightarrow$ *real*
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $\forall\ X.\ (D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T) \longrightarrow (\forall\tau{\in}T.\ (\tau > t_0 \longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)) \wedge$
$(\tau < t_0 \longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau))) \wedge (D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ T)$
shows *diff-invariant* $(\lambda s.\ \nu\ s < \mu\ s)\ f\ T\ S\ t_0\ G$
proof(*simp add*: *diff-invariant-eq ivp-sols-def*, *clarsimp*)
fix $X\ \tau$ **assume** $\tau \in T$ **and** *x-ivp*:$D\ X = (\lambda\tau.\ f\ (X\ \tau))\ on\ T$ $\nu\ (X\ t_0) < \mu\ (X\ t_0)$
{**assume** $\tau \neq t_0$
hence *primed*: $\bigwedge\tau.\ \tau \in T \Longrightarrow \tau > t_0 \Longrightarrow \mu'\ (X\ \tau) \geq \nu'\ (X\ \tau)$
$\bigwedge\tau.\ \tau \in T \Longrightarrow \tau < t_0 \Longrightarrow \mu'\ (X\ \tau) \leq \nu'\ (X\ \tau)$
using *x-ivp assms* **by** *auto*
have *obs1*: $\forall\,t{\in}T.\ D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) \mapsto (\lambda\tau.\ \tau *_R (\mu'\ (X\ t) - \nu'\ (X\ t)))\ at\ t\ within\ T$
using *assms x-ivp* **by** (*auto simp*: *has-vderiv-on-def has-vector-derivative-def*)
have *obs2*: $\{t_0<-\!-<\tau\} \subseteq T$ $\{t_0-\!-\tau\} \subseteq T$
using $\langle\tau \in T\rangle$ *Thyp* $\langle\tau \neq t_0\rangle$ **by** (*auto simp*: *convex-contains-open-segment is-interval-convex-1 closed-segment-subset-interval*)
hence $D\ (\lambda\tau.\ \mu\ (X\ \tau) - \nu\ (X\ \tau)) = (\lambda\tau.\ \mu'\ (X\ \tau) - \nu'\ (X\ \tau))\ on\ \{t_0-\!-\tau\}$
using *obs1 x-ivp* **by** (*auto intro*!: *has-derivative-subset*[*OF* - *obs2*(*2*)]
simp: *has-vderiv-on-def has-vector-derivative-def*)
then obtain t **where** $t \in \{t_0<-\!-<\tau\}$ **and**
$(\mu\ (X\ \tau) - \nu\ (X\ \tau)) - (\mu\ (X\ t_0) - \nu\ (X\ t_0)) = (\lambda\tau.\ \tau * (\mu'\ (X\ t) - \ \nu'\ (X$

t))) $(\tau - t_0)$
  using *mvt-simple-closed-segmentE* ‹$\tau \neq t_0$› **by** *blast*
 hence *mvt*: $\mu\ (X\ \tau) - \nu\ (X\ \tau) = (\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0))$
  by *force*
 have $\tau > t_0 \Longrightarrow t > t_0$ $\neg\ t_0 \leq \tau \Longrightarrow t < t_0$ $t \in T$
  using ‹$t \in \{t_0 <-- < \tau\}$› *obs2* **unfolding** *open-segment-eq-real-ivl* **by** *auto*
 moreover have $t > t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0$ $t < t_0 \Longrightarrow (\mu'\ (X\ t) - \nu'\ (X\ t)) \leq 0$
  using *primed*(*1*,*2*)[*OF* ‹$t \in T$›] **by** *auto*
 ultimately have $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) \geq 0$
  apply(*case-tac* $\tau \geq t_0$) **by** (*force*, *auto simp*: *split-mult-pos-le*)
 hence $(\tau - t_0) * (\mu'\ (X\ t) - \nu'\ (X\ t)) + (\mu\ (X\ t_0) - \nu\ (X\ t_0)) > 0$
  using *x-ivp*(*2*) **by** *auto*
 hence $\nu\ (X\ \tau) < \mu\ (X\ \tau)$
  using *mvt* **by** *simp*}
 thus $\nu\ (X\ \tau) < \mu\ (X\ \tau)$
  using *x-ivp* **by** *blast*
qed

lemma [*diff-invariant-rules*]:
assumes *diff-invariant* I_1 *f T S* t_0 *G*
  and *diff-invariant* I_2 *f T S* t_0 *G*
shows *diff-invariant* $(\lambda s.\ I_1\ s \wedge I_2\ s)$ *f T S* t_0 *G*
 using *assms* **unfolding** *diff-invariant-def* **by** *auto*

lemma [*diff-invariant-rules*]:
assumes *diff-invariant* I_1 *f T S* t_0 *G*
  and *diff-invariant* I_2 *f T S* t_0 *G*
shows *diff-invariant* $(\lambda s.\ I_1\ s \vee I_2\ s)$ *f T S* t_0 *G*
 using *assms* **unfolding** *diff-invariant-def* **by** *auto*

1.3.3 Picard-Lindeloef

A locale with the assumptions of Picard-Lindeloef theorem. It extends *ll-on-open-it* by assuming that $t_0 \in T$.

locale *picard-lindeloef* =
 fixes *f*::*real* ⇒ (′*a*::{*heine-borel*,*banach*}) ⇒ ′*a* **and** *T*::*real set* **and** *S*::′*a set* **and** t_0::*real*
 assumes *open-domain*: *open T open S*
  and *interval-time*: *is-interval T*
  and *init-time*: $t_0 \in T$
  and *cont-vec-field*: $\forall s \in S.$ *continuous-on* T $(\lambda t.\ f\ t\ s)$
  and *lipschitz-vec-field*: *local-lipschitz T S f*
begin

sublocale *ll-on-open-it T f S* t_0
 by (*unfold-locales*) (*auto simp*: *cont-vec-field lipschitz-vec-field interval-time open-domain*)

lemmas *subintervalI* = *closed-segment-subset-domain*

lemma *subintervalD*:
assumes $\{t_1 {-}{-} t_2\} \subseteq T$
shows $t_1 \in T$ **and** $t_2 \in T$
using *assms* **by** *auto*

lemma *csols-eq*: *csols* t_0 $s = \{(X,\ t).\ t \in T \wedge\ X \in$ *ivp-sols* $f\ \{t_0 {-}{-} t\}\ S\ t_0\ s\}$
unfolding *ivp-sols-def csols-def solves-ode-def* **using** *subintervalI*[*OF init-time*]
by *auto*

abbreviation *ex-ivl* $s \equiv$ *existence-ivl* t_0 s

lemma *unique-solution*:
assumes *xivp*: $D\ X = (\lambda t.\ f\ t\ (X\ t))$ *on* $\{t_0 {-}{-} t\}$ $X\ t_0 = s$ $X \in \{t_0 {-}{-} t\} \to S$
and $t \in T$
and *yivp*: $D\ Y = (\lambda t.\ f\ t\ (Y\ t))$ *on* $\{t_0 {-}{-} t\}$ $Y\ t_0 = s$ $Y \in \{t_0 {-}{-} t\} \to S$ **and**
$s \in S$
shows $X\ t = Y\ t$
proof−
have $(X,\ t) \in$ *csols* t_0 s
using *xivp* ‹$t \in T$› **unfolding** *csols-eq ivp-sols-def* **by** *auto*
hence *ivl-fact*: $\{t_0 {-}{-} t\} \subseteq$ *ex-ivl* s
unfolding *existence-ivl-def* **by** *auto*
have *obs*: $\bigwedge z\ T'.\ t_0 \in T' \wedge$ *is-interval* $T' \wedge T' \subseteq$ *ex-ivl* $s \wedge (z$ *solves-ode* $f)\ T'$
$S \Longrightarrow$
$z\ t_0 =$ *flow* $t_0\ s\ t_0 \Longrightarrow (\forall t{\in}T'.\ z\ t =$ *flow* $t_0\ s\ t)$
using *flow-usolves-ode*[*OF init-time* ‹$s \in S$›] **unfolding** *usolves-ode-from-def*
by *blast*
have $\forall \tau{\in}\{t_0 {-}{-} t\}.\ X\ \tau =$ *flow* $t_0\ s\ \tau$
using *obs*[*of* $\{t_0 {-}{-} t\}$ X] *xivp ivl-fact flow-initial-time*[*OF init-time* ‹$s \in S$›]
unfolding *solves-ode-def* **by** *simp*
also have $\forall \tau{\in}\{t_0 {-}{-} t\}.\ Y\ \tau =$ *flow* $t_0\ s\ \tau$
using *obs*[*of* $\{t_0 {-}{-} t\}$ Y] *yivp ivl-fact flow-initial-time*[*OF init-time* ‹$s \in S$›]
unfolding *solves-ode-def* **by** *simp*
ultimately show $X\ t = Y\ t$
by *auto*
qed

lemma *solution-eq-flow*:
assumes *xivp*: $D\ X = (\lambda t.\ f\ t\ (X\ t))$ *on ex-ivl* s $X\ t_0 = s$ $X \in$ *ex-ivl* $s \to S$
and $t \in$ *ex-ivl* s **and** $s \in S$
shows $X\ t =$ *flow* $t_0\ s\ t$
proof−
have *obs*: $\bigwedge z\ T'.\ t_0 \in T' \wedge$ *is-interval* $T' \wedge T' \subseteq$ *ex-ivl* $s \wedge (z$ *solves-ode* $f)\ T'$
$S \Longrightarrow$
$z\ t_0 =$ *flow* $t_0\ s\ t_0 \Longrightarrow (\forall t{\in}T'.\ z\ t =$ *flow* $t_0\ s\ t)$
using *flow-usolves-ode*[*OF init-time* ‹$s \in S$›] **unfolding** *usolves-ode-from-def*

by *blast*
  have $\forall\tau\in ex\text{-}ivl\ s.\ X\ \tau = flow\ t_0\ s\ \tau$
    using *obs*[*of ex-ivl s X*] *existence-ivl-initial-time*[*OF init-time* ‹$s \in S$›]
      xivp flow-initial-time[*OF init-time* ‹$s \in S$›] **unfolding** *solves-ode-def* **by** *simp*
  thus $X\ t = flow\ t_0\ s\ t$
    by (*auto simp*: ‹$t \in ex\text{-}ivl\ s$›)
qed

end

1.3.4 Flows for ODEs

A locale designed for verification of hybrid systems. The user can select both, the interval of existence of her choice, and the computation rule of the flow via the variables T and φ.

locale *local-flow* = *picard-lindeloef* ($\lambda\ t.\ f$) $T\ S\ 0$
  for f::($'a$::{*heine-borel*,*banach*}) $\Rightarrow$ $'a$ **and** $T\ S\ L$ +
  fixes φ :: *real* $\Rightarrow$ $'a \Rightarrow$ $'a$
  assumes *ivp*:$\bigwedge\ t\ s.\ t \in T \Longrightarrow s \in S \Longrightarrow (D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0\text{--}t\})$
    $\bigwedge\ s.\ s \in S \Longrightarrow \varphi\ 0\ s = s$
    $\bigwedge\ t\ s.\ t \in T \Longrightarrow s \in S \Longrightarrow (\lambda t.\ \varphi\ t\ s) \in \{0\text{--}t\} \to S$
begin

lemma *in-ivp-sols-ivl*:
  assumes $t \in T\ s \in S$
  shows $(\lambda t.\ \varphi\ t\ s) \in ivp\text{-}sols\ (\lambda t.\ f)\ \{0\text{--}t\}\ S\ 0\ s$
  apply(*rule ivp-solsI*)
  using *ivp assms* **by** *auto*

lemma *eq-solution-ivl*:
  assumes *xivp*: $D\ X = (\lambda t.\ f\ (X\ t))\ on\ \{0\text{--}t\}\ X\ 0 = s\ X \in \{0\text{--}t\} \to S$
    and *indom*: $t \in T\ s \in S$
  shows $X\ t = \varphi\ t\ s$
  apply(*rule unique-solution*[*OF xivp* ‹$t \in T$›])
  using ‹$s \in S$› *ivp indom* **by** *auto*

lemma *ex-ivl-eq*:
  assumes $s \in S$
  shows $ex\text{-}ivl\ s = T$
  using *existence-ivl-subset*[*of s*] **apply** *safe*
  unfolding *existence-ivl-def csols-eq*
  using *in-ivp-sols-ivl*[*OF - assms*] **by** *blast*

lemma *has-derivative-on-open1*:
  assumes $t > 0\ t \in T\ s \in S$
  obtains B **where** $t \in B$ **and** *open* B **and** $B \subseteq T$
    and $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ at\ t\ within\ B$
proof−

obtain *r*::*real* **where** *rHyp*: $r > 0$ *ball t r* $\subseteq T$
using *open-contains-ball-eq open-domain*(*1*) ‹$t \in T$› **by** *blast*
moreover have $t + r/2 > 0$
using ‹$r > 0$› ‹$t > 0$› **by** *auto*
moreover have $\{0{-}{-}t\} \subseteq T$
using *subintervalI*[*OF init-time* ‹$t \in T$›] **.**
ultimately have *subs*: $\{0{<}{-}{-}{<}t + r/2\} \subseteq T$
unfolding *abs-le-eq abs-le-eq real-ivl-eqs*[*OF* ‹$t > 0$›] *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›]
by *clarify* (*case-tac* $t < x$, *simp-all add*: *cball-def ball-def dist-norm subset-eq field-simps*)
have $t + r/2 \in T$
using *rHyp* **unfolding** *real-ivl-eqs*[*OF rHyp*(*1*)] **by** (*simp add*: *subset-eq*)
hence $\{0{-}{-}t + r/2\} \subseteq T$
using *subintervalI*[*OF init-time*] **by** *blast*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{-}{-}(t + r/2)\})$
using *ivp*(*1*)[*OF* - ‹$s \in S$›] **by** *auto*
hence *vderiv*: $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{<}{-}{-}{<}t + r/2\})$
apply(*rule has-vderiv-on-subset*)
unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **by** *auto*
have $t \in \{0{<}{-}{-}{<}t + r/2\}$
unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **using** *rHyp* ‹$t > 0$› **by** *simp*
moreover have $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ (at\ t\ within\ \{0{<}{-}{-}{<}t + r/2\})$
using *vderiv calculation* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **by** *blast*
moreover have *open* $\{0{<}{-}{-}{<}t + r/2\}$
unfolding *real-ivl-eqs*[*OF* ‹$t + r/2 > 0$›] **by** *simp*
ultimately show *?thesis*
using *subs that* **by** *blast*
qed

lemma *has-derivative-on-open2*:
assumes $t < 0$ $t \in T$ $s \in S$
obtains B **where** $t \in B$ **and** *open* B **and** $B \subseteq T$
and $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ at\ t\ within\ B$
proof−
obtain *r*::*real* **where** *rHyp*: $r > 0$ *ball t r* $\subseteq T$
using *open-contains-ball-eq open-domain*(*1*) ‹$t \in T$› **by** *blast*
moreover have $t - r/2 < 0$
using ‹$r > 0$› ‹$t < 0$› **by** *auto*
moreover have $\{0{-}{-}t\} \subseteq T$
using *subintervalI*[*OF init-time* ‹$t \in T$›] **.**
ultimately have *subs*: $\{0{<}{-}{-}{<}t - r/2\} \subseteq T$
unfolding *open-segment-eq-real-ivl closed-segment-eq-real-ivl*
real-ivl-eqs[*OF rHyp*(*1*)] **by**(*auto simp*: *subset-eq*)
have $t - r/2 \in T$
using *rHyp* **unfolding** *real-ivl-eqs* **by** (*simp add*: *subset-eq*)
hence $\{0{-}{-}t - r/2\} \subseteq T$

using *subintervalI*[*OF init-time*] **by** *blast*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{-}{-}(t - r/2)\})$
using *ivp*(*1*)[*OF* - ‹$s \in S$›] **by** *auto*
hence *vderiv*: $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{<}{-}{-}{<}t - r/2\})$
apply(*rule has-vderiv-on-subset*)
unfolding *open-segment-eq-real-ivl closed-segment-eq-real-ivl* **by** *auto*
have $t \in \{0{<}{-}{-}{<}t - r/2\}$
unfolding *open-segment-eq-real-ivl* **using** *rHyp* ‹$t < 0$› **by** *simp*
moreover have $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))\ (at\ t\ within\ \{0{<}{-}{-}{<}t - r/2\})$
using *vderiv calculation* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **by** *blast*
moreover have *open* $\{0{<}{-}{-}{<}t - r/2\}$
unfolding *open-segment-eq-real-ivl* **by** *simp*
ultimately show *?thesis*
using *subs that* **by** *blast*
qed

lemma *has-derivative-on-open3*:
assumes $s \in S$
obtains B **where** $0 \in B$ **and** *open* B **and** $B \subseteq T$
and $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ 0\ s))\ at\ 0\ within\ B$
proof−
obtain *r*::*real* **where** *rHyp*: $r > 0$ *ball* $0\ r \subseteq T$
using *open-contains-ball-eq open-domain*(*1*) *init-time* **by** *blast*
hence $r/2 \in T\ {-r/2} \in T\ r/2 > 0$
unfolding *real-ivl-eqs* **by** *auto*
hence *subs*: $\{0{-}{-}r/2\} \subseteq T\ \{0{-}{-}(-r/2)\} \subseteq T$
using *subintervalI*[*OF init-time*] **by** *auto*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{-}{-}r/2\})$
$(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{-}{-}(-r/2)\})$
using *ivp*(*1*)[*OF* - ‹$s \in S$›] **by** *auto*
also have $\{0{-}{-}r/2\} = \{0{-}{-}r/2\} \cup closure\ \{0{-}{-}r/2\} \cap closure\ \{0{-}{-}(-r/2)\}$
$\{0{-}{-}(-r/2)\} = \{0{-}{-}(-r/2)\} \cup closure\ \{0{-}{-}r/2\} \cap closure\ \{0{-}{-}(-r/2)\}$
unfolding *closed-segment-eq-real-ivl* ‹$r/2 > 0$› **by** *auto*
ultimately have *vderivs*:
$(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{-}{-}r/2\} \cup closure\ \{0{-}{-}r/2\} \cap closure\ \{0{-}{-}(-r/2)\})$
$(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{0{-}{-}(-r/2)\} \cup closure\ \{0{-}{-}r/2\} \cap closure\ \{0{-}{-}(-r/2)\})$
unfolding *closed-segment-eq-real-ivl* ‹$r/2 > 0$› **by** *auto*
have *obs*: $0 \in \{-r/2{<}{-}{-}{<}r/2\}$
unfolding *open-segment-eq-real-ivl* **using** ‹$r/2 > 0$› **by** *auto*
have *union*: $\{-r/2{-}{-}r/2\} = \{0{-}{-}r/2\} \cup \{0{-}{-}(-r/2)\}$
unfolding *closed-segment-eq-real-ivl* **by** *auto*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{-r/2{-}{-}r/2\})$
using *has-vderiv-on-union*[*OF vderivs*] **by** *simp*
hence $(D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))\ on\ \{-r/2{<}{-}{-}{<}r/2\})$
using *has-vderiv-on-subset*[*OF* - *segment-open-subset-closed*[*of* $-r/2\ r/2$]] **by**

auto
  hence $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ 0\ s))$ (*at 0 within* $\{-r/2<--<r/2\}$)
    unfolding *has-vderiv-on-def has-vector-derivative-def* **using** *obs* **by** *blast*
  moreover have *open* $\{-r/2<--<r/2\}$
    unfolding *open-segment-eq-real-ivl* **by** *simp*
  moreover have $\{-r/2<--<r/2\} \subseteq T$
    using *subs union segment-open-subset-closed* **by** *blast*
  ultimately show *?thesis*
    using *obs that* **by** *blast*
qed

lemma *has-derivative-on-open*:
  assumes $t \in T$ $s \in S$
  obtains B **where** $t \in B$ **and** *open* B **and** $B \subseteq T$
    and $D\ (\lambda\tau.\ \varphi\ \tau\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))$ *at t within* B
  apply(*subgoal-tac* $t < 0 \vee t = 0 \vee t > 0$)
  using *has-derivative-on-open1*[*OF - assms*] *has-derivative-on-open2*[*OF - assms*]
    has-derivative-on-open3[*OF* ‹$s \in S$›] **by** *blast force*

lemma *in-domain*:
  assumes $s \in S$
  shows $(\lambda t.\ \varphi\ t\ s) \in T \to S$
  unfolding *ex-ivl-eq*[*symmetric*] *existence-ivl-def*
  using *local.mem-existence-ivl-subset ivp*(*3*)[*OF - assms*] **by** *blast*

lemma *has-vderiv-on-domain*:
  assumes $s \in S$
  shows $D\ (\lambda t.\ \varphi\ t\ s) = (\lambda t.\ f\ (\varphi\ t\ s))$ *on* T
proof(*unfold has-vderiv-on-def has-vector-derivative-def*, *clarsimp*)
  fix t **assume** $t \in T$
  then obtain B **where** $t \in B$ **and** *open* B **and** $B \subseteq T$
    and *Dhyp*: $D\ (\lambda t.\ \varphi\ t\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))$ *at t within* B
    using *assms has-derivative-on-open*[*OF* ‹$t \in T$›] **by** *blast*
  hence $t \in$ *interior* B
    using *interior-eq* **by** *auto*
  thus $D\ (\lambda t.\ \varphi\ t\ s) \mapsto (\lambda\tau.\ \tau *_R f\ (\varphi\ t\ s))$ *at t within* T
    using *has-derivative-at-within-mono*[*OF -* ‹$B \subseteq T$› *Dhyp*] **by** *blast*
qed

lemma *in-ivp-sols*:
  assumes $s \in S$
  shows $(\lambda t.\ \varphi\ t\ s) \in$ *ivp-sols* $(\lambda t.\ f)\ T\ S\ 0\ s$
  using *has-vderiv-on-domain ivp*(*2*) *in-domain* **apply**(*rule ivp-solsI*)
  using *assms* **by** *auto*

lemma *eq-solution*:
  assumes $X \in$ (*ivp-sols* $(\lambda t.\ f)\ T\ S\ 0\ s$) **and** $t \in T$ **and** $s \in S$
  shows $X\ t = \varphi\ t\ s$
proof–

have $D\ X = (\lambda t.\ f\ (X\ t))$ *on* $(ex\text{-}ivl\ s)$ **and** $X\ 0 = s$ **and** $X \in (ex\text{-}ivl\ s) \rightarrow S$
  using *ivp-solsD*[*OF assms*(*1*)] **unfolding** *ex-ivl-eq*[*OF* ‹$s \in S$›] **by** *auto*
note *solution-eq-flow*[*OF this*]
hence $X\ t = flow\ 0\ s\ t$
  unfolding *ex-ivl-eq*[*OF* ‹$s \in S$›] **using** *assms* **by** *blast*
also have $\varphi\ t\ s = flow\ 0\ s\ t$
  apply(*rule solution-eq-flow ivp*)
    apply(*simp-all add*: *assms*(*2*,*3*) *ivp*(*2*)[*OF* ‹$s \in S$›])
  unfolding *ex-ivl-eq*[*OF* ‹$s \in S$›] **by** (*auto simp*: *has-vderiv-on-domain assms in-domain*)
ultimately show $X\ t = \varphi\ t\ s$
  by *simp*
qed

lemma *ivp-sols-collapse*:
  assumes $T = UNIV$ **and** $s \in S$
  shows *ivp-sols* $(\lambda t.\ f)\ T\ S\ 0\ s = \{(\lambda t.\ \varphi\ t\ s)\}$
  using *in-ivp-sols eq-solution assms* **by** *auto*

lemma *additive-in-ivp-sols*:
  assumes $s \in S$ **and** $\mathcal{P}\ (\lambda\tau.\ \tau + t)\ T \subseteq T$
  shows $(\lambda\tau.\ \varphi\ (\tau + t)\ s) \in$ *ivp-sols* $(\lambda t.\ f)\ T\ S\ 0\ (\varphi\ (0 + t)\ s)$
  apply(*rule ivp-solsI*, *rule vderiv-on-compose-add*)
  using *has-vderiv-on-domain has-vderiv-on-subset assms* **apply** *blast*
  using *in-domain assms* **by** *auto*

lemma *is-monoid-action*:
  assumes $s \in S$ **and** $T = UNIV$
  shows $\varphi\ 0\ s = s$ **and** $\varphi\ (t_1 + t_2)\ s = \varphi\ t_1\ (\varphi\ t_2\ s)$
proof−
  show $\varphi\ 0\ s = s$
    using *ivp assms* **by** *simp*
  have $\varphi\ (0 + t_2)\ s = \varphi\ t_2\ s$
    by *simp*
  also have $\varphi\ t_2\ s \in S$
    using *in-domain assms* **by** *auto*
  finally show $\varphi\ (t_1 + t_2)\ s = \varphi\ t_1\ (\varphi\ t_2\ s)$
    using *eq-solution*[*OF additive-in-ivp-sols*] *assms* **by** *auto*
qed

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

notation *orbit* (γ^φ)

lemma *orbit-eq*[*simp*]:
  assumes $s \in S$
  shows $\gamma^\varphi\ s = \{\varphi\ t\ s|\ t.\ t \in T\}$
  using *eq-solution assms* **unfolding** *orbit-def g-orbital-eq ivp-sols-def*
  by(*auto intro*!: *has-vderiv-on-domain ivp*(*2*) *in-domain*)

lemma *g-orbital-collapses*:
  assumes $s \in S$
  shows *g-orbital f G T S 0 s* = $\{\varphi\ t\ s|\ t.\ t \in T \wedge (\forall\tau\in down\ T\ t.\ G\ (\varphi\ \tau\ s))\}$
proof(*rule subset-antisym*, *simp-all only*: *subset-eq*)
  let *?gorbit* = $\{\varphi\ t\ s\ |t.\ t \in T \wedge (\forall\tau\in down\ T\ t.\ G\ (\varphi\ \tau\ s))\}$
  {fix s' **assume** $s' \in$ *g-orbital f G T S 0 s*
    then obtain X **and** t **where** *x-ivp*:$X \in$ *ivp-sols* $(\lambda t.\ f)$ *T S 0 s*
      and $X\ t = s'$ **and** $t \in T$ **and** *guard*:$(\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\})$
      unfolding *g-orbital-def g-orbit-eq* **by** *auto*
    have *obs*:$\forall\tau\in(down\ T\ t).\ X\ \tau = \varphi\ \tau\ s$
      using *eq-solution*[*OF x-ivp - assms*] **by** *blast*
    hence $\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ G\ s\}$
      using *guard* **by** *auto*
    also have $\varphi\ t\ s = X\ t$
      using *eq-solution*[*OF x-ivp* ‹$t \in T$› *assms*] **by** *simp*
    ultimately have $s' \in$ *?gorbit*
      using ‹$X\ t = s'$› ‹$t \in T$› **by** *auto***}**
  thus $\forall s'\in$ *g-orbital f G T S 0 s*. $s' \in$ *?gorbit*
    by *blast*
next
  let *?gorbit* = $\{\varphi\ t\ s\ |t.\ t \in T \wedge (\forall\tau\in down\ T\ t.\ G\ (\varphi\ \tau\ s))\}$
  {fix s' **assume** $s' \in$ *?gorbit*
    then obtain t **where** $\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ G\ s\}$ **and** $t \in T$ **and** φ
$t\ s = s'$
      by *blast*
    hence $s' \in$ *g-orbital f G T S 0 s*
      using *assms* **by**(*auto intro*!: *g-orbitalI in-ivp-sols*)**}**
  thus $\forall s'\in$*?gorbit*. $s' \in$ *g-orbital f G T S 0 s*
    by *blast*
qed

end

lemma *line-is-local-flow*:
  $0 \in T \Longrightarrow$ *is-interval* $T \Longrightarrow$ *open* $T \Longrightarrow$ *local-flow* $(\lambda\ s.\ c)$ *T UNIV* $(\lambda\ t\ s.\ s$
$+\ t *_R c)$
  apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
   apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=1/2* **in** *exI*, *simp*)
  apply(*rule-tac* $f'1=\lambda\ s.\ 0$ **and** $g'1=\lambda\ s.\ c$ **in** *derivative-intros*(*191*))
  apply(*rule derivative-intros*, *simp*)+
  by *simp-all*

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

begin

Chapter 2

Linear Algebra for Hybrid Systems

Linear systems of ordinary differential equations (ODEs) are those whose vector fields are linear operators. 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. We start by formalising various properties of vector spaces.

2.1 Vector operations

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

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

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

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

lemma *kronecker-delta-simps*[*simp*]:
  fixes $q{::}('a{::}\mathit{semiring\text{-}0})$ **and** $i{::}'n{::}\mathit{finite}$
  shows $(\sum j \in \mathit{UNIV}. \ f \ j * (\delta_K \ j \ i \ q)) = f \ i * q$
    and $(\sum j \in \mathit{UNIV}. \ f \ j * (\delta_K \ i \ j \ q)) = f \ i * q$
    and $(\sum j \in \mathit{UNIV}. \ (\delta_K \ i \ j \ q) * f \ j) = q * f \ i$
    and $(\sum j \in \mathit{UNIV}. \ (\delta_K \ j \ i \ q) * f \ j) = q * f \ i$
  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 *uminus-axis-eq*[*simp*]: $-\ axis\ i\ k = axis\ i\ (-k)$ **for** *k*::*'a::ring*
unfolding *axis-def* **by**(*simp add*: *vec-eq-iff*)

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-normed-algebra-1*) ^'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-normed-algebra-1*) ^'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-normed-algebra-1*) ^'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::*real-normed-algebra-1*) ^'n^'m ⇒ *real* ((*1*‖-‖$_{op}$) [*65*] *61*)
where $\|A\|_{op} \equiv$ *onorm* $(\lambda x.\ A *v\ x)$

lemma *norm-matrix-bound*:
fixes A::('a::*real-normed-algebra-1*) ^'n^'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 $\|(\sum i{\in}UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \le (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$ **by** *simp*}
hence $\bigwedge j.\ \|(A *v\ x)\ \$\ j\| \le ((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j$
unfolding *matrix-vector-mult-def* **by** *simp*
hence $(\sum j{\in}UNIV.\ (\|(A *v\ x)\ \$\ j\|)^2) \le (\sum j{\in}UNIV.\ (\|((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v\ 1)\ \$\ j\|)^2)$
by (*metis* (*mono-tags*, *lifting*) *norm-ge-zero power2-abs power-mono real-norm-def sum-mono*)
thus $\|A *v\ x\| \le \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$
unfolding *norm-vec-def L2-set-def* **by** *simp*
qed

lemma *onorm-set-proptys*:
fixes *A*::(*'a*::*real-normed-algebra-1*) ^*'n*^*'m*
shows *bounded* (*range* ($\lambda x.\ (\|A *v\ x\|)\ /\ (\|x\|)$))
and *bdd-above* (*range* ($\lambda x.\ (\|A *v\ x\|)\ /\ (\|x\|)$))
and (*range* ($\lambda x.\ (\|A *v\ x\|)\ /\ (\|x\|)$)) $\neq \{\}$
unfolding *bounded-def bdd-above-def image-def dist-real-def* **apply**(*rule-tac x=0* **in** *exI*)
apply(*rule-tac* $x=\|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v\ 1\|$ **in** *exI*, *clarsimp*,
subst mult-norm-matrix-sgn-eq[*symmetric*], *clarsimp*,
rule-tac x=sgn - **in** *norm-matrix-bound*, *simp add*: *norm-sgn*)+
by *force*

lemma *op-norm-set-proptys*:
fixes *A*::(*'a*::*real-normed-algebra-1*) ^*'n*^*'m*
shows *bounded* $\{\|A *v\ x\|\ |\ x.\ \|x\| = 1\}$
and *bdd-above* $\{\|A *v\ x\|\ |\ x.\ \|x\| = 1\}$
and $\{\|A *v\ x\|\ |\ x.\ \|x\| = 1\} \neq \{\}$
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 *op-norm-def*:
fixes *A*::(*'a*::*real-normed-algebra-1*) ^*'n*^*'m*
shows $\|A\|_{op} = Sup\ \{\|A *v\ x\|\ |\ x.\ \|x\| = 1\}$
apply(*rule antisym*[*OF onorm-le cSup-least*[*OF op-norm-set-proptys*(*3*)]])
apply(*case-tac* $x = 0$, *simp*)
apply(*subst mult-norm-matrix-sgn-eq*[*symmetric*], *simp*)
apply(*rule cSup-upper*[*OF - op-norm-set-proptys*(*2*)])
apply(*force simp*: *norm-sgn*)
unfolding *onorm-def* **apply**(*rule cSup-upper*[*OF - onorm-set-proptys*(*2*)])
by (*simp add*: *image-def*, *clarsimp*) (*metis div-by-1*)

lemma *norm-matrix-le-op-norm*: $\|x\| = 1 \Longrightarrow \|A *v\ x\| \le \|A\|_{op}$
apply(*unfold onorm-def*, *rule cSup-upper*[*OF - onorm-set-proptys*(*2*)])

unfolding *image-def* **by** (*clarsimp*, *rule-tac x=x* **in** *exI*) *simp*

lemma *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 op-norm-ge-0*)

lemma *norm-matrix-le-mult-op-norm*: $\|A *v\ x\| \leq (\|A\|_{op}) * (\|x\|)$
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 *blin-norm-matrix*: *bounded-linear* ((∗*v*) *A*) **for** *A*::(′*a*::*real-normed-algebra-1*) ^′*n*^′*m*
by (*unfold-locales*) (*auto intro*: *norm-matrix-le-mult-op-norm simp*:
mult.commute matrix-vector-right-distrib vector-scaleR-commute)

lemma *op-norm-zero-iff*: $(\|A\|_{op} = 0) = (A = 0)$ **for** *A*::(′*a*::*real-normed-field*) ^′*n*^′*m*
unfolding *onorm-eq-0*[*OF blin-norm-matrix*] **using** *matrix-axis-0*[*of 1 A*] **by**
fastforce

lemma *op-norm-triangle*: $\|A + B\|_{op} \leq (\|A\|_{op}) + (\|B\|_{op})$
using *onorm-triangle*[*OF blin-norm-matrix*[*of A*] *blin-norm-matrix*[*of B*]]
matrix-vector-mult-add-rdistrib[*symmetric*, *of A - B*] **by** *simp*

lemma *op-norm-scaleR*: $\|c *_R A\|_{op} = |c| * (\|A\|_{op})$
unfolding *onorm-scaleR*[*OF blin-norm-matrix*, *symmetric*] *scaleR-vector-assoc*
..

lemma *op-norm-matrix-matrix-mult-le*:
fixes *A*::(′*a*::*real-normed-algebra-1*) ^′*n*^′*m*
shows $\|A ** B\|_{op} \leq (\|A\|_{op}) * (\|B\|_{op})$
proof(*rule onorm-le*)
have $0 \leq (\|A\|_{op})$
by(*rule onorm-pos-le*[*OF blin-norm-matrix*])
fix *x* **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
finally show $\|A ** B *v\ x\| \leq (\|A\|_{op}) * (\|B\|_{op}) * (\|x\|)$
by *simp*

qed

lemma *norm-matrix-vec-mult-le-transpose*:
$\|x\| = 1 \Longrightarrow (\|A *v\ x\|) \le sqrt\ (\|transpose\ A ** A\|_{op}) * (\|x\|)$ **for** $A{::}real\hat{}'n\hat{}'n$
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 ... $\le (\|x\|) * (\|transpose\ A *v\ (A *v\ x)\|)$
using *norm-cauchy-schwarz* **by** *blast*
also have ... $\le (\|transpose\ A ** A\|_{op}) * (\|x\|)\hat{}2$
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 $((\|A *v\ x\|))\hat{}2 \le (\|transpose\ A ** A\|_{op}) * (\|x\|)\hat{}2$
by *linarith*
thus $(\|A *v\ x\|) \le sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
by (*simp add*: ‹$\|x\| = 1$› *real-le-rsqrt*)
qed

lemma *op-norm-le-sum-column*: $\|A\|_{op} \le (\sum i{\in}UNIV.\ \|column\ i\ A\|)$ **for** $A{::}real\hat{}'n\hat{}'m$

proof(*unfold op-norm-def*, *rule cSup-least*[*OF op-norm-set-proptys*(*3*)], *clarsimp*)
fix $x{::}real\hat{}'n$ **assume** *x-def*:$\|x\| = 1$
hence *x-hyp*:$\bigwedge i.\ \|x\ \$\ i\| \le 1$
by (*simp add*: *norm-bound-component-le-cart*)
have $(\|A *v\ x\|) = \|(\sum i{\in}UNIV.\ x\ \$\ i *s\ column\ i\ A)\|$
by(*subst matrix-mult-sum*[*of A*], *simp*)
also have ... $\le (\sum i{\in}UNIV.\ \|x\ \$\ i *s\ column\ i\ A\|)$
by (*simp add*: *sum-norm-le*)
also have ... $= (\sum i{\in}UNIV.\ (\|x\ \$\ i\|) * (\|column\ i\ A\|))$
by (*simp add*: *mult-norm-matrix-sgn-eq*)
also have ... $\le (\sum i{\in}UNIV.\ \|column\ i\ A\|)$
using *x-hyp* **by** (*simp add*: *mult-left-le-one-le sum-mono*)
finally show $\|A *v\ x\| \le (\sum i{\in}UNIV.\ \|column\ i\ A\|)$ **.**
qed

lemma *op-norm-le-transpose*: $\|A\|_{op} \le \|transpose\ A\|_{op}$ **for** $A{::}real\hat{}'n\hat{}'n$
proof−
have *obs*:$\forall x.\ \|x\| = 1 \longrightarrow (\|A *v\ x\|) \le sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
using *norm-matrix-vec-mult-le-transpose* **by** *blast*
have $(\|A\|_{op}) \le sqrt\ ((\|transpose\ A ** A\|_{op}))$
using *obs* **apply**(*unfold op-norm-def*)
by (*rule cSup-least*[*OF op-norm-set-proptys*(*3*)]) *clarsimp*
hence $((\|A\|_{op}))^2 \le (\|transpose\ A ** A\|_{op})$
using *power-mono*[*of* $(\|A\|_{op})$ - *2*] *op-norm-ge-0* **by** *force*
also have ... $\le (\|transpose\ A\|_{op}) * (\|A\|_{op})$

using *op-norm-matrix-matrix-mult-le* **by** *blast*
finally have $((\|A\|_{op}))^2 \leq (\|transpose\ A\|_{op}) * (\|A\|_{op})$ **by** *linarith*
thus $(\|A\|_{op}) \leq (\|transpose\ A\|_{op})$
using *sq-le-cancel*[*of* $(\|A\|_{op})$] *op-norm-ge-0* **by** *blast*
qed

2.2.2 Matrix maximum norm

abbreviation *max-norm* $(A::real\hat{\ }'n\hat{\ }'m) \equiv Max\ (abs\ `\ (entries\ A))$

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

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

lemma *max-norm-set-proptys*: *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 $\|A\|_{op} \leq real\ CARD('m) * real\ CARD('n) * (\|A\|_{max})$
apply(*rule onorm-le-matrix-component*)
unfolding *max-norm-def* **by**(*rule Max-ge*[*OF max-norm-set-proptys*]) *force*

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 $A::real\hat{\ }'n\hat{\ }'n$
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 (*metis* (*no-types*, *lifting*) *mult.commute mult-right-mono norm-ge-zero power2-eq-square*)
also have $(\|A *v\ sgn\ (x - y)\|) * (\|x - y\|) \leq (\|A\|_{op}) * (\|x - y\|)$
by (*simp add*: *norm-sgn-le-op-norm mult-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 $A::real\hat{\ }'n\hat{\ }'n$
defines $L \equiv (real\ CARD('n))^2 * (\|A\|_{max})$
shows *picard-lindeloef* (λ *t s. A* $*v$ *s*) *UNIV UNIV 0*
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=L* **in** *exI*, *safe*)
using *max-norm-ge-0*[*of A*] **unfolding** *assms* **by** *force* (*rule matrix-lipschitz-constant*)

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 the type class "banach". Hence, we define a type of squared matrices and prove that it is an instance of this class.

2.4.1 Squared matrices operations

typedef $'m$ *sq-mtx* = $UNIV::(real\hat{\ }'m\hat{\ }'m)$ *set*
morphisms *to-vec sq-mtx-chi* **by** *simp*

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

setup-lifting *type-definition-sq-mtx*

lift-definition *sq-mtx-ith*::$'m$ *sq-mtx* $\Rightarrow$ $'m \Rightarrow (real\hat{\ }'m)$ (**infixl** $$ *90*) **is** *vec-nth*
.

lift-definition *sq-mtx-vec-prod*::$'m$ *sq-mtx* $\Rightarrow (real\hat{\ }'m) \Rightarrow (real\hat{\ }'m)$ (**infixl** $*_V$ *90*)
is *matrix-vector-mult* .

lift-definition *sq-mtx-column*::$'m \Rightarrow 'm$ *sq-mtx* $\Rightarrow$ (*real*^$'m$)
 is $\lambda i\ X.$ *column* i (*to-vec* X) **.**

lift-definition *vec-sq-mtx-prod*::(*real*^$'m$) $\Rightarrow 'm$ *sq-mtx* $\Rightarrow$ (*real*^$'m$) **is** *vector-matrix-mult*
.

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

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

lift-definition *sq-mtx-row*::$'m \Rightarrow$ ($'m$::*finite*) *sq-mtx* $\Rightarrow$ *real*^$'m$ (row) **is** *row* **.**

lift-definition *sq-mtx-col*::$'m \Rightarrow$ ($'m$::*finite*) *sq-mtx* $\Rightarrow$ *real*^$'m$ (col) **is** *column* **.**

lift-definition *sq-mtx-rows*::($'m$::*finite*) *sq-mtx* $\Rightarrow$ (*real*^$'m$) *set* **is** *rows* **.**

lift-definition *sq-mtx-cols*::($'m$::*finite*) *sq-mtx* $\Rightarrow$ (*real*^$'m$) *set* **is** *columns* **.**

lemma *to-vec-eq-ith*[*simp*]: (*to-vec* A) \$ i = A \$\$ i
 by *transfer simp*

lemma *sq-mtx-chi-ith*[*simp*]: (*sq-mtx-chi* A) \$\$ *i1* \$ *i2* = A \$ *i1* \$ *i2*
 by *transfer simp*

lemma *sq-mtx-chi-vec-lambda-ith*[*simp*]: *sq-mtx-chi* (χ i j. x i j) \$\$ *i1* \$ *i2* = x *i1*
i2
 by(*simp add*: *sq-mtx-ith-def*)

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*: $m *_V x$ = (χ i. *sum* (λj. ((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*]:$(A^{\dagger}) *_V$ (e i) = row i A
 by *transfer* (*simp add*: *row-def transpose-def axis-def matrix-vector-mult-def*)

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

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

2.4.2 Squared matrices form Banach space

instantiation *sq-mtx* :: (*finite*) *ring*
begin

lift-definition *plus-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** (+) **.**

lift-definition *zero-sq-mtx* :: *'a sq-mtx* **is** *0* **.**

lift-definition *uminus-sq-mtx* ::*'a sq-mtx* ⇒ *'a sq-mtx* **is** *uminus* **.**

lift-definition *minus-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** (−) **.**

lift-definition *times-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** (**) **.**

declare *plus-sq-mtx.rep-eq* [*simp*]
  and *minus-sq-mtx.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-sq-mtx-def*, *transfer*, *simp*)

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

lemma *mtx-vec-prod-add-rdistr*:$(A + B) *_V x = A *_V x + B *_V x$
 unfolding *plus-sq-mtx-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-sq-mtx-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 *sq-mtx* :: (*finite*) *real-normed-vector*
begin

definition *norm-sq-mtx* :: *'a sq-mtx* ⇒ *real* **where** $\|A\| = \|to\text{-}vec\ A\|_{op}$

lift-definition *scaleR-sq-mtx*::*real* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** *scaleR* **.**

definition *sgn-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx*

where *sgn-sq-mtx* $A = (inverse\ (\|A\|)) *_R A$

definition *dist-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *real*
where *dist-sq-mtx* $A\ B = \|A - B\|$

definition *uniformity-sq-mtx* :: (*'a sq-mtx* × *'a sq-mtx*) *filter*
where *uniformity-sq-mtx* = (*INF e*:{*0*<..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-sq-mtx* :: *'a sq-mtx set* ⇒ *bool*
where *open-sq-mtx U* = (∀ *x*∈*U*. $\forall_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *U*)

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

end

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

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

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

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

lemma *norm-le-transpose*: $\|A\| \leq \|A^\dagger\|$
unfolding *norm-sq-mtx-def* **by** *transfer* (*rule op-norm-le-transpose*)

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 *sq-mtx* :: (*finite*) *real-normed-algebra-1*
begin

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

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

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

lemma *sq-mtx-norm-times*: $\|A * B\| \le (\|A\|) * (\|B\|)$ **for** $A::'a\ sq\text{-}mtx$
 unfolding *norm-sq-mtx-def times-sq-mtx-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-sq-mtx-def zero-sq-mtx-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-sq-mtx-def*
    mat-def vec-eq-iff matrix-vector-mult-def)

lemma *Cauchy-cols*:
  fixes $X :: nat \Rightarrow ('a::finite)\ sq\text{-}mtx$
  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*:$\forall m \ge M.\ \forall n \ge M.\ \|X\ m - X\ n\| < \varepsilon$
    using ‹*Cauchy X*› **unfolding** *Cauchy-def* **by**(*simp add*: *dist-sq-mtx-def*) *blast*
  {**fix** $m\ n$ **assume** $m \ge M$ **and** $n \ge M$
    hence $\varepsilon > \|X\ m - X\ n\|$
      using *M-def* **by** *blast*
    moreover have $\|X\ m - X\ n\| \ge \|(X\ m - X\ n) *_V \text{e}\ i\|$
      by(*rule le-mtx-norm*[*of* - $X\ m - X\ n$], *force*)
    moreover have $\|(X\ m - X\ n) *_V \text{e}\ i\| = \|X\ m *_V \text{e}\ i - X\ n *_V \text{e}\ i\|$
      by (*simp add*: *mtx-vec-prod-minus-rdistrib*)
    moreover have ... $= \|\text{col}\ i\ (X\ m) - \text{col}\ i\ (X\ n)\|$
      by (*simp add*: *mtx-vec-prod-minus-rdistrib mtx-vec-prod-canon*)
    ultimately have $\|\text{col}\ i\ (X\ m) - \text{col}\ i\ (X\ n)\| < \varepsilon$
      by *linarith*}
  thus $\exists M.\ \forall m \ge M.\ \forall n \ge M.\ \|\text{col}\ i\ (X\ m) - \text{col}\ i\ (X\ n)\| < \varepsilon$
    by *blast*
qed

lemma *col-convergent*:
  assumes $\forall i.\ (\lambda n.\ \text{col}\ i\ (X\ n)) \longrightarrow L\ \$\ i$
  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** $\forall i. \ \exists \ N. \ \forall n \geq N. \ \|\text{col } i \ (X \ n) - L \ \$ \ i\| < \varepsilon/?a$
unfolding *LIMSEQ-def dist-norm convergent-def* **by** *blast*
then obtain N **where** $\forall i. \ \forall n \geq N. \ \|\text{col } i \ (X \ n) - L \ \$ \ i\| < \varepsilon/?a$
using *finite-nat-minimal-witness*[*of* $\lambda \ i \ n. \ \|\text{col } i \ (X \ n) - L \ \$ \ i\| < \varepsilon/?a$] **by** *blast*
also have $\bigwedge i \ n. \ (\text{col } i \ (X \ n) - L \ \$ \ i) = (\text{col } i \ (X \ n - ?L))$
unfolding *minus-sq-mtx-def* **by**(*transfer*, *simp add*: *transpose-def vec-eq-iff column-def*)
ultimately have *N-def*:$\forall i. \ \forall n \geq N. \ \|\text{col } i \ (X \ n - ?L)\| < \varepsilon/?a$
by *auto*
have $\forall n \geq N. \ \|X \ n - ?L\| < \varepsilon$
proof(*rule allI*, *rule impI*)
fix n::*nat* **assume** $N \leq n$
hence $\forall \ i. \ \|\text{col } i \ (X \ n - ?L)\| < \varepsilon/?a$
using *N-def* **by** *blast*
hence $(\sum i \in UNIV. \ \|\text{col } i \ (X \ n - ?L)\|) < (\sum (i::'a) \in UNIV. \ \varepsilon/?a)$
using *sum-strict-mono*[*of* - $\lambda i. \ \|\text{col } i \ (X \ n - ?L)\|$] **by** *force*
moreover have $\|X \ n - ?L\| \leq (\sum i \in UNIV. \ \|\text{col } i \ (X \ n - ?L)\|)$
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 *sq-mtx* :: (*finite*) *banach*
proof(*standard*)
fix X::*nat* $\Rightarrow$ *'a sq-mtx*
assume *Cauchy X*
have $\bigwedge i.$ *Cauchy* $(\lambda n. \text{ col } i \ (X \ n))$
using ‹*Cauchy X*› *Cauchy-cols* **by** *blast*
hence *obs*:$\forall i. \ \exists! \ L. \ (\lambda n. \text{ col } i \ (X \ n)) \longrightarrow L$
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** $\forall i. \ (\lambda n. \text{ col } i \ (X \ n)) \longrightarrow L \ \$ \ i$
using *theI-unique*[*of* $\lambda L. \ (\lambda n. \text{ col - } (X \ n)) \longrightarrow L \ L \ \$$ -] **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. We show that they all satisfy the *local-flow* locale.

```
lemma mtx-vec-prod-has-derivative-mtx-vec-prod:
  assumes ⋀ i j. D (λt. (A t) $$ i $ j) ↦ (λτ. τ *R (A' t) $$ i $ j) (at t within s)
    and (λτ. τ *R (A' t) *V x) = g'
  shows D (λt. A t *V x) ↦ g' at t within s
  using assms(2) unfolding sq-mtx-vec-mult-sum-cols apply safe
  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=λτ. 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 ↦ (λh. h *R A' x) at x within s
  shows D (λt. A t $$ i) ↦ (λh. h *R A' x $$ 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 < ε
  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) sq-mtx)
  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) sq-mtx
  defines L ≡ (real CARD('n))^2 * (‖to-vec A‖_max)
  shows picard-lindeloef (λ t s. A *V s) UNIV UNIV 0
  apply(unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def, clarsimp)
  apply(rule-tac x=1 in exI, clarsimp, rule-tac x=L in exI, safe)
  using max-norm-ge-0[of to-vec A] unfolding assms apply force
  by transfer (rule matrix-lipschitz-constant)

lemma local-flow-exp:
  fixes A::('n::finite) sq-mtx
  shows local-flow ((*V) A) UNIV UNIV (λt s. exp (t *R A) *V s)
  unfolding local-flow-def local-flow-axioms-def apply safe
  using picard-lindeloef-sq-mtx 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-dyn-sys 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* $= 'a \Rightarrow$ *bool*

no-notation *bres* (**infixr** $\rightarrow$ *60*)

no-notation *dagger* ($\text{-}^{\dagger}$ [*101*] *100*)

3.1 Verification of regular programs

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

lemma $fb_{\mathcal{F}}\ F\ S = \{s.\ F\ s \subseteq S\}$
  unfolding *ffb-def map-dual-def klift-def kop-def dual-set-def*
  by(*auto simp*: *Compl-eq-Diff-UNIV fun-eq-iff f2r-def converse-def r2f-def*)

lemma *ffb-eq*: $fb_{\mathcal{F}}\ F\ S = \{s.\ \forall y.\ y \in F\ s \longrightarrow y \in S\}$
  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-eta*[*simp*]: $fb_{\mathcal{F}}\ \eta\ S = S$
  unfolding *ffb-def* **by**(*simp add*: *kop-def klift-def map-dual-def*)

lemma *ffb-iso*: $P \leq Q \Longrightarrow fb_{\mathcal{F}}\ F\ P \leq fb_{\mathcal{F}}\ F\ Q$
  unfolding *ffb-eq* **by** *auto*

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

by *auto*
qed

lemma *ffb-invariants*:
assumes $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ F\ \{s.\ I\ s\}$ **and** $\{s.\ J\ s\} \leq fb_{\mathcal{F}}\ F\ \{s.\ J\ s\}$
shows $\{s.\ I\ s \wedge J\ s\} \leq fb_{\mathcal{F}}\ F\ \{s.\ I\ s \wedge J\ s\}$
and $\{s.\ I\ s \vee J\ s\} \leq fb_{\mathcal{F}}\ F\ \{s.\ I\ s \vee J\ s\}$
using *assms* **unfolding** *ffb-eq* **by** *auto*

Next, we introduce assignments and their wlps.

definition *vec-upd* :: $('a\hat{\ }'n) \Rightarrow 'n \Rightarrow 'a \Rightarrow 'a\hat{\ }'n$
where *vec-upd s i a* $\equiv \chi\ j.\ (((\$)\ s)(i := a))\ j$

definition *assign* :: $'n \Rightarrow ('a\hat{\ }'n \Rightarrow 'a) \Rightarrow ('a\hat{\ }'n) \Rightarrow ('a\hat{\ }'n)\ set$ ((*2*- ::= -) [*70*, *65*] *61*)
where $(x ::= e) \equiv (\lambda s.\ \{vec\text{-}upd\ s\ x\ (e\ s)\})$

lemma *ffb-assign*[*simp*]: $fb_{\mathcal{F}}\ (x ::= e)\ Q = \{s.\ (\chi\ j.\ (((\$)\ s)(x := (e\ s)))\ j) \in Q\}$
unfolding *vec-upd-def assign-def* **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*)

lemma *ffb-kcomp-ge*:
assumes $P \leq fb_{\mathcal{F}}\ F\ R\ R \leq fb_{\mathcal{F}}\ G\ Q$
shows $P \leq fb_{\mathcal{F}}\ (F \circ_K G)\ Q$
apply(*subst ffb-kcomp*)
by (*rule order.trans*[*OF assms*(*1*)]) (*rule ffb-iso*[*OF assms*(*2*)])

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*:
$fb_{\mathcal{F}}\ (IF\ T\ THEN\ X\ ELSE\ Y\ FI)\ Q = \{s.\ T\ s \longrightarrow s \in fb_{\mathcal{F}}\ X\ Q\} \cap \{s.\ \neg\ T\ s \longrightarrow s \in fb_{\mathcal{F}}\ Y\ Q\}$
unfolding *ffb-eq ifthenelse-def* **by** *auto*

lemma *ffb-if-then-else-ge*:
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-elseI*:
assumes $T\ s \longrightarrow s \in fb_{\mathcal{F}}\ X\ Q$
and $\neg\ T\ s \longrightarrow s \in fb_{\mathcal{F}}\ Y\ Q$
shows $s \in fb_{\mathcal{F}}\ (IF\ T\ THEN\ X\ ELSE\ Y\ FI)\ Q$
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*: $I \leq fb_{\mathcal{F}}\ F\ I \Longrightarrow I \subseteq fb_{\mathcal{F}}\ (kstar\ F)\ I$
unfolding *kstar-def ffb-eq* **apply** *clarsimp*
using *kstar-inv* **by** *blast*

lemma *ffb-kstarI*:
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−
have $I \subseteq fb_{\mathcal{F}}\ (kstar\ F)\ I$
using *assms*(*2*) *ffb-star-induct-self* **by** *blast*
hence $P \leq fb_{\mathcal{F}}\ (kstar\ F)\ I$
using *assms*(*1*) **by** *auto*
also have $fb_{\mathcal{F}}\ (kstar\ F)\ I \leq fb_{\mathcal{F}}\ (kstar\ F)\ Q$
by (*rule ffb-iso*[*OF assms*(*3*)])
finally show *?thesis* **.**
qed

3.2 Verification of hybrid programs

notation *g-orbital* ((*1x´=- & - on - - @ -*))

abbreviation *g-evol* ::(('a::banach)⇒'a) ⇒ 'a pred ⇒ 'a ⇒ 'a set
((*1x´=- & -*)) **where** (*x´=f & G*) *s* ≡ (*x´=f & G on UNIV UNIV @ 0*) *s*

3.2.1 Verification by providing solutions

lemma *ffb-g-orbital-eq*: $fb_{\mathcal{F}}$ (*x´=f & G on T S @* t_0) *Q* =
$\{s.\ \forall X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t \in T.\ (\mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow \mathcal{P}\ X\ (down\ T\ t) \subseteq Q\}$
unfolding *ffb-eq g-orbital-eq image-le-pred subset-eq* **apply**(*clarsimp*, *safe*)
apply(*erule-tac x=X xa* **in** *allE*, *erule impE*, *force*, *simp*)
by (*erule-tac x=X* **in** *ballE*, *simp-all*)

lemma *ffb-g-orbital*: $fb_{\mathcal{F}}$ (*x´=f & G on T S @* t_0) *Q* =

$\{s.\ \forall X \in \textit{ivp-sols}\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t \in T.\ (\forall \tau \in \textit{down}\ T\ t.\ G\ (X\ \tau)) \longrightarrow (X\ t) \in Q\}$
unfolding *ffb-eq g-orbital-eq* **by** *auto*

context *local-flow*
begin

lemma *ffb-g-orbit*: $fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ 0)\ Q =$
$\{s.\ s \in S \longrightarrow (\forall t \in T.\ (\forall \tau \in \textit{down}\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow (\varphi\ t\ s) \in Q)\}$ (**is** - = *?wlp*)
unfolding *ffb-g-orbital* **apply**(*safe*, *clarsimp*)
apply(*erule-tac* $x = \lambda t.\ \varphi\ t\ x$ **in** *ballE*)
using *in-ivp-sols* **apply**(*force*, *force*, *force simp*: *init-time ivp-sols-def*)
apply(*subgoal-tac* $\forall \tau \in \textit{down}\ T\ t.\ X\ \tau = \varphi\ \tau\ x$, *simp-all*, *clarsimp*)
apply(*subst eq-solution*, *simp-all add*: *ivp-sols-def*)
using *init-time* **by** *auto*

lemma *ffb-orbit*: $fb_{\mathcal{F}}\ \gamma^{\varphi}\ Q = \{s.\ s \in S \longrightarrow (\forall\ t \in T.\ \varphi\ t\ s \in Q)\}$
unfolding *orbit-def ffb-g-orbit* **by** *simp*

end

3.2.2 Verification with differential invariants

lemma *ffb-g-orbital-guard*:
assumes $H = (\lambda s.\ G\ s \wedge Q\ s)$
shows $fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ H\ s\} = fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ Q\ s\}$
unfolding *ffb-g-orbital* **using** *assms* **by** *auto*

lemma *ffb-g-orbital-inv*:
assumes $P \leq I$ **and** $I \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ I$ **and** $I \leq Q$
shows $P \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ Q$
using *assms*(*1*) **apply**(*rule order.trans*)
using *assms*(*2*) **apply**(*rule order.trans*)
by (*rule ffb-iso*[*OF assms*(*3*)])

lemma *diff-invariant* $I\ f\ T\ S\ t_0\ G = (((\textit{g-orbital}\ f\ G\ T\ S\ t_0)^{\dagger})\ \{s.\ I\ s\} \subseteq \{s.\ I\ s\})$
unfolding *klift-def diff-invariant-def* **by** *simp*

lemma *bdf-diff-inv*:
diff-invariant $I\ f\ T\ S\ t_0\ G = (bd_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\} \leq \{s.\ I\ s\})$
unfolding *ffb-fbd-galois-var* **by** (*auto simp*: *diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *ffb-diff-inv*:
$(\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x' = f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\}) =$ *diff-invariant* $I\ f\ T\ S\ t_0\ G$
by (*auto simp*: *diff-invariant-def ivp-sols-def ffb-eq g-orbital-eq*)

lemma *diff-inv-guard-ignore*:
assumes $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x'{=}f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\}$
shows $\{s.\ I\ s\} \leq fb_{\mathcal{F}}\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ I\ s\}$
using *assms* **unfolding** *ffb-diff-inv diff-invariant-eq image-le-pred* **by** *auto*

context *local-flow*
begin

lemma *ffb-diff-inv-eq*: $diff\text{-}invariant\ I\ f\ T\ S\ 0\ (\lambda s.\ True) =$
$(\{s.\ s \in S \longrightarrow I\ s\} = fb_{\mathcal{F}}\ (x'{=}f\ \&\ (\lambda s.\ True)\ on\ T\ S\ @\ 0)\ \{s.\ s \in S \longrightarrow I\ s\})$
unfolding *ffb-diff-inv*[*symmetric*] *ffb-g-orbital*
using *init-time* **apply**(*auto simp*: *subset-eq ivp-sols-def*)
apply(*subst ivp*(*2*)[*symmetric*], *simp*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ x$ **in** *allE*)
using *in-domain has-vderiv-on-domain ivp*(*2*) *init-time* **by** *force*

lemma *diff-inv-eq-inv-set*:
$diff\text{-}invariant\ I\ f\ T\ S\ 0\ (\lambda s.\ True) = (\forall s.\ I\ s \longrightarrow \gamma^{\varphi}\ s \subseteq \{s.\ I\ s\})$
unfolding *diff-inv-eq-inv-set orbit-def* **by** *simp*

end

3.2.3 Derivation of the rules of dL

We derive domain specific rules of differential dynamic logic (dL).

lemma *diff-solve-axiom*:
fixes $c{::}'a{::}\{heine\text{-}borel,\ banach\}$
assumes $0 \in T$ **and** *is-interval T open T*
shows $fb_{\mathcal{F}}\ (x'{=}(\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0)\ Q =$
$\{s.\ \forall t{\in}T.\ (\mathcal{P}\ (\lambda\tau.\ s + \tau *_R c)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow (s + t *_R c) \in Q\}$
apply(*subst local-flow.ffb-g-orbit*[*of* $\lambda s.\ c$ - - $(\lambda t\ s.\ s + t *_R c)$])
using *line-is-local-flow assms* **unfolding** *image-le-pred* **by** *auto*

lemma *diff-solve-rule*:
assumes *local-flow f T UNIV* φ
and $\forall s.\ s \in P \longrightarrow (\forall\ t{\in}T.\ (\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow (\varphi\ t$
$s) \in Q)$
shows $P \leq fb_{\mathcal{F}}\ (x'{=}f\ \&\ G\ on\ T\ UNIV\ @\ 0)\ Q$
using *assms* **by**(*subst local-flow.ffb-g-orbit*) *auto*

lemma *diff-weak-axiom*: $fb_{\mathcal{F}}\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ Q = fb_{\mathcal{F}}\ (x'{=}f\ \&\ G\ on$
$T\ S\ @\ t_0)\ \{s.\ G\ s \longrightarrow s \in Q\}$
unfolding *ffb-g-orbital image-def* **by** *force*

lemma *diff-weak-rule*: $\{s.\ G\ s\} \leq Q \Longrightarrow P \leq fb_{\mathcal{F}}\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ Q$
by(*auto intro*: *g-orbitalD simp*: *le-fun-def g-orbital-eq ffb-eq*)

lemma *ffb-g-orbital-eq-univD*:
assumes $fb_{\mathcal{F}}\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \{s.\ C\ s\} = UNIV$

```
    and ∀τ∈(down T t). x τ ∈ (x´=f & G on T S @ t_0) s
  shows ∀τ∈(down T t). C (x τ)
proof
  fix τ assume τ ∈ (down T t)
  hence x τ ∈ (x´=f & G on T S @ t_0) s
    using assms(2) by blast
  also have ∀y. y ∈ (x´=f & G on T S @ t_0) s ⟶ C y
    using assms(1) ffb-eq-univD by fastforce
  ultimately show C (x τ) by blast
qed

lemma diff-cut-axiom:
  assumes Thyp: is-interval T t_0 ∈ T
    and fb_ℱ (x´=f & G on T S @ t_0) {s. C s} = UNIV
  shows fb_ℱ (x´=f & G on T S @ t_0) Q = fb_ℱ (x´=f & (λs. G s ∧ C s) on T
S @ t_0) Q
proof(rule-tac f=λ x. fb_ℱ x Q in HOL.arg-cong, rule ext, rule subset-antisym)
  fix s
  {fix s′ assume s′ ∈ (x´=f & G on T S @ t_0) s
    then obtain τ::real and X where x-ivp: X ∈ ivp-sols (λt. f) T S t_0 s
      and X τ = s′ and τ ∈ T and guard-x:𝒫 X (down T τ) ⊆ {s. G s}
      using g-orbitalD[of s′ f G T S t_0 s]  by blast
    have ∀t∈(down T τ). 𝒫 X (down T t) ⊆ {s. G s}
      using guard-x by (force simp: image-def)
    also have ∀t∈(down T τ). t ∈ T
      using ‹τ ∈ T› Thyp closed-segment-subset-interval by auto
    ultimately have ∀t∈(down T τ). X t ∈ (x´=f & G on T S @ t_0) s
      using g-orbitalI[OF x-ivp] by (metis (mono-tags, lifting))
    hence ∀t∈(down T τ). C (X t)
      using assms by (meson ffb-eq-univD mem-Collect-eq)
    hence s′ ∈ (x´=f & (λs. G s ∧ C s) on T S @ t_0) s
      using g-orbitalI[OF x-ivp ‹τ ∈ T›] guard-x ‹X τ = s′›
      unfolding image-le-pred by fastforce}
  thus (x´=f & G on T S @ t_0) s ⊆ (x´=f & (λs. G s ∧ C s) on T S @ t_0) s
    by blast
next show ⋀s. (x´=f & (λs. G s ∧ C s) on T S @ t_0) s ⊆ (x´=f & G on T S
@ t_0) s
    by (auto simp: g-orbital-eq)
qed

lemma diff-cut-rule:
  assumes Thyp: is-interval T t_0 ∈ T
    and ffb-C: P ≤ fb_ℱ (x´=f & G on T S @ t_0) {s. C s}
    and ffb-Q: P ≤ fb_ℱ (x´=f & (λs. G s ∧ C s) on T S @ t_0) Q
  shows P ≤ fb_ℱ (x´=f & G on T S @ t_0) Q
proof(subst ffb-eq, subst g-orbital-eq, clarsimp)
  fix t::real and X::real ⇒ ′a and s assume s ∈ P and t ∈ T
    and x-ivp:X ∈ ivp-sols (λt. f) T S t_0 s
    and guard-x:𝒫 X (down T t) ⊆ {s. G s}
```

```
  have ∀ r∈(down T t). X r ∈ (x´=f & G on T S @ t_0) s
    using g-orbitalI[OF x-ivp] guard-x unfolding image-le-pred by auto
  hence ∀ t∈(down T t). C (X t)
    using ffb-C ‹s ∈ P› by (subst (asm) ffb-eq, auto)
  hence X t ∈ (x´=f & (λs. G s ∧ C s) on T S @ t_0) s
    using guard-x ‹t ∈ T› by (auto intro!: g-orbitalI x-ivp)
  thus (X t) ∈ Q
    using ‹s ∈ P› ffb-Q by (subst (asm) ffb-eq) auto
qed

lemma solve:
  assumes local-flow f UNIV UNIV φ
    and ∀ s. s ∈ P ⟶ (∀ t. (∀ τ≤t. G (φ τ s)) ⟶ (φ t s) ∈ Q)
  shows P ≤ fb_𝓕 (x´=f & G) Q
  apply(rule diff-solve-rule[OF assms(1)])
  using assms(2) unfolding image-le-pred by simp

lemma DS:
  fixes c::'a::{heine-borel, banach}
  shows fb_𝓕 (x´=(λs. c) & G) Q = {x. ∀ t. (∀ τ≤t. G (x + τ *_R c)) ⟶ (x + t
*_R c) ∈ Q}
  by (subst diff-solve-axiom[of UNIV]) auto

lemma DW: fb_𝓕 (x´=f & G) Q = fb_𝓕 (x´=f & G) {s. G s ⟶ s ∈ Q}
  by (rule diff-weak-axiom)

lemma dW: {s. G s} ≤ Q ⟹ P ≤ fb_𝓕 (x´=f & G) Q
  by (rule diff-weak-rule)

lemma DC:
  assumes fb_𝓕 (x´=f & G) {s. C s} = UNIV
  shows fb_𝓕 (x´=f & G) Q = fb_𝓕 (x´=f & (λs. G s ∧ C s)) Q
  by (rule diff-cut-axiom) (auto simp: assms)

lemma dC:
  assumes P ≤ fb_𝓕 (x´=f & G) {s. C s}
    and P ≤ fb_𝓕 (x´=f & (λs. G s ∧ C s)) Q
  shows P ≤ fb_𝓕 (x´=f & G) Q
  apply(rule diff-cut-rule)
  using assms by auto

lemma dI:
  assumes P ≤ {s. I s} and diff-invariant I f UNIV UNIV 0 G and {s. I s} ≤ Q
  shows P ≤ fb_𝓕 (x´=f & G) Q
  apply(rule ffb-g-orbital-inv[OF assms(1) - assms(3)])
  using ffb-diff-inv[symmetric] assms(2) by force

end
theory cat2funcset-examples
```

imports *../hs-prelims-matrices cat2funcset*

begin

3.2.4 Examples

Preliminary preparation for the examples.

— Finite set of program variables.

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

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

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* {''x'',''y''}])
   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-univ-eq*: (*UNIV*::*program-vars set*) = {$\upharpoonright_V$''x'', $\upharpoonright_V$''y''}
  apply *auto* **by** (*metis to-str to-str-inverse insertE singletonD*)

lemma *program-vars-exhaust*: $x = \upharpoonright_V$''x'' $\vee$ $x = \upharpoonright_V$''y''
  using *program-vars-univ-eq* **by** *auto*

— Alternative to the finite set of program variables.

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

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

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

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

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

Circular Motion

— Verified with differential invariants.

abbreviation *circular-motion-kinematics* :: *real^program-vars* ⇒ *real^program-vars*

where $circular\text{-}motion\text{-}kinematics\ s \equiv (\chi\ i.\ if\ i{=}(\restriction_V {''x''})\ then\ s\$(\restriction_V {''y''})\ else\ {-s\$(\restriction_V {''x''})})$

notation *circular-motion-kinematics* (C)

lemma *circle-invariant*:
$diff\text{-}invariant\ (\lambda s.\ (r::real)^2 = (s\$(\restriction_V {''x''}))^2 + (s\$(\restriction_V {''y''}))^2)\ C\ UNIV\ UNIV\ 0\ G$
apply(*rule-tac diff-invariant-rules*, *clarsimp*, *simp*, *clarsimp*)
apply(*frule-tac* $i{=}\restriction_V {''x''}$ **in** *has-vderiv-on-vec-nth*, *drule-tac* $i{=}\restriction_V {''y''}$ **in** *has-vderiv-on-vec-nth*)
by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *circular-motion-invariant*:
$\{s.\ r^2 = (s\$\restriction_V {''x''})^2 + (s\$\restriction_V {''y''})^2\} \leq fb_{\mathcal{F}}\ (x'{=}C\ \&\ G)\ \{s.\ r^2 = (s\$\restriction_V {''x''})^2 + (s\$\restriction_V {''y''})^2\}$
unfolding *ffb-diff-inv* **using** *circle-invariant* **by** *simp*

— Verified with the flow.

abbreviation $circular\text{-}motion\text{-}flow\ t\ s \equiv$
$(\chi\ i.\ if\ i{=}\restriction_V {''x''}\ then\ s\$(\restriction_V {''x''}) \cdot cos\ t + s\$(\restriction_V {''y''}) \cdot sin\ t$
$else\ -\ s\$(\restriction_V {''x''}) \cdot sin\ t + s\$(\restriction_V {''y''}) \cdot cos\ t)$

notation *circular-motion-flow* (φ_C)

lemma *picard-lindeloef-circ-motion*: $picard\text{-}lindeloef\ (\lambda t.\ C)\ UNIV\ UNIV\ 0$
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac* $x{=}1$ **in** *exI*, *clarsimp*, *rule-tac* $x{=}1$ **in** *exI*)
by(*simp add*: *dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject power2-commute*)

lemma *local-flow-circ-motion*: *local-flow C UNIV UNIV* φ_C
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
apply(*rule picard-lindeloef-circ-motion*, *simp-all add*: *vec-eq-iff*)
apply(*rule has-vderiv-on-vec-lambda*, *clarify*)
apply(*case-tac* $i = \restriction_V {''x''}$, *simp*)
apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)

apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)
using *program-vars-exhaust* **by** *force*

lemma *circular-motion*:
$\{s.\ r^2 = (s\$\restriction_V{''x''})^2 + (s\$\restriction_V{''y''})^2\} \le fb_{\mathcal{F}}\ (x' {=} C\ \&\ G)\ \{s.\ r^2 = (s\$\restriction_V{''x''})^2 + (s\$\restriction_V{''y''})^2\}$
by (*subst local-flow.ffb-g-orbit*[*OF local-flow-circ-motion*]) (*auto simp*: *to-var-inject*)

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

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

— Verified as a linear system (using uniqueness).

abbreviation *circular-motion-sq-mtx* :: *2 sq-mtx*
where *circular-motion-sq-mtx* $\equiv$ *sq-mtx-chi* (χ *i*. *if i=0 then* $-$ e *1 else* e *0*)

abbreviation *circular-motion-mtx-flow* :: *real* $\Rightarrow$ *real^2* $\Rightarrow$ *real^2*
where *circular-motion-mtx-flow t s* $\equiv$
(χ *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-sq-mtx* (C)

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

lemma *circular-motion-mtx-exp-eq*: $exp\ (t *_R C) *_V s = \varphi_C\ t\ s$
apply(*rule local-flow.eq-solution*[*OF local-flow-exp*, *symmetric*])
apply(*rule ivp-solsI*, *rule has-vderiv-on-vec-lambda*, *clarsimp*)
unfolding *sq-mtx-vec-prod-def matrix-vector-mult-def* **apply** *simp*
apply(*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)
using *exhaust-2 two-eq-zero* **by** (*force simp*: *vec-eq-iff*, *auto*)

lemma *circular-motion-sq-mtx*:
$\{s.\ r^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\} \le fb_{\mathcal{F}}\ (x' {=} (*_V)\ C\ \&\ G)\ \{s.\ r^2 = (s\ \$\ 0)^2 + (s\ \$\ 1)^2\}$
unfolding *local-flow.ffb-g-orbit*[*OF local-flow-exp*] *circular-motion-mtx-exp-eq* **by** *auto*

no-notation *circular-motion-sq-mtx* (C)

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

Bouncing Ball

— Verified with differential invariants.

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

lemma [*bb-real-arith*]:

assumes $0 > g$ **and** *inv*: $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$
shows $(x{::}real) \leq h$
proof−
have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot h \wedge 0 > g$
using *inv* **and** $\langle 0 > g \rangle$ **by** *auto*
hence *obs*:$v \cdot v = 2 \cdot g \cdot (x - h) \wedge 0 > g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
hence $(v \cdot v)/(2 \cdot g) = (x - h)$
by *auto*
also from *obs* **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

abbreviation *constant-acceleration-kinematics* $g\ s \equiv (\chi\ i.\ \mathit{if}\ i{=}(\restriction_V{}''x'')\ \mathit{then}\ s\$(\restriction_V{}''y'')\ \mathit{else}\ g)$

notation *constant-acceleration-kinematics* (K)

lemma *energy-conservation-invariant*:
fixes $g\ h :: real$
defines *dinv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\restriction_V{}''x'') - 2 \cdot g \cdot h - (s\$(\restriction_V{}''y'') \cdot s\$(\restriction_V{}''y'')) = 0)$
shows *diff-invariant* $I\ (K\ g)$ *UNIV UNIV 0 G*
unfolding *dinv* **apply**(*rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
apply(*frule-tac* $i{=}\restriction_V{}''y''$ **in** *has-vderiv-on-vec-nth*)
apply(*drule-tac* $i{=}\restriction_V{}''x''$ **in** *has-vderiv-on-vec-nth*)
by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *bouncing-ball-invariants*:
fixes $h{::}real$
assumes $g < 0$ **and** $h \geq 0$
defines *diff-inv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\restriction_V{}''x'') - 2 \cdot g \cdot h - (s\$(\restriction_V{}''y'') \cdot s\$(\restriction_V{}''y'')) = 0)$
shows $\{s.\ s\$(\restriction_V{}''x'') = h \wedge s\$(\restriction_V{}''y'') = 0\} \leq fb_{\mathcal{F}}$
$(kstar\ ((x'{=}K\ g\ \&\ (\lambda\ s.\ s\$(\restriction_V{}''x'') \geq 0)) \circ_K$
$(IF\ (\lambda\ s.\ s\$(\restriction_V{}''x'') = 0)\ THEN\ ((\restriction_V{}''y'') ::= (\lambda s.\ -\ s\$(\restriction_V{}''y'')))\ ELSE\ \eta\ FI)))$
$\{s.\ 0 \leq s\$(\restriction_V{}''x'') \wedge s\$(\restriction_V{}''x'') \leq h\}$
apply(*rule ffb-kstarI*[*of* - $\{s.\ 0 \leq s\$(\restriction_V{}''x'') \wedge I\ s\}$])
using $\langle h \geq 0 \rangle$ **apply**(*subst diff-inv*, *clarsimp*, *simp only*: *ffb-kcomp*)
apply(*rule-tac* $b{=}fb_{\mathcal{F}}\ (x'{=}(K\ g)\ \&\ (\lambda\ s.\ s\$(\restriction_V{}''x''){\geq}0))\ \{s.\ 0{\leq}s\$(\restriction_V{}''x'') \wedge I\ s\}$ **in** *order.trans*)
apply(*simp add*: *ffb-g-orbital-guard*)
apply(*rule-tac* $b{=}\{s.\ I\ s\}$ **in** *order.trans*, *force*)
unfolding *ffb-diff-inv* **apply**(*simp-all add*: *diff-inv*)
using *energy-conservation-invariant* **apply** *force*

```
 apply(rule ffb-iso)
 using assms unfolding diff-inv ffb-if-then-else
 by (auto simp: bb-real-arith to-var-inject)
```

— Verified with the flow.

```
lemma picard-lindeloef-cnst-acc:
  fixes g::real
  shows picard-lindeloef (λt. K g) UNIV UNIV 0
  apply(unfold-locales)
  apply(unfold-locales, simp-all add: local-lipschitz-def lipschitz-on-def, clarsimp)
  apply(rule-tac x=1/2 in exI, clarsimp, rule-tac x=1 in exI)
 by(simp add: dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject)

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

notation constant-acceleration-kinematics-flow (φK)

lemma local-flow-cnst-acc: local-flow (K g) UNIV UNIV (φK g)
  unfolding local-flow-def local-flow-axioms-def apply safe
  using picard-lindeloef-cnst-acc apply blast
   apply(rule has-vderiv-on-vec-lambda, clarify)
   apply(case-tac i = ↾V ''x'')
  using program-vars-exhaust by(auto intro!: poly-derivatives simp: to-var-inject
vec-eq-iff)

lemma [bb-real-arith]:
  assumes invar: 2 · g · x = 2 · g · h + v · v
    and pos: g · τ² / 2 + v · τ + (x::real) = 0
  shows 2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) = 0
proof−
  from pos have g · τ²  + 2 · v · τ + 2 · x = 0 by auto
  then have g²  · τ²  + 2 · g · v · τ + 2 · g · 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 g² · τ² + 2 · g · v · τ + v² + 2 · g · h = 0
    using invar by (simp add: monoid-mult-class.power2-eq-square)
  hence obs: (g · τ + v)² + 2 · g · 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))
  thus 2 · g · h + (g · τ · (g · τ + v) + v · (g · τ + v)) = 0
    by (simp add: monoid-mult-class.power2-eq-square)
  have  2 · g · h + (− ((g · τ) + v))² = 0
    using obs by (metis Groups.add-ac(2) power2-minus)
qed
```

lemma [*bb-real-arith*]:
assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
shows $2 \cdot g \cdot (g \cdot \tau^2 \mathbin{/} 2 + v \cdot \tau + (x{::}real)) =$
$2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v))$ (**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*:
fixes *h*::*real*
assumes $g < 0$ **and** $h \geq 0$
defines *loop-inv*: $I \equiv (\lambda s.\ 0 \leq s\$(\restriction_V ''x'') \wedge\ 2 \cdot g \cdot s\$(\restriction_V ''x'') =$
$2 \cdot g \cdot h + (s\$(\restriction_V ''y'') \cdot s\$(\restriction_V ''y'')))$
shows $\{s.\ s\$(\restriction_V ''x'') = h \wedge s\$(\restriction_V ''y'') = 0\} \leq fb_{\mathcal{F}}$
$(kstar\ ((x\acute{}=K\ g\ \&\ (\lambda\ s.\ s\ \$\ (\restriction_V\ ''x'') \geq 0)) \circ_K$
$(IF\ (\lambda\ s.\ s\$(\restriction_V ''x'') = 0)\ THEN\ ((\restriction_V ''y'') ::= (\lambda s.\ -\ s\$(\restriction_V ''y'')))\ ELSE\ \eta$
$FI)))$
$\{s.\ 0 \leq s\$(\restriction_V ''x'') \wedge s\$(\restriction_V ''x'') \leq h\}$
apply(*rule ffb-kstarI*[*of* - {*s*. *I s*}])
unfolding *loop-inv* **using** ‹$h \geq 0$› **apply**(*clarsimp*, *simp only*: *ffb-kcomp*)
apply(*subst local-flow.ffb-g-orbit*[*OF local-flow-cnst-acc*], *simp*)
unfolding *ffb-if-then-else* **using** *assms*
by (*auto simp*: *bb-real-arith to-var-inject*)

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

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

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

— Verified as a linear system (computing exponential).

abbreviation *constant-acceleration-kinematics-sq-mtx* :: *3 sq-mtx*
where *constant-acceleration-kinematics-sq-mtx* ≡
sq-mtx-chi (χ *i*::*3*. *if i*=*0 then* e *1 else if i*=*1 then* e *2 else 0*)

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

lemma *const-acc-mtx-pow2*: K^2 = *sq-mtx-chi* (χ *i. if i=0 then* e *2 else 0*)
unfolding *power2-eq-square times-sq-mtx-def*
by(*simp add*: *sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def*)

lemma *const-acc-mtx-powN*: $n > 2 \Longrightarrow (\tau *_R K)\hat{}n = 0$
apply(*induct n*, *simp*, *case-tac* $n \leq 2$)
apply(*simp only*: *le-less-Suc-eq power-Suc*, *simp*)
by(*auto simp*: *const-acc-mtx-pow2 sq-mtx-chi-inject vec-eq-iff times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def*)

lemma *exp-cnst-acc-sq-mtx*: $exp\ (\tau *_R K) = ((\tau *_R K)^2 /_R\ 2) + (\tau *_R K) + 1$
unfolding *exp-def* **apply**(*subst suminf-eq-sum*[*of 2*])
using *const-acc-mtx-powN* **by** (*simp-all add*: *numeral-2-eq-2*)

lemma *exp-cnst-acc-sq-mtx-simps*:
$exp\ (\tau *_R K)$ \$\$ *0* \$ *0* = *1* $exp\ (\tau *_R K)$ \$\$ *0* \$ *1* = τ $exp\ (\tau *_R K)$ \$\$ *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* = τ
$exp\ (\tau *_R K)$ \$\$ *2* \$ *0* = *0* $exp\ (\tau *_R K)$ \$\$ *2* \$ *1* = *0* $exp\ (\tau *_R K)$ \$\$ *2* \$ *2* = *1*
unfolding *exp-cnst-acc-sq-mtx scaleR-power const-acc-mtx-pow2*
by (*auto simp*: *plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def mat-def scaleR-vec-def axis-def plus-vec-def*)

lemma *bouncing-ball-K*:
{*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* & (λ *s. s* \$ *0* $\geq$ *0*)) $\circ_K$
(*IF* (λ *s. s* \$ *0* = *0*) *THEN* (*1* ::= (λ*s.* − *s* \$ *1*)) *ELSE* η *FI*)))
{*s. 0* $\leq$ *s* \$ *0* $\wedge$ *s* \$ *0* $\leq$ *h*}
apply(*rule ffb-kstarI*[*of* - {*s. 0*≤*s*\$*0* $\wedge$ *0*>*s*\$*2* $\wedge$ *2* · *s*\$*2* · *s*\$*0* = *2* · *s*\$*2* · *h* + (*s*\$*1* · *s*\$*1*)}])
apply(*clarsimp*, *simp only*: *ffb-kcomp*)
apply(*subst local-flow.ffb-g-orbit*[*OF local-flow-exp*])
apply(*subst ffb-if-then-else*, *simp add*: *sq-mtx-vec-prod-eq*)
unfolding *UNIV-3* **apply**(*simp add*: *exp-cnst-acc-sq-mtx-simps*, *safe*)
subgoal for *x* **using** *bb-real-arith*(*2*)[*of x*\$*2*]
by (*simp add*: *add.commute mult.commute*)
subgoal for *x* τ **using** *bb-real-arith*(*3*)[**where** *g*=*x*\$*2* **and** *v*=*x*\$*1*]
by(*simp add*: *add.commute mult.commute*)
by (*force simp*: *bb-real-arith*)

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

end
theory *cat2rel*
imports
../*hs-prelims-dyn-sys*

../../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* ($r2s$)
and *VC-KAD.gets* (- ::= - [70, 65] 61)

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 *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

lemma *p2r-r2p-wp*: $\lceil \lfloor wp\ R\ P \rfloor \rceil = wp\ R\ P$
apply(*subst d-p2r*[*symmetric*])
using *wp-simp*[*symmetric*, *of R P*] **by** *blast*

lemma *p2r-r2p-simps*:
$\lfloor \lceil P \sqcap Q \rceil \rfloor = (\lambda\ s.\ \lfloor \lceil P \rceil \rfloor\ s \wedge \lfloor \lceil Q \rceil \rfloor\ s)$
$\lfloor \lceil P \sqcup Q \rceil \rfloor = (\lambda\ s.\ \lfloor \lceil P \rceil \rfloor\ s \vee \lfloor \lceil Q \rceil \rfloor\ s)$
$\lfloor \lceil P \rceil \rfloor = P$
unfolding *p2r-def r2p-def* **by** (*auto simp*: *fun-eq-iff*)

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

abbreviation *vec-upd* :: $('a\hat{\ }'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a\hat{\ }'b$
  where *vec-upd x i a* $\equiv$ *vec-lambda* ((*vec-nth x*)($i := a$))

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

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

lemma *wp-assign-var* [*simp*]: $\lfloor$*wp* $(x ::= e)$ $\lceil Q \rceil\rfloor = (\lambda s.\ Q$ (*vec-upd s x* $(e\ s)))$
  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*: $|$*if p then x else y 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−
  have $d\ i \le |x]\ (d\ i)$
    using ‹$d\ i \le |x]\ i$› *local.fbox-simp* **by** *auto*
  hence $|1]\ p \le |x^\star]\ i$
    using ‹$d\ p \le d\ i$› **by** (*metis* (*no-types*) *dual-order.trans*
      fbox-one fbox-simp fbox-star-induct-var)
  thus *?thesis*
    using ‹$d\ i \le d\ q$› **by** (*metis* (*full-types*) *fbox-mult*
      fbox-one fbox-seq-var 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*
  hence *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

abbreviation *g-evolution* ::(('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ *real set* ⇒ '*a set* ⇒ *real* ⇒ '*a rel* ((1*x*´=- & - *on* - - @ -))
where $(x´=f\ \&\ G\ on\ T\ S\ @\ t_0) \equiv \{(s,s') \mid s\ s'.\ s' \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s\}$

abbreviation *g-evol* ::(('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ '*a rel* ((1*x*´=- & -))
where $(x´=f\ \&\ G) \equiv (x´=f\ \&\ G\ on\ UNIV\ UNIV\ @\ 0)$

4.2.1 Verification by providing solutions

lemma *wp-g-evolution*: $wp\ (x´=f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil=$
$\lceil\lambda\ s.\ \forall X{\in}ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t{\in}T.\ (\forall\tau{\in}down\ T\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t)\rceil$
unfolding *g-orbital-eq wp-rel ivp-sols-def image-le-pred* **by** *auto*

context *local-flow*
begin

lemma *wp-g-orbit*: $wp\ (x´=f\ \&\ G\ on\ T\ S\ @\ 0)\ \lceil Q\rceil =$
$\lceil\lambda\ s.\ s \in S \longrightarrow (\forall t{\in}T.\ (\forall\tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
unfolding *wp-g-evolution* **apply**(*clarsimp*, *safe*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ s$ **in** *ballE*)
using *in-ivp-sols* **apply**(*force*, *force*, *force simp*: *init-time ivp-sols-def*)
apply(*subgoal-tac* $\forall\tau{\in}down\ T\ t.\ X\ \tau = \varphi\ \tau\ s$, *simp-all*, *clarsimp*)
apply(*subst eq-solution*, *simp-all add*: *ivp-sols-def*)
using *init-time* **by** *auto*

lemma *wp-orbit*: $wp\ (\{(s,s') \mid s\ s'.\ s' \in \gamma^{\varphi}\ s\})\ \lceil Q\rceil = \lceil\lambda\ s.\ s \in S \longrightarrow (\forall\ t \in T.\ Q\ (\varphi\ t\ s))\rceil$
unfolding *orbit-def wp-g-orbit* **by** *auto*

end

4.2.2 Verification with differential invariants

lemma *wp-g-evolution-guard*:
assumes $H = (\lambda s.\ G\ s \wedge Q\ s)$
shows $wp\ (x´=f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil H\rceil = wp\ (x´=f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
unfolding *wp-g-evolution* **using** *assms* **by** *auto*

lemma *wp-g-evolution-inv*:
assumes $\lceil P\rceil \le \lceil I\rceil$ **and** $\lceil I\rceil \le wp\ (x´=f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$ **and** $\lceil I\rceil \le \lceil Q\rceil$
shows $\lceil P\rceil \le wp\ (x´=f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$

using *assms*(*1*) **apply**(*rule order.trans*)
using *assms*(*2*) **apply**(*rule order.trans*)
apply(*rule rel-antidomain-kleene-algebra.fbox-iso*)
using *assms*(*3*) **by** *auto*

lemma *wp-diff-inv*: $(\lceil I \rceil \leq wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I \rceil) = diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G$
unfolding *diff-invariant-eq wp-g-evolution image-le-pred* **by**(*auto simp*: *p2r-def*)

4.2.3 Derivation of the rules of dL

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.

lemma *diff-solve-axiom*:
fixes $c::'a::\{heine\text{-}borel,\ banach\}$
assumes $0 \in T$ **and** *is-interval* T *open* T
shows $wp\ (x'{=}(\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q \rceil =$
$\lceil \lambda\ s.\ \forall\, t \in T.\ (\mathcal{P}\ (\lambda\ t.\ s + t *_R c)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (s + t *_R c) \rceil$
apply(*subst local-flow.wp-g-orbit*[**where** $f{=}\lambda s.\ c$ **and** $\varphi{=}(\lambda\ t\ x.\ x + t *_R c)$])
using *line-is-local-flow assms* **unfolding** *image-le-pred* **by** *auto*

lemma *diff-solve-rule*:
assumes *local-flow* $f\ T\ UNIV\ \varphi$
and $\forall s.\ P\ s \longrightarrow (\forall\ t \in T.\ (\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\lceil P \rceil \leq wp\ (x'{=}f\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q \rceil$
using *assms* **by**(*subst local-flow.wp-g-orbit*, *auto*)

lemma *diff-weak-axiom*: $wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil = wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil \lambda\ s.\ G\ s \longrightarrow Q\ s \rceil$
unfolding *wp-g-evolution image-def* **by** *force*

lemma *diff-weak-rule*:
assumes $\lceil G \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$
using *assms* **apply**(*subst wp-rel*)
by(*auto simp*: *g-orbital-eq*)

lemma *wp-g-orbit-IdD*:
assumes $wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil = Id$
and $\forall \tau \in (down\ T\ t).\ (s,\ x\ \tau) \in (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)$
shows $\forall \tau \in (down\ T\ t).\ C\ (x\ \tau)$
proof
fix τ **assume** $\tau \in (down\ T\ t)$
hence $x\ \tau \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
using *assms*(*2*) **by** *blast*

also have $\forall y.\ y \in (\textit{g-orbital}\ f\ G\ T\ S\ t_0\ s) \longrightarrow C\ y$
using *assms(1)* **unfolding** *wp-rel* **by**(*auto simp*: *p2r-def*)
ultimately show $C\ (x\ \tau)$
by *blast*
qed

lemma *diff-cut-axiom*:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C\rceil = Id$
shows $wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil = wp\ (x\acute{}{=}f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
proof(*rule-tac* $f{=}\lambda\ x.\ wp\ x\ \lceil Q\rceil$ **in** *HOL.arg-cong*, *clarsimp*, *rule subset-antisym*, *safe*)
{**fix** s **and** s' **assume** $s' \in \textit{g-orbital}\ f\ G\ T\ S\ t_0\ s$
then obtain τ::*real* **and** X **where** *x-ivp*: $X \in \textit{ivp-sols}\ (\lambda t.\ f)\ T\ S\ t_0\ s$
and $X\ \tau = s'$ **and** $\tau \in T$ **and** *guard-x*:$(\mathcal{P}\ X\ (down\ T\ \tau) \subseteq \{s.\ G\ s\})$
using *g-orbitalD*[*of* s' f G T S t_0 s] **by** *blast*
have $\forall t{\in}(down\ T\ \tau).\ \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}$
using *guard-x* **by** (*force simp*: *image-def*)
also have $\forall t{\in}(down\ T\ \tau).\ t \in T$
using ‹$\tau \in T$› *Thyp* **by** *auto*
ultimately have $\forall t{\in}(down\ T\ \tau).\ X\ t \in \textit{g-orbital}\ f\ G\ T\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))
hence $\forall t{\in}(down\ T\ \tau).\ C\ (X\ t)$
using *wp-g-orbit-IdD*[*OF assms(3)*] **by** *blast*
hence $s' \in \textit{g-orbital}\ f\ (\lambda s.\ G\ s \wedge C\ s)\ T\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp* ‹$\tau \in T$›] *guard-x* ‹$X\ \tau = s'$›
unfolding *image-le-pred* **by** *fastforce*}
thus $\bigwedge s\ s'.\ s' \in \textit{g-orbital}\ f\ G\ T\ S\ t_0\ s \Longrightarrow s' \in \textit{g-orbital}\ f\ (\lambda s.\ G\ s \wedge C\ s)\ T\ S\ t_0\ s$
by *blast*
next show $\bigwedge s\ s'.\ s' \in \textit{g-orbital}\ f\ (\lambda s.\ G\ s \wedge C\ s)\ T\ S\ t_0\ s \Longrightarrow s' \in \textit{g-orbital}\ f\ G\ T\ S\ t_0\ s$
by (*auto simp*: *g-orbital-eq*)
qed

lemma *diff-cut-rule*:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and *wp-C*: $\lceil P\rceil \le wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C\rceil$
and *wp-Q*: $\lceil P\rceil \subseteq wp\ (x\acute{}{=}f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
shows $\lceil P\rceil \subseteq wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
proof(*subst wp-rel*, *simp add*: *g-orbital-eq p2r-def image-le-pred*, *clarsimp*)
fix t::*real* **and** X::*real* $\Rightarrow$ *'a* **and** s **assume** $P\ s$ **and** $t \in T$
and *x-ivp*:$X \in \textit{ivp-sols}\ (\lambda t.\ f)\ T\ S\ t_0\ s$
and *guard-x*:$\forall x.\ x \in T \wedge x \le t \longrightarrow G\ (X\ x)$
have $\forall t{\in}(down\ T\ t).\ X\ t \in \textit{g-orbital}\ f\ G\ T\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp*] *guard-x* **unfolding** *image-le-pred* **by** *auto*
hence $\forall t{\in}(down\ T\ t).\ C\ (X\ t)$
using *wp-C* ‹$P\ s$› **by** (*subst* (*asm*) *wp-rel*, *auto*)

hence $X\ t \in$ *g-orbital f* $(\lambda s.\ G\ s \wedge C\ s)\ T\ S\ t_0\ s$
using *guard-x* $\langle t \in T\rangle$ **by** (*auto intro*!: *g-orbitalI x-ivp*)
thus $Q\ (X\ t)$
using $\langle P\ s\rangle$ *wp-Q* **by** (*subst* (*asm*) *wp-rel*) *auto*
qed

lemma *DS*:
fixes c::$'a$::{*heine-borel*, *banach*}
shows $wp\ (x' {=} (\lambda s.\ c)\ \&\ G)\ \lceil Q\rceil = \lceil \lambda x.\ \forall t.\ (\forall \tau {\le} t.\ G\ (x + \tau *_R c)) \longrightarrow Q\ (x + t *_R c)\rceil$
by (*subst diff-solve-axiom*[*of UNIV*]) *auto*

lemma *solve*:
assumes *local-flow f UNIV UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall t.\ (\forall \tau {\le} t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\lceil P\rceil \le wp\ (x'{=}f\ \&\ G)\ \lceil Q\rceil$
apply(*rule diff-solve-rule*[*OF assms*(*1*)])
using *assms*(*2*) **unfolding** *image-le-pred* **by** *simp*

lemma *DW*: $wp\ (x'{=}f\ \&\ G)\ \lceil Q\rceil = wp\ (x'{=}f\ \&\ G)\ \lceil \lambda s.\ G\ s \longrightarrow Q\ s\rceil$
by (*rule diff-weak-axiom*)

lemma *dW*: $\lceil G\rceil \le \lceil Q\rceil \Longrightarrow \lceil P\rceil \le wp\ (x'{=}f\ \&\ G)\ \lceil Q\rceil$
by (*rule diff-weak-rule*)

lemma *DC*:
assumes $wp\ (x'{=}f\ \&\ G)\ \lceil C\rceil = Id$
shows $wp\ (x'{=}f\ \&\ G)\ \lceil Q\rceil = wp\ (x'{=}f\ \&\ (\lambda s.\ G\ s \wedge C\ s))\ \lceil Q\rceil$
apply (*rule diff-cut-axiom*)
using *assms* **by** *auto*

lemma *dC*:
assumes $\lceil P\rceil \le wp\ (x'{=}f\ \&\ G)\ \lceil C\rceil$
and $\lceil P\rceil \le wp\ (x'{=}f\ \&\ (\lambda s.\ G\ s \wedge C\ s))\ \lceil Q\rceil$
shows $\lceil P\rceil \le wp\ (x'{=}f\ \&\ G)\ \lceil Q\rceil$
apply(*rule diff-cut-rule*)
using *assms* **by** *auto*

lemma *dI*:
assumes $\lceil P\rceil \le \lceil I\rceil$ **and** *diff-invariant I f UNIV UNIV 0 G* **and** $\lceil I\rceil \le \lceil Q\rceil$
shows $\lceil P\rceil \le wp\ (x'{=}f\ \&\ G)\ \lceil Q\rceil$
apply(*rule wp-g-evolution-inv*[*OF assms*(*1*) - *assms*(*3*)])
unfolding *wp-diff-inv* **using** *assms*(*2*) **.**

end
theory *cat2rel-examples*
imports ../*hs-prelims-matrices cat2rel*

begin

4.2.4 Examples

Preliminary preparation for the examples.

no-notation *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor$-$\rfloor$)

— Finite set of program variables.

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

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

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* {''x'',''y''}])
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-univ-eq*: (*UNIV*::*program-vars set*) = {$\restriction_V$''x'', $\restriction_V$''y''}
apply *auto* **by** (*metis to-str to-str-inverse insertE singletonD*)

lemma *program-vars-exhaust*: $x = \restriction_V$''x'' $\lor$ $x = \restriction_V$''y''
using *program-vars-univ-eq* **by** *auto*

— Alternative to the finite set of program variables.

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

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

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

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

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

Circular Motion

— Verified with differential invariants.

abbreviation *circular-motion-kinematics* :: *real^program-vars* $\Rightarrow$ *real^program-vars*

where *circular-motion-kinematics s* $\equiv$ $(\chi\ i.\ if\ i=(\restriction_V ''x'')\ then\ s\$(\restriction_V ''y'')\ else\ -s\$(\restriction_V ''x''))$

notation *circular-motion-kinematics* (C)

lemma *circular-motion-invariant*:
diff-invariant $(\lambda s.\ (r::real)^2 = (s\$(\restriction_V ''x''))^2 + (s\$(\restriction_V ''y''))^2)\ C\ UNIV\ UNIV\ 0\ G$
apply(*rule-tac diff-invariant-rules*, *clarsimp*, *simp*, *clarsimp*)
apply(*frule-tac* $i=\restriction_V ''x''$ **in** *has-vderiv-on-vec-nth*, *drule-tac* $i=\restriction_V ''y''$ **in** *has-vderiv-on-vec-nth*)
by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *circular-motion-invariants*:
$\lceil \lambda s.\ r^2 = (s\$\restriction_V ''x'')^2 + (s\$\restriction_V ''y'')^2 \rceil \leq wp\ (x' = C\ \&\ G)\ \lceil \lambda s.\ r^2 = (s\$\restriction_V ''x'')^2 + (s\$\restriction_V ''y'')^2 \rceil$
unfolding *wp-diff-inv* **using** *circular-motion-invariant* **by** *auto*

— Verified with the flow.

abbreviation *circular-motion-flow t s* $\equiv$
$(\chi\ i.\ if\ i=\restriction_V ''x''\ then\ s\$(\restriction_V ''x'') \cdot cos\ t + s\$(\restriction_V ''y'') \cdot sin\ t$
$else\ -\ s\$(\restriction_V ''x'') \cdot sin\ t + s\$(\restriction_V ''y'') \cdot cos\ t)$

notation *circular-motion-flow* (φ_C)

lemma *picard-lindeloef-circ-motion*: *picard-lindeloef* $(\lambda t.\ C)$ *UNIV UNIV 0*
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac* $x=1$ **in** *exI*, *clarsimp*, *rule-tac* $x=1$ **in** *exI*)
by(*simp add*: *dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject power2-commute*)

lemma *local-flow-circ-motion*: *local-flow C UNIV UNIV* φ_C
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
apply(*rule picard-lindeloef-circ-motion*, *simp-all add*: *vec-eq-iff*)
apply(*rule has-vderiv-on-vec-lambda*, *clarify*)
apply(*case-tac* $i = \restriction_V ''x''$, *simp*)
apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)
apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)

using *program-vars-exhaust* **by** *force*

lemma *circular-motion*:
$\lceil\lambda s.\ r^2 = (s\$\restriction_V\ ''x'')^2 + (s\$\restriction_V\ ''y'')^2\rceil \leq wp\ (x\acute{}= C\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$\restriction_V\ ''x'')^2 + (s\$\restriction_V\ ''y'')^2\rceil$
by (*subst local-flow.wp-g-orbit*[*OF local-flow-circ-motion*]) (*auto simp*: *to-var-inject*)

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

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

— Verified as a linear system (using uniqueness).

abbreviation *circular-motion-sq-mtx* :: *2 sq-mtx*
where *circular-motion-sq-mtx* ≡ *sq-mtx-chi* (χ *i*. *if i=0 then* − e *1 else* e *0*)

abbreviation *circular-motion-mtx-flow* :: *real* ⇒ *real^2* ⇒ *real^2*
where *circular-motion-mtx-flow t s* ≡
(χ *i*. *if i=(0::2) then s\$0 · cos t − s\$1 · sin t else s\$0 · sin t + s\$1 · cos t*)

notation *circular-motion-sq-mtx* (C)

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

lemma *circular-motion-mtx-exp-eq*: $exp\ (t *_R C) *_V s = \varphi_C\ t\ s$
apply(*rule local-flow.eq-solution*[*OF local-flow-exp*, *symmetric*])
apply(*rule ivp-solsI*, *rule has-vderiv-on-vec-lambda*, *clarsimp*)
unfolding *sq-mtx-vec-prod-def matrix-vector-mult-def* **apply** *simp*
apply(*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)
using *exhaust-2 two-eq-zero* **by** (*force simp*: *vec-eq-iff*, *auto*)

lemma *circular-motion-sq-mtx*:
$\lceil\lambda s.\ r^2 = (s\$0)^2 + (s\$1)^2\rceil \leq wp\ (x\acute{}= ((*_V)\ C)\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$0)^2 + (s\$1)^2\rceil$
unfolding *local-flow.wp-g-orbit*[*OF local-flow-exp*] *circular-motion-mtx-exp-eq* **by** *auto*

no-notation *circular-motion-sq-mtx* (C)

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

Bouncing Ball

— Verified with differential invariants.

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

lemma [*bb-real-arith*]:

assumes $0 > g$ **and** *inv*: $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$
shows $(x::real) \leq h$
proof−
have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot h \wedge 0 > g$
using *inv* **and** ⟨$0 > g$⟩ **by** *auto*
hence *obs*:$v \cdot v = 2 \cdot g \cdot (x - h) \wedge 0 > g \wedge v \cdot v \geq 0$
using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
hence $(v \cdot v)/(2 \cdot g) = (x - h)$
by *auto*
also from *obs* **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

abbreviation *constant-acceleration-kinematics* $g\ s \equiv (\chi\ i.\ if\ i=(\restriction_V{''x''})\ then\ s\$(\restriction_V{''y''})\ else\ g)$

notation *constant-acceleration-kinematics* (K)

lemma *energy-conservation-invariant*:
fixes $g\ h :: real$
defines *dinv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\restriction_V{''x''}) - 2 \cdot g \cdot h - (s\$(\restriction_V{''y''}) \cdot s\$(\restriction_V{''y''})) = 0)$
shows *diff-invariant* $I\ (K\ g)$ *UNIV UNIV* $0\ G$
unfolding *dinv* **apply**(*rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
apply(*frule-tac* $i=\restriction_V{''y''}$ **in** *has-vderiv-on-vec-nth*)
apply(*drule-tac* $i=\restriction_V{''x''}$ **in** *has-vderiv-on-vec-nth*)
by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *bouncing-ball-invariants*:
fixes $h::real$
assumes $g < 0$ **and** $h \geq 0$
defines *diff-inv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\restriction_V{''x''}) - 2 \cdot g \cdot h - (s\$(\restriction_V{''y''}) \cdot s\$(\restriction_V{''y''})) = 0)$
shows $\lceil \lambda s.\ s\$(\restriction_V{''x''}) = h \wedge s\$(\restriction_V{''y''}) = 0 \rceil \leq$
$wp\ (((x\acute{}=K\ g\ \&\ (\lambda\ s.\ s\$(\restriction_V{''x''}) \geq 0));$
$(IF\ (\lambda\ s.\ s\$(\restriction_V{''x''}) = 0)\ THEN\ ((\restriction_V{''y''}) ::= (\lambda s.\ - s\$(\restriction_V{''y''})))\ ELSE\ Id\ FI))^*)$
$\lceil \lambda s.\ 0 \leq s\$(\restriction_V{''x''}) \wedge s\$(\restriction_V{''x''}) \leq h \rceil$
apply(*rule-tac* $I=\lceil \lambda s.\ 0 \leq s\$(\restriction_V{''x''}) \wedge I\ s\rceil$ **in** *rel-ad-mka-starI*)
using ⟨$h \geq 0$⟩ **apply**(*simp add*: *diff-inv*, *simp only*: *rel-antidomain-kleene-algebra.fbox-seq*)
apply(*subst p2r-r2p-wp*[*symmetric*, *of* (*IF - THEN - ELSE Id FI*)])
apply(*rule order.trans*[**where** $b=wp\ (x\acute{}=K\ g\ \&\ (\lambda s.\ s\$(\restriction_V{''x''}) \geq 0))\ \lceil \lambda s.\ 0 \leq s\$(\restriction_V{''x''}) \wedge I\ s\rceil$])
apply(*simp only*: *wp-g-evolution-guard*)
apply(*rule order.trans*[**where** $b=\lceil I \rceil$], *simp*)
apply(*subst wp-diff-inv*, *unfold diff-inv*)

using *energy-conservation-invariant* **apply** *force*
apply(*subst wp-trafo*, *rule rel-antidomain-kleene-algebra.fbox-iso*)
using *assms* **unfolding** *rel-antidomain-kleene-algebra.cond-def image-le-pred rel-antidomain-kleene-algebra.ads-d-def* **by**(*auto simp*: *p2r-def rel-ad-def bb-real-arith*)

— Verified with the flow.

lemma *picard-lindeloef-cnst-acc*:
fixes *g*::*real*
shows *picard-lindeloef* ($\lambda t.$ *K g*) *UNIV UNIV 0*
apply(*unfold-locales*)
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
by(*simp add*: *dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject*)

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

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

lemma *local-flow-cnst-acc*: *local-flow* (*K g*) *UNIV UNIV* (φ_K *g*)
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-cnst-acc* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*, *clarify*)
apply(*case-tac i* = $\upharpoonright_V$ $''x''$)
using *program-vars-exhaust* **by**(*auto intro*!: *poly-derivatives simp*: *to-var-inject vec-eq-iff*)

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*)
hence *obs*: $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$
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*))
thus $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
by (*simp add*: *monoid-mult-class.power2-eq-square*)
have $2 \cdot g \cdot h + (-\, ((g \cdot \tau) + v))^2 = 0$
using *obs* **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*)
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*:
  fixes *h*::*real*
  assumes $g < 0$ **and** $h \geq 0$
  defines *loop-inv*: $I \equiv (\lambda s.\ 0 \leq s\$(\upharpoonright_V{''x''}) \wedge\ 2 \cdot g \cdot s\$(\upharpoonright_V{''x''}) =$
  $2 \cdot g \cdot h + (s\$(\upharpoonright_V{''y''}) \cdot s\$(\upharpoonright_V{''y''})))$
  shows $\lceil \lambda s.\ s\$(\upharpoonright_V{''x''}) = h \wedge s\$(\upharpoonright_V{''y''}) = 0 \rceil \leq$
  wp $(((x\acute{} = K\ g\ \&\ (\lambda\ s.\ s\$(\upharpoonright_V{''x''}) \geq 0));$
  $(IF\ (\lambda\ s.\ s\$(\upharpoonright_V{''x''}) = 0)\ THEN\ ((\upharpoonright_V{''y''}) ::= (\lambda s.\ -\ s\$(\upharpoonright_V{''y''})))\ ELSE\ Id$
$FI))^*)$
  $\lceil \lambda s.\ 0 \leq s\$(\upharpoonright_V{''x''}) \wedge s\$(\upharpoonright_V{''x''}) \leq h \rceil$
 apply(*rule-tac* $I = \lceil I \rceil$ **in** *rel-ad-mka-starI*)
 using ‹$h \geq 0$› **apply**(*simp add*: *loop-inv*, *simp only*: *rel-antidomain-kleene-algebra.fbox-seq*)
  apply(*subst p2r-r2p-wp*[*symmetric*, *of* (*IF - THEN - ELSE Id FI*)])
  apply(*subst local-flow.wp-g-orbit*[*OF local-flow-cnst-acc*])
  apply(*subst wp-trafo*, *simp add*: *rel-antidomain-kleene-algebra.cond-def p2r-def*)
 apply(*simp add*: *rel-antidomain-kleene-algebra.ads-d-def rel-ad-def*)
 unfolding *loop-inv* **using** ‹$g < 0$› ‹$h \geq 0$› **by** (*auto simp*: *to-var-inject bb-real-arith*)

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

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

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

— Verified as a linear system (computing exponential).

abbreviation *constant-acceleration-kinematics-sq-mtx* :: *3 sq-mtx*
where *constant-acceleration-kinematics-sq-mtx* ≡
sq-mtx-chi (χ *i*::*3*. *if i=0 then* e *1 else if i=1 then* e *2 else 0*)

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

lemma *const-acc-mtx-pow2*: K^2 = *sq-mtx-chi* (χ *i*. *if i=0 then* e *2 else 0*)
unfolding *monoid-mult-class.power2-eq-square times-sq-mtx-def*
by (*simp add*: *sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def*)

lemma *const-acc-mtx-powN*: *n* > *2* ⟹ ($\tau *_R K$)ˆ*n* = *0*
apply(*induct n*, *simp*, *case-tac n* ≤ *2*)
apply(*simp only*: *le-less-Suc-eq power-class.power.simps*(*2*), *simp*)
by (*auto simp*: *const-acc-mtx-pow2 sq-mtx-chi-inject vec-eq-iff*
times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def)

lemma *exp-cnst-acc-sq-mtx*: *exp* ($\tau *_R K$) = (($\tau *_R K)^2/_R$ *2*) + ($\tau *_R K$) + *1*
unfolding *exp-def* **apply**(*subst suminf-eq-sum*[*of 2*])
using *const-acc-mtx-powN* **by** (*simp-all add*: *numeral-2-eq-2*)

lemma *exp-cnst-acc-sq-mtx-simps*:
exp ($\tau *_R K$) $$ *0* $ *0* = *1* *exp* ($\tau *_R K$) $$ *0* $ *1* = τ *exp* ($\tau *_R K$) $$ *0* $ *2* = τˆ*2*/*2*
exp ($\tau *_R K$) $$ *1* $ *0* = *0* *exp* ($\tau *_R K$) $$ *1* $ *1* = *1* *exp* ($\tau *_R K$) $$ *1* $ *2* = τ
exp ($\tau *_R K$) $$ *2* $ *0* = *0* *exp* ($\tau *_R K$) $$ *2* $ *1* = *0* *exp* ($\tau *_R K$) $$ *2* $ *2* = *1*
unfolding *exp-cnst-acc-sq-mtx scaleR-power const-acc-mtx-pow2*
by (*auto simp*: *plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def*
mat-def scaleR-vec-def axis-def plus-vec-def)

lemma *bouncing-ball-K*:
⌈λ*s*. *0* ≤ *s* $ *0* ∧ *s* $ *0* = *h* ∧ *s* $ *1* = *0* ∧ *0* > *s* $ *2*⌉ ⊆
wp (((*x*´=($*_V$) *K* & (λ *s*. *s* $ *0* ≥ *0*));
(*IF* (λ *s*. *s* $ *0* = *0*) *THEN* (*1* ::= (λ*s*. − *s* $ *1*)) *ELSE Id FI*))*)
⌈λ*s*. *0* ≤ *s* $ *0* ∧ *s* $ *0* ≤ *h*⌉
apply(*rule-tac I*=⌈λ*s*. *0* ≤ *s*$*0* ∧ *0* > *s*$*2* ∧
2 · *s*$*2* · *s*$*0* = *2* · *s*$*2* · *h* + (*s*$*1* · *s*$*1*)⌉ **in** *rel-ad-mka-starI*)
apply(*simp*, *simp only*: *rel-antidomain-kleene-algebra.fbox-seq*)
apply(*subst p2r-r2p-wp*[*symmetric*, *of* (*IF* (λ*s*. *s* $ *0* = *0*) *THEN* (*1* ::= (λ*s*. − *s* $ *1*)) *ELSE Id FI*)])
apply(*subst local-flow.wp-g-orbit*[*OF local-flow-exp*])
apply(*subst rel-antidomain-kleene-algebra.fbox-cond-var*)
apply(*simp add*: *wp-rel sq-mtx-vec-prod-eq*)
apply(*simp add*: *p2r-r2p-simps*)
unfolding *UNIV-3 image-le-pred* **apply**(*simp add*: *exp-cnst-acc-sq-mtx-simps*, *safe*)
subgoal for *x* **using** *bb-real-arith*(*3*)[*of x* $ *2*]

```
    by (simp add: add.commute mult.commute)
  subgoal for x τ 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 p2r-def)

no-notation constant-acceleration-kinematics-sq-mtx (K)

end
theory kat2rel
  imports
  ../hs-prelims-dyn-sys
  ../../afpModified/VC-KAT

begin
```

Chapter 5

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 *Relation.Domain* (*r2s*)
  and *VC-KAT.gets* (- ::= - [*70*, *65*] *61*)
  and *tau* (τ)

5.1 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 $r2p\ R = (\lambda x.\ x \in Domain\ R)$.

thm *sH-H*

lemma *sH-weaken-pre*: $rel\text{-}kat.H\ \lceil P2 \rceil\ R\ \lceil Q \rceil \Longrightarrow \lceil P1 \rceil \subseteq \lceil P2 \rceil \Longrightarrow rel\text{-}kat.H\ \lceil P1 \rceil\ R\ \lceil Q \rceil$
  unfolding *sH-H* **by** *auto*

Next, we introduce assignments and compute their Hoare triple.

abbreviation *vec-upd* :: $('a\hat{\ }'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a\hat{\ }'b$
  where $vec\text{-}upd\ x\ i\ a \equiv vec\text{-}lambda\ ((vec\text{-}nth\ x)(i := a))$

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

lemma *sH-assign-iff* [*simp*]: $rel\text{-}kat.H\ \lceil P \rceil\ (x ::= e)\ \lceil Q \rceil \longleftrightarrow (\forall s.\ P\ s \longrightarrow Q\ (vec\text{-}upd\ s\ x\ (e\ s)))$
  unfolding *sH-H* **by** *simp*

Next, the Hoare rule of the composition:

lemma *sH-relcomp*: *rel-kat.H* $\lceil P \rceil$ *X* $\lceil R \rceil \Longrightarrow$ *rel-kat.H* $\lceil R \rceil$ *Y* $\lceil Q \rceil \Longrightarrow$ *rel-kat.H* $\lceil P \rceil$ $(X\ ;\ Y)$ $\lceil Q \rceil$
using *rel-kat.H-seq-swap* **by** *force*

lemma *rel-kat.H* $\lceil P \rceil$ $(X\ ;\ Y)$ $\lceil Q \rceil$ = *rel-kat.H* $\lceil P \rceil$ (X) $\{(s,s') \mid s\ s'.\ (s,s') \in Y \longrightarrow Q\ s'\}$
unfolding *rel-kat.H-def* **apply**(*auto simp*: *subset-eq p2r-def Int-def*)
oops

There is also already an implementation of the conditional operator *if p then x else y fi* $= t\ p \cdot x + !p \cdot y$ and its Hoare triple rule: $[\![$*PRE P* $\sqcap$ *T X POST Q*; *PRE P* $\sqcap -$ *T Y POST Q*$]\!] \Longrightarrow$ *PRE P* (*IF T THEN X ELSE Y FI*) *POST Q*.

Finally, we add a Hoare triple rule for a simple finite iteration.

lemma (**in** *kat*) *H-star-self*: $H\ (t\ i)\ x\ i \Longrightarrow H\ (t\ i)\ (x^\star)\ i$
unfolding *H-def* **by** (*simp add*: *local.star-sim2*)

lemma (**in** *kat*) *H-star*:
assumes $t\ p \leq t\ i$ **and** $H\ (t\ i)\ x\ i$ **and** $t\ i \leq t\ q$
shows $H\ (t\ p)\ (x^\star)\ q$
proof−
have $H\ (t\ i)\ (x^\star)\ i$
using *assms*(*2*) *H-star-self* **by** *blast*
hence $H\ (t\ p)\ (x^\star)\ i$
apply(*simp add*: *H-def*)
using *assms*(*1*) *local.phl-cons1* **by** *blast*
thus *?thesis*
unfolding *H-def* **using** *assms*(*3*) *local.phl-cons2* **by** *blast*
qed

lemma *sH-star*:
assumes $\lceil P \rceil \subseteq \lceil I \rceil$ **and** *rel-kat.H* $\lceil I \rceil$ *R* $\lceil I \rceil$ **and** $\lceil I \rceil \subseteq \lceil Q \rceil$
shows *rel-kat.H* $\lceil P \rceil$ (R^*) $\lceil Q \rceil$
using *rel-kat.H-star*[*of* $\lceil P \rceil$ $\lceil I \rceil$ *R* $\lceil Q \rceil$] *assms* **by** *auto*

5.2 Verification of hybrid programs

abbreviation *g-evolution* ::(('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ *real set* ⇒ '*a set* ⇒ *real* ⇒ '*a rel* ((*1x*´=- & - *on* - - @ -))
where (*x*´=*f* & *G on T S* @ t_0) ≡ $\{(s,s') \mid s\ s'.\ s' \in$ *g-orbital f G T S* $t_0\ s\}$

abbreviation *g-evol* ::(('*a*::*banach*)⇒'*a*) ⇒ '*a pred* ⇒ '*a rel* ((*1x*´=- & -))
where (*x*´=*f* & *G*) ≡ (*x*´=*f* & *G on UNIV UNIV* @ *0*)

5.2.1 Verification by providing solutions

lemma *sH-g-evolution*:
assumes $\forall s.\ P\ s \longrightarrow (\forall X{\in}ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t))$
shows $rel\text{-}kat.H\ \lceil P\rceil\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
using *assms* **unfolding** *g-orbital-eq*(*1*) *sH-H image-le-pred* **by** *auto*

context *local-flow*
begin

lemma *sH-g-orbit*:
assumes $\forall s.\ s \in S \longrightarrow P\ s \longrightarrow (\forall t{\in}T.\ (\forall \tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))$
shows $rel\text{-}kat.H\ \lceil P\rceil\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ 0)\ \lceil Q\rceil$
apply(*rule sH-g-evolution*)
using *assms* **apply**(*safe*, *simp add*: *ivp-sols-def*, *clarsimp*)
apply(*erule-tac x=X 0* **in** *allE*, *erule impE*)
using *init-time* **apply** *force*
apply(*subgoal-tac* $\forall \tau{\in}down\ T\ t.\ X\ \tau = \varphi\ \tau\ (X\ 0)$, *simp-all*, *clarsimp*)
apply(*subst eq-solution*, *simp-all add*: *ivp-sols-def*)
using *init-time* **by** *auto*

lemma *sH-orbit*:
assumes $\forall s.\ s \in S \longrightarrow P\ s \longrightarrow (\forall\ t \in T.\ Q\ (\varphi\ t\ s))$
shows $rel\text{-}kat.H\ \lceil P\rceil\ (\{(s,s')\mid s\ s'.\ s' \in \gamma^{\varphi}\ s\})\ \lceil Q\rceil$
unfolding *orbit-def* **apply**(*rule sH-g-orbit*)
using *assms* **by** *auto*

end

5.2.2 Verification with differential invariants

lemma *sH-g-evolution-guard*:
assumes $R = (\lambda s.\ G\ s \wedge Q\ s)$ **and** $rel\text{-}kat.H\ \lceil P\rceil\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
shows $rel\text{-}kat.H\ \lceil P\rceil\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil R\rceil$
using *assms* **unfolding** *g-orbital-eq sH-H ivp-sols-def* **by** *auto*

lemma *sH-g-evolution-inv*:
assumes $\lceil P\rceil \le \lceil I\rceil$ **and** $rel\text{-}kat.H\ \lceil I\rceil\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$ **and** $\lceil I\rceil \le \lceil Q\rceil$
shows $rel\text{-}kat.H\ \lceil P\rceil\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
using *assms*(*1*) **apply**(*rule-tac* $p'{=}\lceil I\rceil$ **in** *rel-kat.H-cons-1*, *simp*)
using *assms*(*3*) **apply**(*rule-tac* $q'{=}\lceil I\rceil$ **in** *rel-kat.H-cons-2*, *simp*)
using *assms*(*2*) **by** *simp*

lemma *sH-diff-inv*: $rel\text{-}kat.H\ \lceil I\rceil\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil = diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G$
unfolding *diff-invariant-eq sH-H g-orbital-eq image-le-pred* **by** *auto*

5.2.3 Derivation of the rules of dL

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.

lemma *diff-solve-axiom*:
  fixes *c*::*'a*::{*heine-borel*, *banach*}
  assumes *0* ∈ *T* **and** *is-interval T open T*
    and ∀ *s*. *P s* ⟶ (∀ *t*∈*T*. (𝒫 (λ *t*. *s* + *t* $*_R$ *c*) (*down T t*) ⊆ {*s*. *G s*}) ⟶ *Q* (*s* + *t* $*_R$ *c*))
  shows *rel-kat.H* ⌈*P*⌉ (*x´*=(λ*s*. *c*) & *G on T UNIV* @ *0*) ⌈*Q*⌉
  apply(*subst local-flow.sH-g-orbit*[**where** *f*=λ*s*. *c* **and** φ=(λ *t x*. *x* + *t* $*_R$ *c*)])
  using *line-is-local-flow assms* **unfolding** *image-le-pred* **by** *auto*

lemma *diff-solve-rule*:
  assumes *local-flow f T UNIV* φ
    and ∀ *s*. *P s* ⟶ (∀ *t*∈*T*. (𝒫 (λ*t*. φ *t s*) (*down T t*) ⊆ {*s*. *G s*}) ⟶ *Q* (φ *t s*))
  shows *rel-kat.H* ⌈*P*⌉ (*x´*=*f* & *G on T UNIV* @ *0*) ⌈*Q*⌉
  using *assms* **by**(*subst local-flow.sH-g-orbit*, *auto*)

lemma *diff-weak-rule*:
  assumes ⌈*G*⌉ ≤ ⌈*Q*⌉
  shows *rel-kat.H* ⌈*P*⌉ (*x´*=*f* & *G on T S* @ t_0) ⌈*Q*⌉
  using *assms* **unfolding** *g-orbital-eq sH-H ivp-sols-def* **by** *auto*

lemma *diff-cut-rule*:
  assumes *Thyp*: *is-interval T* t_0 ∈ *T*
    and *wp-C*:*rel-kat.H* ⌈*P*⌉ (*x´*=*f* & *G on T S* @ t_0) ⌈*C*⌉
    and *wp-Q*:*rel-kat.H* ⌈*P*⌉ (*x´*=*f* & (λ *s*. *G s* ∧ *C s*) *on T S* @ t_0) ⌈*Q*⌉
  shows *rel-kat.H* ⌈*P*⌉ (*x´*=*f* & *G on T S* @ t_0) ⌈*Q*⌉
proof(*subst sH-H*, *simp add*: *g-orbital-eq p2r-def image-le-pred*, *clarsimp*)
  fix *t*::*real* **and** *X*::*real* ⇒ *'a* **and** *s* **assume** *P s* **and** *t* ∈ *T*
    and *x-ivp*:*X* ∈ *ivp-sols* (λ*t*. *f*) *T S* t_0 *s*
    and *guard-x*:∀ *x*. *x* ∈ *T* ∧ *x* ≤ *t* ⟶ *G* (*X x*)
  have ∀ *t*∈(*down T t*). *X t* ∈ *g-orbital f G T S* t_0 *s*
    using *g-orbitalI*[*OF x-ivp*] *guard-x* **unfolding** *image-le-pred* **by** *auto*
  hence ∀ *t*∈(*down T t*). *C* (*X t*)
    using *wp-C* ‹*P s*› **by** (*subst* (*asm*) *sH-H*, *auto*)
  hence *X t* ∈ *g-orbital f* (λ*s*. *G s* ∧ *C s*) *T S* t_0 *s*
    using *guard-x* ‹*t* ∈ *T*› **by** (*auto intro*!: *g-orbitalI x-ivp*)
  thus *Q* (*X t*)
    using ‹*P s*› *wp-Q* **by** (*subst* (*asm*) *sH-H*) *auto*
qed

end
theory *kat2rel-examples*
  imports ../*hs-prelims-matrices kat2rel*

begin

5.2.4 Examples

Preliminary preparation for the examples.

no-notation *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor$-$\rfloor$)

— Finite set of program variables.

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

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

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* {''*x*'',''*y*''}])
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-univ-eq*: (*UNIV*::*program-vars set*) = {$\restriction_V$''*x*'', $\restriction_V$''*y*''}
apply *auto* **by** (*metis to-str to-str-inverse insertE singletonD*)

lemma *program-vars-exhaust*: *x* = $\restriction_V$''*x*'' $\vee$ *x* = $\restriction_V$''*y*''
using *program-vars-univ-eq* **by** *auto*

— Alternative to the finite set of program variables.

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

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

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

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

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

lemma *sum-axis-UNIV-3*[*simp*]: ($\sum j \in$(*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*

Circular Motion

— Verified with differential invariants.

abbreviation *circular-motion-kinematics* :: *real*^*program-vars* ⇒ *real*^*program-vars*

where *circular-motion-kinematics s* ≡ (*χ i*. *if i*=($\restriction_V$''*x*'') *then s*\$($\restriction_V$''*y*'') *else* −*s*\$($\restriction_V$''*x*''))

notation *circular-motion-kinematics* (*C*)

lemma *circular-motion-invariant*:
diff-invariant (*λs*. (*r*::*real*)^2 = (*s*\$($\restriction_V$''*x*''))^2 + (*s*\$($\restriction_V$''*y*''))^2) *C UNIV UNIV 0 G*
apply(*rule-tac diff-invariant-rules*, *clarsimp*, *simp*, *clarsimp*)
apply(*frule-tac i*=$\restriction_V$''*x*'' **in** *has-vderiv-on-vec-nth*, *drule-tac i*=$\restriction_V$''*y*'' **in** *has-vderiv-on-vec-nth*)
by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *circular-motion-invariants*: *rel-kat.H*
⌈*λs*. *r*^{2} = (*s*\$($\restriction_V$''*x*''))^2 + (*s*\$($\restriction_V$''*y*''))^2⌉ (*x*´= *C* & *G*) ⌈*λs*. *r*^{2} = (*s*\$($\restriction_V$''*x*''))^2 + (*s*\$($\restriction_V$''*y*''))^2⌉
unfolding *sH-diff-inv* **using** *circular-motion-invariant* **by** *auto*

— Verified with the flow.

abbreviation *circular-motion-flow τ s* ≡
(*χ i*. *if i*=$\restriction_V$''*x*'' *then s*\$($\restriction_V$''*x*'') · *cos τ* + *s*\$($\restriction_V$''*y*'') · *sin τ*
else − *s*\$($\restriction_V$''*x*'') · *sin τ* + *s*\$($\restriction_V$''*y*'') · *cos τ*)

notation *circular-motion-flow* (φ_C)

lemma *picard-lindeloef-circ-motion*: *picard-lindeloef* (*λt*. *C*) *UNIV UNIV 0*
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x*=*1* **in** *exI*, *clarsimp*, *rule-tac x*=*1* **in** *exI*)
by(*simp add*: *dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject power2-commute*)

lemma *local-flow-circ-motion*: *local-flow C UNIV UNIV* φ_C
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
apply(*rule picard-lindeloef-circ-motion*, *simp-all add*: *vec-eq-iff*)
apply(*rule has-vderiv-on-vec-lambda*, *clarify*)

apply(*case-tac* $i = \upharpoonright_V ''x''$, *simp*)
apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)
apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)
using *program-vars-exhaust* **by** *force*

lemma *circular-motion*:*rel-kat.H*
$\lceil \lambda s.\ r^2 = (s\$(\upharpoonright_V ''x''))^2 + (s\$(\upharpoonright_V ''y''))^2 \rceil$ $(x\acute{} = C\ \&\ G)$ $\lceil \lambda s.\ r^2 = (s\$(\upharpoonright_V ''x''))^2 + (s\$(\upharpoonright_V ''y''))^2 \rceil$
by (*rule local-flow.sH-g-orbit*[*OF local-flow-circ-motion*]) (*auto simp*: *to-var-inject*)

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

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

— Verified as a linear system (using uniqueness).

abbreviation *circular-motion-sq-mtx* :: *2 sq-mtx*
where *circular-motion-sq-mtx* $\equiv$ *sq-mtx-chi* (χ *i. if i=0 then* $-$ e *1 else* e *0*)

abbreviation *circular-motion-mtx-flow* :: *real* $\Rightarrow$ *real^2* $\Rightarrow$ *real^2*
where *circular-motion-mtx-flow* τ s $\equiv$
(χ *i. if i=(0::2) then* $s\$0 \cdot \cos\tau - s\$1 \cdot \sin\tau$ *else* $s\$0 \cdot \sin\tau + s\$1 \cdot \cos\tau$)

notation *circular-motion-sq-mtx* (C)

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

lemma *circular-motion-mtx-exp-eq*: *exp* $(\tau *_R C) *_V s = \varphi_C\ \tau\ s$
apply(*rule local-flow.eq-solution*[*OF local-flow-exp*, *symmetric*])
apply(*rule ivp-solsI*, *rule has-vderiv-on-vec-lambda*, *clarsimp*)
unfolding *sq-mtx-vec-prod-def matrix-vector-mult-def* **apply** *simp*
apply(*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)
using *exhaust-2 two-eq-zero* **by** (*force simp*: *vec-eq-iff*, *auto*)

lemma *circular-motion-sq-mtx*: *rel-kat.H*
$\lceil \lambda s.\ r^2 = (s\$0)^2 + (s\$1)^2 \rceil$ $(x\acute{} = ((*_V)\ C)\ \&\ G)$ $\lceil \lambda s.\ r^2 = (s\$0)^2 + (s\$1)^2 \rceil$
apply(*rule local-flow.sH-g-orbit*[*OF local-flow-exp*])
unfolding *circular-motion-mtx-exp-eq* **by** *auto*

no-notation *circular-motion-sq-mtx* (C)

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

Bouncing Ball

— Verified with differential invariants.

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

lemma [*bb-real-arith*]:
  assumes $0 > g$ **and** *inv*: $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$
  shows $(x::real) \leq h$
proof−
  have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot h \wedge 0 > g$
    using *inv* **and** $\langle 0 > g \rangle$ **by** *auto*
  hence *obs*:$v \cdot v = 2 \cdot g \cdot (x - h) \wedge 0 > g \wedge v \cdot v \geq 0$
    using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
  hence $(v \cdot v)/(2 \cdot g) = (x - h)$
    by *auto*
  also from *obs* **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

abbreviation *constant-acceleration-kinematics* $g\ s \equiv (\chi\ i.\ if\ i=(\upharpoonright_V{''x''})\ then\ s\$(\upharpoonright_V{''y''})\ else\ g)$

notation *constant-acceleration-kinematics* (K)

lemma *energy-conservation-invariant*:
  fixes $g\ h :: real$
  defines *dinv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\upharpoonright_V{''x''}) - 2 \cdot g \cdot h - (s\$(\upharpoonright_V{''y''}) \cdot s\$(\upharpoonright_V{''y''})) = 0)$
  shows *diff-invariant* $I\ (K\ g)$ *UNIV UNIV 0 G*
  unfolding *dinv* **apply**(*rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
  apply(*frule-tac* $i=\upharpoonright_V{''y''}$ **in** *has-vderiv-on-vec-nth*)
  apply(*drule-tac* $i=\upharpoonright_V{''x''}$ **in** *has-vderiv-on-vec-nth*)
  by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *bouncing-ball-invariants*:
  fixes $h::real$
  assumes $g < 0$ **and** $h \geq 0$
  defines *diff-inv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\upharpoonright_V{''x''}) - 2 \cdot g \cdot h - (s\$(\upharpoonright_V{''y''}) \cdot s\$(\upharpoonright_V{''y''})) = 0)$
  shows *rel-kat.H*
  $\lceil \lambda s.\ s\$(\upharpoonright_V{''x''}) = h \wedge s\$(\upharpoonright_V{''y''}) = 0 \rceil$
  $(((x\acute{}=K\ g\ \&\ (\lambda\ s.\ s\$(\upharpoonright_V{''x''}) \geq 0));$
  $(IF\ (\lambda\ s.\ s\$(\upharpoonright_V{''x''}) = 0)\ THEN\ ((\upharpoonright_V{''y''}) ::= (\lambda s.\ - s\$(\upharpoonright_V{''y''})))\ ELSE\ Id\ FI))^*)$
  $\lceil \lambda s.\ 0 \leq s\$(\upharpoonright_V{''x''}) \wedge s\$(\upharpoonright_V{''x''}) \leq h \rceil$
  apply(*rule sH-star*[*of* - $\lambda s.\ 0 \leq s\$(\upharpoonright_V{''x''}) \wedge I\ s$])
  using $\langle h \geq 0 \rangle$ **apply**(*simp add*: *diff-inv*)
   apply(*rule sH-relcomp*[**where** $R=\lambda s.\ 0 \leq s\$(\upharpoonright_V{''x''}) \wedge I\ s$])
    apply(*rule sH-g-evolution-guard*, *simp*)
    apply(*rule-tac* $p'=\lceil I \rceil$ **in** *rel-kat.H-cons-1*, *simp*)

apply(*unfold diff-inv*, *subst sH-diff-inv*)
using *energy-conservation-invariant* **apply** *force*
apply(*rule sH-cond*, *subst sH-assign-iff*, *force simp*: *bb-real-arith*)
using *assms* **by** (*subst sH-H*, *simp-all*) (*force simp*: *bb-real-arith*)

— Verified with the flow.

lemma *picard-lindeloef-cnst-acc*:
fixes *g*::*real*
shows *picard-lindeloef* ($\lambda t.$ *K g*) *UNIV UNIV 0*
apply(*unfold-locales*)
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x=1/2* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
by(*simp add*: *dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject*)

abbreviation *constant-acceleration-kinematics-flow* $g\ \tau\ s \equiv$
$(\chi\ i.\ \mathit{if}\ i{=}(\upharpoonright_V\ ''x'')\ \mathit{then}\ g \cdot \tau\ \hat{}\ 2/2 + s\ \$\ (\upharpoonright_V\ ''y'') \cdot \tau + s\ \$\ (\upharpoonright_V\ ''x'')$
$\mathit{else}\ g \cdot \tau + s\ \$\ (\upharpoonright_V\ ''y''))$

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

lemma *local-flow-cnst-acc*: *local-flow* (*K g*) *UNIV UNIV* (φ_K *g*)
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-cnst-acc* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*, *clarify*)
apply(*case-tac* $i = \upharpoonright_V\ ''x''$)
using *program-vars-exhaust* **by**(*auto intro*!: *poly-derivatives simp*: *to-var-inject vec-eq-iff*)

lemma [*bb-real-arith*]:
assumes *invar*: $2 \cdot g \cdot x = 2 \cdot g \cdot h + v \cdot v$
and *pos*: $g \cdot \tau^2\ /\ 2 + v \cdot \tau + (x{::}\mathit{real}) = 0$
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*)
hence *obs*: $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$
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*))
thus $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
by (*simp add*: *monoid-mult-class.power2-eq-square*)
have $2 \cdot g \cdot h + (-\ ((g \cdot \tau) + v))^2 = 0$

using *obs* **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*)
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*:
fixes *h*::*real*
assumes $g < 0$ **and** $h \geq 0$
defines *loop-inv*: $I \equiv (\lambda s.\ 0 \leq s\$(\upharpoonright_V{''x''}) \wedge\ 2 \cdot g \cdot s\$(\upharpoonright_V{''x''}) =$
$2 \cdot g \cdot h + (s\$(\upharpoonright_V{''y''}) \cdot s\$(\upharpoonright_V{''y''})))$
shows *rel-kat.H*
$\lceil \lambda s.\ s\$(\upharpoonright_V{''x''}) = h \wedge s\$(\upharpoonright_V{''y''}) = 0 \rceil$
$(((x' {=} K\ g\ \&\ (\lambda\ s.\ s\$(\upharpoonright_V{''x''}) \geq 0));$
$(IF\ (\lambda\ s.\ s\$(\upharpoonright_V{''x''}) = 0)\ THEN\ ((\upharpoonright_V{''y''}) ::= (\lambda s.\ -\ s\$(\upharpoonright_V{''y''})))\ ELSE\ Id$
$FI))^*)$
$\lceil \lambda s.\ 0 \leq s\$(\upharpoonright_V{''x''}) \wedge s\$(\upharpoonright_V{''x''}) \leq h \rceil$
apply(*rule sH-star*[*of - I*])
using ‹$h \geq 0$› **apply**(*simp add*: *loop-inv*)
apply(*rule sH-relcomp*[**where** *R*=*I*])
apply(*rule local-flow.sH-g-orbit*[*OF local-flow-cnst-acc*])
apply(*simp add*: *loop-inv to-var-inject*)
apply(*force simp*: *bb-real-arith*)
apply(*rule sH-cond*, *subst sH-assign-iff*)
using *assms* **by**(*auto simp*: *sH-H bb-real-arith*)

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

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

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

— Verified as a linear system (computing exponential).

abbreviation *constant-acceleration-kinematics-sq-mtx* :: *3 sq-mtx*
where *constant-acceleration-kinematics-sq-mtx* ≡
sq-mtx-chi (χ *i::3. if i=0 then* e *1 else if i=1 then* e *2 else 0*)

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

lemma *const-acc-mtx-pow2*: K^2 = *sq-mtx-chi* (χ *i. if i=0 then* e *2 else 0*)
unfolding *monoid-mult-class.power2-eq-square times-sq-mtx-def*
by (*simp add*: *sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def*)

lemma *const-acc-mtx-powN*: $m > 2 \Longrightarrow (\tau *_R K)\hat{\ }m = 0$
apply(*induct m*, *simp*, *case-tac* $m \leq 2$)
apply(*simp only*: *le-less-Suc-eq power-class.power.simps*(*2*), *simp*)
by (*auto simp*: *const-acc-mtx-pow2 sq-mtx-chi-inject vec-eq-iff*
times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def)

lemma *exp-cnst-acc-sq-mtx*: *exp* $(\tau *_R K) = ((\tau *_R K)^2/_R\ 2) + (\tau *_R K) + 1$
unfolding *exp-def* **apply**(*subst suminf-eq-sum*[*of 2*])
using *const-acc-mtx-powN* **by** (*simp-all add*: *numeral-2-eq-2*)

lemma *exp-cnst-acc-sq-mtx-simps*:
exp $(\tau *_R K)$ \$\$ *0* \$ *0* = *1* *exp* $(\tau *_R K)$ \$\$ *0* \$ *1* = τ *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*
= τ
exp $(\tau *_R K)$ \$\$ *2* \$ *0* = *0* *exp* $(\tau *_R K)$ \$\$ *2* \$ *1* = *0* *exp* $(\tau *_R K)$ \$\$ *2* \$ *2*
= *1*
unfolding *exp-cnst-acc-sq-mtx scaleR-power const-acc-mtx-pow2*
by (*auto simp*: *plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def*
mat-def scaleR-vec-def axis-def plus-vec-def)

lemma *bouncing-ball-K*: *rel-kat.H*
$\lceil\lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 = h \wedge s\ \$\ 1 = 0 \wedge 0 > s\ \$\ 2\rceil$
$(((x' {=} (*_V)\ K\ \&\ (\lambda\ s.\ s\ \$\ 0 \geq 0));$
$(IF\ (\lambda\ s.\ s\ \$\ 0 = 0)\ THEN\ (1 ::= (\lambda s.\ -\ s\ \$\ 1))\ ELSE\ Id\ FI))^*)$
$\lceil\lambda s.\ 0 \leq s\ \$\ 0 \wedge s\ \$\ 0 \leq h\rceil$
apply(*rule sH-star* [*of* - $\lambda s.\ 0 \leq s\$0 \wedge 0 > s\$2 \wedge\ 2 \cdot s\$2 \cdot s\$0 = 2 \cdot s\$2 \cdot h$
$+ (s\$1 \cdot s\$1)$], *simp*)
apply(*rule sH-relcomp*[**where** $R{=}\lambda s.\ 0 \leq s\$0 \wedge 0 > s\$2 \wedge\ 2 \cdot s\$2 \cdot s\$0 =$
$2 \cdot s\$2 \cdot h + (s\$1 \cdot s\$1)$])
apply(*subst local-flow.sH-g-orbit*[*OF local-flow-exp*], *simp-all add*: *sq-mtx-vec-prod-eq*)
unfolding *UNIV-3 image-le-pred*
apply(*simp add*: *exp-cnst-acc-sq-mtx-simps field-simps monoid-mult-class.power2-eq-square*)
by (*auto simp*: *bb-real-arith sH-H*)

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

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

begin

Chapter 6

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* ($\lfloor$-$\rfloor$)
  and *bres* (**infixr** $\rightarrow$ *60*)

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

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

6.1 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.

declare *Abs-nd-fun-inverse* [*simp*]

— Analog of already existing $(\bigwedge x.\ f\ x = g\ x) \Longrightarrow f = g$.

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*)

lemma *nd-fun-eq-iff*: $(\forall x.\ (f_{\bullet})\ x = (g_{\bullet})\ x) = (f = g)$

by (*auto simp*: *nd-fun-ext*)

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

lift-definition *antidomain-op-nd-fun* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun*
is $\lambda f.\ (\lambda x.\ \mathit{if}\ ((f_\bullet)\ x = \{\})\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})^\bullet$.

lift-definition *zero-nd-fun* :: $'a$ *nd-fun*
is $\zeta^\bullet$.

lift-definition *star-nd-fun* :: $'a$ *nd-fun* $\Rightarrow$ $'a$ *nd-fun*
is $\lambda(f{::}'a\ \mathit{nd\text{-}fun}).\mathit{qstar}\ f$.

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

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

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

lemma *nd-fun-leq*[*nd-fun-aka*]: $((x{::}'a\ \mathit{nd\text{-}fun}) \leq y) = (x + y = y)$
and *nd-fun-leq-add*[*nd-fun-aka*]: $z \cdot x \leq z \cdot (x + y)$
apply(*transfer*)
apply(*metis* (*no-types*, *lifting*) *less-eq-nd-fun.transfer sup.absorb-iff2 sup-nd-fun.transfer*)
by(*transfer*, *simp add*: *kcomp-isol*)

lemma *nd-fun-ad-zero*[*nd-fun-aka*]: $ad\ (x{::}'a\ \mathit{nd\text{-}fun}) \cdot x = 0$
and *nd-fun-ad*[*nd-fun-aka*]: $ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$
and *nd-fun-ad-one*[*nd-fun-aka*]: $ad\ (ad\ x) + ad\ x = 1$
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*]: $1 + (x{::}'a\ \mathit{nd\text{-}fun}) \cdot x^\star \leq x^\star$
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 obtain a relationship between both of these.

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

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*[*simp*]: $(\lceil P\rceil = \lceil Q\rceil) = (P = Q)$
by(*subst eq-iff*, *auto simp*: *fun-eq-iff*)

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* :: $'a\ nd\text{-}fun \Rightarrow\ 'a \Rightarrow bool$ $((1\lfloor\text{-}\rfloor))$
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*)

6.2 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 $wp\ f \equiv fbox\ (f::'a\ nd\text{-}fun)$

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

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

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

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

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

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

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 $ffb\text{-}is\text{-}wp$: $fb_{\mathcal{F}}\ (F_\bullet)\ \{x.\ P\ x\} = \{s.\ \lfloor wp\ F\ \lceil P \rceil \rfloor\ s\}$
  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* :: $('a\hat{}'b) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'a\hat{}'b$
where *vec-upd x i a* $\equiv$ *vec-lambda* ((*vec-nth x*)(*i* := *a*))

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

lemma *wp-assign*[*simp*]: *wp* (x ::= e) $\lceil Q \rceil = \lceil \lambda s.\ Q$ (*vec-upd s x* (*e 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** *if p then x else y fi* $= d\ p \cdot x + ad\ p \cdot y$

lemma *fbox-export1*: $ad\ p + |x]\ q = |d\ p \cdot x]\ q$
using *a-d-add-closure fbox-def fbox-mult*
by (*metis* (*mono-tags*, *lifting*) *a-de-morgan ads-d-def*)

lemma *fbox-cond-var*[*simp*]: $|$*if p then x else y fi*$]\ q = (ad\ p + |x]\ q) \cdot (d\ p + |y]\ q)$
using *cond-def a-closure′ ads-d-def ans-d-def fbox-add2 fbox-export1* **by** (*metis* (*no-types*, *lifting*))

abbreviation *cond-sugar* :: $'a$ *pred* $\Rightarrow 'a$ *nd-fun* $\Rightarrow 'a$ *nd-fun* $\Rightarrow 'a$ *nd-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 $\lceil \lambda s.\ P\ s \wedge T\ s \rceil \leq wp\ X\ \lceil Q \rceil$
and $\lceil \lambda s.\ P\ s \wedge \neg\ T\ s \rceil \leq wp\ Y\ \lceil Q \rceil$
shows $\lceil P \rceil \leq wp$ (*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 $d\ p \leq d\ i$ **and** $d\ i \leq |x]\ i$ **and** $d\ i \leq d\ q$
  shows $d\ p \leq |x^\star]\ q$
  by (*meson assms local.dual-order.trans local.fbox-iso local.fbox-star-induct-var*)

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

lemma *nd-fun-top-ads-d*:$(x::'a\ nd\text{-}fun) \leq 1 \Longrightarrow d\ x = x$
  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 $P \leq I$ **and** $I \leq wp\ F\ I$ **and** $I \leq Q$
  shows $P \leq wp\ (qstar\ F)\ Q$
proof−
  have $P \leq 1$
    using *assms*(*1*,*2*) **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*)
  hence $d\ P \leq d\ I \wedge d\ I \leq wp\ F\ I \wedge d\ I \leq d\ Q$
    using *assms* **by** (*metis* (*no-types*) *ads-d-mono assms*)
  hence $d\ P \leq wp\ (F^\star)\ Q$
    by(*simp add*: *fbox-starI*[*of - I*])
  thus $P \leq wp\ (qstar\ F)\ Q$
    using ‹$d\ P = P$› **by** (*transfer*, *simp*)
qed

6.3 Verification of hybrid programs

abbreviation *g-evolution* ::$(('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow real\ set \Rightarrow 'a\ set \Rightarrow real \Rightarrow 'a\ nd\text{-}fun$ ((*1x´=- & - on - - @ -*))
  where ($x´=f$ & G *on* $T\ S$ @ t_0) $\equiv$ ($\lambda\ s.\ g\text{-}orbital\ f\ G\ T\ S\ t_0\ s)^\bullet$

abbreviation *g-evol* ::$(('a::banach) \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow 'a\ nd\text{-}fun$ ((*1x´=- & -*))
  where ($x´=f$ & G) $\equiv$ ($x´=f$ & G *on UNIV UNIV* @ *0*)

6.3.1 Verification by providing solutions

lemma *wp-g-evolution*: *wp* ($x´=f$ & G *on* $T\ S$ @ t_0) $\lceil Q \rceil$=
  $\lceil \lambda\ s.\ \forall X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s.\ \forall t \in T.\ (\forall \tau \in down\ T\ t.\ G\ (X\ \tau)) \longrightarrow Q\ (X\ t) \rceil$
  unfolding *g-orbital-eq*(*1*) *wp-nd-fun* **by** (*auto simp*: *fun-eq-iff image-le-pred*)

context *local-flow*

begin

lemma *wp-g-orbit*: $wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ 0)\ \lceil Q\rceil =$
$\lceil\lambda\ s.\ s \in S \longrightarrow (\forall\, t{\in}T.\ (\forall\,\tau{\in}down\ T\ t.\ G\ (\varphi\ \tau\ s)) \longrightarrow Q\ (\varphi\ t\ s))\rceil$
unfolding *wp-g-evolution* **apply**(*clarsimp*, *simp add*: *fun-eq-iff*, *safe*)
apply(*erule-tac* $x{=}\lambda t.\ \varphi\ t\ x$ **in** *ballE*)
using *in-ivp-sols* **apply**(*force*, *force*, *force simp*: *init-time ivp-sols-def*)
apply(*subgoal-tac* $\forall\,\tau{\in}down\ T\ t.\ X\ \tau = \varphi\ \tau\ x$, *simp-all*, *clarsimp*)
apply(*subst eq-solution*, *simp-all add*: *ivp-sols-def*)
using *init-time* **by** *auto*

lemma *wp-orbit*: $wp\ (\gamma^{\varphi\bullet})\ \lceil Q\rceil = \lceil\lambda\ s.\ s \in S \longrightarrow (\forall\ t \in T.\ Q\ (\varphi\ t\ s))\rceil$
unfolding *orbit-def wp-g-orbit* **by** *auto*

end

6.3.2 Verification with differential invariants

lemma *wp-g-evolution-guard*:
assumes $H = (\lambda s.\ G\ s \wedge Q\ s)$
shows $wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil H\rceil = wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
unfolding *wp-g-evolution* **using** *assms* **by** *auto*

lemma *wp-g-evolution-inv*:
assumes $\lceil P\rceil \le \lceil I\rceil$ **and** $\lceil I\rceil \le wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil$ **and** $\lceil I\rceil \le$ $\lceil Q\rceil$
shows $\lceil P\rceil \le wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q\rceil$
using *assms*(*1*) **apply**(*rule order.trans*)
using *assms*(*2*) **apply**(*rule order.trans*)
apply(*rule fbox-iso*)
using *assms*(*3*) **by** *auto*

lemma *wp-diff-inv*: $(\lceil I\rceil \le wp\ (x\acute{}{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil I\rceil) = diff\text{-}invariant\ I\ f\ T\ S\ t_0\ G$
unfolding *diff-invariant-eq wp-g-evolution image-le-pred* **by**(*auto simp*: *fun-eq-iff*)

6.3.3 Derivation of the rules of dL

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.

lemma *diff-solve-axiom*:
fixes $c{::}'a{::}\{heine\text{-}borel,\ banach\}$
assumes $0 \in T$ **and** *is-interval T open T*
shows $wp\ (x\acute{}{=}(\lambda s.\ c)\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q\rceil =$
$\lceil\lambda\ s.\ \forall\, t{\in}T.\ (\mathcal{P}\ (\lambda\ t.\ s + t *_R\ c)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (s + t *_R\ c)\rceil$
apply(*subst local-flow.wp-g-orbit*[**where** $f{=}\lambda s.\ c$ **and** $\varphi{=}(\lambda\ t\ s.\ s + t *_R\ c)$])

using *line-is-local-flow*[*OF assms*] **unfolding** *image-le-pred* **by** *auto*

lemma *diff-solve-rule*:
assumes *local-flow f T UNIV* φ
and $\forall s.\ P\ s \longrightarrow (\forall\ t{\in}T.\ (\mathcal{P}\ (\lambda t.\ \varphi\ t\ s)\ (down\ T\ t) \subseteq \{s.\ G\ s\}) \longrightarrow Q\ (\varphi\ t\ s))$
shows $\lceil P \rceil \leq wp\ (x'{=}f\ \&\ G\ on\ T\ UNIV\ @\ 0)\ \lceil Q \rceil$
using *assms* **by**(*subst local-flow.wp-g-orbit*, *auto*)

lemma *diff-weak-axiom*: $wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil = wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil \lambda\ s.\ G\ s \longrightarrow Q\ s \rceil$
unfolding *wp-g-evolution image-def* **by** *force*

lemma *diff-weak-rule*: $\lceil G \rceil \leq \lceil Q \rceil \Longrightarrow \lceil P \rceil \leq wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$
by (*subst wp-nd-fun*) (*auto simp*: *g-orbital-eq*)

lemma *wp-g-orbit-IdD*:
assumes $wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil = \eta^\bullet$
and $\forall \tau{\in}(down\ T\ t).\ x\ \tau \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
shows $\forall \tau{\in}(down\ T\ t).\ C\ (x\ \tau)$
proof
fix τ **assume** $\tau \in (down\ T\ t)$
hence $x\ \tau \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
using *assms*(*2*) **by** *blast*
also have $\forall y.\ y \in (g\text{-}orbital\ f\ G\ T\ S\ t_0\ s) \longrightarrow C\ y$
using *assms*(*1*) **unfolding** *wp-nd-fun* **by** (*subst* (*asm*) *nd-fun-eq-iff*[*symmetric*]) *auto*
ultimately show $C\ (x\ \tau)$
by *blast*
qed

lemma *diff-cut-axiom*:
assumes *Thyp*: *is-interval* T $t_0 \in T$
and $wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil C \rceil = \eta^\bullet$
shows $wp\ (x'{=}f\ \&\ G\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil = wp\ (x'{=}f\ \&\ (\lambda s.\ G\ s \wedge C\ s)\ on\ T\ S\ @\ t_0)\ \lceil Q \rceil$
proof(*rule-tac* $f{=}\lambda\ x.\ wp\ x\ \lceil Q \rceil$ **in** *HOL.arg-cong*, *rule nd-fun-ext*, *rule subset-antisym*, *simp-all*)
fix s
{**fix** s' **assume** $s' \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
then obtain τ::*real* **and** X **where** *x-ivp*: $X \in ivp\text{-}sols\ (\lambda t.\ f)\ T\ S\ t_0\ s$
and $X\ \tau = s'$ **and** $\tau \in T$ **and** *guard-x*:$(\mathcal{P}\ X\ (down\ T\ \tau) \subseteq \{s.\ G\ s\})$
using *g-orbitalD*[*of* $s'\ f\ G\ T\ S\ t_0\ s$] **by** *blast*
have $\forall t{\in}(down\ T\ \tau).\ \mathcal{P}\ X\ (down\ T\ t) \subseteq \{s.\ G\ s\}$
using *guard-x* **by** (*force simp*: *image-def*)
also have $\forall t{\in}(down\ T\ \tau).\ t \in T$
using ‹$\tau \in T$› *Thyp* **by** *auto*
ultimately have $\forall t{\in}(down\ T\ \tau).\ X\ t \in g\text{-}orbital\ f\ G\ T\ S\ t_0\ s$
using *g-orbitalI*[*OF x-ivp*] **by** (*metis* (*mono-tags*, *lifting*))

```
    hence ∀ t∈(down T τ). C (X t)
      using wp-g-orbit-IdD[OF assms(3)] by blast
    hence s′ ∈ g-orbital f (λs. G s ∧ C s) T S t0 s
      using g-orbitalI[OF x-ivp ‹τ ∈ T›] guard-x ‹X τ = s′›
      unfolding image-le-pred by fastforce}
  thus g-orbital f G T S t0 s ⊆ g-orbital f (λs. G s ∧ C s) T S t0 s
    by blast
next
  fix s
  show g-orbital f (λs. G s ∧ C s) T S t0 s ⊆ g-orbital f G T S t0 s
    by (auto simp: g-orbital-eq)
qed

lemma diff-cut-rule:
  assumes Thyp: is-interval T t0 ∈ T
    and wp-C: ⌈P⌉ ≤ wp (x´=f & G on T S @ t0) ⌈C⌉
    and wp-Q: ⌈P⌉ ≤ wp (x´=f & (λs. G s ∧ C s) on T S @ t0) ⌈Q⌉
  shows ⌈P⌉ ≤ wp (x´=f & G on T S @ t0) ⌈Q⌉
proof(simp add: wp-nd-fun g-orbital-eq image-le-pred, clarsimp)
  fix t::real and X::real ⇒ ′a and s assume P s and t ∈ T
    and x-ivp:X ∈ ivp-sols (λt. f) T S t0 s
    and guard-x:∀ x. x ∈ T ∧ x ≤ t ⟶ G (X x)
  have ∀ t∈(down T t). X t ∈ g-orbital f G T S t0 s
    using g-orbitalI[OF x-ivp] guard-x unfolding image-le-pred by auto
  hence ∀ t∈(down T t). C (X t)
    using wp-C ‹P s› by (subst (asm) wp-nd-fun, auto)
  hence X t ∈ g-orbital f (λs. G s ∧ C s) T S t0 s
    using guard-x ‹t ∈ T› by (auto intro!: g-orbitalI x-ivp)
  thus Q (X t)
    using ‹P s› wp-Q by (subst (asm) wp-nd-fun) auto
qed

lemma DS:
  fixes c::′a::{heine-borel, banach}
  shows wp (x´=(λs. c) & G) ⌈Q⌉ = ⌈λx. ∀ t. (∀ τ≤t. G (x + τ *R c)) ⟶ Q (x
+ t *R c)⌉
  by (subst diff-solve-axiom[of UNIV]) (auto simp: fun-eq-iff)

lemma solve:
  assumes local-flow f UNIV UNIV φ
    and ∀ s. P s ⟶ (∀ t. (∀ τ≤t. G (φ τ s)) ⟶ Q (φ t s))
  shows ⌈P⌉ ≤ wp (x´=f & G) ⌈Q⌉
  apply(rule diff-solve-rule[OF assms(1)])
  using assms(2) unfolding image-le-pred by simp

lemma DW: wp (x´=f & G) ⌈Q⌉ = wp (x´=f & G) ⌈λs. G s ⟶ Q s⌉
  by (rule diff-weak-axiom)

lemma dW: ⌈G⌉ ≤ ⌈Q⌉ ⟹ ⌈P⌉ ≤ wp (x´=f & G) ⌈Q⌉
```

by (*rule diff-weak-rule*)

lemma *DC*:
assumes *wp* (x'=*f* & *G*) $\lceil C \rceil = \eta^\bullet$
shows *wp* (x'=*f* & *G*) $\lceil Q \rceil$ = *wp* (x'=*f* & ($\lambda s.\ G\ s \wedge C\ s$)) $\lceil Q \rceil$
apply (*rule diff-cut-axiom*)
using *assms* **by** *auto*

lemma *dC*:
assumes $\lceil P \rceil \leq$ *wp* (x'=*f* & *G*) $\lceil C \rceil$
and $\lceil P \rceil \leq$ *wp* (x'=*f* & ($\lambda s.\ G\ s \wedge C\ s$)) $\lceil Q \rceil$
shows $\lceil P \rceil \leq$ *wp* (x'=*f* & *G*) $\lceil Q \rceil$
apply(*rule diff-cut-rule*)
using *assms* **by** *auto*

lemma *dI*:
assumes $\lceil P \rceil \leq \lceil I \rceil$ **and** *diff-invariant I f UNIV UNIV 0 G* **and** $\lceil I \rceil \leq \lceil Q \rceil$
shows $\lceil P \rceil \leq$ *wp* (x'=*f* & *G*) $\lceil Q \rceil$
apply(*rule wp-g-evolution-inv*[*OF assms*(*1*) - *assms*(*3*)])
unfolding *wp-diff-inv* **using** *assms*(*2*) **.**

end
theory *cat2ndfun-examples*
imports ../*hs-prelims-matrices cat2ndfun*

begin

6.3.4 Examples

Preliminary preparation for the examples.

no-notation *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
and *Archimedean-Field.floor-ceiling-class.floor* ($\lfloor$-$\rfloor$)

— Finite set of program variables.

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

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

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* {''x'',''y''}])
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-univ-eq*: $(UNIV::program\text{-}vars\ set) = \{\upharpoonright_V{''x''}, \upharpoonright_V{''y''}\}$
apply *auto* **by** (*metis to-str to-str-inverse insertE singletonD*)

lemma *program-vars-exhaust*: $x = \upharpoonright_V{''x''} \vee x = \upharpoonright_V{''y''}$
using *program-vars-univ-eq* **by** *auto*

— Alternative to the finite set of program variables.

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

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

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

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

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

Circular Motion

— Verified with differential invariants.

abbreviation *circular-motion-kinematics* :: $real\hat{\ }program\text{-}vars \Rightarrow real\hat{\ }program\text{-}vars$

where *circular-motion-kinematics* $s \equiv (\chi\ i.\ if\ i{=}(\upharpoonright_V{''x''})\ then\ s\$(\upharpoonright_V{''y''})\ else\ -s\$(\upharpoonright_V{''x''}))$

notation *circular-motion-kinematics* (C)

lemma *circular-motion-invariant*:
diff-invariant $(\lambda s.\ (r::real)^2 = (s\$(\upharpoonright_V{''x''}))^2 + (s\$(\upharpoonright_V{''y''}))^2)\ C\ UNIV\ UNIV\ 0\ G$
apply(*rule-tac diff-invariant-rules*, *clarsimp*, *simp*, *clarsimp*)
apply(*frule-tac* $i{=}\upharpoonright_V{''x''}$ **in** *has-vderiv-on-vec-nth*, *drule-tac* $i{=}\upharpoonright_V{''y''}$ **in** *has-vderiv-on-vec-nth*)
by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *circular-motion-invariants*:
$\lceil\lambda s.\ r^2 = (s\$\upharpoonright_V{''x''})^2 + (s\$\upharpoonright_V{''y''})^2\rceil \leq wp\ (x\acute{} = C\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$\upharpoonright_V{''x''})^2 + (s\$\upharpoonright_V{''y''})^2\rceil$
unfolding *wp-diff-inv* **using** *circular-motion-invariant* **by** *auto*

— Verified with the flow.

abbreviation *circular-motion-flow t s* ≡
$(\chi\ i.\ if\ i{=}\upharpoonright_V{''x''}\ then\ s\$(\upharpoonright_V{''x''}) \cdot cos\ t + s\$(\upharpoonright_V{''y''}) \cdot sin\ t$
$else\ -\ s\$(\upharpoonright_V{''x''}) \cdot sin\ t + s\$(\upharpoonright_V{''y''}) \cdot cos\ t)$

notation *circular-motion-flow* (φ_C)

lemma *picard-lindeloef-circ-motion*: *picard-lindeloef* $(\lambda t.\ C)$ *UNIV UNIV 0*
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac x=1* **in** *exI*, *clarsimp*, *rule-tac x=1* **in** *exI*)
by(*simp add*: *dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject power2-commute*)

lemma *local-flow-circ-motion*: *local-flow C UNIV UNIV* φ_C
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
apply(*rule picard-lindeloef-circ-motion*, *simp-all add*: *vec-eq-iff*)
apply(*rule has-vderiv-on-vec-lambda*, *clarify*)
apply(*case-tac* $i = \upharpoonright_V{''x''}$, *simp*)
apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)
apply(*force intro*!: *poly-derivatives derivative-intros simp*: *to-var-inject*)
using *program-vars-exhaust* **by** *force*

lemma *circular-motion*:
$\lceil\lambda s.\ r^2 = (s\$\upharpoonright_V{''x''})^2 + (s\$\upharpoonright_V{''y''})^2\rceil \leq wp\ (x\acute{} = C\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$\upharpoonright_V{''x''})^2 + (s\$\upharpoonright_V{''y''})^2\rceil$
by (*subst local-flow.wp-g-orbit*[*OF local-flow-circ-motion*]) (*auto simp*: *to-var-inject*)

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

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

— Verified as a linear system (using uniqueness).

abbreviation *circular-motion-sq-mtx* :: *2 sq-mtx*
where *circular-motion-sq-mtx* ≡ *sq-mtx-chi* $(\chi\ i.\ if\ i{=}0\ then\ -\ \mathrm{e}\ 1\ else\ \mathrm{e}\ 0)$

abbreviation *circular-motion-mtx-flow* :: *real* ⇒ *real^2* ⇒ *real^2*
where *circular-motion-mtx-flow t s* ≡
$(\chi\ i.\ if\ i{=}(0{::}2)\ then\ s\$0 \cdot cos\ t - s\$1 \cdot sin\ t\ else\ s\$0 \cdot sin\ t + s\$1 \cdot cos\ t)$

notation *circular-motion-sq-mtx* (C)

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

lemma *circular-motion-mtx-exp-eq*: $exp\ (t *_R C) *_V s = \varphi_C\ t\ s$
  apply(*rule local-flow.eq-solution*[*OF local-flow-exp*, *symmetric*])
    apply(*rule ivp-solsI*, *rule has-vderiv-on-vec-lambda*, *clarsimp*)
  unfolding *sq-mtx-vec-prod-def matrix-vector-mult-def* **apply** *simp*
    apply(*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)
  using *exhaust-2 two-eq-zero* **by** (*force simp*: *vec-eq-iff*, *auto*)

lemma *circular-motion-sq-mtx*:
  $\lceil\lambda s.\ r^2 = (s\$0)^2 + (s\$1)^2\rceil \leq wp\ (x' = ((*_V)\ C)\ \&\ G)\ \lceil\lambda s.\ r^2 = (s\$0)^2 + (s\$1)^2\rceil$
 unfolding *local-flow.wp-g-orbit*[*OF local-flow-exp*] *circular-motion-mtx-exp-eq* **by** *auto*

no-notation *circular-motion-sq-mtx* (C)

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

Bouncing Ball

— Verified with differential invariants.

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

lemma [*bb-real-arith*]:
  assumes $0 > g$ **and** *inv*: $2 \cdot g \cdot x - 2 \cdot g \cdot h = v \cdot v$
  shows (x::*real*) $\leq h$
proof−
  have $v \cdot v = 2 \cdot g \cdot x - 2 \cdot g \cdot h \wedge 0 > g$
    using *inv* **and** ‹$0 > g$› **by** *auto*
  hence *obs*:$v \cdot v = 2 \cdot g \cdot (x - h) \wedge 0 > g \wedge v \cdot v \geq 0$
    using *left-diff-distrib mult.commute* **by** (*metis zero-le-square*)
  hence $(v \cdot v)/(2 \cdot g) = (x - h)$
    by *auto*
  also from *obs* **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

abbreviation *constant-acceleration-kinematics* $g\ s \equiv (\chi\ i.\ if\ i{=}(\restriction_V{''x''})\ then\ s\$(\restriction_V{''y''})\ else\ g)$

notation *constant-acceleration-kinematics* (K)

lemma *energy-conservation-invariant*:
  fixes $g\ h$ :: *real*

defines *dinv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\restriction_V {''x''}) - 2 \cdot g \cdot h - (s\$(\restriction_V {''y''}) \cdot s\$(\restriction_V {''y''})) = 0)$
shows *diff-invariant* I $(K\ g)$ *UNIV UNIV 0 G*
unfolding *dinv* **apply**(*rule diff-invariant-rules*, *simp*, *simp*, *clarify*)
apply(*frule-tac* $i{=}\restriction_V {''y''}$ **in** *has-vderiv-on-vec-nth*)
apply(*drule-tac* $i{=}\restriction_V {''x''}$ **in** *has-vderiv-on-vec-nth*)
by(*auto intro*!: *poly-derivatives simp*: *to-var-inject*)

lemma *bouncing-ball-invariants*:
fixes *h*::*real*
assumes $g < 0$ **and** $h \geq 0$
defines *diff-inv*: $I \equiv (\lambda s.\ 2 \cdot g \cdot s\$(\restriction_V {''x''}) - 2 \cdot g \cdot h - (s\$(\restriction_V {''y''}) \cdot s\$(\restriction_V {''y''})) = 0)$
shows $\lceil \lambda s.\ s\$(\restriction_V {''x''}) = h \wedge s\$(\restriction_V {''y''}) = 0 \rceil \leq$
$wp\ (((x\acute{}{=}K\ g\ \&\ (\lambda\ s.\ s\$(\restriction_V {''x''}) \geq 0)) \cdot$
$(IF\ (\lambda\ s.\ s\$(\restriction_V {''x''}) = 0)\ THEN\ ((\restriction_V {''y''}) ::= (\lambda s.\ -\ s\$(\restriction_V {''y''})))\ ELSE\ \eta^\bullet\ FI))^\star)$
$\lceil \lambda s.\ 0 \leq s\$(\restriction_V {''x''}) \wedge s\$(\restriction_V {''x''}) \leq h \rceil$
apply(*subst star-nd-fun.abs-eq*)
apply(*rule-tac* $I{=}\lceil \lambda s.\ 0 \leq s\$(\restriction_V {''x''}) \wedge I\ s \rceil$ **in** *wp-starI*)
using ‹$h \geq 0$› **apply**(*simp add*: *diff-inv*, *simp only*: *fbox-mult*)
apply(*subst p2ndf-ndf2p-wp*[*symmetric*, *of* (*IF - THEN - ELSE* $\eta^\bullet$ *FI*)])
apply(*rule order.trans*[**where** $b{=}wp\ (x\acute{}{=}K\ g\ \&\ (\lambda s.\ s\$(\restriction_V {''x''}){\geq}0))\ \lceil \lambda s.\ 0{\leq}s\$(\restriction_V {''x''}) \wedge I\ s \rceil$])
apply(*simp only*: *wp-g-evolution-guard*)
apply(*rule order.trans*[**where** $b{=}\lceil I \rceil$], *simp*)
apply(*simp add*: *wp-diff-inv*, *unfold diff-inv*)
using *energy-conservation-invariant* **apply** *force*
apply(*rule fbox-iso*)
apply(*simp add*: *plus-nd-fun-def f2r-def times-nd-fun-def kcomp-def*)
using *assms* **by** (*auto simp*: *bb-real-arith le-fun-def*)

— Verified with the flow.

lemma *picard-lindeloef-cnst-acc*:
fixes *g*::*real*
shows *picard-lindeloef* $(\lambda t.\ K\ g)$ *UNIV UNIV 0*
apply(*unfold-locales*)
apply(*unfold-locales*, *simp-all add*: *local-lipschitz-def lipschitz-on-def*, *clarsimp*)
apply(*rule-tac* $x{=}1/2$ **in** *exI*, *clarsimp*, *rule-tac* $x{=}1$ **in** *exI*)
by(*simp add*: *dist-norm norm-vec-def L2-set-def program-vars-univ-eq to-var-inject*)

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

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

lemma *local-flow-cnst-acc*: *local-flow* $(K\ g)$ *UNIV UNIV* $(\varphi_K\ g)$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-cnst-acc* **apply** *blast*
apply(*rule has-vderiv-on-vec-lambda*, *clarify*)
apply(*case-tac* $i = \upharpoonright_V\ ''x''$)
using *program-vars-exhaust* **by**(*auto intro*!: *poly-derivatives simp*: *to-var-inject vec-eq-iff*)

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 $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
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*)
hence *obs*: $(g \cdot \tau + v)^2 + 2 \cdot g \cdot h = 0$
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*))
thus $2 \cdot g \cdot h + (g \cdot \tau \cdot (g \cdot \tau + v) + v \cdot (g \cdot \tau + v)) = 0$
by (*simp add*: *monoid-mult-class.power2-eq-square*)
have $2 \cdot g \cdot h + (-\ ((g \cdot \tau) + v))^2 = 0$
using *obs* **by** (*metis Groups.add-ac*(*2*) *power2-minus*)
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*:
  fixes *h*::*real*
  assumes $g < 0$ **and** $h \geq 0$
  defines *loop-inv*: $I \equiv (\lambda s.\ 0 \leq s\$(\upharpoonleft_V{}''x'') \wedge\ 2 \cdot g \cdot s\$(\upharpoonleft_V{}''x'') =$
    $2 \cdot g \cdot h + (s\$(\upharpoonleft_V{}''y'') \cdot s\$(\upharpoonleft_V{}''y'')))$
  shows $\lceil\lambda s.\ s\$(\upharpoonleft_V{}''x'') = h \wedge s\$(\upharpoonleft_V{}''y'') = 0\rceil \leq$
  $wp\ (((x\acute{}=K\ g\ \&\ (\lambda\ s.\ s\$(\upharpoonleft_V{}''x'') \geq 0)) \cdot$
  $(IF\ (\lambda\ s.\ s\$(\upharpoonleft_V{}''x'') = 0)\ THEN\ ((\upharpoonleft_V{}''y'')\ ::= (\lambda s.\ -\ s\$(\upharpoonleft_V{}''y'')))\ ELSE\ \eta^\bullet$
$FI))^\star)$
  $\lceil\lambda s.\ 0 \leq s\$(\upharpoonleft_V{}''x'') \wedge s\$(\upharpoonleft_V{}''x'') \leq h\rceil$
  apply(*subst star-nd-fun.abs-eq*)
  apply(*rule-tac* $I=\lceil I\rceil$ **in** *wp-starI*)
   unfolding *loop-inv* **using** ‹$h \geq 0$› **apply**(*simp*, *simp only*: *fbox-mult*)
   apply(*subst p2ndf-ndf2p-wp*[*symmetric*, *of* (*IF* - *THEN* - *ELSE* $\eta^\bullet$ *FI*)])
    apply(*subst local-flow.wp-g-orbit*[*OF local-flow-cnst-acc*])
   apply(*subst ndf2p-wpD*)
   unfolding *cond-def* **apply**(*simp add*: *plus-nd-fun-def f2r-def times-nd-fun-def*
kcomp-def)
  using *assms* **by** (*auto simp*: *bb-real-arith le-fun-def to-var-inject*)

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

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

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

— Verified as a linear system (computing exponential).

abbreviation *constant-acceleration-kinematics-sq-mtx* :: *3 sq-mtx*
  where *constant-acceleration-kinematics-sq-mtx* ≡
    sq-mtx-chi (χ *i*::*3*. *if i=0 then* e *1 else if i=1 then* e *2 else 0*)

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

lemma *const-acc-mtx-pow2*: K^2 = *sq-mtx-chi* (χ *i*. *if i=0 then* e *2 else 0*)
  unfolding *power2-eq-square times-sq-mtx-def*
  by(*simp add*: *sq-mtx-chi-inject vec-eq-iff matrix-matrix-mult-def*)

lemma *const-acc-mtx-powN*: $n > 2 \Longrightarrow (\tau *_R K)\hat{}n = 0$
  apply(*induct n*, *simp*, *case-tac* $n \leq 2$)
   apply(*simp only*: *le-less-Suc-eq power-Suc*, *simp*)
  by(*auto simp*: *const-acc-mtx-pow2 sq-mtx-chi-inject vec-eq-iff*
      times-sq-mtx-def zero-sq-mtx-def matrix-matrix-mult-def)

lemma *exp-cnst-acc-sq-mtx*: $exp\ (\tau *_R K) = ((\tau *_R K)^2/_R\ 2) + (\tau *_R K) + 1$
  unfolding *exp-def* **apply**(*subst suminf-eq-sum*[*of 2*])
  using *const-acc-mtx-powN* **by** (*simp-all add*: *numeral-2-eq-2*)

lemma *exp-cnst-acc-sq-mtx-simps*:

exp $(\tau *_R K)$ \$\$ *0* \$ *0* = *1* *exp* $(\tau *_R K)$ \$\$ *0* \$ *1* = τ *exp* $(\tau *_R K)$ \$\$ *0* \$ *2* = τ ^*2*/*2*
exp $(\tau *_R K)$ \$\$ *1* \$ *0* = *0* *exp* $(\tau *_R K)$ \$\$ *1* \$ *1* = *1* *exp* $(\tau *_R K)$ \$\$ *1* \$ *2* = τ
exp $(\tau *_R K)$ \$\$ *2* \$ *0* = *0* *exp* $(\tau *_R K)$ \$\$ *2* \$ *1* = *0* *exp* $(\tau *_R K)$ \$\$ *2* \$ *2* = *1*
unfolding *exp-cnst-acc-sq-mtx scaleR-power const-acc-mtx-pow2*
by (*auto simp*: *plus-sq-mtx-def scaleR-sq-mtx-def one-sq-mtx-def mat-def scaleR-vec-def axis-def plus-vec-def*)

lemma *bouncing-ball-K*:
⌈λ*s*. *0* ≤ *s* \$ *0* ∧ *s* \$ *0* = *h* ∧ *s* \$ *1* = *0* ∧ *0* > *s* \$ *2*⌉ ≤
wp (((*x*´=($*_V$) *K* & (λ *s*. *s* \$ *0* ≥ *0*)) ·
(*IF* (λ *s*. *s* \$ *0* = *0*) *THEN* (*1* ::= (λ*s*. − *s* \$ *1*)) *ELSE* $\eta^\bullet$ *FI*))*)
⌈λ*s*. *0* ≤ *s* \$ *0* ∧ *s* \$ *0* ≤ *h*⌉
apply(*subst star-nd-fun.abs-eq*)
apply(*rule-tac I*=⌈λ*s*. *0*≤*s*\$*0* ∧ *0* > *s*\$*2* ∧ *2* · *s*\$*2* · *s*\$*0* = *2* · *s*\$*2* · *h* + (*s*\$*1* · *s*\$*1*)⌉ **in** *wp-starI*)
apply(*simp*, *simp only*: *fbox-mult*)
apply(*subst p2ndf-ndf2p-wp*[*symmetric*, *of* (*IF* - *THEN* - *ELSE* $\eta^\bullet$ *FI*)])
apply(*subst local-flow.wp-g-orbit*[*OF local-flow-exp*], *clarsimp*)
apply(*simp add*: *plus-nd-fun-def times-nd-fun-def f2r-def kcomp-def*)
apply(*rule-tac x*=*exp* $(t *_R K) *_V$ *s* **in** *exI*)
apply(*simp add*: *sq-mtx-vec-prod-def matrix-vector-mult-def*)
unfolding *UNIV-3* **apply**(*simp add*: *exp-cnst-acc-sq-mtx-simps*, *safe*)
subgoal for *x* **using** *bb-real-arith*(*2*)[*of x* \$ *2*]
by (*simp add*: *add.commute mult.commute*)
subgoal for *x* τ **using** *bb-real-arith*(*3*)[**where** *g*=*x* \$ *2* **and** *v*=*x* \$ *1*]
by(*simp add*: *add.commute mult.commute*)
apply(*simp add*: *field-simps power2-eq-square*)
by (*force simp*: *bb-real-arith*)

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

end

6.4 VC_diffKAD

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

begin

6.4.1 Stack Theories Preliminaries: VC_KAD and ODEs

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

no-notation *Archimedean-Field.ceiling* ($\lceil$-$\rceil$)
  and *Archimedean-Field.floor* ($\lfloor$-$\rfloor$)
  and *Set.image* (‘)
  and *Range-Semiring.antirange-semiring-class.ars-r* (r)

notation *p2r* ($\lceil$-$\rceil$)
  and *r2p* ($\lfloor$-$\rfloor$)
  and *Set.image* (-$(\!|$-$|\!)$)
  and *Product-Type.prod.fst* (π_1)
  and *Product-Type.prod.snd* (π_2)
  and *List.zip* (**infixl** $\otimes$ *63*)
  and *rel-ad* ($\Delta^c{}_1$)

This and more notation is explained by the following lemmata.

lemma shows $\lceil P\rceil = \{(s, s) \mid s.\ P\ s\}$
  and $\lfloor R\rfloor = (\lambda x.\ x \in r2s\ R)$
  and $r2s\ R = \{x \mid x.\ \exists\ y.\ (x,y) \in R\}$
  and $\pi_1\ (x,y) = x \wedge \pi_2\ (x,y) = y$
  and $\Delta^c{}_1\ R = \{(x, x) \mid x.\ \nexists y.\ (x, y) \in R\}$
  and $wp\ R\ Q = \Delta^c{}_1\ (R\ ;\ \Delta^c{}_1\ Q)$
  and $[x1,x2,x3,x4] \otimes [y1,y2] = [(x1,y1),(x2,y2)]$
  and $\{a..b\} = \{x.\ a \leq x \wedge x \leq b\}$
  and $\{a<..<b\} = \{x.\ a < x \wedge x < b\}$
  and $(x\ solves\text{-}ode\ f)\ \{0..t\}\ R = ((x\ has\text{-}vderiv\text{-}on\ (\lambda t.\ f\ t\ (x\ t)))\ \{0..t\} \wedge x \in \{0..t\} \to R)$
  and $f \in A \to B = (f \in \{f.\ \forall\ x.\ x \in A \longrightarrow (f\ x) \in B\})$
  and $(x\ has\text{-}vderiv\text{-}on\ x')\{0..t\} =$
    $(\forall\, r\in\{0..t\}.\ (x\ has\text{-}vector\text{-}derivative\ x'\ r)\ (at\ r\ within\ \{0..t\}))$
  and $(x\ has\text{-}vector\text{-}derivative\ x'\ r)\ (at\ r\ within\ \{0..t\}) =$
    $(x\ has\text{-}derivative\ (\lambda x.\ x *_R x'\ r))\ (at\ r\ within\ \{0..t\})$
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 $\pi_1(\!|R|\!) = r2s\ R$
by (*simp add*: *fst-eq-Domain*)

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

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

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

proposition *rdom-p2r-contents*:$(a,\ b) \in rdom\ \lceil P\rceil = ((a = b) \wedge P\ a)$
proof−
have $(a,\ b) \in rdom\ \lceil P\rceil = ((a = b) \wedge (a,\ a) \in rdom\ \lceil P\rceil)$ **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*: $R \subseteq Id \Longrightarrow (x,x) \notin R \Longrightarrow (x,x) \in \Delta^c{}_1\ R$
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*:$(x,x) \in wp\ R\ \lceil P\rceil = (\forall\ y.\ (x,y) \in R \longrightarrow P\ y)$
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 $\langle d\ i \leq |x]\ i\rangle$ **have** $d\ i \leq |x]\ (d\ i)$
  using *local.fbox-simp* **by** *auto*
hence $|1]\ p \leq |x^\star]\ i$ **using** $\langle d\ p \leq d\ i\rangle$ **by** (*metis* (*no-types*)
  local.dual-order.trans local.fbox-one local.fbox-simp local.fbox-star-induct-var)
thus *?thesis* **using** $\langle d\ i \leq d\ q\rangle$ **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*]

6.4.2 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* ⇒ *string* (∂ - [*55*] *70*) **where**
(∂ *x*) = ″*d*[″@*x*@″]″

definition *varDiffs* :: *string set* **where**
varDiffs = {*y*. ∃ *x*. *y* = ∂ *x*}

proposition *vdiff-inj*:(∂ *x*) = (∂ *y*) ⟹ *x* = *y*
by(*simp add*: *vdiff-def*)

proposition *vdiff-noFixPoints*:*x* ≠ (∂ *x*)
by(*simp add*: *vdiff-def*)

lemma *varDiffsI*:*x* = (∂ *z*) ⟹ *x* ∈ *varDiffs*
by(*simp add*: *varDiffs-def vdiff-def*)

lemma *varDiffsE*:
assumes *x* ∈ *varDiffs*
obtains *y* **where** *x* = ″*d*[″@*y*@″]″
using *assms* **unfolding** *varDiffs-def vdiff-def* **by** *auto*

proposition *vdiff-invarDiffs*:(∂ *x*) ∈ *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* $\neq$ ''*d*['' $\Longrightarrow$ (*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*)
($(\pi_1\ (\pi_2\ uxf)) := (\pi_1\ uxf)\ t\ s$,
$\partial\ (\pi_1\ (\pi_2\ uxf)) := (if\ t = 0\ then\ (\pi_2\ (\pi_2\ uxf))\ s$
$else\ vderiv\text{-}of\ (\lambda\ r.\ (\pi_1\ uxf)\ r\ s)\ \{0{<}..{<}\ (2 *_R t)\}\ t))$

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*,
  $\partial\ x := if\ t = 0\ then\ f\ s\ else\ vderiv\text{-}of\ (\lambda\ r.\ u\ r\ s)\ \{0{<}..{<}\ (2 *_R t)\}\ t))\ x$ **by** *simp*
  also have ... = *u t s* **by** (*simp add*: *vdiff-def*)
  ultimately show *?thesis* **by** *simp*
next
  assume *yTailHyp*:(*u*, *x*, *f*) ∈ *set* (*utail* ⊗ *xftail*)
  from *this* **and** *indHyp* **have** *3*:(*s*[*xftail*←*utail*] *t*) *x* = *u t s* **by** *fastforce*
  from *yTailHyp* **and** *distHyp* **have** *2*:*y* ≠ *x* **using** *set-zip-left-rightD* **by** *force*
  from *yTailHyp* **and** *varHyp* **have** *1*:*x* ≠ ∂ *y*
  using *set-zip-left-rightD vdiff-invarDiffs* **by** *fastforce*
  from *1* **and** *2* **have** (*s*[(*y*, *g*) # *xftail*←*v* # *utail*] *t*) *x* = (*s*[*xftail*←*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 $(s[[]\leftarrow utail]\ 0) = override\text{-}on\ s\ s\ varDiffs$
unfolding *override-on-def* **by** *simp*
thus *?case* **by** *force*
next
case (*4 xf xfTail u uTail*)
then have $\exists\, g.\ (s[xfTail\leftarrow uTail]\ 0) = override\text{-}on\ s\ g\ varDiffs$ **by** *simp*
thus *?case* **using** *inductive-state-list-cross-upd-its-dvars 4.prems* **by** *blast*
qed

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

lemma *closed-vderiv-on-cong-to-open-vderiv*:
assumes $gHyp{:}\forall\ \tau > 0.\ f\ \tau = g\ \tau$
and $fHyp{:}\forall\, t{\ge}0.\ (f\ has\text{-}vderiv\text{-}on\ f')\ \{0..t\}$
and $tHyp{:}\ t>0$ **and** $cHyp{:}\ c > 1$
shows $vderiv\text{-}of\ g\ \{0{<}..{<}\ (c *_R\ t)\}\ t = f'\ t$
proof−
have $ctHyp{:}c \cdot t > 0$ **using** *tHyp* **and** *cHyp* **by** *auto*
from *fHyp* **have** $(f\ has\text{-}vderiv\text{-}on\ f')\ \{0{<}..{<}c \cdot t\}$ **using** *has-vderiv-on-subset*
by (*metis greaterThanLessThan-subseteq-atLeastAtMost-iff less-eq-real-def*)
then have $derivHyp{:}(g\ has\text{-}vderiv\text{-}on\ f')\ \{0{<}..{<}c \cdot t\}$
using *gHyp ctHyp* **and** *has-vderiv-on-cong-open-interval* **by** *blast*
hence $f'Hyp{:}\forall\ f''.\ (g\ has\text{-}vderiv\text{-}on\ f'')\ \{0{<}..{<}c \cdot t\} \longrightarrow (\forall\ \tau \in \{0{<}..{<}\ c \cdot t\}.$
$f'\ \tau = f''\ \tau)$
using *vderiv-unique-within-open-interval ctHyp* **by** *blast*
also have $(g\ has\text{-}vderiv\text{-}on\ (vderiv\text{-}of\ g\ \{0{<}..{<}\ (c *_R\ t)\}))\ \{0{<}..{<}c \cdot t\}$
by(*simp add*: *vderiv-of-def*, *metis derivHyp someI-ex*)
ultimately show$vderiv\text{-}of\ g\ \{0{<}..{<}c *_R\ t\}\ t = f'\ t$ **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 ∂ *x* $\neq$ ∂ *y* **by**(*simp add*: *vdiff-def*)
  have *x* $\neq$ ∂ *y* **using** *vars vdiff-invarDiffs* **by** *auto*
  obtain *h2* **where** *h2Def*:(*sol s*[*xfs*$\leftarrow$*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*$\leftarrow$*u* # *us*] *0*) (∂ *y*) = *g* (*sol s*[*xfs*$\leftarrow$*us*] *0*)
  using *tailHyp indHyp* ‹*x* $\neq$ ∂ *y*› **and** ‹∂ *x* $\neq$ ∂ *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*$\leftarrow$*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*:$\forall$ *s*. $\forall$ *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 *ygHyp*:(*y*, *g*) $\in$ *set xfList*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *0*)(∂ *y*) = *g* (*sol s*[*xfList*$\leftarrow$*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*: $\forall$ *xf*$\in$*set* ((*y*, *g*) # *xfs*). π_1 *xf* $\notin$ *varDiffs*
and *indHyp*:(*u*,*x*,*f*) $\in$ *set* (*vs* $\otimes$ *xfs*) $\Longrightarrow$ (*s*[*xfs*$\leftarrow$*vs*]*t*)(∂ *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* $\otimes$ *xfs*) **and** *tHyp*:*t* > *0*
shows (*s*[(*y*, *g*) # *xfs*$\leftarrow$*v* # *vs*] *t*) (∂ *x*) = *vderiv-of* (λ*r*. *u r s*) {*0*<..<*2* $*_R$ *t*} *t*
proof−
let *?lhs* = ((*s*[*xfs*$\leftarrow$*vs*] *t*)(*y* := *v t s*, ∂ *y* := *vderiv-of* (λ *r*. *v r s*) {*0*<..< (*2* $\cdot$ *t*)} *t*)) (∂ *x*)
let *?rhs* = *vderiv-of* (λ*r*. *u r s*) {*0*<..< (*2* $\cdot$ *t*)} *t*
have (*s*[(*y*, *g*) # *xfs*$\leftarrow$*v* # *vs*] *t*) (∂ *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* $\cdot$ *t*)} *t* **by** *simp*
also have *vderiv-of* (λ *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 ∂ *x* $\neq$ ∂ *y* **by**(*simp add*: *vdiff-def*)
have *y* $\neq$ ∂ *x* **using** *varHyp* **using** *vdiff-invarDiffs* **by** *auto*
then have *?lhs* = (*s*[*xfs*$\leftarrow$*vs*] *t*) (∂ *x*) **using** ‹*y* $\neq$ ∂ *x*› **and** ‹∂ *x* $\neq$ ∂ *y*› **by** *simp*
also have (*s*[*xfs*$\leftarrow$*vs*] *t*) (∂ *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* π_1 *xfList*)
and *lengthHyp*:*length xfList* = *length uInput*
and *varsHyp*:$\forall$ *xf* $\in$ *set xfList*. π_1 *xf* $\notin$ *varDiffs*
and *uxfHyp*:(u, x, f) $\in$ *set* (*uInput* $\otimes$ *xfList*) **and** *tHyp*:$t > 0$
shows (s[*xfList*$\leftarrow$*uInput*] t) (∂ x) = *vderiv-of* (λ 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*$\llbracket$ - $\rrbracket_t$)) **where**
$\llbracket t_C\ r \rrbracket_t = (\lambda\ s.\ r)|$
$\llbracket t_V\ x \rrbracket_t = (\lambda\ s.\ s\ x)|$
$\llbracket \ominus\ \vartheta \rrbracket_t = (\lambda\ s.\ -\ (\llbracket \vartheta \rrbracket_t)\ s)|$
$\llbracket \vartheta \oplus \eta \rrbracket_t = (\lambda\ s.\ (\llbracket \vartheta \rrbracket_t)\ s + (\llbracket \eta \rrbracket_t)\ s)|$
$\llbracket \vartheta \odot \eta \rrbracket_t = (\lambda\ s.\ (\llbracket \vartheta \rrbracket_t)\ s \cdot (\llbracket \eta \rrbracket_t)\ s)$

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*$\llbracket$-$\rrbracket_P$)) **where**
$\llbracket \vartheta \doteq \eta \rrbracket_P = (\lambda\ s.\ (\llbracket \vartheta \rrbracket_t)\ s = (\llbracket \eta \rrbracket_t)\ s)|$
$\llbracket \vartheta \prec \eta \rrbracket_P = (\lambda\ s.\ (\llbracket \vartheta \rrbracket_t)\ s < (\llbracket \eta \rrbracket_t)\ s)|$
$\llbracket \vartheta \preceq \eta \rrbracket_P = (\lambda\ s.\ (\llbracket \vartheta \rrbracket_t)\ s \leq (\llbracket \eta \rrbracket_t)\ s)|$
$\llbracket \varphi \sqcap \psi \rrbracket_P = (\lambda\ s.\ (\llbracket \varphi \rrbracket_P)\ s \wedge (\llbracket \psi \rrbracket_P)\ s)|$
$\llbracket \varphi \sqcup \psi \rrbracket_P = (\lambda\ s.\ (\llbracket \varphi \rrbracket_P)\ s \vee (\llbracket \psi \rrbracket_P)\ s)$

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**
$trmVars\ (t_C\ r) = \{\}|$
$trmVars\ (t_V\ x) = \{x\}|$
$trmVars\ (\ominus\ \vartheta) = trmVars\ \vartheta|$
$trmVars\ (\vartheta \oplus \eta) = trmVars\ \vartheta \cup trmVars\ \eta|$
$trmVars\ (\vartheta \odot \eta) = trmVars\ \vartheta \cup trmVars\ \eta$

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

proposition *substList-on-compl-of-varDiffs*:
assumes $trmVars\ \eta \subseteq (UNIV - varDiffs)$
and $set\ (map\ \pi_1\ xtList) \subseteq varDiffs$
shows $xtList\langle\eta\rangle = \eta$
using *assms* **apply**(*induction* η, *simp-all add*: *varDiffs-def*)
by(*induction xtList*, *auto*)

lemma *substList-help1*:$set\ (map\ \pi_1\ ((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)) \subseteq varDiffs$
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* ψ

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** $\tau=t$ **and** $t=2{\cdot}t$ **in** *vderiv-unique-within-open-interval*)
using $\langle t \in \{0{<}..{<}2 \cdot t\}\rangle$ **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** $t{=}2 \cdot t$ **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* $f'1{=}\lambda\ x.\ a$ **and** $g'1{=}\lambda\ x.\ 0$ **in** *derivative-intros*(*191*))
unfolding *has-vderiv-on-def* **by**(*auto intro*: *derivative-eq-intros*)

lemma [*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** $\tau{=}t$ **and** $t{=}2{\cdot}t$ **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

6.4.3 Phase Space Relational Semantics

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

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 $\forall\ t \geq 0.\ \forall\ xf \in set\ xfList.\ ((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))\ solves\text{-}ode\ (\lambda\ t.\lambda\ r.(\pi_2\ xf)$ $(\varphi_S\ t)))\ \{0..t\}\ UNIV$
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 $\forall\ t \geq 0.\ \forall\ xf \in set\ xfList.\ ((\lambda\ t.\ \varphi_S\ t\ (\pi_1\ xf))\ solves\text{-}ode\ (\lambda\ t.\lambda\ r.(\pi_2\ xf)$ $(\varphi_S\ t)))\ \{0..t\}\ UNIV$
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 φ_S *0 y = s y*
using *assms solves-store-ivpE*(*6*) **by** *simp*

definition *guarDiffEqtn* :: (*string* × (*real store* ⇒ *real*)) *list* ⇒ (*real store pred*) ⇒
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*$\}$

6.4.4 Derivation of Differential Dynamic Logic Rules

"Differential Weakening"

lemma *wlp-evol-guard*:*Id* ⊆ *wp* (*ODEsystem xfList with G*) ⌈*G*⌉
by(*simp add*: *rel-antidomain-kleene-algebra.fbox-def rel-ad-def guarDiffEqtn-def p2r-def*, *force*)

theorem *dWeakening*:
assumes *guardImpliesPost*: ⌈*G*⌉ ⊆ ⌈*Q*⌉
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*) ⌈*Q*⌉ = *wp* (*ODEsystem xfList with G*) ⌈$\lambda s.\ G\ s \longrightarrow Q\ s$⌉
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*=φ_S **in** *exI*) **using** *assms* **by** *simp*

lemma *condAfterEvol-remainsAlongEvol*:
assumes *boxDiffC*:(*s*, *s*) ∈ *wp* (*ODEsystem xfList with G*) ⌈*C*⌉
and *FisSol*:*solvesStoreIVP* φ_S *xfList s* $\wedge (\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r)) \wedge 0 \leq t$
shows $\forall\ r \in \{0..t\}.\ G\ (\varphi_S\ r) \wedge C\ (\varphi_S\ r)$
proof−
from *boxDiffC* **have** $\forall\ c.\ (s,c) \in$ (*ODEsystem xfList with G*) ⟶ *C c*
 by (*simp add*: *boxProgrPred-chrctrztn*)
also from *FisSol* **have** $\forall\ r \in \{0..t\}.\ (s,\ \varphi_S\ r) \in$ (*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* $\wedge$ φ_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* $\wedge$ φ_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* (λ*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*=φ_S **in** *exI*, *simp*)
qed

Solve Differential Equation

lemma *prelim-dSolve*:

assumes *solHyp*:(λt. *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*:$t{\geq}0$ $\wedge$ φ_S *t* = *c* $\wedge$ *solvesStoreIVP* φ_S *xfList s* $\wedge$ ($\forall$ $r \in \{0..t\}$. *G* (φ_S *r*))
using *guarDiffEqtn-def* **by** *auto*
from *this* **and** *uniqHyp* **have** (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) = φ_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* (λt. *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*=λt. *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* (λt. *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*: $\forall$ *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(*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*:$\forall$ *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*. (($\lambda\tau$. (*sol s*[*xfList*$\leftarrow$*uInput*] τ) *x*) *has-vderiv-on* ($\lambda\tau$. *f* (*sol s*[*xfList*$\leftarrow$*uInput*] τ))) {*0*..*t*}
and *xfHyp*:(*x*, *f*) $\in$ *set xfList* **and** *tHyp*:*t* $\geq$ *0*
shows (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (∂ *x*) = *f* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*)
proof−
from *xfHyp* **obtain** *u* **where** *xfuHyp*: (*u*,*x*,*f*) $\in$ *set* (*uInput* $\otimes$ *xfList*)
by (*metis in-set-impl-in-set-zip2 lengthHyp*)
show (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (∂ *x*) =*f* (*sol s*[*xfList*$\leftarrow$*uInput*] *t*)
  proof(*cases t*=*0*)
  case *True*
    have (*sol s*[*xfList*$\leftarrow$*uInput*] *0*) (∂ *x*) = *f* (*sol s*[*xfList*$\leftarrow$*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*$\leftarrow$*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*$\leftarrow$*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*.
((λ*t*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on* (λ*t*. 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 (λt *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*:$\forall\ xf \in set\ xfList.\ continuous\text{-}on\ (\{0..t\} \times UNIV)$
$(\lambda(t,\ (r{::}real)).\ (\pi_2\ xf)\ (sol\ s[xfList{\leftarrow}uInput]\ t))$
and *eqDerivs*:$\forall\ xf \in set\ xfList.\ \forall\ \tau \in \{0..t\}.\ (\pi_2\ xf)\ (\varphi_s\ \tau) = (\pi_2\ xf)\ (sol$
$s[xfList{\leftarrow}uInput]\ \tau)$
and *Fsolves*:$solvesStoreIVP\ \varphi_s\ xfList\ s$
and *solHyp*:$solvesStoreIVP\ (\lambda\ \tau.\ (sol\ s[xfList{\leftarrow}uInput]\ \tau))\ xfList\ s$
shows $(sol\ s[xfList{\leftarrow}uInput]\ t) = \varphi_s\ t$
proof
  fix $x{::}string$ **show** $(sol\ s[xfList{\leftarrow}uInput]\ t)\ x = \varphi_s\ t\ x$
  proof(*cases* $x \in (\pi_1(\!|set\ xfList|\!)) \cup varDiffs$)
  case *False*
    then have *notInVars*:$x \notin (\pi_1(\!|set\ xfList|\!)) \cup varDiffs$ **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** $\varphi_s\ t\ x = s\ x$ **using** *tHyp notInVars solves-store-ivpD(1)*
by *blast*
    ultimately show $(sol\ s[xfList{\leftarrow}uInput]\ t)\ x = \varphi_s\ t\ x$ **by** *simp*
  next case *True*
    then have $x \in (\pi_1(\!|set\ xfList|\!)) \vee x \in varDiffs$ **by** *simp*
    from *this* **show** *?thesis*
    proof
      assume $x \in (\pi_1(\!|set\ xfList|\!))$
      from *this* **obtain** f **where** *xfHyp*:$(x,\ f) \in set\ xfList$ **by** *fastforce*

      then have *expand1*:$\forall\ xf \in set\ xfList.((\lambda\tau.\ \varphi_s\ \tau\ (\pi_1\ xf))\ solves\text{-}ode$
      $(\lambda\tau\ r.\ (\pi_2\ xf)\ (\varphi_s\ \tau)))\{0..t\}\ UNIV \wedge \varphi_s\ 0\ (\pi_1\ xf) = s\ (\pi_1\ xf)$
      using *Fsolves tHyp* **by** (*simp add*:*solvesStoreIVP-def*)
      hence *expand2*:$\forall\ xf \in set\ xfList.\ \forall\ \tau \in \{0..t\}.\ ((\lambda r.\ \varphi_s\ r\ (\pi_1\ xf))$
       $has\text{-}vector\text{-}derivative\ (\lambda r.\ (\pi_2\ xf)\ (sol\ s[xfList{\leftarrow}uInput]\ \tau))\ \tau)\ (at\ \tau\ within$
$\{0..t\})$
      using *eqDerivs* **by** (*simp add*: *solves-ode-def has-vderiv-on-def*)

      then have $\forall\ xf \in set\ xfList.\ ((\lambda\tau.\ \varphi_s\ \tau\ (\pi_1\ xf))\ solves\text{-}ode$
       $(\lambda\tau\ r.\ (\pi_2\ xf)\ (sol\ s[xfList{\leftarrow}uInput]\ \tau)))\{0..t\}\ UNIV \wedge \varphi_s\ 0\ (\pi_1\ xf) = s$
$(\pi_1\ xf)$
      by (*simp add*: *has-vderiv-on-def solves-ode-def expand1 expand2*)
     then have *1*:$((\lambda\tau.\ \varphi_s\ \tau\ x)\ solves\text{-}ode\ (\lambda\tau\ r.\ f\ (sol\ s[xfList{\leftarrow}uInput]\ \tau)))\{0..t\}$
$UNIV \wedge$
      $\varphi_s\ 0\ x = s\ x$ **using** *xfHyp* **by** *fastforce*

     from *solHyp* **and** *xfHyp* **have** *2*:$((\lambda\ \tau.\ (sol\ s[xfList{\leftarrow}uInput]\ \tau)\ x)\ solves\text{-}ode$

      $(\lambda\tau\ r.\ f\ (sol\ s[xfList{\leftarrow}uInput]\ \tau)))\ \{0..t\}\ UNIV \wedge (sol\ s[xfList{\leftarrow}uInput]\ 0)$
$x = s\ x$
      using *solvesStoreIVP-def tHyp* **by** *fastforce*

     from *tHyp* **and** *contHyp* **have** $\forall\ xf \in set\ xfList.\ unique\text{-}on\text{-}bounded\text{-}closed\ 0$
$\{0..t\}\ (s\ (\pi_1\ xf))$

```
    (λτ r. (π₂ 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) (λτ 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 = φs 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 = φs t x
      proof(cases y ∈ set (map π₁ xfList))
      case True
        then obtain f where xfHyp:(y, f) ∈ set xfList by fastforce
        from tHyp and Fsolves have φs t x = f (φs 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 φs 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)). (π₂ xf) (sol s[xfList←uInput] t))
and solHyp:∀ s. solvesStoreIVP (λ t. (sol s[xfList←uInput] t)) xfList s
and uniqHyp:∀ s. ∀ φs. φs solvesTheStoreIVP xfList withInitState s ⟶
(∀ t ≥ 0. ∀ xf ∈ set xfList. ∀ r ∈ {0..t}. (π₂ xf) (φs r) = (π₂ xf) (sol s[xfList←uInput]
r))
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(rule-tac uInput=uInput in dSolve)
prefer 2 subgoal proof(clarify)
fix s::real store and φs::real ⇒ real store and t::real
assume isSol:solvesStoreIVP φs xfList s and sHyp:0 ≤ t
from this and uniqHyp have ∀ xf ∈ set xfList. ∀ t ∈ {0..t}.
(π₂ xf) (φs t) = (π₂ xf) (sol s[xfList←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*. ((λ*t*. (*sol s*[*xfList*$\leftarrow$*uInput*] *t*) (π_1 *xf*)) *has-vderiv-on*
(λ*t*. 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*. ((λ*t*. *F t* (π_1 *xf*)) *solves-ode* (λ*t 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$$\{0..t\}$. $\forall$ *xf* $\in$ *set xfList*.
((λ *t*. *F t* (π_1 *xf*)) *has-vector-derivative F r* (∂ (π_1 *xf*))) (*at r within* $\{0..t\}$)
proof(*clarify*, *rename-tac t r x f*)
fix *x f* **and** *t r*::*real*
assume *tHyp*:*0* $\leq$ *t* **and** *xfHyp*:(*x*, *f*) $\in$ *set xfList* **and** *rHyp*:*r* $\in$ $\{0..t\}$
from *this* **and** *vars* **have** ((λ*t*. *F t x*) *solves-ode* (λ*t r*. *f* (*F t*))) $\{0..t\}$ *UNIV*
using *tHyp* **by** *fastforce*
hence $*$:$\forall$ *r*$\in$$\{0..t\}$. (($\lambda$ *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 $\forall$ *t* $\geq$ *0*. $\forall$ *r*$\in$$\{0..t\}$. $\forall$ *xf* $\in$ *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 $((\lambda t.\ F\ t\ (\pi_1\ (x, f)))$ *has-vector-derivative* $F\ r\ (\partial\ (\pi_1\ (x, f))))$ $($*at* r *within* $\{0..t\})$
using $*$ *rHyp* **by** *auto*
qed

lemma *derivationLemma-baseCase*:
fixes F::*real* $\Rightarrow$ *real store*
assumes *solves*:*solvesStoreIVP F xfList a*
shows $\forall\ x \in (UNIV - varDiffs).\ \forall\ t \geq 0.\ \forall r \in \{0..t\}.$
$((\lambda\ t.\ F\ t\ x)$ *has-vector-derivative* $F\ r\ (\partial\ x))$ $($*at* r *within* $\{0..t\})$
proof
fix x
assume $x \in UNIV - varDiffs$
then have *notVarDiff*:$\forall\ z.\ x \neq \partial\ z$ **using** *varDiffs-def* **by** *fastforce*
show $\forall t \geq 0.\ \forall r \in \{0..t\}.\ ((\lambda t.\ F\ t\ x)$ *has-vector-derivative* $F\ r\ (\partial\ x))$ $($*at* r *within* $\{0..t\})$
proof(*cases* $x \in$ *set* (*map* π_1 *xfList*))
case *True*
from *this* **and** *solves* **have** $\forall\ t \geq 0.\ \forall r \in \{0..t\}.\ \forall\ xf \in$ *set xfList*.
$((\lambda\ t.\ F\ t\ (\pi_1\ xf))$ *has-vector-derivative* $F\ r\ (\partial\ (\pi_1\ xf)))$ $($*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*:$\forall\ t \geq 0.\ F\ t\ x = a\ x$
using *solves-store-ivpD*(*1*) **by** (*simp add*: *varDiffs-def*)
have *constD*:$\forall\ t \geq 0.\ \forall r \in \{0..t\}.\ ((\lambda\ r.\ a\ x)$ *has-vector-derivative* $0)$ $($*at* r *within* $\{0..t\})$
by (*auto intro*: *derivative-eq-intros*)
{fix t r::*real*
assume $t \geq 0$ **and** $r \in \{0..t\}$
hence $((\lambda\ s.\ a\ x)$ *has-vector-derivative* $0)$ $($*at* r *within* $\{0..t\})$ **by** (*simp add*: *constD*)
moreover have $\bigwedge s.\ s \in \{0..t\} \Longrightarrow (\lambda\ r.\ F\ r\ x)\ s = (\lambda\ r.\ a\ x)\ s$
using *const* **by** (*simp add*: $\langle 0 \leq t \rangle$)
ultimately have $((\lambda\ s.\ F\ s\ x)$ *has-vector-derivative* $0)$ $($*at* r *within* $\{0..t\})$
using *has-vector-derivative-transform* **by** (*metis* $\langle r \in \{0..t\} \rangle$)**}**
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* η)
  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* η)
  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** $\vartheta{::}trms$ **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*) ∈ *ODEsystem xfList with G*
from *this* **obtain** *t*::*real* **and** *F*::*real* ⇒ *real store*
where *tcHyp*:*t*≥*0* ∧ *F t* = *c* ∧ *solvesStoreIVP F xfList a* ∧ (∀ *r*∈{*0*..*t*}. *G* (*F r*))

using *guarDiffEqtn-def* **by** *auto*
then have ∀ *x*. *x* ∉ *varDiffs* ⟶ *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*:∀ *r*∈{*0*..*t*}. ((λ*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 ∀ *r*∈{*0*..*t*}. ∀ *xf* ∈ *set xfList*. *F r* (∂ (π_1 *xf*)) = π_2 *xf* (*F r*)
using *tcHyp solves-store-ivpD*(*3*) **by** *fastforce*
hence ∀ *r*∈{*0*..*t*}. $[\![\partial_t\ \eta]\!]_t$ (*F r*) = $[\![$((*map* (*vdiff* ∘ π_1) *xfList*) ⊗ *uInput*) ⟨∂_t η⟩$]\!]_t$
(*F r*)
using *tcHyp diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** ∀ *r*∈{*0*..*t*}. $[\![$((*map* (*vdiff* ∘ π_1) *xfList*) ⊗ *uInput*)⟨∂_t
η⟩$]\!]_t$ (*F r*) = *0*
using *solves-store-ivpD*(*2*) *tcHyp* **by** *fastforce*
ultimately have ∀ *r*∈{*0*..*t*}. ((λ*s*. $[\![\eta]\!]_t$ (*F s*)) *has-vector-derivative 0*) (*at r within*
{*0*..*t*})
using *obs2* **by** *auto*
from *this* **and** *tcHyp* **have** ∀ *s*∈{*0*..*t*}. ((λ*x*. $[\![\eta]\!]_t$ (*F x*)) *has-derivative* (λ*x*. *x* $*_R$
0))
(*at s within* {*0*..*t*}) **by** (*metis has-vector-derivative-def*)
hence $[\![\eta]\!]_t$ (*F t*) − $[\![\eta]\!]_t$ (*F 0*) = (λ*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 ∀ *st*. *G st* ⟶ (∀ *str*. *str* ∉ (π_1(|*set xfList*|)) ⟶ *st* (∂ *str*) = *0*) ⟶
$[\![$((*map* (*vdiff* ∘ π_1) *xfList*) ⊗ *uInput*) ⟨∂_t η⟩$]\!]_t$ *st* = *0*
and *termVarsHyp*:*trmVars* η ⊆ (*UNIV* − *varDiffs*)
and *listsHyp*:*map* π_2 *xfList* = *map tval uInput*
and *eta-f*:*f* = $[\![\eta]\!]_t$
shows *PRE* (λ *s*. *f s* = *0*) (*ODEsystem xfList with G*) *POST* (λ *s*. *f s* = *0*)
using *eta-f* **proof**(*clarsimp*)
fix *a b*
assume (*a*, *b*) ∈ ⌈λ*s*. $[\![\eta]\!]_t$ *s* = *0*⌉ **and** *f* = $[\![\eta]\!]_t$
from *this* **have** *aHyp*:*a* = *b* ∧ $[\![\eta]\!]_t$ *a* = *0* **by** (*metis* (*full-types*) *d-p2r rdom-p2r-contents*)
have $[\![\eta]\!]_t$ *a* = *0* ⟶ (∀ *c*. (*a*,*c*) ∈ (*ODEsystem xfList with G*) ⟶ $[\![\eta]\!]_t$ *c* = *0*)
using *assms dInvForTrms-prelim* **by** *metis*
from *this* **and** *aHyp* **have** ∀ *c*. (*a*,*c*) ∈ (*ODEsystem xfList with G*) ⟶ $[\![\eta]\!]_t$ *c* =
0 **by** *blast*
thus (*a*, *b*) ∈ *wp* (*ODEsystem xfList with G*) ⌈λ*s*. $[\![\eta]\!]_t$ *s* = *0*⌉
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 $[\![\partial_P\ \varphi]\!]_P\ (F\ t) = [\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\restriction\partial_P\ \varphi\restriction]\!]_P\ (F\ t)$
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 $0 \leq [\![\eta]\!]_t\ a \longrightarrow (\forall\, c.\ (a,\ c) \in \mathit{ODEsystem\ xfList\ with\ G}\ \longrightarrow 0 \leq [\![\eta]\!]_t\ c)$
proof(*clarify*)
fix *c* **assume** aHyp:$[\![\eta]\!]_t\ a \geq 0$ **and** cHyp:$(a,\ c) \in \mathit{ODEsystem\ xfList\ with\ G}$
from *this* **obtain** t::*real* **and** F::*real* $\Rightarrow$ *real store*
where tcHyp:$t{\geq}0 \wedge F\ t = c \wedge \mathit{solvesStoreIVP}\ F\ \mathit{xfList}\ a \wedge (\forall\, r{\in}\{0..t\}.\ G\ (F\ r))$

using *guarDiffEqtn-def* **by** *auto*
then have $\forall\, x.\ x \notin \mathit{varDiffs} \longrightarrow F\ 0\ x = a\ x$ **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 $(\forall\, t{\geq}0.\ \forall\ \mathit{xf} \in \mathit{set\ xfList}.\ F\ t\ (\partial\ (\pi_1\ \mathit{xf})) = \pi_2\ \mathit{xf}\ (F\ t))$
using *tcHyp solves-store-ivpD*(*3*) **by** *blast*
from *this* **and** *tcHyp* **have** $\forall\, r{\in}\{0..t\}.\ [\![\partial_t\ \eta]\!]_t\ (F\ r) =$
$[\![((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ \mathit{xfList}) \otimes \mathit{uInput})\ \langle\partial_t\ \eta\rangle]\!]_t\ (F\ r)$
using *diff-subst-prprty-4terms termVarsHyp listsHyp* **by** *fastforce*
also from *substHyp* **have** $\forall\, r{\in}\{0..t\}.\ [\![((\mathit{map}\ (\mathit{vdiff} \circ \pi_1)\ \mathit{xfList}) \otimes \mathit{uInput})\ \langle\partial_t$
$\eta\rangle]\!]_t\ (F\ r) \geq 0$
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 $[\![((\mathit{map}\ (\mathit{vdiff}\ \circ \pi_1)\ \mathit{xfList}) \otimes \mathit{uInput})\restriction\partial_P\ (\vartheta \prec \eta)\restriction]\!]_P\ st$
shows $[\![((\mathit{map}\ (\mathit{vdiff}{\circ}\pi_1)\ \mathit{xfList}) \otimes \mathit{uInput})\langle\partial_t\ (\eta \oplus (\ominus\ \vartheta))\rangle]\!]_t\ st \geq 0$
using *assms* **by**(*auto*)

lemma *leq-pval-to-tval*:
assumes $[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\upharpoonright\partial_P\ (\vartheta \preceq \eta)\upharpoonright]\!]_P\ st$
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*:$\forall\ st.\ G\ st \longrightarrow\ (\forall str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\upharpoonright\partial_P\ \varphi\upharpoonright]\!]_P\ st$
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*:$\forall st.\ G\ st \longrightarrow\ (\forall str.\ str \notin (\pi_1(\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff \circ \pi_1)\ xfList) \otimes uInput)\upharpoonright\partial_P\ (\vartheta \doteq \eta)\upharpoonright]\!]_P\ st$ **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 $phi\text{-}p{:}P = [\![\varphi]\!]_P$
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 $\forall\ st.\ G\ st \longrightarrow (\forall\, str.\ str \notin (\pi_1 (\!|set\ xfList|\!)) \longrightarrow st\ (\partial\ str) = 0) \longrightarrow$
$[\![((map\ (vdiff\ \circ \pi_1)\ xfList) \otimes uInput)\restriction \partial_P\ \varphi \restriction]\!]_P\ st$
and $termVarsHyp{:}propVars\ \varphi \subseteq (UNIV - varDiffs)$
and $listsHyp{:}map\ \pi_2\ xfList = map\ tval\ uInput$
and $impls{:}\lceil P \rceil \subseteq \lceil F \rceil \wedge \lceil F \rceil \subseteq \lceil Q \rceil$
and $phi\text{-}f{:}F = [\![\varphi]\!]_P$
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* ($\lambda s.$ $G\ s \wedge F\ s$)) *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

6.4.5 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*:(($\lambda t.$ φ_s *t v*) *solves-ode* ($\lambda t\ r.$ φ_s *t a*)) $\{0..t\}$ *UNIV* $\wedge$ φ_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* ($\lambda t\ r.$ φ_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 φ_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 (λ *s. s ''y''* < *s ''x''* $\wedge$ *s ''v''* $\geq$ *0* $\wedge$ *s ''a''* > *0*)
(*ODEsystem* [(*''x''*,(λ *s. s ''v''*)),(*''v''*,(λ *s. s ''a''*))] *with* (λ *s. True*))
POST (λ *s.* (*s ''y''* < *s ''x''*))
apply(*rule-tac uInput*=[λ *t s. s ''a''* $\cdot$ *t* ^ *2*/*2* + *s ''v''* $\cdot$ *t* + *s ''x''*,
λ *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* φ_s *t r* **apply**(*rule flow-vel-is-galilean-vel*[*of* φ_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 φ_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* $\wedge$ φ_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 $(\lambda\ s.\ 0 \le s\ ''x'' \wedge s\ ''x'' = H \wedge s\ ''v'' = 0 \wedge s\ ''g'' > 0 \wedge 1 \ge c \wedge c \ge 0)$
(((*ODEsystem* $[(''x'', \lambda\ s.\ s\ ''v''),(''v'',\lambda\ s.\ -\ s\ ''g'')]$ *with* $(\lambda\ s.\ 0 \le s\ ''x'')$));
(*IF* $(\lambda\ s.\ s\ ''x'' = 0)$ *THEN* $(''v'' ::= (\lambda\ s.\ -\ c \cdot s\ ''v''))$ *ELSE* $(''v'' ::= (\lambda\ s.\ s\ ''v''))$ *FI*))
POST $(\lambda\ s.\ 0 \le s\ ''x'' \wedge s\ ''x'' \le H)$
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*:
$\quad PRE\ (\lambda\ s.\ s\ ''x'' = 0)$
$\quad (ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''x'' + 1))]\ with\ (\lambda\ s.\ s\ ''x'' \geq 0))$
$\quad POST\ (\lambda\ s.\ s\ ''x'' \geq 0)$
using *dWeakening* **by** *blast*

lemma *system-where-the-guard-implies-the-postcondition2*:
$\quad PRE\ (\lambda\ s.\ s\ ''x'' = 0)$
$\quad (ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''x'' + 1))]\ with\ (\lambda\ s.\ s\ ''x'' \geq 0))$
$\quad POST\ (\lambda\ s.\ s\ ''x'' \geq 0)$
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*:
$\quad PRE\ (\lambda\ s.\ (s\ ''x'') \cdot (s\ ''x'') + (s\ ''y'') \cdot (s\ ''y'') - (s\ ''r'') \cdot (s\ ''r'') = 0)$
$\quad (ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''y'')),(''y'',(\lambda\ s.\ -\ s\ ''x''))]\ with\ G)$
$\quad POST\ (\lambda\ s.\ (s\ ''x'') \cdot (s\ ''x'') + (s\ ''y'') \cdot (s\ ''y'') - (s\ ''r'') \cdot (s\ ''r'') = 0)$
apply($rule\text{-}tac\ \eta{=}(t_V\ ''x'')\odot(t_V\ ''x'') \oplus (t_V\ ''y'')\odot(t_V\ ''y'') \oplus (\ominus(t_V\ ''r'')\odot(t_V\ ''r''))$
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*:
$\quad PRE\ (\lambda\ s.\ s\ ''x'' > s\ ''y'' \wedge s\ ''v'' > 0)$
$\quad (ODEsystem\ [(''x'',\ \lambda\ s.\ s\ ''v'')]\ with\ (\lambda\ s.\ True))$
$\quad POST\ (\lambda\ s.\ s\ ''x''>\ s\ ''y'')$
apply($rule\text{-}tac\ C = \lambda\ s.\ \ s\ ''v'' > 0$ **in** *dCut*)
apply($rule\text{-}tac\ \varphi = (t_C\ 0) \prec (t_V\ ''v'')$ **and** $uInput{=}[t_V\ ''v'']$**in** *dInvFinal*)
apply(*simp-all add*: *vdiff-def varDiffs-def*, *clarify*, *erule-tac* $x{=}''v''$ **in** *allE*, *simp*)
apply($rule\text{-}tac\ C = \lambda\ s.\ \ s\ ''x'' > s\ ''y''$ **in** *dCut*)
apply($rule\text{-}tac\ \varphi{=}(t_V\ ''y'') \prec (t_V\ ''x'')$ **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*:
$\quad PRE\ (\lambda\ s.\ s\ ''y'' < s\ ''x''\ \ \wedge s\ ''v'' \geq 0 \wedge s\ ''a'' > 0)$
$\quad (ODEsystem\ [(''x'',(\lambda\ s.\ s\ ''v'')),(''v'',(\lambda\ s.\ s\ ''a''))]\ with\ (\lambda\ s.\ True))$
$\quad POST\ (\lambda\ s.\ (s\ ''y'' < s\ ''x''))$
apply($rule\text{-}tac\ C = \lambda\ s.\ \ s\ ''a'' > 0$ **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